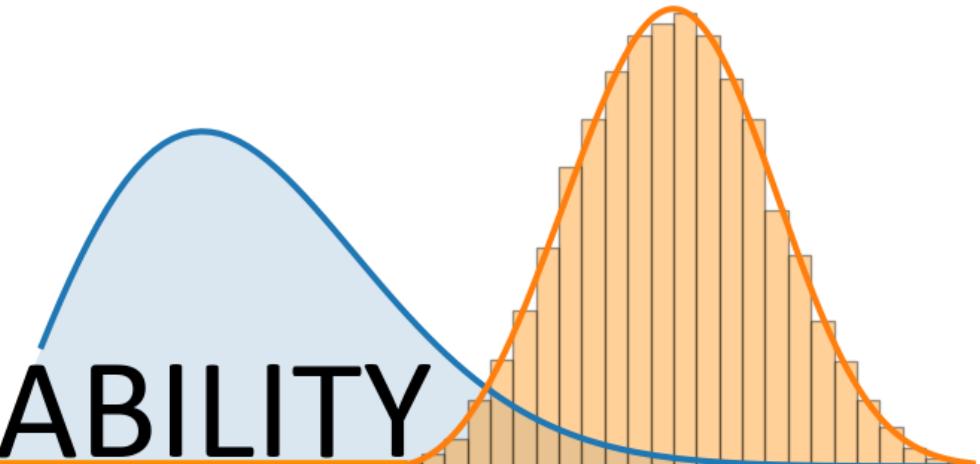


reliability

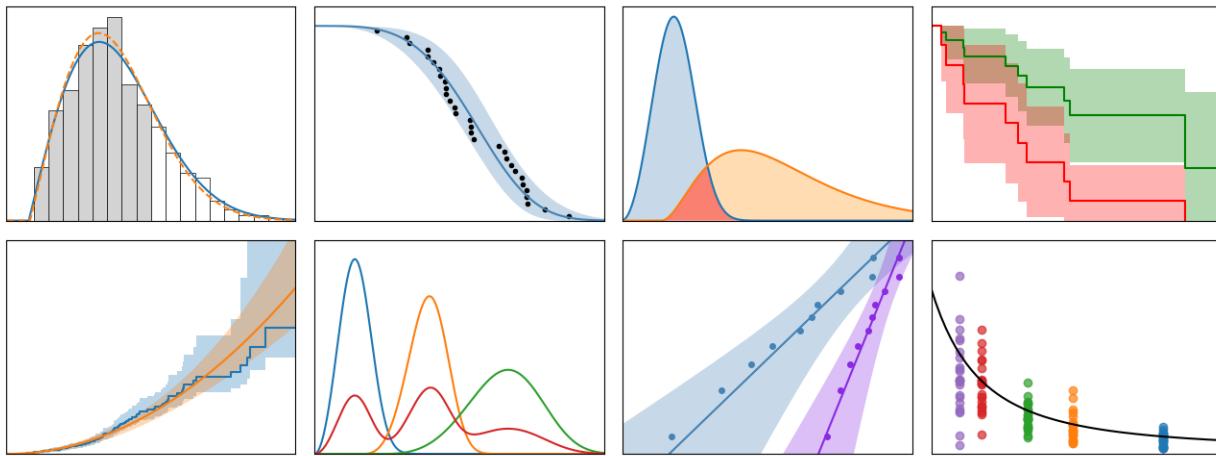
Jan 04, 2025



RELIABILITY

A Python library for reliability engineering

reliability is a Python library for [reliability engineering](#) and [survival analysis](#). It significantly extends the functionality of `scipy.stats` and also includes many specialist tools that are otherwise only available in proprietary software.



If you frequently use the Python Reliability Library, please consider filling out a quick [survey](#) to help guide the development of the library and this documentation.



RELIABILITY
A Python library for reliability engineering

QUICKSTART FOR RELIABILITY

1.1 Installation and upgrading

If you are new to using Python, you will first need to install a [Python 3](#) interpreter and also install an IDE so that you can interact with the code. There are many [good IDEs](#) available including [Pycharm](#), [Spyder](#) and [Jupyter](#).

Once you have Python installed, to install *reliability* for the first time, open your command prompt and type:

```
pip install reliability
```

To upgrade a previous installation of *reliability* to the most recent version, open your command prompt and type:

```
pip install --upgrade reliability
```

If you would like to be notified by email of when a new release of *reliability* is uploaded to PyPI, there is a free service to do exactly that called [NewReleases.io](#).

1.2 A quick example

In this example, we will create a Weibull Distribution, and from that distribution we will draw 20 random samples. Using those samples we will Fit a 2-parameter Weibull Distribution. The fitting process generates the probability plot. We can then access the distribution object to plot the survival function.

```
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_2P
from reliability.Probability_plotting import plot_points
import matplotlib.pyplot as plt

dist = Weibull_Distribution(alpha=30, beta=2) # creates the distribution object
data = dist.random_samples(20, seed=42) # draws 20 samples from the distribution.
# Seeded for repeatability
plt.subplot(121)
fit = Fit_Weibull_2P(failures=data) # fits a Weibull distribution to the data and
# generates the probability plot
plt.subplot(122)
fit.distribution.SF(label='fitted distribution') # uses the distribution object from
# Fit_Weibull_2P and plots the survival function
dist.SF(label='original distribution', linestyle='--') # plots the survival function of
# the original distribution
plot_points(failures=data, func='SF') # overlays the original data on the survival
# function
```

(continues on next page)

(continued from previous page)

```
plt.legend()
plt.show()

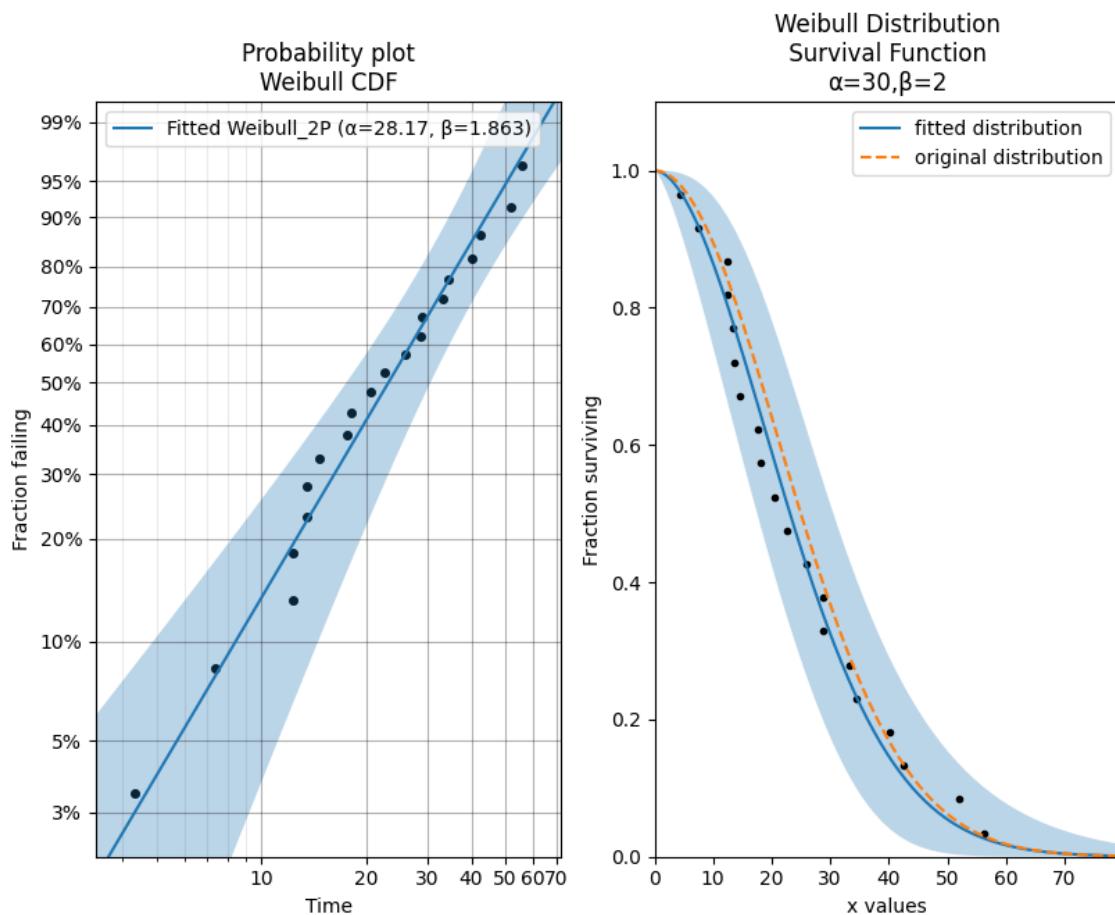
"""

Results from Fit_Weibull_2P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Failures / Right censored: 20/0 (0% right censored)
```

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha	28.1696	3.57032	21.9733	36.1131
Beta	1.86309	0.32449	1.32428	2.62111

Goodness of fit Value
 Log-likelihood -79.5482
 AICc 163.802
 BIC 165.088
 AD 0.837278

```
""
```



A key feature of *reliability* is that probability distributions are created as objects, and these objects have many properties (such as the mean) that are set once the parameters of the distribution are defined. Using the dot operator allows us to

access these properties as well as a large number of methods (such as drawing random samples as seen in the example above).

Each distribution may be visualised in five different plots. These are the Probability Density Function (PDF), Cumulative Distribution Function (CDF), Survival Function (SF) [also known as the reliability function], Hazard Function (HF), and the Cumulative Hazard Function (CHF). Accessing the plot of any of these is as easy as any of the other methods. Eg. `dist.SF()` in the above example is what plots the survival function using the distribution object that was returned from the fitter.

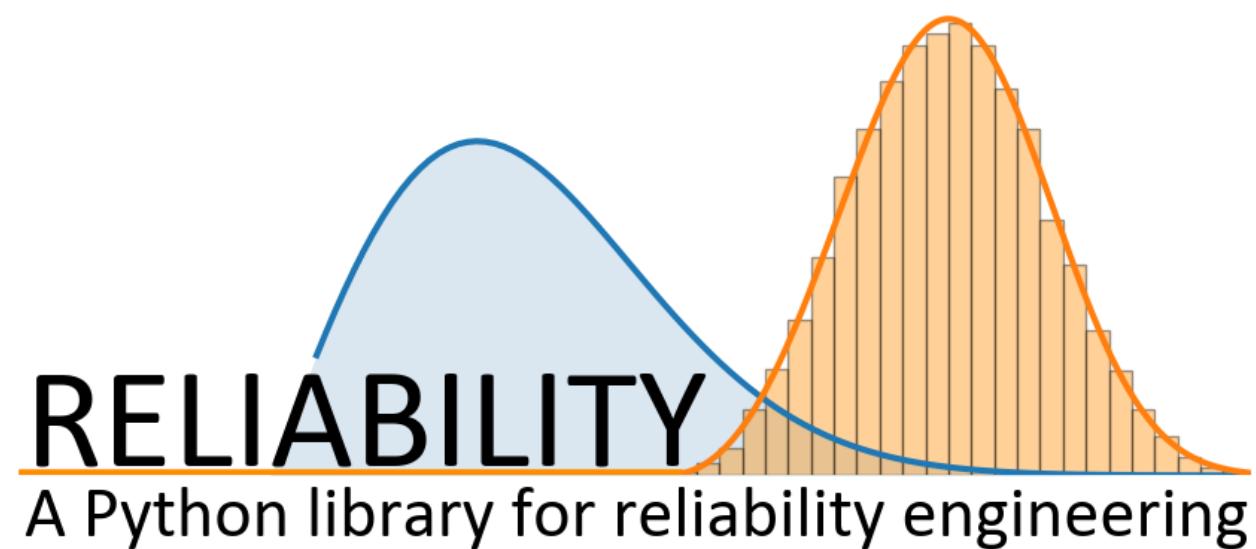
INTRODUCTION TO THE FIELD OF RELIABILITY ENGINEERING

Reliability engineering is a field of study that deals with the estimation, prevention, and management of failures by combining statistics, risk analysis, and physics. By understanding how failures may occur or have occurred, we are able to better predict the lifespan of a product or system, allowing us to manage its lifecycle and the risks associated with its failure. All engineering systems, components, and structures will eventually fail, and knowing how and when that failure will occur is of great interest to the owners and operators of those systems. Due to the similarities between the lifecycle of engineering systems and the lifecycle of humans, the field of study known as [survival analysis](#) has many concepts that are used in reliability engineering.

Everyone is acutely aware of the importance of reliability, particularly when something doesn't work at a time we expect it to. Whether it be your car not starting, your television failing, or the chance of being delayed on the runway because your aircraft's airconditioning unit just stopped working, we know that system failure is something we all want to avoid. When it can't be avoided, we at least want to know when it is likely to occur so we can conduct preventative maintenance before the need for corrective maintenance arises. Reliability engineering is most frequently used for systems which are of critical safety importance (such as in the nuclear industry), or in systems which are numerous (such as vehicles or electronics) where the cost of fleetwide reliability problems can quickly become very expensive.

Much of reliability engineering involves the analysis of data (such as time to failure data), to uncover the patterns in how failures occur. Once we understand how things are failing, we can use those patterns to forecast how the failures will occur throughout the lifetime of a population of items, or the lifetime of one or more repairable items. It is the data analysis part of reliability engineering that this Python library is designed to help with.

Further reading is available on [Wikipedia](#) and in many [other reliability resources](#).



RECOMMENDED RESOURCES

The following collection of resources are things I have found useful during my reliability engineering studies and also while writing the Python reliability library. There are many other resources available (especially textbooks and academic papers), so I encourage you to do your own research. If you find something you think is worth adding here, please send me an email (alpha.reliability@gmail.com).

Textbooks

- Reliability Engineering and Risk Analysis: A practical Guide, Third Edition (2017), by M. Modarres, M. Kaminskiy, and V. Krivtsov.
- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson.
- [Probability Distributions Used in Reliability Engineering \(2011\)](#), by A. O'Conner, M. Modarres, and A. Mosleh.
- Practical Reliability Engineering, Fifth Edition (2012), by P. O'Conner and A. Kleyner.
- Recurrent Events Data Analysis for Product Repairs, Disease Recurrences, and Other Applications (2003), by W. Nelson
- Reliability Analysis Using MINITAB and Python (2022), by J. Hwang. This textbook provides many examples using *reliability* and also a few examples with *surpyval*. The version of *reliability* used in the book is 0.8.1 which is significantly different from the most recent version.
- Reliasoft has compiled a much more comprehensive [list of textbooks](#).
- The reliability analytics toolkit (linked below in free online tools and calculators) has also compiled a much more comprehensive [list of textbooks](#).

Free software

- [Lifelines](#) - a Python library for survival analysis. Very powerful collection of tools, only a few of which overlap with the Python reliability library.
- [Surpyval](#) - a Python library for survival analysis. Similar to reliability, but with an API more aligned with Scipy. Mostly focussed on fitting models (probability distributions and nonparametric).
- [Parameter Solver v3.0](#) - a biostatistics tool for quickly making some simple calculations with probability distributions.
- [Orange](#) - a standalone data mining and data visualization program that runs using Python. Beautifully interactive data analysis workflows with a large toolbox. Not much reliability related content but good for data preprocessing.
- [R \(Programming Language\)](#) - R is one of the most popular programming languages for data science, and it has several libraries that are targeted towards reliability engineering and survival analysis. These include [WeibullR](#), [abrem](#), [flexsurv](#), and [survival](#).
- [CumFreq](#) - a program for cumulative frequency analysis with probability distribution fitting for a wide range of distributions. Limited functionality beyond fitting distributions.

- [OpenTURNS](#) - a Python library for the treatment of uncertainties, risks and statistics. This library contains many powerful statistical functions, some of which are applicable to reliability engineering (mainly the fitting of distributions). The syntax of the library requires many steps as shown in the tutorials.

Paid software

The listing of a software package here does not imply my endorsement, and is only intended to give readers an understanding of the broad range of reliability engineering software packages that are available. It is difficult to find a comprehensive list of software resources since most developers of proprietary software rarely acknowledge the existence of any software other than their own. I have not used most of the paid software listed here due to the high cost, so most of my comments in this section are based purely on the content from their websites.

- [Minitab](#) - a great collection of statistical tools. A few reliability focussed tools included.
- [Reliasoft](#) - the industry leader for reliability engineering software.
- [SAS JMP](#) - lots of statistical tools for data modelling and visualization. A few purpose built reliability tools. Its utility for reliability engineering will depend on your application. SAS has also released the [SAS University Edition](#) which is a free software package that runs in VirtualBox and offers a reduced set of tools compared to the paid package.
- [PTC Windchill](#) - a powerful tool for risk and reliability. Similar to Reliasoft but it forms one part of the larger PTC suite of tools.
- [Isograph Reliability Workbench](#) - A collection of tools designed specifically for reliability engineering.
- [Item Software](#) - A collection of tools for reliability engineering including FMECA, fault trees, reliability prediction, and many others.
- [SuperSMITH](#) - This software is designed specifically for reliability engineering and has many useful tools. The user interface looks like it is from the early 1990s but the methods used are no less relevant today. This software was developed alongside the New Weibull Handbook, an excellent resource for interpreting the results of reliability engineering software.
- [RAM Commander](#) - A software tool for Reliability and Maintainability Analysis and Prediction, Spares Optimisation, FMEA/FMECA, Testability, Fault Tree Analysis, Event Tree Analysis and Safety Assessment.
- [RelCalc](#) - RelCalc for Windows automates the reliability prediction procedure of Telcordia SR-332, or MIL-HDBK-217, providing an alternative to tedious, time consuming, and error prone manual methods.
- [Relyence](#) - Relyence offers a range of products, similar to Reliasoft, each with a focus on a different area including Life Data Analysis, Accelerated Life Testing, Reliability Block Diagrams, FMEA, and several more.
- [@RISK](#) - A comprehensive Excel addon that allows for distribution fitting, reliability modelling, MC simulation and much more.
- [Quanterion Automated Reliability Toolkit \(QuART\)](#) - A collection of reliability tools including reliability prediction, FMECA, derating, stress-strength interference, and many other. Quanterion produces several software products so their tools are not all available in one place.
- [TopEvent FTA](#) - Fault Tree Analysis software. Tailored specifically for fault tree analysis so it lacks other RAM tools but it is good at its intended function. A demo version is available with size and data export limitations.
- [Maintenance Aware Design \(MADe\)](#) - FMECA and RCM software that is extremely useful at the product design stage to inform the design and service plan which then improves the inherent reliability and maintainability. There is an academic license which allows non-profit users to run the software for free.

Free online tools and calculators

- [Reliability Analytics Toolkit](#) - a collection of tools which run using the Google App Engine. Includes a [tool](#) for fitting a Weibull_2P distribution.

- [Weibull App](#) - An online tool for fitting a Weibull_2P distribution. Download the example template to see what format the app is expecting your data to be in before you can upload your own data. The backend is powered by the abrem R package. This tool has limited applications beyond fitting a Weibull_2P distribution.
- [Distributome](#) - Provides PDF and CDF of a large number of probability distributions that can be easily changed using sliders for their parameters. It also includes a quantile / CDF calculator. Similar to the Distribution calculator below.
- [Distribution Calculator](#) - Provides PDF and CDF of a large number of probability distributions that can be easily changed using sliders for their parameters. It also includes a quantile / CDF calculator. Similar to Distributome above.
- [Kijima G-renewal process](#) - an online calculator for simulating the G-renewal process.
- [Prediction of future recurrent events](#) - an online calculator for predicting future recurrent events with different underlying probability functions.
- [Maintenance optimization](#) - an online calculator for optimal replacement policy (time) under Kijima imperfect repair model.
- [e-Fatigue](#) - This website provides stress concentration factors (Kt) for various notched geometries. You will need this if using the functions for fracture mechanics in the Physics of Failure section.
- [Fault Tree Analyser](#) - A simple online tool where you can build a fault tree, give each branch a failure rate and run a variety of reports including reliability prediction at time, minimal cut sets, and several others.
- [Wolfram Alpha](#) - an amazing computational knowledge engine. Great for checking your calculations.
- [Derivative calculator](#) - calculates derivatives. Slightly more user friendly input method than Wolfram alpha and doesn't time out as easily for big calculations.
- [Integral calculator](#) - calculates integrals. Slightly more user friendly input method than Wolfram alpha and doesn't time out as easily for big calculations.
- [GeoGebra](#) - An interactive calculator that is extremely useful for plotting equations. Also includes many mathematical operations (such as integrals and derivatives) that allow you to keep your equations in symbolic form. You can download your current calculator to save it. The only downside is that there are not many probability distribution functions inbuilt so you will need to enter the [equations](#) manually.
- [NewReleases.io](#) - This website allows you to setup email notifications for when a new release of *reliability* (or any other package) is uploaded to PyPI. While not exactly a tool for reliability engineering, it is very useful to let you know when it's time to upgrade your version of *reliability*.

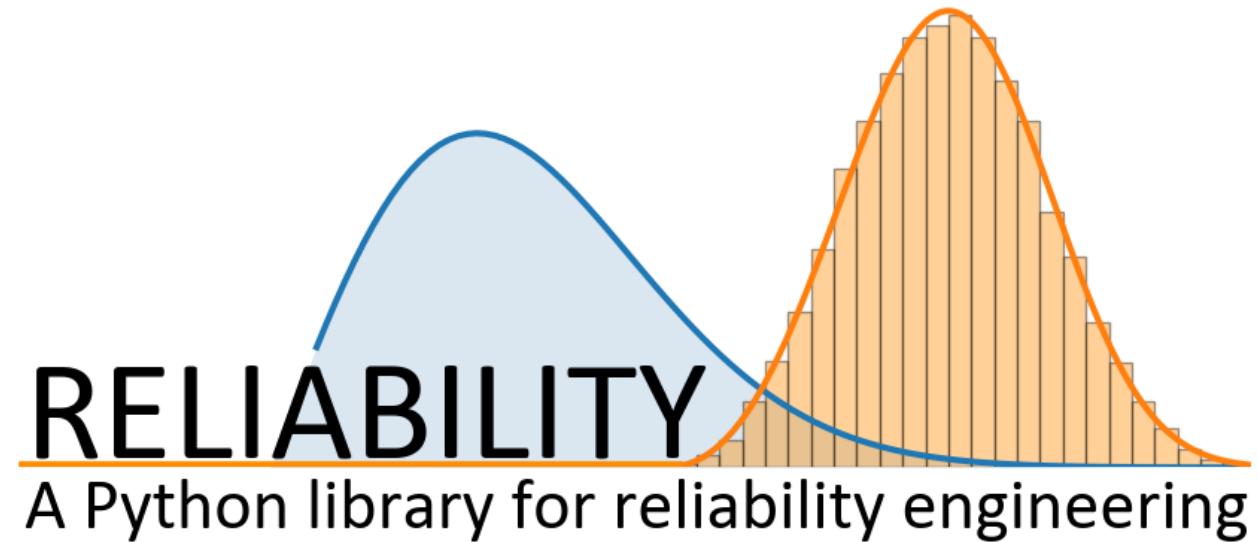
Online information resources

- [Reliawiki](#) - an excellent reference written by Reliasoft that is intended largely as a guide to reliability engineering when using Reliasoft's software but is equally as good to understand concepts without using their software.
- [Reliasoft's Accelerated Life Testing Data Analysis Reference](#)
- [Reliasoft's collection of Military Directives, Handbooks and Standards Related to Reliability](#)
- [Univariate distributions relationships](#) - a great interactive diagram for understanding more about probability distributions and how they are related. Some strange parametrisations are used in the documentation.
- [Cross Validated](#) - a forum for asking statistics and mathematics questions. Check for existing answers before posting your own question.
- [Stack Overflow](#) - a forum for programmers where you can post questions and answers related to programming. Check for existing answers before posting your own question.
- [Wikipedia](#) - it's always worth checking if there's an article on there about the topic you're trying to understand.

Getting free access to academic papers

- [arXiv](#) - a database run by Cornell university that provides open access to over 1.5 million academic papers that have been submitted. If you can't find it here then check on Sci-Hub.
- [Sci-Hub](#) - paste in a DOI to get a copy of the academic paper. Accessing academic knowledge should be free and this site makes it possible.

Found a broken link? If so, please email me (alpha.reliability@gmail.com) so I can update it.



EQUATIONS OF SUPPORTED DISTRIBUTIONS

The following expressions provide the equations for the Probability Density Function (PDF), Cumulative Distribution Function (CDF), Survival Function (SF) (this is the same as the reliability function $R(t)$), Hazard Function (HF), and Cumulative Hazard Function (CHF) of all supported distributions. Readers should note that there are many ways to write the equations for probability distributions and careful attention should be afforded to the parametrization to ensure you understand each parameter. For more equations of these distributions, see the textbook “Probability Distributions Used in Reliability Engineering” listed in [recommended resources](#).

4.1 Weibull Distribution

α = scale parameter ($\alpha > 0$)

β = shape parameter ($\beta > 0$)

Limits ($t \geq 0$)

$$\begin{aligned}\text{PDF:} \quad f(t) &= \frac{\beta t^{\beta-1}}{\alpha^\beta} e^{-(\frac{t}{\alpha})^\beta} \\ &= \frac{\beta}{\alpha} \left(\frac{t}{\alpha}\right)^{(\beta-1)} e^{-(\frac{t}{\alpha})^\beta}\end{aligned}$$

$$\text{CDF:} \quad F(t) = 1 - e^{-(\frac{t}{\alpha})^\beta}$$

$$\text{SF:} \quad R(t) = e^{-(\frac{t}{\alpha})^\beta}$$

$$\text{HF:} \quad h(t) = \frac{\beta}{\alpha} \left(\frac{t}{\alpha}\right)^{\beta-1}$$

$$\text{CHF:} \quad H(t) = \left(\frac{t}{\alpha}\right)^\beta$$

4.2 Exponential Distribution

λ = scale parameter ($\lambda > 0$)

Limits ($t \geq 0$)

$$\text{PDF:} \quad f(t) = \lambda e^{-\lambda t}$$

$$\text{CDF:} \quad F(t) = 1 - e^{-\lambda t}$$

$$\text{SF:} \quad R(t) = e^{-\lambda t}$$

$$\text{HF:} \quad h(t) = \lambda$$

$$\text{CHF:} \quad H(t) = \lambda t$$

Note that some parametrizations of the Exponential distribution (such as the one in `scipy.stats`) use $\frac{1}{\lambda}$ in place of λ .

4.3 Normal Distribution

μ = location parameter ($-\infty < \mu < \infty$)

σ = scale parameter ($\sigma > 0$)

Limits ($-\infty < t < \infty$)

$$\text{PDF: } f(t) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right]$$

$$= \frac{1}{\sigma} \phi\left[\frac{t-\mu}{\sigma}\right]$$

where ϕ is the standard normal PDF with $\mu = 0$ and $\sigma = 1$

$$\text{CDF: } F(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^t \exp\left[-\frac{1}{2}\left(\frac{\theta-\mu}{\sigma}\right)^2\right] d\theta$$

$$= \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{t-\mu}{\sigma\sqrt{2}}\right)$$

$$= \Phi\left(\frac{t-\mu}{\sigma}\right)$$

where Φ is the standard normal CDF with $\mu = 0$ and $\sigma = 1$

$$\text{SF: } R(t) = 1 - \Phi\left(\frac{t-\mu}{\sigma}\right)$$

$$= \Phi\left(\frac{\mu-t}{\sigma}\right)$$

$$\text{HF: } h(t) = \frac{\phi\left[\frac{t-\mu}{\sigma}\right]}{\sigma\left(\Phi\left[\frac{\mu-t}{\sigma}\right]\right)}$$

$$\text{CHF: } H(t) = -\ln\left[\Phi\left(\frac{\mu-t}{\sigma}\right)\right]$$

4.4 Lognormal Distribution

μ = scale parameter ($-\infty < \mu < \infty$)

σ = shape parameter ($\sigma > 0$)

Limits ($t \geq 0$)

$$\text{PDF: } f(t) = \frac{1}{\sigma t \sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{\ln(t)-\mu}{\sigma}\right)^2\right]$$

$$= \frac{1}{\sigma t} \phi\left[\frac{\ln(t)-\mu}{\sigma}\right]$$

where ϕ is the standard normal PDF with $\mu = 0$ and $\sigma = 1$

$$\text{CDF: } F(t) = \frac{1}{\sigma\sqrt{2\pi}} \int_0^t \frac{1}{\theta} \exp\left[-\frac{1}{2}\left(\frac{\ln(\theta)-\mu}{\sigma}\right)^2\right] d\theta$$

$$= \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{\ln(t)-\mu}{\sigma\sqrt{2}}\right)$$

$$= \Phi\left(\frac{\ln(t)-\mu}{\sigma}\right)$$

where Φ is the standard normal CDF with $\mu = 0$ and $\sigma = 1$

$$\text{SF: } R(t) = 1 - \Phi\left(\frac{\ln(t)-\mu}{\sigma}\right)$$

$$\text{HF: } h(t) = \frac{\phi\left[\frac{\ln(t)-\mu}{\sigma}\right]}{t\sigma\left(1-\Phi\left(\frac{\ln(t)-\mu}{\sigma}\right)\right)}$$

$$\text{CHF: } H(t) = -\ln\left[1 - \Phi\left(\frac{\ln(t)-\mu}{\sigma}\right)\right]$$

4.5 Gamma Distribution

α = scale parameter ($\alpha > 0$)

β = shape parameter ($\beta > 0$)

Limits ($t \geq 0$)

$$\text{PDF: } f(t) = \frac{t^{\beta-1}}{\Gamma(\beta)\alpha^\beta} e^{-\frac{t}{\alpha}}$$

where $\Gamma(x)$ is the complete gamma function. $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$

$$\text{CDF: } F(t) = \frac{1}{\Gamma(\beta)} \gamma\left(\beta, \frac{t}{\alpha}\right)$$

where $\gamma(x, y)$ is the lower incomplete gamma function. $\gamma(x, y) = \frac{1}{\Gamma(x)} \int_0^y t^{x-1} e^{-t} dt$

$$\text{SF: } R(t) = \frac{1}{\Gamma(\beta)} \Gamma\left(\beta, \frac{t}{\alpha}\right)$$

where $\Gamma(x, y)$ is the upper incomplete gamma function. $\Gamma(x, y) = \frac{1}{\Gamma(x)} \int_y^\infty t^{x-1} e^{-t} dt$

$$\text{HF: } h(t) = \frac{t^{\beta-1} \exp\left(-\frac{t}{\alpha}\right)}{\alpha^\beta \Gamma\left(\beta, \frac{t}{\alpha}\right)}$$

$$\text{CHF: } H(t) = -\ln \left[\frac{1}{\Gamma(\beta)} \Gamma\left(\beta, \frac{t}{\alpha}\right) \right]$$

Note that some parametrizations of the Gamma distribution use $\frac{1}{\alpha}$ in place of α . There is also an alternative parametrization which uses shape and rate instead of shape and scale. See [Wikipedia](#) for an example of this.

4.6 Beta Distribution

α = shape parameter ($\alpha > 0$)

β = shape parameter ($\beta > 0$)

Limits ($0 \leq t \leq 1$)

$$\begin{aligned} \text{PDF: } f(t) &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \cdot t^{\alpha-1} (1-t)^{\beta-1} \\ &= \frac{1}{B(\alpha, \beta)} \cdot t^{\alpha-1} (1-t)^{\beta-1} \end{aligned}$$

where $\Gamma(x)$ is the complete gamma function. $\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt$

where $B(x, y)$ is the complete beta function. $B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$

$$\begin{aligned} \text{CDF: } F(t) &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^t \theta^{\alpha-1} (1-\theta)^{\beta-1} d\theta \\ &= \frac{B_t(t|\alpha, \beta)}{B(\alpha, \beta)} \\ &= I_t(t|\alpha, \beta) \end{aligned}$$

where $B_t(t|x, y)$ is the incomplete beta function. $B_t(t|x, y) = \int_0^t \theta^{x-1} (1-\theta)^{y-1} d\theta$

where $I_t(t|x, y)$ is the regularized incomplete beta function which is defined in terms of the incomplete beta function and the complete beta function. $I_t(t|x, y) = \frac{B_t(t|x, y)}{B(x, y)}$

$$\text{SF: } R(t) = 1 - I_t(t|\alpha, \beta)$$

$$\text{HF: } h(t) = \frac{t^{\alpha-1} (1-t)}{B(\alpha, \beta) - B_t(t|\alpha, \beta)}$$

$$\text{CHF: } H(t) = -\ln [1 - I_t(t|\alpha, \beta)]$$

Note that there is a parameterization of the Beta distribution that changes the lower and upper limits beyond 0 and 1. For this parametrization, see the reference listed in the opening paragraph of this page.

4.7 Loglogistic Distribution

α = scale parameter ($\alpha > 0$)

β = shape parameter ($\beta > 0$)

Limits ($t \geq 0$)

$$\text{PDF: } f(t) = \frac{\left(\frac{\beta}{\alpha}\right)\left(\frac{t}{\alpha}\right)^{\beta-1}}{\left(1+\left(\frac{t}{\alpha}\right)^{\beta}\right)^2}$$

$$\begin{aligned} \text{CDF: } F(t) &= \frac{1}{1+\left(\frac{t}{\alpha}\right)^{-\beta}} \\ &= \frac{\left(\frac{t}{\alpha}\right)^{\beta}}{1+\left(\frac{t}{\alpha}\right)^{\beta}} \\ &= \frac{t^{\beta}}{\alpha^{\beta}+t^{\beta}} \end{aligned}$$

$$\text{SF: } R(t) = \frac{1}{1+\left(\frac{t}{\alpha}\right)^{\beta}}$$

$$\text{HF: } h(t) = \frac{\left(\frac{\beta}{\alpha}\right)\left(\frac{t}{\alpha}\right)^{\beta-1}}{1+\left(\frac{t}{\alpha}\right)^{\beta}}$$

$$\text{CHF: } H(t) = \ln\left(1 + \left(\frac{t}{\alpha}\right)^{\beta}\right)$$

There is another parameterization of the loglogistic distribution using μ and σ which is designed to look more like the parameterization of the [logistic distribution](#) and is related to the above parametrization by $\mu = \ln(\alpha)$ and $\sigma = \frac{1}{\beta}$. This parametrisation can be found [here](#).

4.8 Gumbel Distribution

μ = location parameter ($-\infty < \mu < \infty$)

σ = scale parameter ($\sigma > 0$)

Limits ($-\infty < t < \infty$)

$$\text{PDF: } f(t) = \frac{1}{\sigma} e^{z-e^z}$$

where $z = \frac{t-\mu}{\sigma}$

$$\text{CDF: } F(t) = 1 - e^{-e^z}$$

$$\text{SF: } R(t) = e^{-e^z}$$

$$\text{HF: } h(t) = \frac{e^z}{\sigma}$$

$$\text{CHF: } H(t) = e^z$$

The parametrization of the Gumbel Distribution shown above is also known as the Smallest Extreme Value (SEV) distribution. There are several types of extreme value distributions, and the article on [Wikipedia](#) is for the Largest Extreme Value (LEV) distribution. There is only a slight difference in the parametrisation between SEV and LEV distributions, but this change effectively flips the PDF about μ to give the LEV positive skewness (a longer tail to the right), while the SEV has negative skewness (a longer tail to the left).

4.9 Location shifting the distributions

Within *reliability* the parametrization of the Exponential, Weibull, Gamma, Lognormal, and Loglogistic distributions allows for location shifting using the gamma parameter. This will simply shift the distribution's lower limit to the right from 0 to γ . In the location shifted form of the distributions, the equations listed above are almost identical, except everywhere you see t replace it with $t - \gamma$. The reason for using the location shifted form of the distribution is because some phenomena that can be modelled well by a certain probability distribution do not begin to occur immediately, so it becomes necessary to shift the lower limit of the distribution so that the data can be accurately modelled by the distribution.

If implementing this yourself, ensure you set all y-values to 0 for $t \leq \gamma$ as the raw form of the location shifted distributions above will not automatically zeroise these values for you and may result in negative values. This zeroizing is done automatically within *reliability*.

4.10 Relationships between the five functions

The PDF, CDF, SF, HF, CHF of a probability distribution are inter-related and any of these functions can be obtained by applying the correct transformation to any of the others. The following list of transformations are some of the most useful:

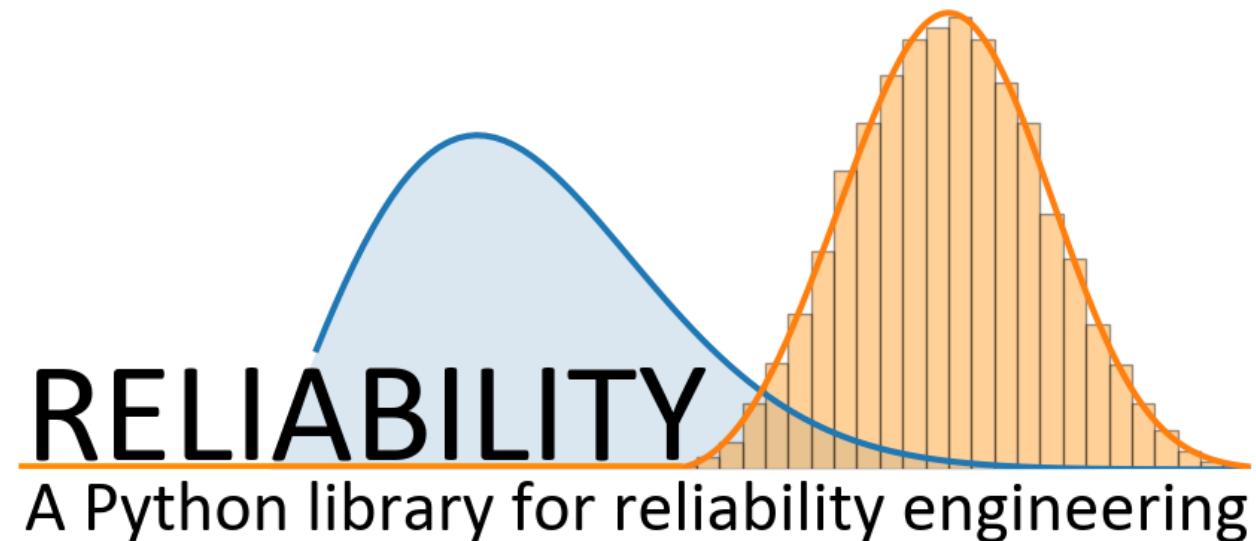
$$\text{PDF} = \frac{d}{dt} \text{CDF}$$

$$\text{CDF} = \int_{-\infty}^t \text{PDF}$$

$$\text{SF} = 1 - \text{CDF}$$

$$\text{HF} = \frac{\text{PDF}}{\text{SF}}$$

$$\text{CHF} = -\ln(\text{SF})$$



CREATING AND PLOTTING DISTRIBUTIONS

There are 8 standard probability distributions available in *reliability.Distributions*. These are:

- Weibull Distribution (, ,)
- Exponential Distribution (,)
- Gamma Distribution (, ,)
- Normal Distribution (,)
- Lognormal Distribution (, ,)
- Logistic Distribution (, ,)
- Gumbel Distribution (,)
- Beta Distribution (,)

API Reference

For inputs and outputs see the [API reference](#).

Probability distributions within *reliability* are Python objects, which allows us to specify just the type of distribution and its parameters. Once the distribution object is created, we can access a large number of methods (such as PDF() or plot()). Some of the methods require additional input and some have optional inputs.

In all of the distributions which use `loc`, the `loc` parameter is used to location shift the distribution to the right. If used, the `loc` parameter must be greater than or equal to 0.

The Beta distribution is only defined in the range 0 to 1. All distributions except the Normal and Gumbel distributions are defined in the positive domain only ($x > 0$).

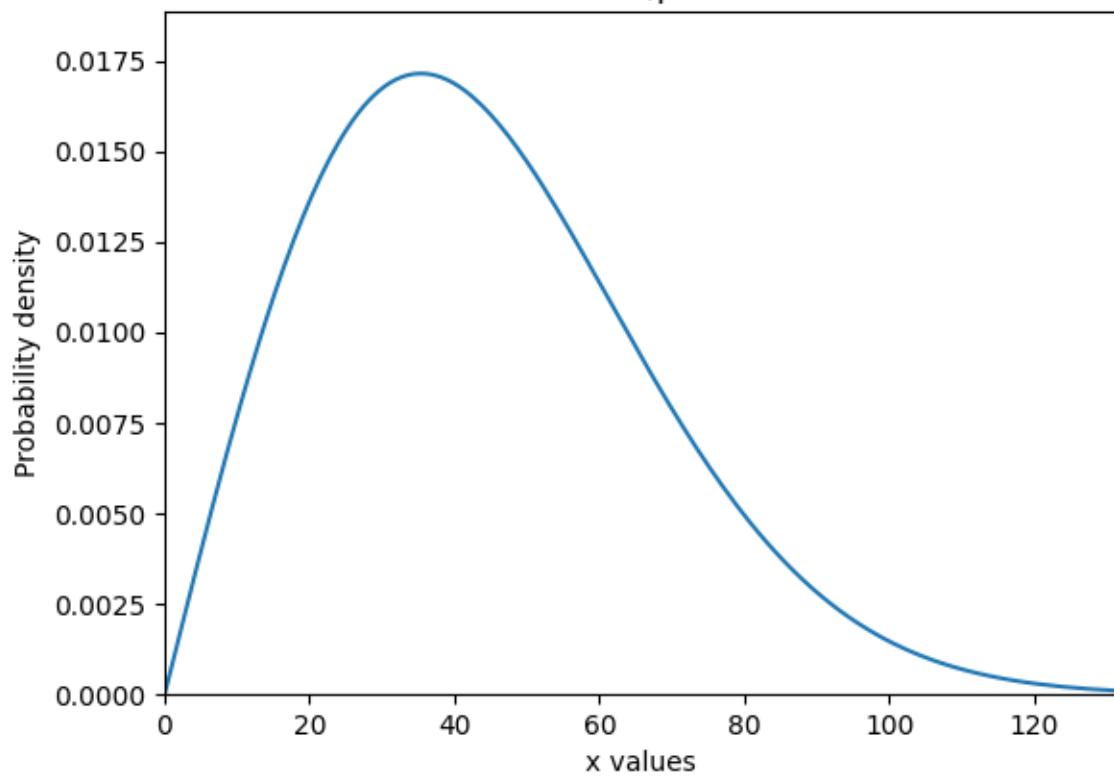
5.1 Example 1

Understanding how to create and plot distributions is easiest with an example. In this first example, we will create a Weibull Distribution with parameters `alpha = 50` and `beta = 2`. We will then plot the PDF of the distribution.

```
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt

dist = Weibull_Distribution(alpha=50, beta=2) # this created the distribution object
dist.PDF() # this creates the plot of the PDF
plt.show()
```

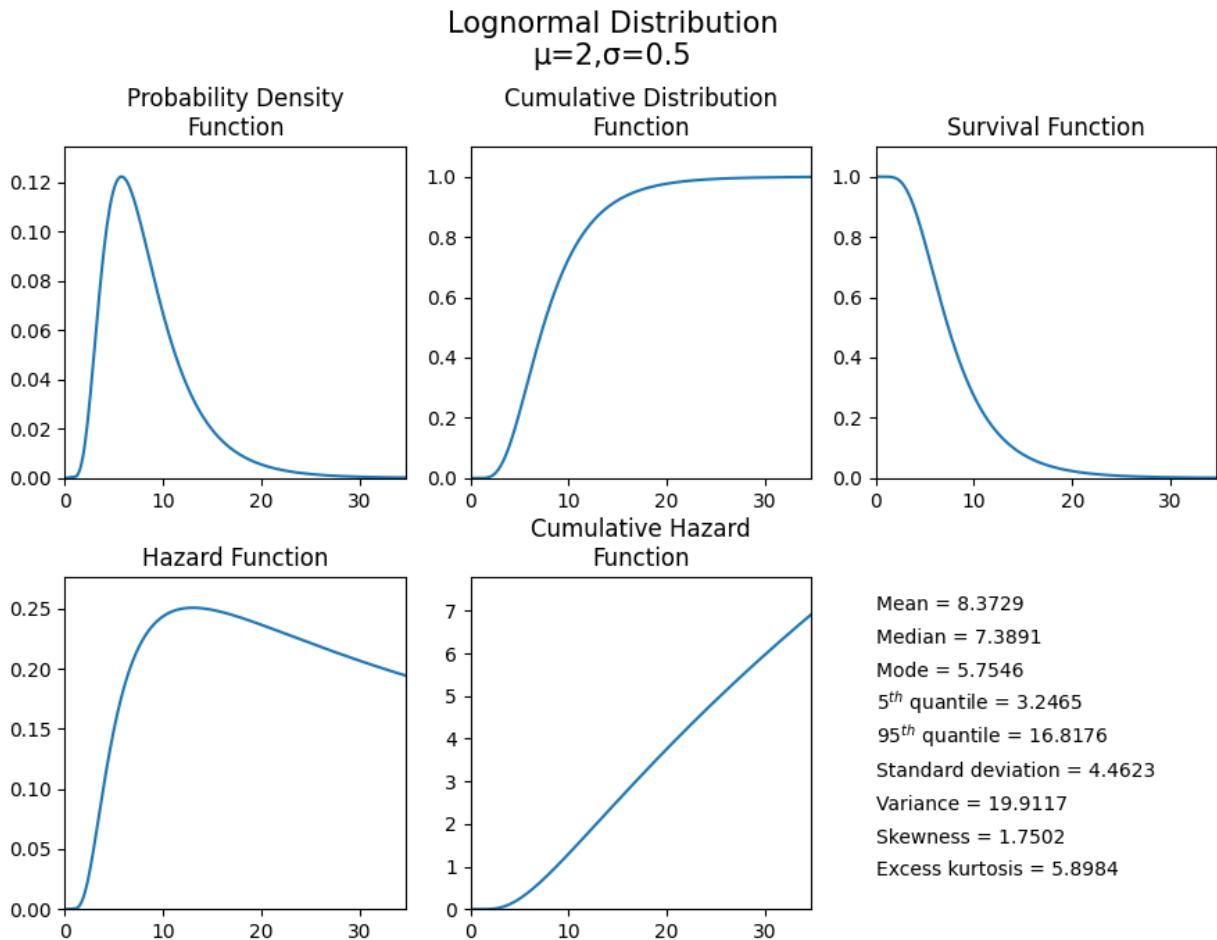
Weibull Distribution
Probability Density Function
 $\alpha=50, \beta=2$



5.2 Example 2

Just as easily as we plotted the PDF in the above example, we can plot any of the 5 characteristic functions (PDF, CDF, SF, HF, CHF). If you would like to view all of these functions together, you can use the `plot()` method. In this second example, we will create a Lognormal Distribution with parameters $\mu=2$ and $\sigma=0.5$. From this distribution, we will use the `plot()` method to visualise the five functions and also provide a summary of the descriptive statistics.

```
from reliability.Distributions import Lognormal_Distribution
dist = Lognormal_Distribution(mu=2, sigma=0.5)
dist.plot()
```



For all of the individual plotting functions (PDF, CDF, SF, HF, CHF), all standard matplotlib plotting keywords (such as label, color, linestyle, etc.) are accepted and used. If not specified they are preset. In specifying the plotting positions for the x-axis, there are optional keywords to be used. The first of these is ‘xvals’ which accepts a list of x-values to use for the horizontal axis. Alternatively, the user may specify ‘xmin’ and/or ‘xmax’ if there is a desired minimum or maximum value. If left unspecified these will be set automatically. xvals overrides xmin and xmax.

Note that .plot() does not require plt.show() to be used as it will automatically show, however the other 5 plotting functions will not be displayed until plt.show() is used. This is to allow the user to overlay multiple plots on the figure or change titles, labels, and legends as required. The plot can be turned off by specifying show_plot=False.

5.3 Example 3

Each of the 5 functions (PDF, CDF, SF, HF, CHF) will always return the y-values for a given set of x-values (xvals). In this example, we want to know the value of the Survival Function at x=20.

```
from reliability.Distributions import Weibull_Distribution

dist = Weibull_Distribution(alpha=50, beta=2)
sf = dist.SF(20)
print('The value of the SF at 20 is', round(sf * 100, 2), '%') # we are converting the
# decimal answer (0.8521...) to a percentage
```

(continues on next page)

(continued from previous page)

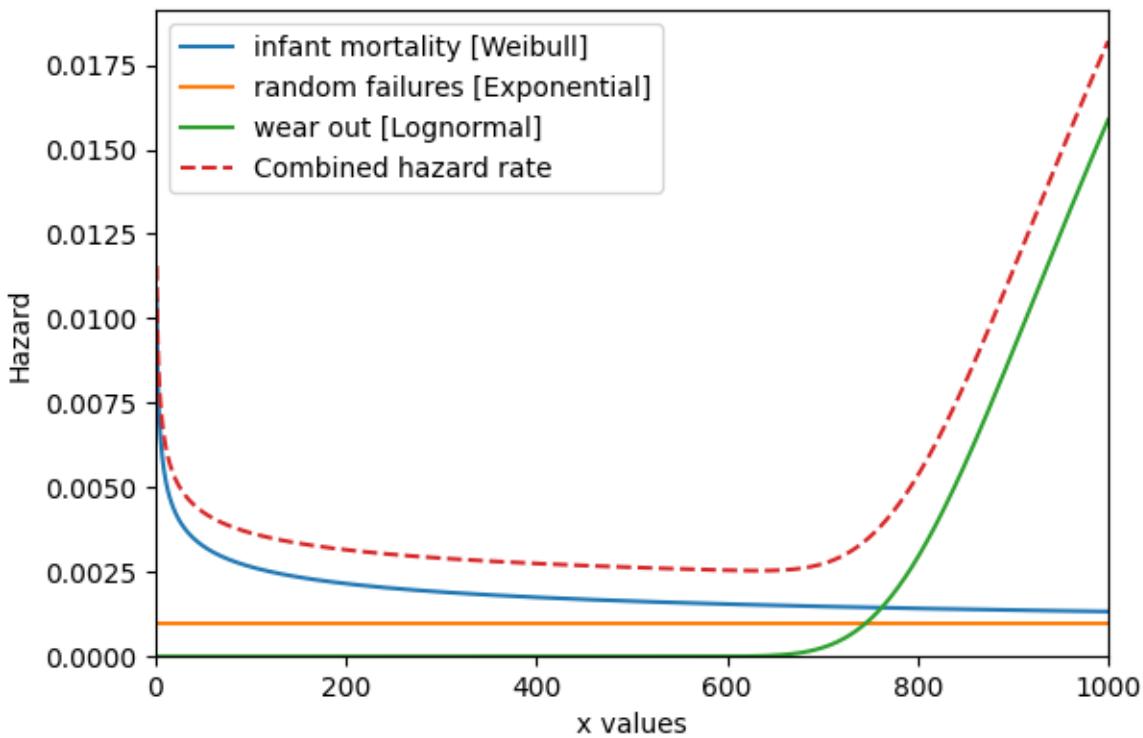
```
"""
The value of the SF at 20 is 85.21 %
""
```

5.4 Example 4

As a final example, we will create a bathtub curve by creating and layering several distributions. The bathtub curve is only for the Hazard function as it shows how a variety of failure modes throughout the life of a population can shape the hazard into a bathtub shape. The three distinct regions are infant mortality, random failures, and wear out. In this example, the returned y-values are added together to produce the ‘combined’ array which is then plotted using matplotlib against the xvals. By specifying xvals in each HF we can ensure that the xvals used will all align. Leaving xvals unspecified would not be appropriate in this example as the default xvals will depend on the shape of the plot.

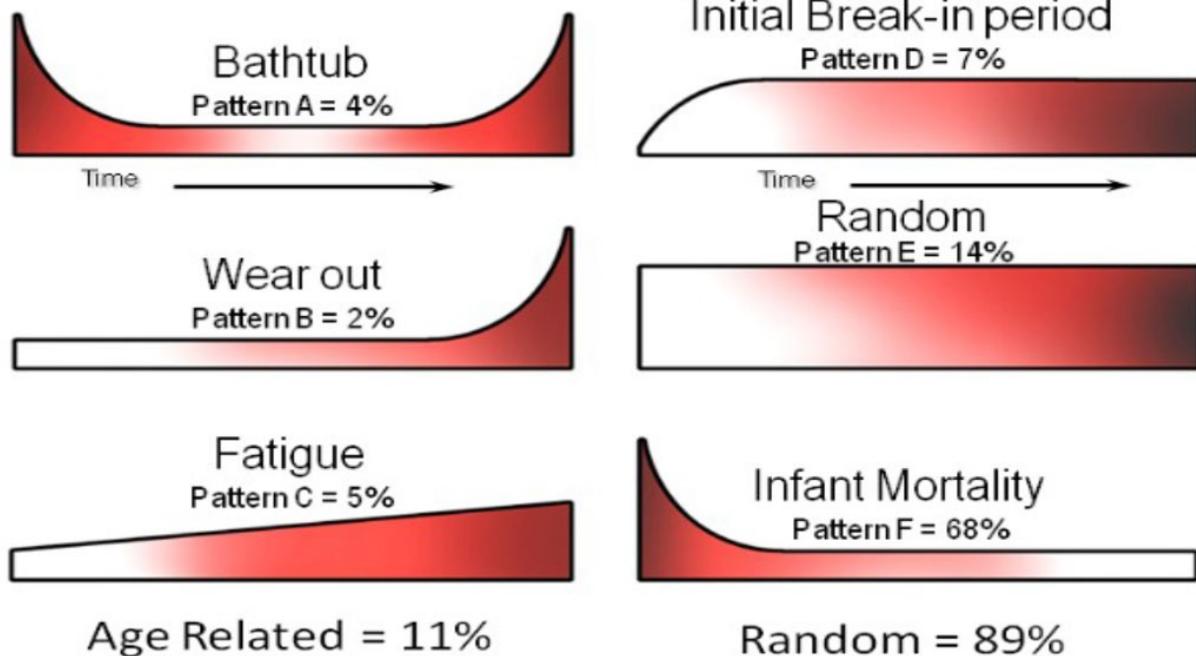
```
from reliability.Distributions import Weibull_Distribution, Lognormal_Distribution,
    Exponential_Distribution
import matplotlib.pyplot as plt
import numpy as np
xvals = np.linspace(0,1000,1000)
infant_mortality = Weibull_Distribution(alpha=400,beta=0.7).HF(xvals=xvals,label='infant_
    mortality [Weibull]')
random_failures = Exponential_Distribution(Lambda=0.001).HF(xvals=xvals,label='random_
    failures [Exponential]')
wear_out = Lognormal_Distribution(mu=6.8,sigma=0.1).HF(xvals=xvals,label='wear out_
    [Lognormal]')
combined = infant_mortality+random_failures+wear_out
plt.plot(xvals,combined,linestyle='--',label='Combined hazard rate')
plt.legend()
plt.title('Example of how multiple failure modes at different stages of\nlife can create_
    a "Bathtub curve" for the total Hazard function')
plt.xlim(0,1000)
plt.ylim(bottom=0)
plt.show()
```

Example of how multiple failure modes at different stages of life can create a "Bathtub curve" for the total Hazard function



On the topic of the Bathtub curve generated in Example 4, it is important to understand that despite its well known name, the bathtub shape of the hazard function is actually much more uncommon than its reputation may suggest. A series of studies (United Airlines 1978, Broberg 1973, SSMD 1993, SUBMEPP 2001) have analysed the failure patterns of large numbers of components and found that there are six characteristic failure patterns (named A to F). Three of these (including the bathtub curve - pattern A) exhibit wear out, while the other three show no signs of wear out. Of all components analysed, just 4% (from the 1978 study) were found to exhibit a bathtub curve, and only 11% showed evidence of wear out (failure modes A,B,C). With 89% of components analysed showing no evidence of wear out it is surprising how many of our maintenance programs to this day still have maintenance policies based on wear out, and how ubiquitous the term "bathtub curve" has become in the maintenance community. Before assuming something is wearing out, we should let its data tell the story.

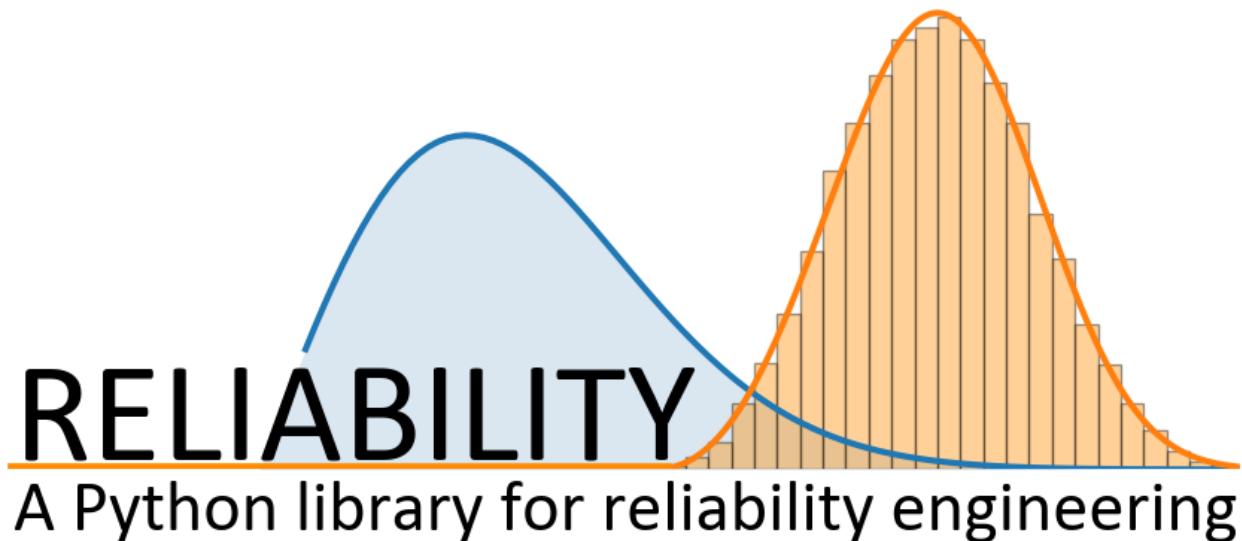
Failure Patterns



Source: John Moubray

If you would like access the API Reference programatically, you can use the help function within Python. Simply type:

```
from reliability.Distributions import Lognormal_Distribution
print(help(Lognormal_Distribution))
```



FITTING A SPECIFIC DISTRIBUTION TO DATA

API Reference

For inputs and outputs see the [API reference](#).

The module *reliability.Fitters* provides many probability distribution fitting functions as shown below.

Functions for fitting non-location shifted distributions:

- Fit_Exponential_1P
- Fit_Weibull_2P
- Fit_Gamma_2P
- Fit_Lognormal_2P
- Fit_Loglogistic_2P
- Fit_Normal_2P
- Fit_Gumbel_2P
- Fit_Beta_2P

Functions for fitting location shifted distributions:

- Fit_Exponential_2P
- Fit_Weibull_3P
- Fit_Gamma_3P
- Fit_Lognormal_3P
- Fit_Loglogistic_3P

All of the distributions can be fitted to both complete and incomplete (right censored) data. All distributions in the Fitters module are named with their number of parameters (eg. Fit_Weibull_2P uses „, whereas Fit_Weibull_3P uses „„). This is intended to remove ambiguity about what distribution you are fitting.

Distributions are fitted simply by using the desired function and specifying the data as failures or right_censored data. You must have at least as many failures as there are distribution parameters or the fit would be under-constrained. It is generally advisable to have at least 4 data points as the accuracy of the fit is proportional to the amount of data. Once fitted, the results are assigned to an object and the fitted parameters can be accessed by name, as shown in the examples below. The goodness of fit criterions are also available as AICc (Akaike Information Criterion corrected), BIC (Bayesian Information Criterion), AD (Anderson-Darling), and loglik (log-likelihood), though these are more useful when comparing the fit of multiple distributions such as in the [Fit_Everything](#) function. As a matter of convenience, each of the modules in Fitters also generates a distribution object that has the parameters of the fitted distribution.

The Beta distribution is only for data in the range 0 to 1. Specifying data outside of this range will cause an error. The fitted Beta distribution does not include confidence intervals.

If you have a very large amount of data (>100000 samples) then it is likely that your computer will take significant time to compute the results. This is a limitation of interpreted languages like Python compared to compiled languages like C++ which many commercial reliability software packages are written in. If you have very large volumes of data, you may want to consider using commercial software for faster computation time. The function `Fit_Weibull_2P_grouped` is designed to accept a dataframe which has multiple occurrences of some values (eg. multiple values all right censored to the same value when the test was ended). Depending on the size of the data set and the amount of grouping in your data, `Fit_Weibull_2P_grouped` may be much faster than `Fit_Weibull_2P` and achieve the same level of accuracy. This difference is not noticeable if you have less than 10000 samples. For more information, see the example below on [using `Fit_Weibull_2P_grouped`](#).

Heavily censored data (>99.9% censoring) may result in a failure of the optimizer to find a solution. If you have heavily censored data, you may have a limited failure population problem. It is recommended that you do not try fitting one of these standard distributions to such a dataset as your results (while they may have achieved a successful fit) will be a poor description of your overall population statistic and you risk drawing the wrong conclusions when the wrong model is fitted. The limited failure population model is planned for a future release of *reliability*, though development on this model is yet to commence. In the meantime, see JMP Pro's model for [Defective Subpopulations](#).

If you do not know which distribution you want to fit, then please see the [section](#) on using the `Fit_Everything` function which will find the best distribution to describe your data. It is highly recommended that you always try to fit everything and accept the best fit rather than choosing a particular distribution for subjective reasons.

If you have tried fitting multiple distributions and nothing seems to work well, or you can see the scatter points on the probability plot have an S-shape or a bend, then you may have data from a mixture of sources. In this case consider fitting a [Mixture model](#) or a [Competing Risks Model](#).

6.1 Example 1

To learn how we can fit a distribution, we will start by using a simple example with 30 failure times. These times were generated from a Weibull distribution with $\alpha=50$, $\beta=3$. Note that the output also provides the confidence intervals and standard error of the parameter estimates. The probability plot is generated by default (you will need to specify `plt.show()` to show it). See the section on [probability plotting](#) to learn how to interpret this plot.

```
from reliability.Fitters import Fit_Weibull_2P
import matplotlib.pyplot as plt
data = [58,75,36,52,63,65,22,17,28,64,23,40,73,45,52,36,52,60,13,55,82,55,34,57,23,42,66,
       35,34,25] # made using Weibull Distribution(alpha=50,beta=3)
wb = Fit_Weibull_2P(failures=data)
plt.show()

"""
Results from Fit_Weibull_2P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 30/0 (0% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
Alpha      51.858      3.55628    45.3359    59.3183
Beta       2.80086      0.41411    2.09624    3.74233

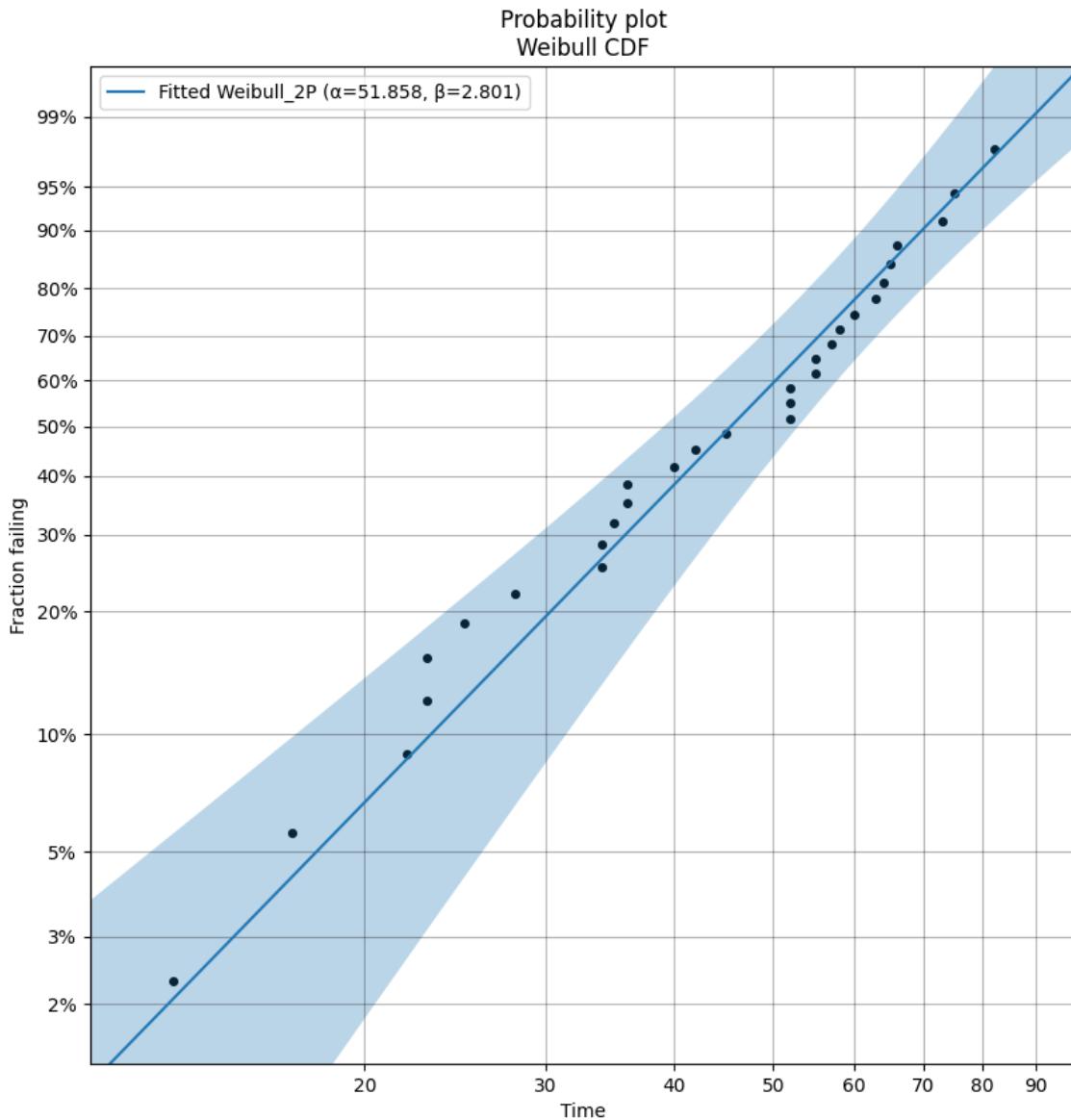
Goodness of fit      Value
Log-likelihood -129.063
AICc      262.57
```

(continues on next page)

(continued from previous page)

BIC 264.928
 AD 0.759805

'''



The above probability plot is the typical way to visualise how the CDF (the blue line) models the failure data (the black points). If you would like to view the failure points alongside the PDF, CDF, SF, HF, or CHF without the axis being scaled then you can generate the scatter plot using the function `plot_points` which is available within `reliability.Probability_plotting`. In the example below we create some data, then fit a Weibull distribution to the data (ensuring we turn off the probability plot). From the fitted distribution object we plot the Survival Function (SF). We then use `plot_points` to generate a scatter plot of the plotting positions for the survival function.

API Reference

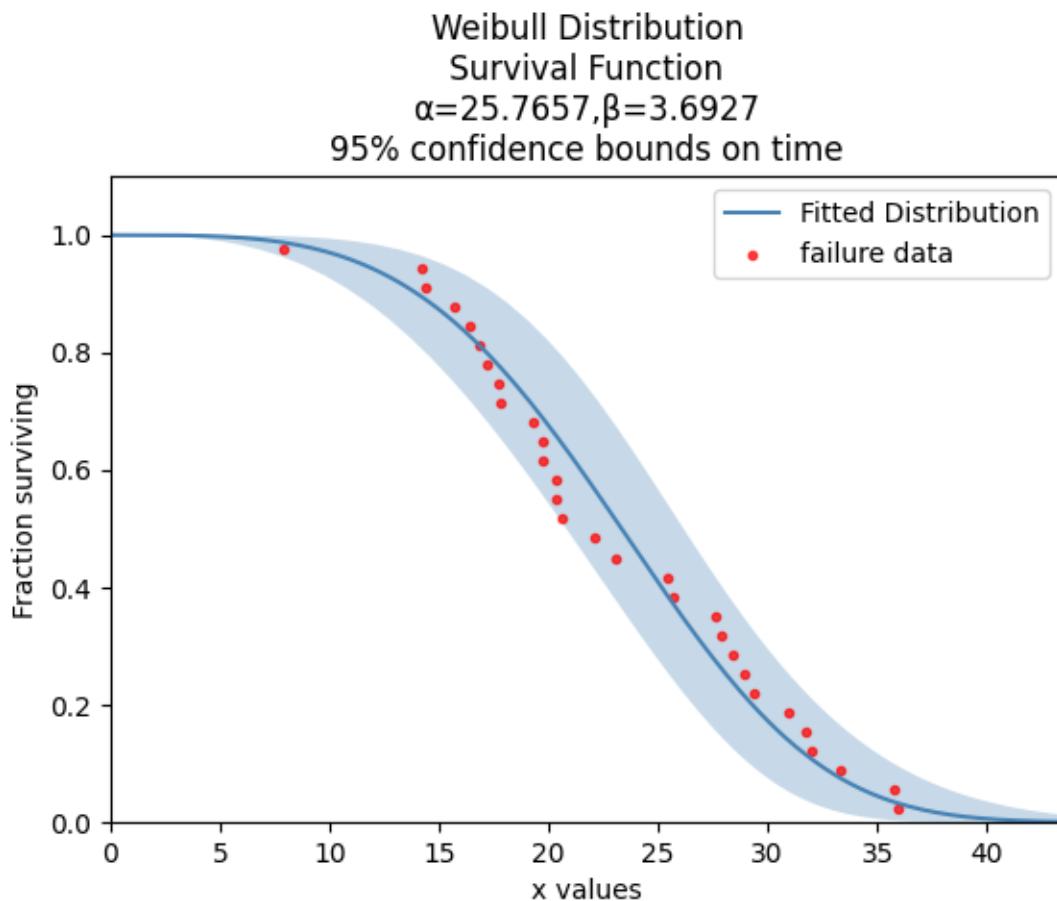
For inputs and outputs of the `plot_points` function see the [API reference](#).

6.2 Example 2

This example shows how to use the `plot_points` function.

```
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_2P
from reliability.Probability_plotting import plot_points
import matplotlib.pyplot as plt

data = Weibull_Distribution(alpha=25, beta=4).random_samples(30)
weibull_fit = Fit_Weibull_2P(failures=data, show_probability_plot=False, print_
    _results=False)
weibull_fit.distribution.SF(label='Fitted Distribution', color='steelblue')
plot_points(failures=data, func='SF', label='failure data', color='red', alpha=0.7)
plt.legend()
plt.show()
```



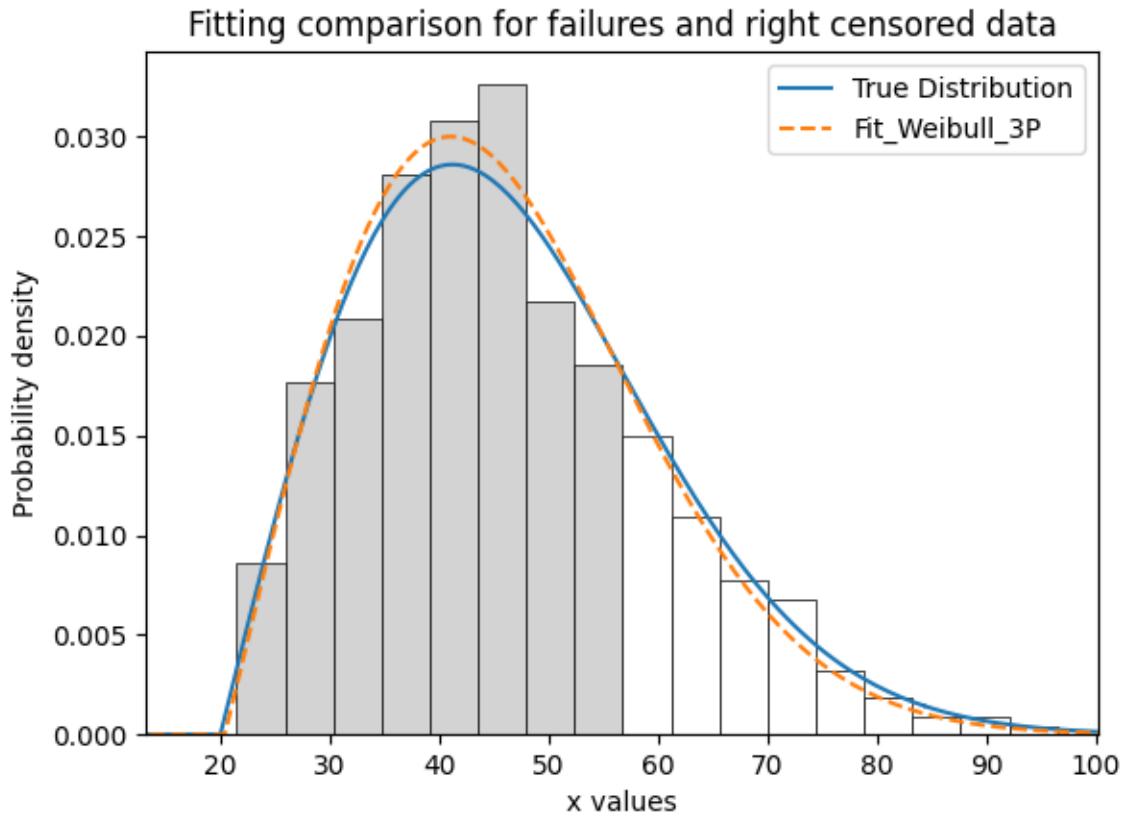
6.3 Example 3

It is beneficial to see the effectiveness of the fitted distribution in comparison to the original distribution. In this example, we are creating 500 samples from a Weibull distribution and then we will right censor all of the data above our chosen threshold. Then we are fitting a Weibull_3P distribution to the data. Note that we need to specify “show_probability_plot=False, print_results=False” in the Fit_Weibull_3P to prevent the normal outputs of the fitting function from being displayed.

```
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_3P
from reliability.Other_functions import make_right_censored_data, histogram
import matplotlib.pyplot as plt

a = 30
b = 2
g = 20
threshold=55
dist = Weibull_Distribution(alpha=a, beta=b, gamma=g) # generate a weibull distribution
raw_data = dist.random_samples(500, seed=2) # create some data from the distribution
data = make_right_censored_data(raw_data, threshold=threshold) #right censor some of the data
print('There are', len(data.right_censored), 'right censored items.')
wbf = Fit_Weibull_3P(failures=data.failures, right_censored=data.right_censored, show_probability_plot=False, print_results=False) # fit the Weibull_3P distribution
print('Fit_Weibull_3P parameters:\nAlpha:', wbf.alpha, '\nBeta:', wbf.beta, '\nGamma:', wbf.gamma)
histogram(raw_data,white_above=threshold) # generates the histogram using optimal bin width and shades the censored part as white
dist.PDF(label='True Distribution') # plots the true distribution's PDF
wbf.distribution.PDF(label='Fit_Weibull_3P', linestyle='--') # plots to PDF of the fitted Weibull_3P
plt.title('Fitting comparison for failures and right censored data')
plt.legend()
plt.show()

"""
There are 118 right censored items.
Fit_Weibull_3P parameters:
Alpha: 28.874745169627886
Beta: 2.0294944619390654
Gamma 20.383959629725744
""
```



6.4 Example 4

As another example, we will fit a Gamma_2P distribution to some partially right censored data. To provide a comparison of the fitting accuracy as the number of samples increases, we will do the same experiment with varying sample sizes. The results highlight that the accuracy of the fit is proportional to the amount of samples, so you should always try to obtain more data if possible.

```
from reliability.Distributions import Gamma_Distribution
from reliability.Fitters import Fit_Gamma_2P
from reliability.Other_functions import make_right_censored_data, histogram
import matplotlib.pyplot as plt

a = 30
b = 4
threshold = 180 # this is used when right censoring the data
trials = [10, 100, 1000, 10000]
subplot_id = 221
plt.figure(figsize=(9, 7))
for sample_size in trials:
    dist = Gamma_Distribution(alpha=a, beta=b)
    raw_data = dist.random_samples(sample_size, seed=2) # create some data. Seeded for
    # repeatability
    data = make_right_censored_data(raw_data, threshold=threshold) # right censor the
    # data
```

(continues on next page)

(continued from previous page)

```

→data
gf = Fit_Gamma_2P(failures=data.failures, right_censored=data.right_censored, show_
→probability_plot=False, print_results=False) # fit the Gamma_2P distribution
print('\nFit_Gamma_2P parameters using', sample_size, 'samples:', '\nAlpha:', gf.
→alpha, '\nBeta:', gf.beta) # print the results
plt.subplot(subplot_id)
histogram(raw_data,white_above=threshold) # plots the histogram using optimal bin_
→width and shades the right censored part white
dist.PDF(label='True') # plots the true distribution
gf.distribution.PDF(label='Fitted', linestyle='--') # plots the fitted Gamma_2P_
→distribution
plt.title(str(str(sample_size) + ' samples\n' + r'$\alpha$ error: ' +_
→str(round(abs(gf.alpha - a) / a * 100, 2)) + '%\n' + r'$\beta$ error: ' +_
→str(round(abs(gf.beta - b) / b * 100, 2)) + '%'))
plt.ylim([0, 0.012])
plt.xlim([0, 500])
plt.legend()
subplot_id += 1
plt.subplots_adjust(left=0.11, bottom=0.08, right=0.95, top=0.89, wspace=0.33, hspace=0.
→58)
plt.show()

"""

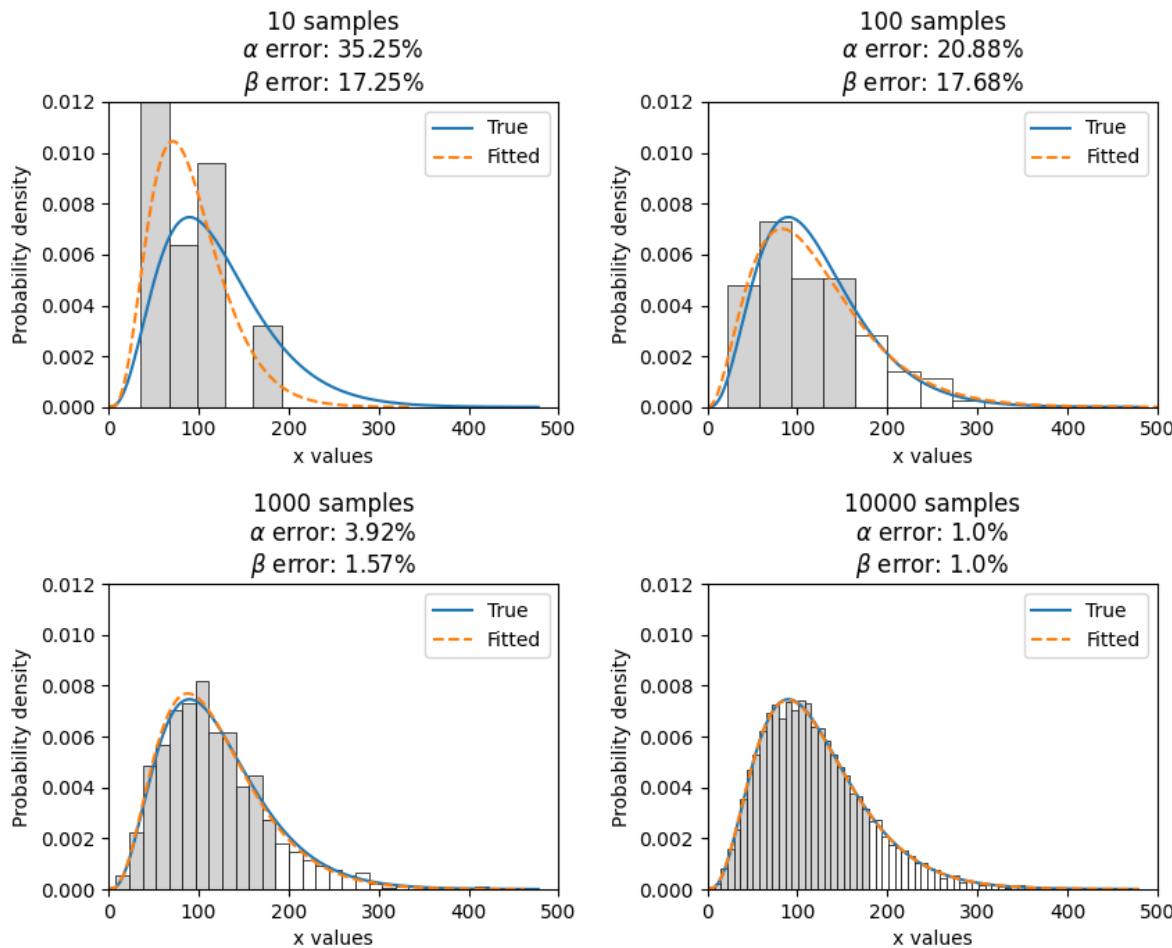
Fit_Gamma_2P parameters using 10 samples:
Alpha: 19.42603577754394
Beta: 4.6901283424759255

Fit_Gamma_2P parameters using 100 samples:
Alpha: 36.26411284656554
Beta: 3.2929448936077534

Fit_Gamma_2P parameters using 1000 samples:
Alpha: 28.825423280158407
Beta: 4.062909060146121

Fit_Gamma_2P parameters using 10000 samples:
Alpha: 30.301232862075587
Beta: 3.96009153189253
"""

```



6.5 Example 5

To obtain details of the quantiles (y-values from the CDF) which include the lower estimate, point estimate, and upper estimate, we can use the quantiles input for each Fitter. In this example, we will create some data and fit a Weibull_2P distribution. When quantiles is specified the results printed includes both the table of results and the table of quantiles. Setting quantiles as True will use a default list of quantiles (as shown in the first output). Alternatively we can specify the exact quantiles to use (as shown in the second output). The use of the `crosshairs` function is also shown which was used to annotate the plot manually. Note that the quantiles provided are the quantiles of the confidence bounds on time. You can extract the confidence bounds on reliability using the fitted distribution object as shown [here](#).

```
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_2P
from reliability.Other_functions import crosshairs
import matplotlib.pyplot as plt

dist = Weibull_Distribution(alpha=500, beta=6)
data = dist.random_samples(50, seed=1) # generate some data
# this will produce the large table of quantiles below the first table of results
Fit_Weibull_2P(failures=data, quantiles=True, CI=0.8, show_probability_plot=False)
print('-----')
# repeat the process but using specified quantiles.
```

(continues on next page)

(continued from previous page)

```

output = Fit_Weibull_2P(failures=data, quantiles=[0.05, 0.5, 0.95], CI=0.8)
# these points have been manually annotated on the plot using crosshairs
crosshairs()
plt.show()

# the values from the quantiles dataframe can be extracted using pandas:
lower_estimates = output.quantiles['Lower Estimate'].values
print('Lower estimates:', lower_estimates)

#alternatively, the bounds can be extracted from the distribution object
lower,point,upper = output.distribution.CDF(CI_y=[0.05, 0.5, 0.95], CI=0.8)
print('Upper estimates:', upper)

"""

Results from Fit_Weibull_2P (80% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 50/0 (0% right censored)

```

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha	489.117	13.9217	471.597	507.288
Beta	5.20798	0.589269	4.505	6.02066

Goodness of fit	Value
Log-likelihood	-301.658
AICc	607.571
BIC	611.14
AD	0.48267

Table of quantiles (80% CI bounds on time):

Quantile	Lower Estimate	Point Estimate	Upper Estimate
0.01	175.215	202.212	233.368
0.05	250.235	276.521	305.569
0.1	292.686	317.508	344.435
0.2	344.277	366.719	390.623
0.25	363.578	385.05	407.79
0.5	437.69	455.879	474.824
0.75	502.94	520.776	539.245
0.8	517.547	535.917	554.938
0.9	553.267	574.068	595.651
0.95	580.174	603.82	628.43
0.99	625.682	655.79	687.347

Results from Fit_Weibull_2P (80% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 50/0 (0% right censored)

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha	489.117	13.9217	471.597	507.288
Beta	5.20798	0.589269	4.505	6.02066

(continues on next page)

(continued from previous page)

Goodness of fit *Value*
Log-likelihood -301.658
AICc 607.571
BIC 611.14
AD 0.48267

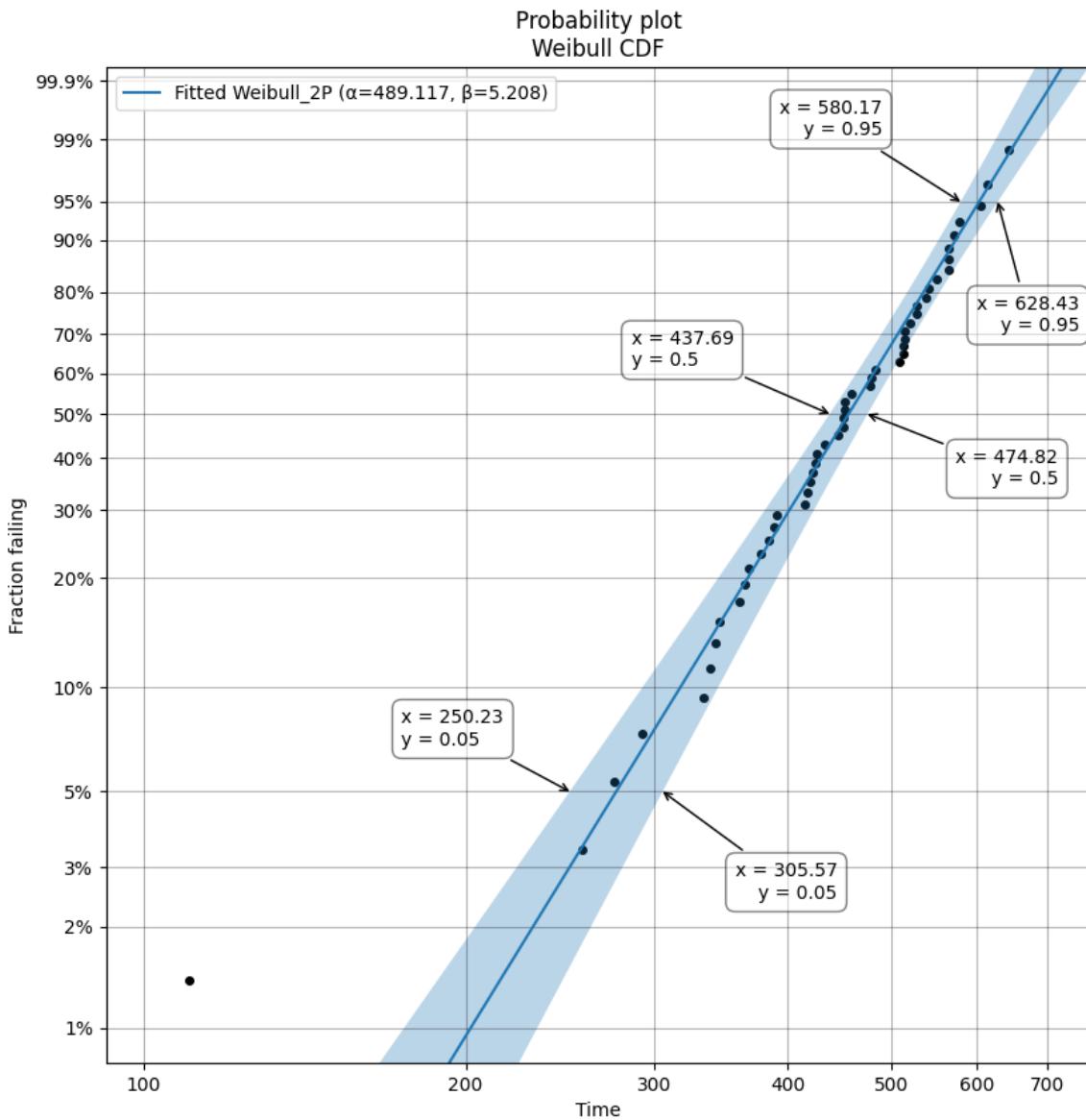
Table of quantiles (80% CI bounds on time):

Quantile	Lower Estimate	Point Estimate	Upper Estimate
0.05	250.235	276.521	305.569
0.5	437.69	455.879	474.824
0.95	580.174	603.82	628.43

Lower estimates: [250.23461473 437.69015375 580.17421254]

Upper estimates: [305.56872227 474.82362169 628.43042835]

"



6.6 Using Fit_Weibull_2P_grouped for large data sets

The function `Fit_Weibull_2P_grouped` is effectively the same as `Fit_Weibull_2P`, except for a few small differences that make it more efficient at handling grouped data sets. Grouped data sets are typically found in very large data that may be heavily censored. The function includes a choice between two optimizers and a choice between two initial guess methods for the initial guess that is given to the optimizer. These help in cases where the data is very heavily censored (>99.9%). The defaults for these options are usually the best but you may want to try different options to see which one gives you the lowest log-likelihood.

API Reference

For inputs and outputs see the [API reference](#).

6.7 Example 6

The following example shows how we can use Fit_Weibull_2P_grouped to fit a Weibull_2P distribution to grouped data from a spreadsheet (shown below) on the Windows desktop. If you would like to access this data, it is available in reliability.Datasets.electronics and includes both the failures and right_censored format as well as the dataframe format. An example of this is provided in the code below (option 2).

	A	B	C
1	time	quantity	category
2	220	1	F
3	179	1	F
4	123	1	F
5	146	1	F
6	199	1	F
7	181	1	F
8	191	1	F
9	216	1	F
10	1	1	F
11	73	1	F
12	44798	817	C
13	62715	823	C
14	81474	815	C
15	80632	813	C
16	62716	804	C

```
from reliability.Fitters import Fit_Weibull_2P_grouped
import pandas as pd

# option 1 for importing this dataset (from an excel file on your desktop)
filename = 'C:\\\\Users\\\\Current User\\\\Desktop\\\\data.xlsx'
df = pd.read_excel(io=filename)

## option 2 for importing this dataset (from the dataset in reliability)
# from reliability.Datasets import electronics
# df = electronics().dataframe

print(df.head(15), '\n')
Fit_Weibull_2P_grouped(dataframe=df, show_probability_plot=False)

"""

      time  quantity category
0      220         1        F
1      179         1        F
2      123         1        F
3      146         1        F
4      199         1        F
5      181         1        F
6      191         1        F
7      216         1        F
```

(continues on next page)

(continued from previous page)

8	1	1	F
9	73	1	F
10	44798	817	C
11	62715	823	C
12	81474	815	C
13	80632	813	C
14	62716	804	C

Results from Fit_Weibull_2P_grouped (95% CI):

Analysis method: Maximum Likelihood Estimation (MLE)

Optimizer: TNC

Failures / Right censored: 10/4072 (99.75502% right censored)

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha	6.19454e+21	7.7592e+22	1.34889e+11	2.84473e+32
Beta	0.153742	0.0485886	0.0827523	0.285632

Goodness of fit Value

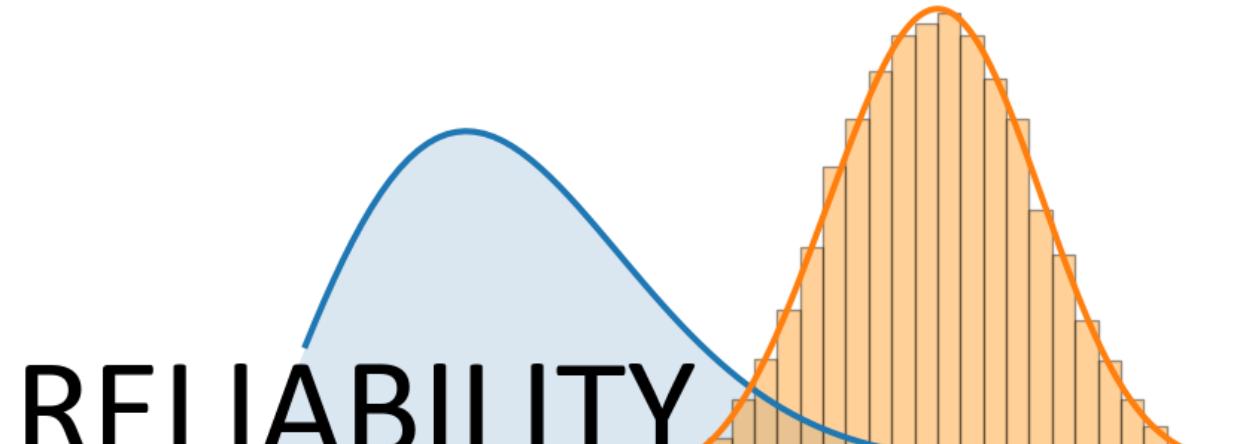
Log-likelihood -144.617

AICc 293.236

BIC 305.862

AD 264.999

...



RELIABILITY
A Python library for reliability engineering

FITTING ALL AVAILABLE DISTRIBUTIONS TO DATA

API Reference

For inputs and outputs see the [API reference](#).

To fit all of the [distributions available](#) in *reliability*, is a similar process to fitting a specific distribution. The user needs to specify the failures and any right censored data. The Beta distribution will only be fitted if you specify data that is in the range 0 to 1 and does not include confidence intervals on the plot. The selection of what can be fitted is all done automatically based on the data provided. Manual exclusion of probability distributions is also possible. If you only provide 2 failures then all distributions with more than 2 parameters will automatically be excluded from the fitting process.

Confidence intervals are shown on the plots but they are not reported for each of the fitted parameters as this would be a large number of outputs. If you need the confidence intervals for the fitted parameters you can repeat the fitting using just a specific distribution and the results will include the confidence intervals. Confidence intervals are not yet available for the Weibull DS, Weibull Mixture, and Weibull CR models.

The distributions Weibull_ZI and Weibull_DSZI are not included when using `Fit_Everything` as these distributions are only applicable when the dataset contains zeros. If your data contains zeros you should fit these distributions individually.

7.1 Example 1

In this first example, we will use `Fit_Everything` on some data and will return only the dataframe of results. Note that we are actively supressing the 4 plots that would normally be shown to provide graphical goodness of fit indications. The table of results has been ranked by BIC to show us that Weibull_2P was the best fitting distribution for this dataset. This is what we expected since the data was generated using `Weibull_Distribution(alpha=50,beta=2)`.

```
from reliability.Fitters import Fit_Everything
# data created using Weibull_Distribution(alpha=50,beta=2), and rounded to nearest integer
data = [92, 44, 94, 56, 54, 24, 96, 3, 27, 37, 61, 23, 70, 101, 21, 47, 4, 34, 10, 88, 37, 86, 62, 70, 21, 13, 47, 21, 57, 36, 43, 83, 42, 16, 20, 44, 43, 50, 35, 51, 35, 49, 60, 22, 34, 41, 53, 27, 44, 49]
Fit_Everything(failures=data, show_histogram_plot=False, show_probability_plot=False, show_PP_plot=False, show_best_distribution_probability_plot=False)

"""
Results from Fit_Everything:
Analysis method: MLE
```

(continues on next page)

(continued from previous page)

Failures / Right censored: 50/0 (0% right censored)

Distribution	Alpha	Beta	Gamma	Alpha 1	Beta 1	Alpha 2	Beta 2	Proportion 1	DS
↳ Mu	Sigma	Lambda	Log-likelihood	AICc	BIC	AD optimizer			↳
Weibull_2P	51.1908	1.92376		-228.338	460.932	464.501	0.613083	TNC	↳
↳	Gamma_2P	16.5098	2.75836	-229.902	464.06	467.628	0.779371	TNC	↳
↳	Weibull_CR			52.292	1.78639	98.2941	27.141		↳
↳	Weibull_3P	51.1908	1.92376	-226.049	460.987	467.746	0.654939	TNC	↳
↳	↳		0						↳
↳	Weibull_DS	51.1908	1.92376	-228.338	463.198	468.413	0.613083	TNC	1
↳	↳		0						↳
Normal_2P				-228.338	463.198	468.413	0.613083	TNC	↳
↳	45.54	24.2959							↳
Weibull_Mixture				-230.462	465.18	468.748	0.967238	TNC	0.880535
↳				44.0526	2.21658	94.6341	17.6943		↳
↳	Gamma_3P	16.5098	2.75836	-225.092	461.547	469.744	0.61163	TNC	↳
↳	↳		0						↳
Loglogistic_2P	40.6775	2.72212		-229.902	466.326	471.54	0.779371	TNC	↳
↳	Loglogistic_3P	40.6775	2.72212	-232.426	469.108	472.677	0.754563	TNC	↳
↳	↳		0						↳
Lognormal_2P				-232.426	471.374	476.589	0.754563	TNC	3.
↳	62651	0.7149							3.
Gumbel_2P				-235.492	475.239	478.808	1.52542	TNC	
↳	58.2756	25.7469		58.2756	25.7469	-237.148	478.551	482.12	2.19655
Lognormal_3P				58.2756	25.7469	0			
↳	62651	0.7149				-235.492	477.505	482.72	1.52542
Exponential_2P				62651	0.7149	2.9999			
↳						-237.522	479.3	482.869	4.27822
Exponential_1P									
↳				0.0235072	0.0219587	-240.93	483.942	485.771	5.05245
""									

7.2 Example 2

In this second example, we will create some right censored data and use *Fit_Everything*. All outputs are shown, and the best fitting distribution is accessed and printed.

```
from reliability.Fitters import Fit_Everything
from reliability.Distributions import Weibull_Distribution
from reliability.Other_functions import make_right_censored_data

raw_data = Weibull_Distribution(alpha=12, beta=3).random_samples(100, seed=2) # create
# some data
data = make_right_censored_data(raw_data, threshold=14) # right censor the data
results = Fit_Everything(failures=data.failures, right_censored=data.right_censored) # fit
# all the models
print('The best fitting distribution was', results.best_distribution_name, 'which had
```

(continues on next page)

(continued from previous page)

```
↪parameters', results.best_distribution.parameters)
```

```
""
```

Results from Fit_Everything:

Analysis method: MLE

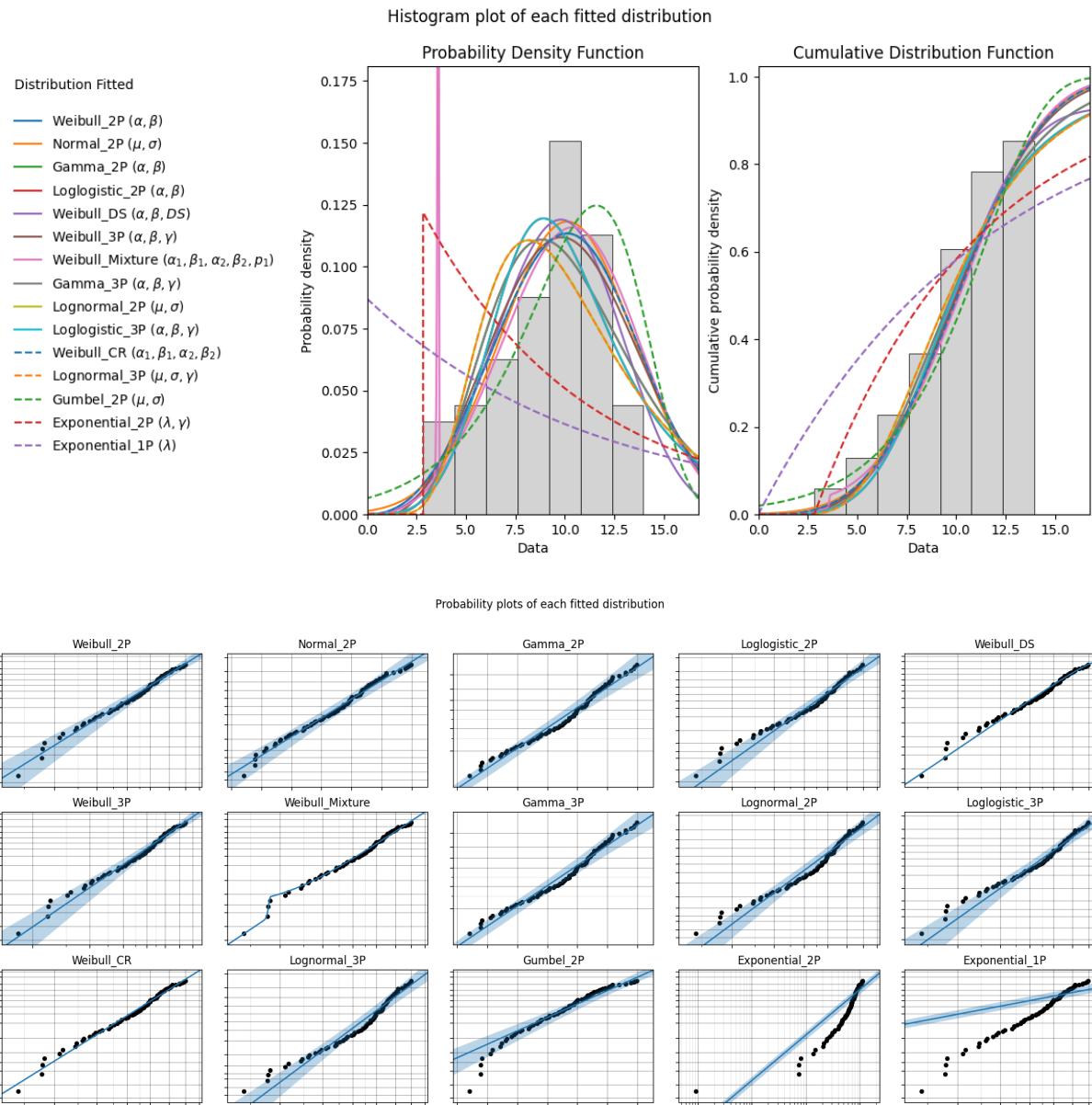
Failures / Right censored: 86/14 (14.0% right censored)

Distribution	Alpha	Beta	Gamma	Alpha 1	Beta 1	Alpha 2	Beta 2	Proportion 1	
DS	Mu	Sigma	Lambda	Log-likelihood	AICc	BIC	AD	optimizer	
Weibull_2P	11.2773	3.30301			-241.959	488.041	493.128	44.945	TNC
Normal_2P	10.1194	3.37466			-242.479	489.082	494.169	44.9098	TNC
Gamma_2P	1.42301	7.21417			-243.235	490.594	495.68	45.2817	TNC
Loglogistic_2P	9.86245	4.48433			-243.588	491.301	496.387	45.2002	TNC
Weibull_DS	10.7383	3.57496			-241.594	489.437	497.003	44.9447	0.
930423									TNC
Weibull_3P	10.0786	2.85824	1.15083		-241.779	489.807	497.373	44.9927	TNC
Weibull_Mixture					3.59763	113.232	11.4208	3.54076	0.0276899
Gamma_3P	1.42301	7.21417	0		-237.392	485.421	497.809	44.9283	TNC
Lognormal_2P	2.26524	0.406436			-243.235	492.72	500.286	45.2817	TNC
Loglogistic_3P	9.86245	4.48433	0		-245.785	495.694	500.78	45.6874	TNC
Weibull_CR					-243.588	493.427	500.992	45.2002	TNC
Lognormal_3P	2.26524	0.406436	0		12.72	3.30301	15.8031	3.30301	
Gumbel_2P	11.5926	2.94944			-241.959	492.338	502.338	44.945	TNC
Exponential_2P			2.82892		-245.785	497.82	505.385	45.6874	TNC
Exponential_1P			0.121884		-248.348	500.819	505.906	45.4624	L-BFGS-B
			0.0870024		-267.003	538.129	543.216	51.7852	TNC
					-295.996	594.034	596.598	56.8662	TNC

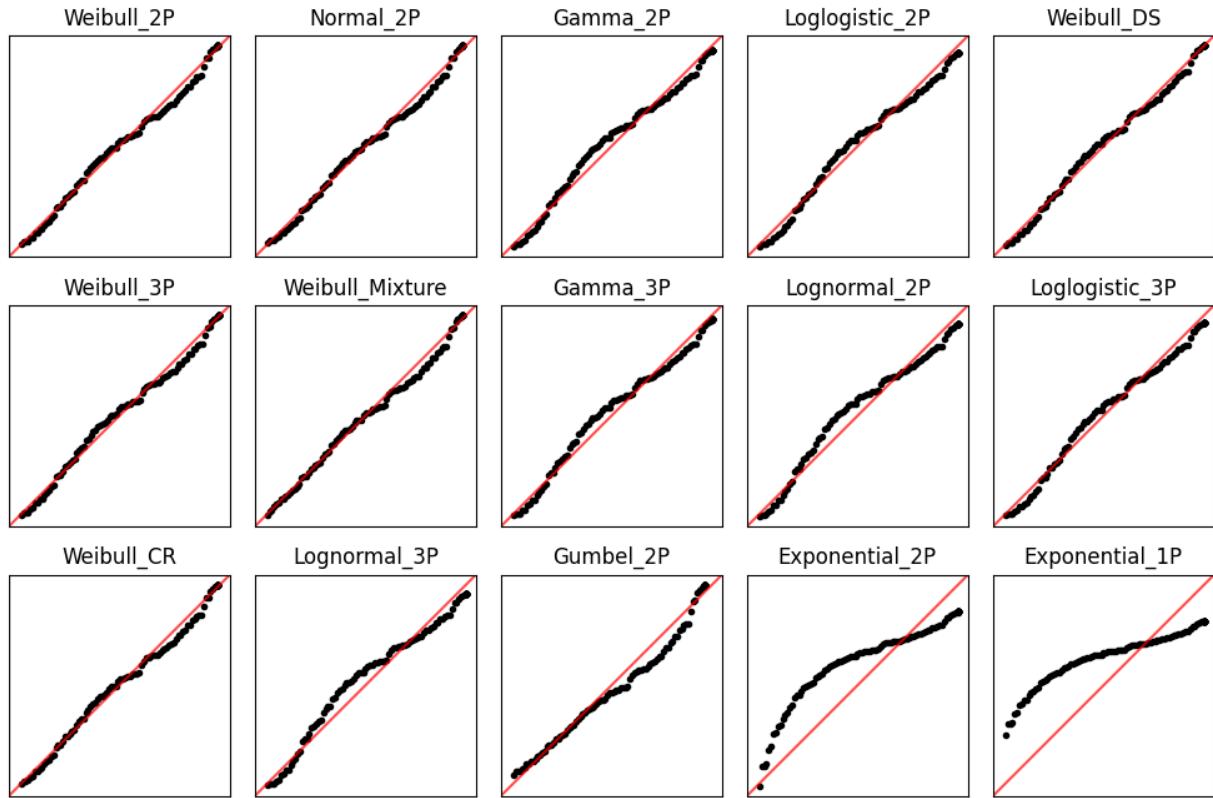
The best fitting distribution was Weibull_2P which had parameters [11.27730641 3.

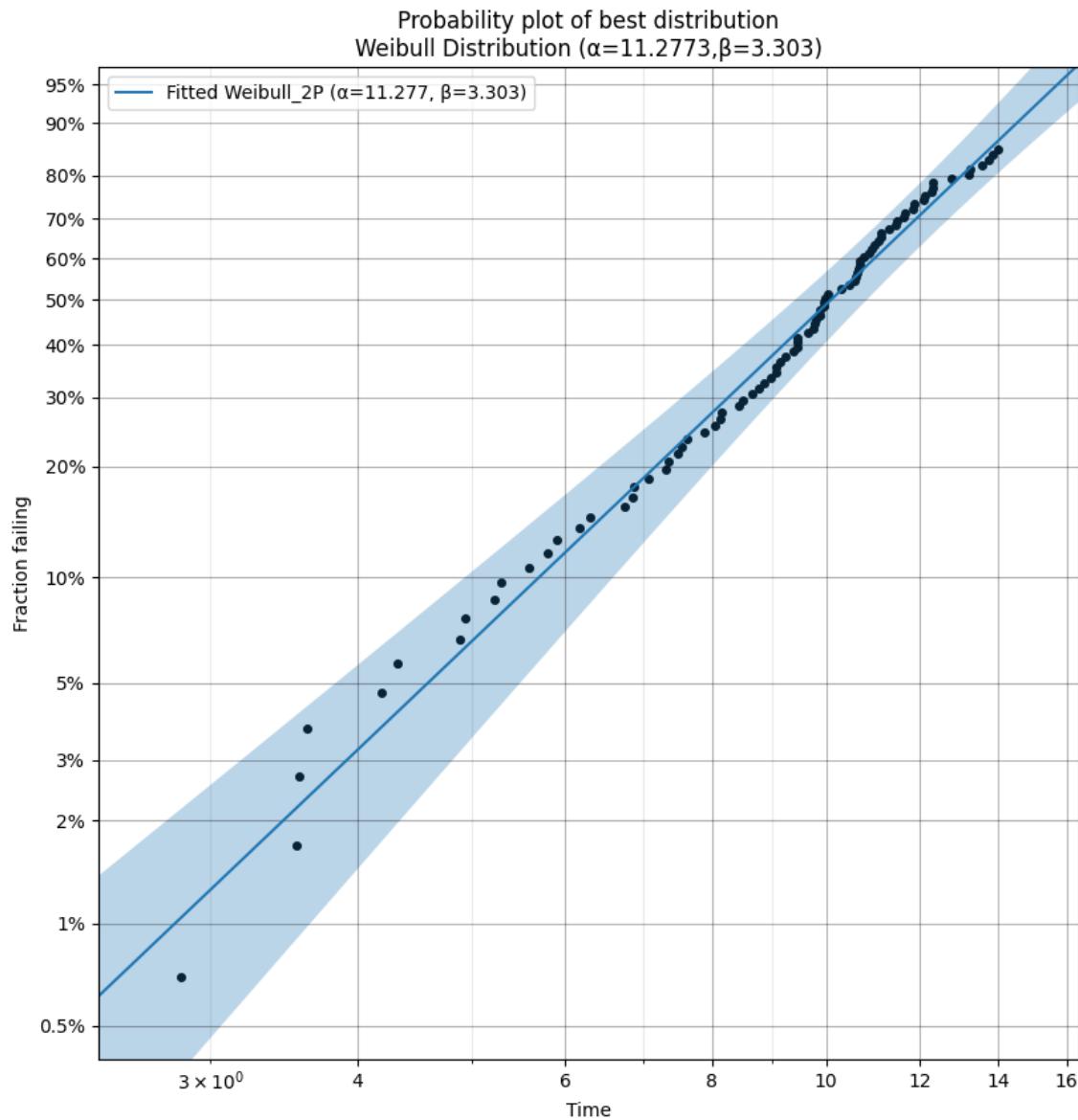
```
↪30300712 0. ]
```

```
""
```



Semi-parametric Probability-Probability plots of each fitted distribution
Parametric (x-axis) vs Non-Parametric (y-axis)





All plots are ordered based on the goodness of fit order of the results. For the histogram this is reflected in the order of the legend. For the probability plots and PP plots, these are ordered from top left to bottom right.



RELIABILITY
A Python library for reliability engineering

WORKING WITH FITTED DISTRIBUTIONS

Probability distributions can be created by specifying the parameters or by fitting the model to data. The main difference between the two distribution objects is the presence of additional parameters required to generate the confidence bounds on the plot. This document shows how a fitted distribution can be used to make certain predictions, with a focus on extracting values from the confidence bounds.

The additional parameters that are added to the distribution object when it is created by a Fitter are the standard error of each parameter (eg. alpha_SE, beta_SE) and the covariance between the parameters (e.g. Cov_alpha_beta). The confidence interval is also required, though this defaults to 0.95 for 95% confidence bounds. It may be specified using the CI argument.

The confidence bounds available are bounds on time or bounds on reliability. For an detailed explaination of how these are calculated, please see the theory document on confidence intervals. The important thing to note here is that they are different, and which one you should use depends on what you want to know. If you want to know the system reliability at a certain time, then you are specifying time (CI_x) and seeking bounds on reliability (CI_type='reliability'). If you want to know the time that the system will reach a certain reliability, then you are specifying reliability (CI_y) and seeking bounds on time (CI_type='time'). Note that these are paired, so CI_y only works with CI_type='time' and CI_x only works with CI_type='reliability'. The only exception to this is the Exponential Distribution which does not accept CI_type since bounds on time and reliability are identical.

8.1 Example 1

In this example, we see how the confidence bounds on time or reliability should be used based on what we are trying to predict.

```
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_2P
import matplotlib.pyplot as plt

dist = Weibull_Distribution(alpha=500,beta=3)
data = dist.random_samples(10,seed=1)
fit = Fit_Weibull_2P(failures=data,show_probability_plot=False,print_results=False)

plt.figure(figsize=(10,7))
plt.subplot(121)
arrow_x = 25
arrow_y = 0.025

X_lower,X_point,X_upper = fit.distribution.CDF(CI_type='time',CI_y=0.7)
plt.arrow(x=0,y=0.7,dx=X_upper,dy=0,color='red',head_width=arrow_y,head_length=arrow_x,
          length_includes_head=True)
plt.arrow(x=X_lower,y=0.7,dx=0,dy=-0.7,color='red',head_width=arrow_x,head_length=arrow_y)
```

(continues on next page)

(continued from previous page)

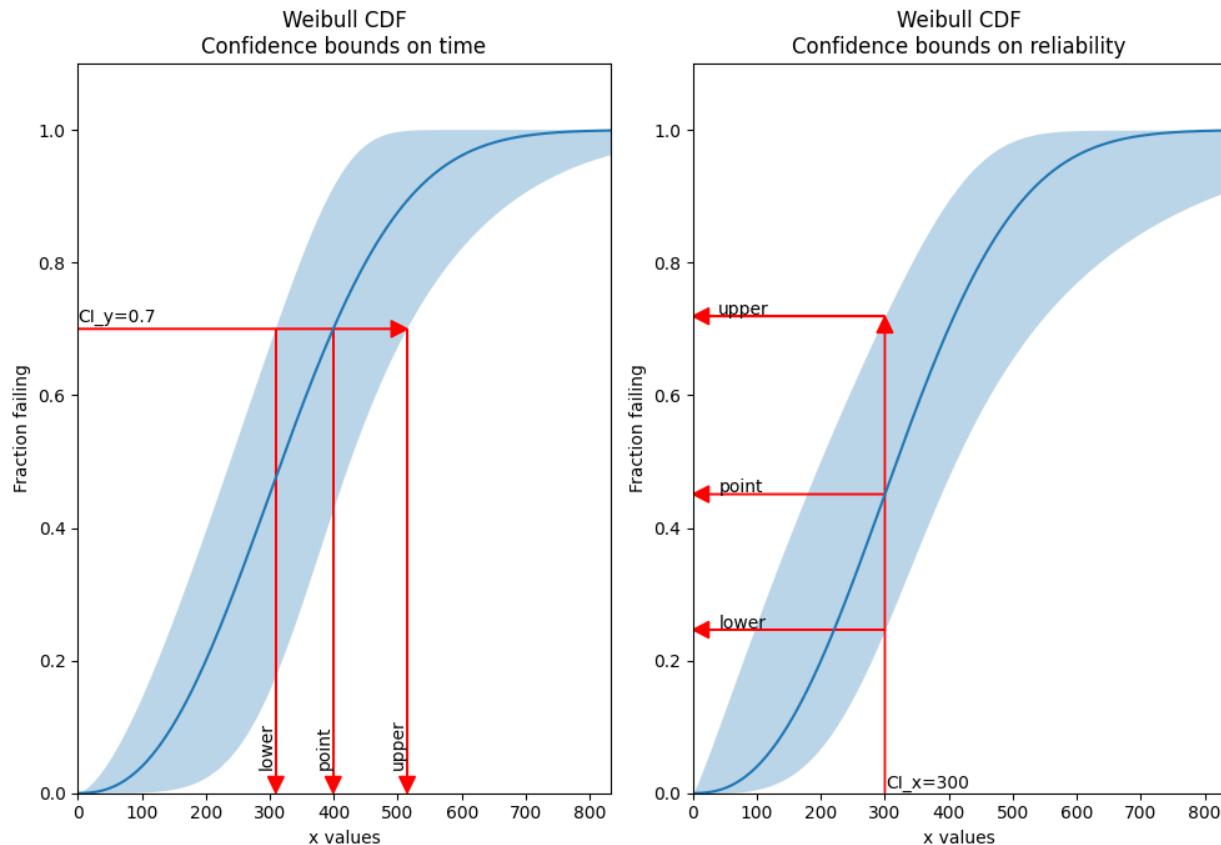
```

→y,length_includes_head=True)
plt.arrow(x=X_point,y=0.7,dx=0,dy=-0.7,color='red',head_width=arrow_x,head_length=arrow_
→y,length_includes_head=True)
plt.arrow(x=X_upper,y=0.7,dx=0,dy=-0.7,color='red',head_width=arrow_x,head_length=arrow_
→y,length_includes_head=True)
plt.xlim(0,dist.quantile(0.99))
plt.ylim(0,1.1)
plt.text(x=0,y=0.705,s='CI_y=0.7',va='bottom')
plt.text(x=X_lower,y=0.035,s='lower',va='bottom',ha='right',rotation=90)
plt.text(x=X_point,y=0.035,s='point',va='bottom',ha='right',rotation=90)
plt.text(x=X_upper,y=0.035,s='upper',va='bottom',ha='right',rotation=90)
plt.title('Weibull CDF\nConfidence bounds on time')

plt.subplot(122)
Y_lower,Y_point,Y_upper = fit.distribution.CDF(CI_type='reliability',CI_x=300)
plt.arrow(x=300,y=0,dx=0,dy=Y_upper,color='red',head_width=arrow_y,head_length=arrow_y,
→length_includes_head=True)
plt.arrow(x=300,y=Y_lower,dx=-300,dy=0,color='red',head_width=arrow_y,head_length=arrow_
→x,length_includes_head=True)
plt.arrow(x=300,y=Y_point,dx=-300,dy=0,color='red',head_width=arrow_y,head_length=arrow_
→x,length_includes_head=True)
plt.arrow(x=300,y=Y_upper,dx=-300,dy=0,color='red',head_width=arrow_y,head_length=arrow_
→x,length_includes_head=True)
plt.xlim(0,dist.quantile(0.99))
plt.ylim(0,1.1)
plt.text(x=301,y=0.001,s='CI_x=300',va='bottom')
plt.text(x=40,y=Y_lower+0.002,s='lower')
plt.text(x=40,y=Y_point+0.002,s='point')
plt.text(x=40,y=Y_upper+0.002,s='upper')

plt.title('Weibull CDF\nConfidence bounds on reliability')
plt.tight_layout()
plt.show()

```



8.2 Example 2

There are 5 plots available (PDF, CDF, SF, HF, CHF) and confidence bounds are only available for 3 of them (CDF, SF, CHF). The following example shows how these plots can be generated, as well as extracting the confidence bounds (the red, purple, and blue points) using `CI_x` and `CI_y`. You can also turn off the confidence bounds if you set `plot_CI=False`.

```
import matplotlib.pyplot as plt
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_2P

dist = Weibull_Distribution(alpha=50, beta=2)
data = dist.random_samples(10, seed=1)
fit = Fit_Weibull_2P(failures=data, show_probability_plot=False, print_results=False)
CI_x = [dist.quantile(0.25), dist.quantile(0.5), dist.quantile(0.75)]
CI_y = [0.25, 0.5, 0.75]

plt.figure(figsize=(16,9))
plt.subplot(3,5,1)
fit.distribution.PDF()
plt.title('PDF')

plt.subplot(3,5,2)
fit.distribution.CDF(plot_CI=False)
plt.title('CDF')
```

(continues on next page)

(continued from previous page)

```

plt.subplot(3,5,7)
lower, point, upper = fit.distribution.CDF(CI_y=CI_y,CI_type='time')
plt.scatter(lower,CI_y,color='blue')
plt.scatter(point,CI_y,color='purple')
plt.scatter(upper,CI_y,color='red')
plt.title('CDF time')

plt.subplot(3,5,12)
lower, point, upper = fit.distribution.CDF(CI_x=CI_x,CI_type='rel')
plt.scatter(CI_x,lower,color='blue')
plt.scatter(CI_x,point,color='purple')
plt.scatter(CI_x,upper,color='red')
plt.title('CDF reliability')

plt.subplot(3,5,3)
fit.distribution.SF(plot_CI=False)
plt.title('SF')

plt.subplot(3,5,8)
lower, point, upper = fit.distribution.SF(CI_y=CI_y,CI_type='time')
plt.scatter(lower,CI_y,color='blue')
plt.scatter(point,CI_y,color='purple')
plt.scatter(upper,CI_y,color='red')
plt.title('SF time')

plt.subplot(3,5,13)
lower, point, upper = fit.distribution.SF(CI_x=CI_x,CI_type='rel')
plt.scatter(CI_x,lower,color='blue')
plt.scatter(CI_x,point,color='purple')
plt.scatter(CI_x,upper,color='red')
plt.title('SF reliability')

plt.subplot(3,5,4)
fit.distribution.HF()
plt.title('HF')

plt.subplot(3,5,5)
fit.distribution.CHF(plot_CI=False)
plt.title('CHF')

plt.subplot(3,5,10)
CI_y_chf = [1,3,5]
lower, point, upper = fit.distribution.CHF(CI_y=CI_y_chf,CI_type='time')
plt.scatter(lower,CI_y_chf,color='blue')
plt.scatter(point,CI_y_chf,color='purple')
plt.scatter(upper,CI_y_chf,color='red')
plt.title('CHF time')

plt.subplot(3,5,15)
lower, point, upper = fit.distribution.CHF(CI_x=CI_x,CI_type='rel')
plt.scatter(CI_x,lower,color='blue')

```

(continues on next page)

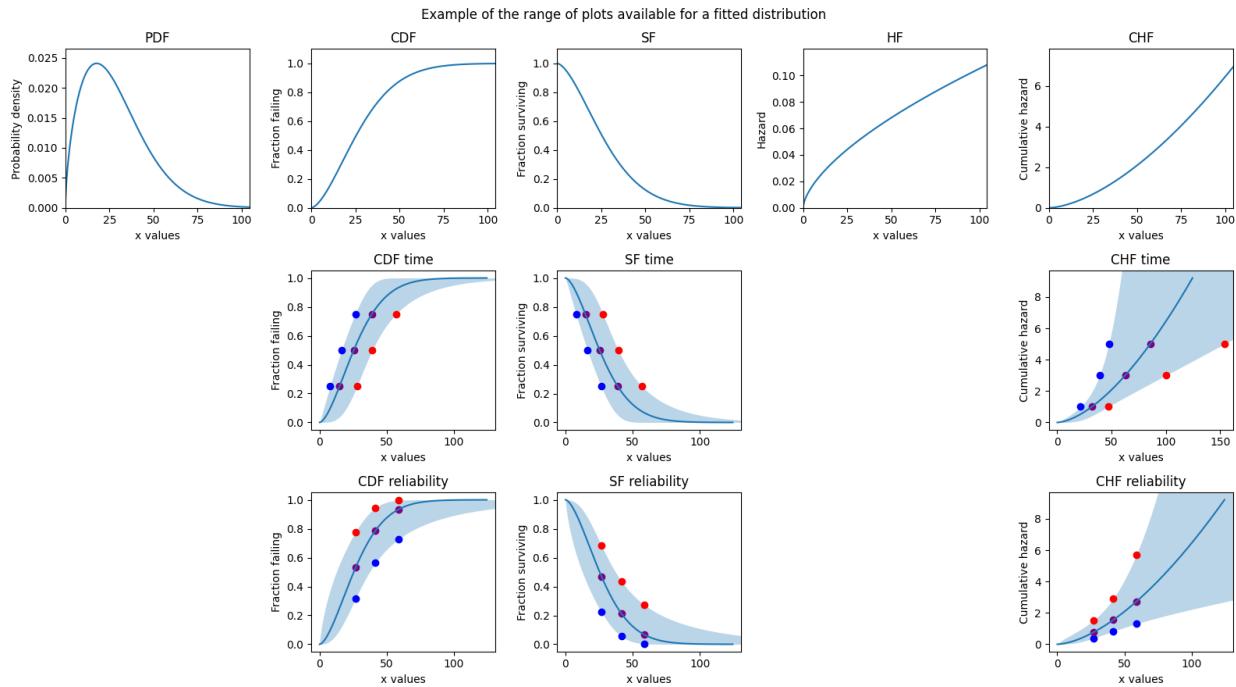
(continued from previous page)

```

plt.scatter(CI_x,point,color='purple')
plt.scatter(CI_x,upper,color='red')
plt.title('CHF reliability')

plt.suptitle('Example of the range of plots available for a fitted distribution')
plt.tight_layout()
plt.show()

```



8.3 Example 3

Now, let's get into a realistic example. We begin by importing the automotive dataset from the Datasets module. This dataset provides failure times (in miles) and right censored times (in miles) for a fleet of vehicles on test. We will fit a Weibull Distribution, and then from the fitted distribution, we want to know the system reliability after 100000 miles.

```

from reliability.Datasets import automotive
from reliability.Fitters import Fit_Weibull_2P
import matplotlib.pyplot as plt

fit = Fit_Weibull_2P(failures=automotive().failures,right_censored=automotive().right_censored,show_probability_plot=False)

# we want to know the system reliability after 100000 miles
lower,point,upper = fit.distribution.SF(CI_x=100000,CI_type='reliability',CI=0.8)
plt.scatter([100000,100000,100000],[lower,point,upper],color='black')

print('Failures:',automotive().failures)
print('Right censored:',automotive().right_censored)
print('')

```

(continues on next page)

(continued from previous page)

```
print("The 80% reliability estimates at 100000 miles are:")
print("lower bound: "+"{:.2%}".format(lower))
print("point estimate: "+"{:.2%}".format(point))
print("upper bound: "+"{:.2%}".format(upper))
plt.show()
```

```
""
```

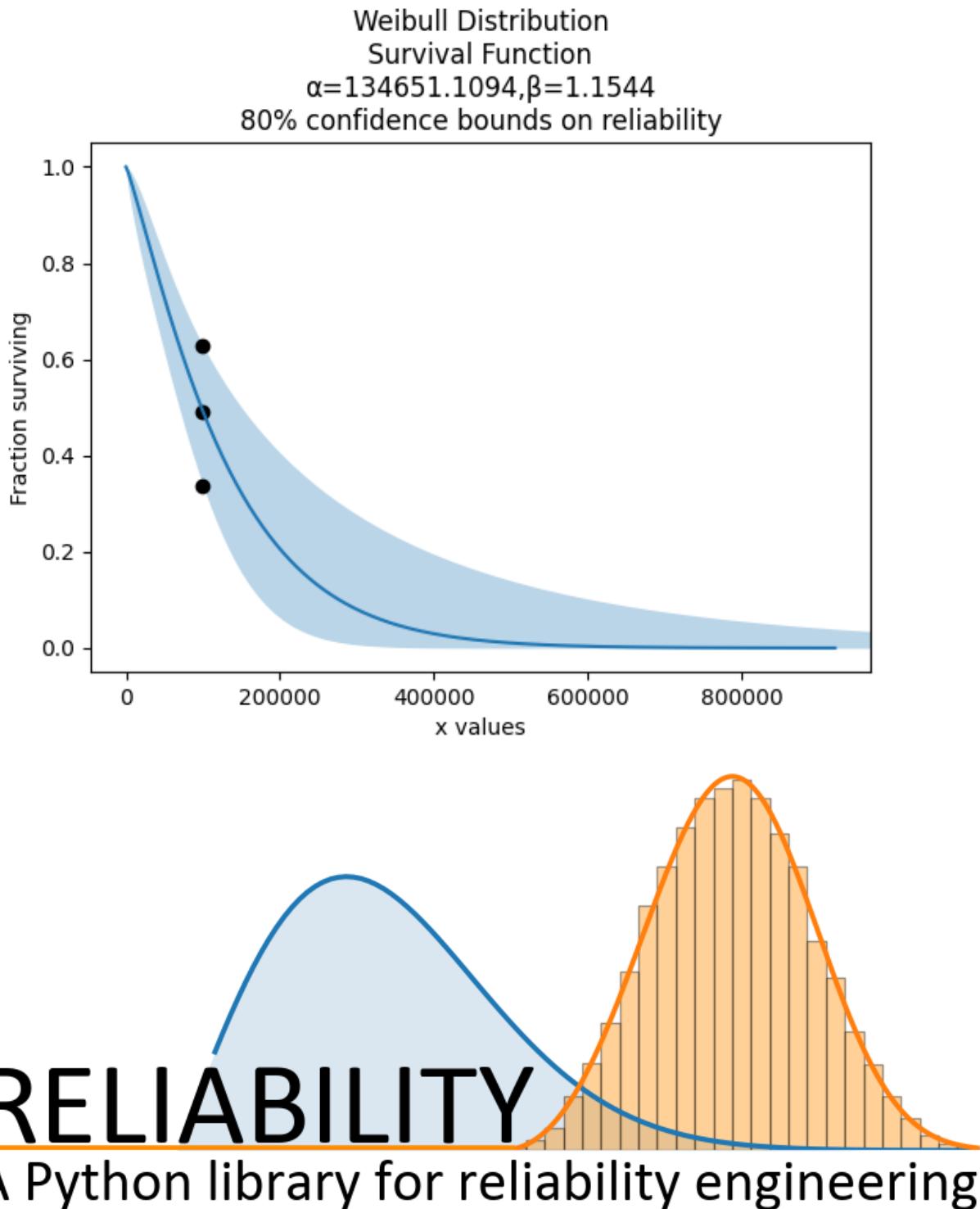
Results from Fit_Weibull_2P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 10/21 (67.74194% right censored)

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha	134651	42767.3	72252.9	250937
Beta	1.15443	0.29614	0.698249	1.90863

Goodness of fit	Value
Log-likelihood	-128.974
AICc	262.376
BIC	264.816
AD	35.6052

*Failures: [5248, 7454, 16890, 17200, 38700, 45000, 49390, 69040, 72280, 131900]
 Right censored: [3961, 4007, 4734, 6054, 7298, 10190, 23060, 27160, 28690, 37100, 40060, ↪
 ↪ 45670, 53000, 67000, 69630, 77350, 78470, 91680, 105700, 106300, 150400]*

*The 80% reliability estimates at 100000 miles are:
 lower bound: 33.83%
 point estimate: 49.20%
 upper bound: 62.87%
 ""*



MIXTURE MODELS

9.1 What are mixture models?

Mixture models are a combination of two or more distributions added together to create a distribution that has a shape with more flexibility than a single distribution. Each of the mixture's components must be multiplied by a proportion, and the sum of all the proportions is equal to 1. The mixture is generally written in terms of the PDF, but since the CDF is the integral (cumulative sum) of the PDF, we can equivalently write the Mixture model in terms of the PDF or CDF. For a mixture model with 2 distributions, the equations are shown below:

$$PDF_{mixture} = p \times PDF_1 + (1 - p) \times PDF_2$$

$$CDF_{mixture} = p \times CDF_1 + (1 - p) \times CDF_2$$

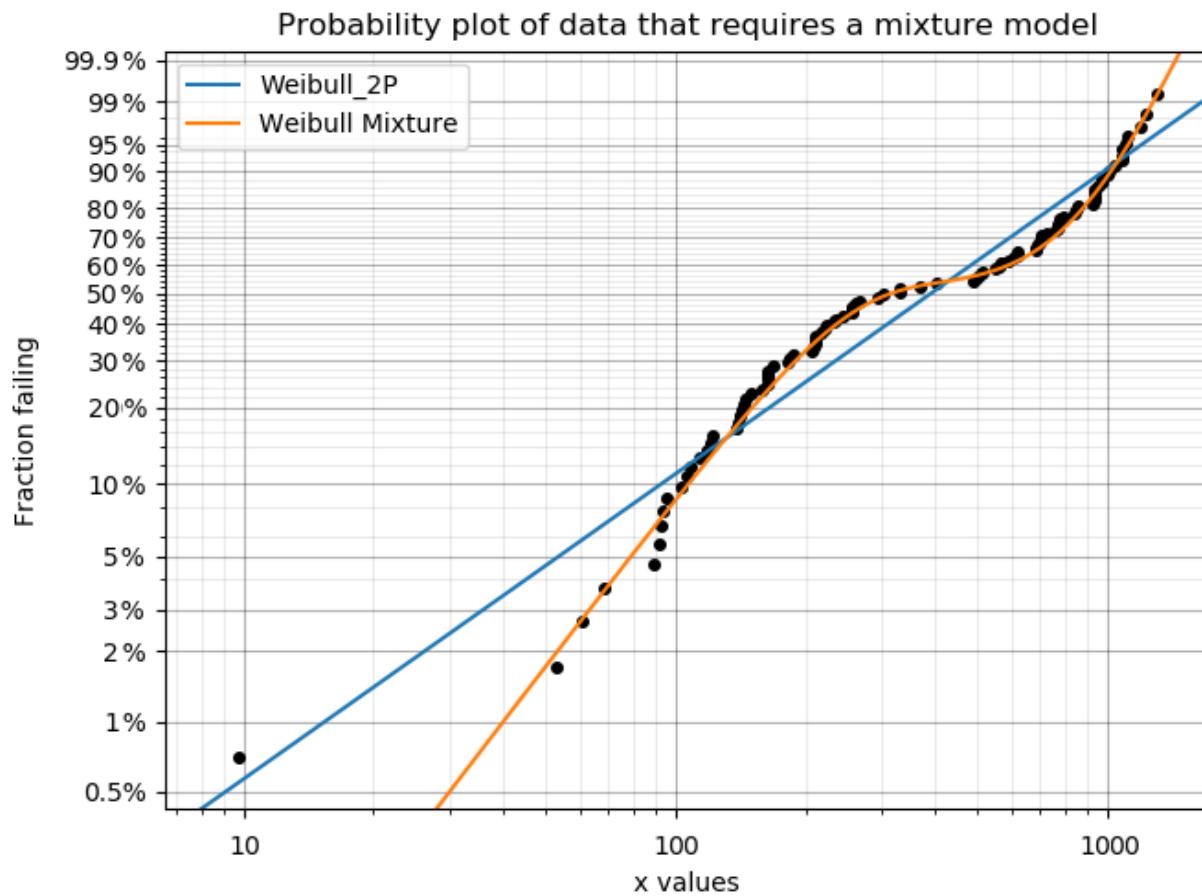
$$SF_{mixture} = 1 - CDF_{mixture}$$

$$HF_{mixture} = \frac{PDF_{mixture}}{SF_{mixture}}$$

$$CHF_{mixture} = -\ln(SF_{mixture})$$

Mixture models are useful when there is more than one failure mode that is generating the failure data. This can be recognised by the shape of the PDF and CDF being outside of what any single distribution can accurately model. On a probability plot, a mixture of failure modes can be identified by bends or S-shapes in the data that you might otherwise expect to be linear. An example of this is shown in the image below. You should not use a mixture model just because it can fit almost anything really well, but you should use a mixture model if you suspect that there are multiple failure modes contributing to the failure data you are observing. To judge whether a mixture model is justified, look at the goodness of fit criterion (AICc or BIC) which penalises the score based on the number of parameters in the model. The closer the goodness of fit criterion is to zero, the better the fit. Using AD or log-likelihood for this check is not appropriate as these goodness of fit criterions do not penalise the score based on the number of parameters in the model and are therefore prone to overfitting.

See also [competing risk models](#) for another method of combining distributions using the product of the SF rather than the sum of the CDF.



9.2 Creating a mixture model

Within `reliability.Distributions` is the `Mixture_Model`. This function accepts an array or list of standard distribution objects created using the `reliability.Distributions` module (available distributions are Exponential, Weibull, Gumbel, Normal, Lognormal, Loglogistic, Gamma, Beta). There is no limit to the number of components you can add to the mixture, but it is generally preferable to use as few as are required to fit the data appropriately (typically 2 or 3). In addition to the distributions, you can specify the proportions contributed by each distribution in the mixture. These proportions must sum to 1. If not specified the proportions will be set as equal for each component.

As this process is additive for the survival function, and may accept many distributions of different types, the mathematical formulation quickly gets complex. For this reason, the algorithm combines the models numerically rather than empirically so there are no simple formulas for many of the descriptive statistics (mean, median, etc.). Also, the accuracy of the model is dependent on `xvals`. If the `xvals` array is small (<100 values) then the answer will be “blocky” and inaccurate. The variable `xvals` is only accepted for `PDF`, `CDF`, `SF`, `HF`, and `CHF`. The other methods (like random samples) use the default `xvals` for maximum accuracy. The default number of values generated when `xvals` is not given is 1000. Consider this carefully when specifying `xvals` in order to avoid inaccuracies in the results.

API Reference

For inputs and outputs see the [API reference](#).

9.2.1 Example 1

The following example shows how the Mixture_Model object can be created, visualised and used.

```
from reliability.Distributions import Lognormal_Distribution, Gamma_Distribution, Weibull_Distribution, Mixture_Model
import matplotlib.pyplot as plt

# create the mixture model
d1 = Lognormal_Distribution(mu=2, sigma=0.8)
d2 = Weibull_Distribution(alpha=50, beta=5, gamma=100)
d3 = Gamma_Distribution(alpha=5, beta=3, gamma=30)
mixture_model = Mixture_Model(distributions=[d1, d2, d3], proportions=[0.3, 0.4, 0.3])

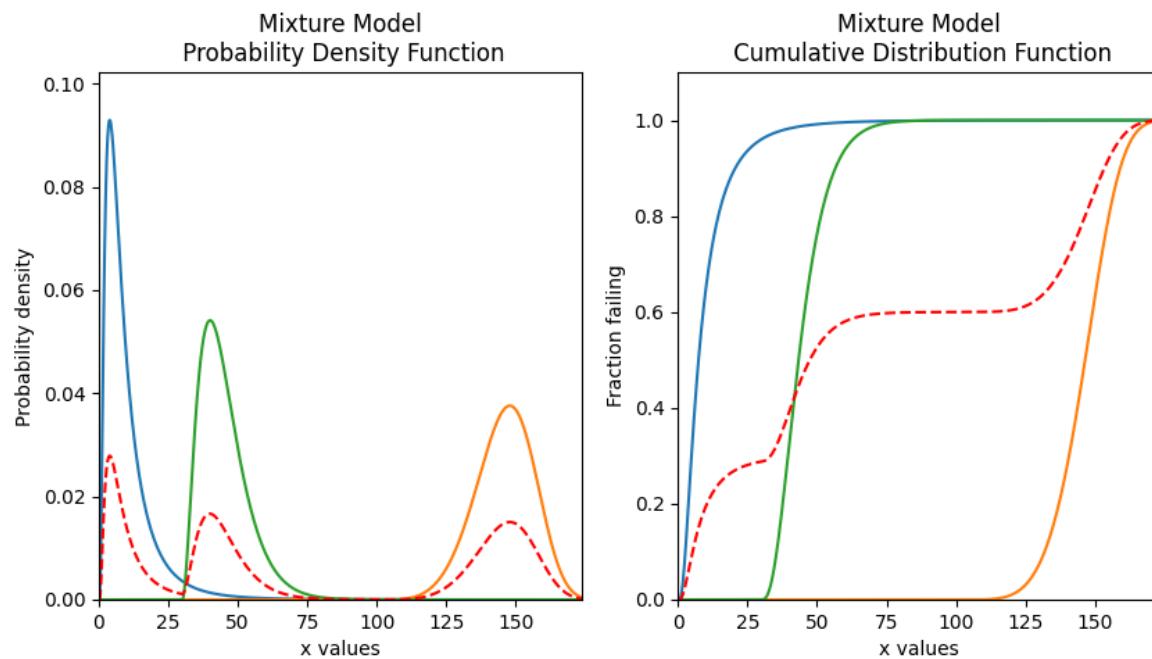
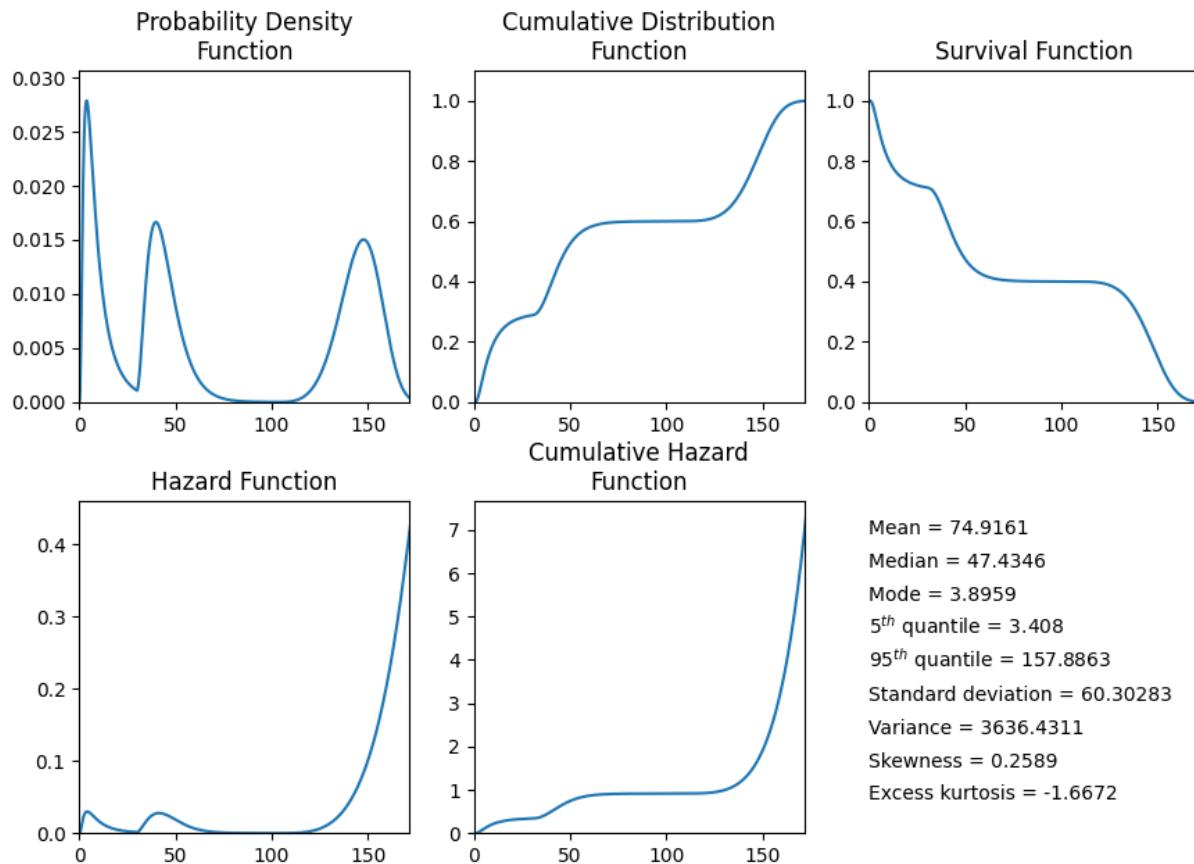
# plot the 5 functions using the plot() function
mixture_model.plot()

# plot the PDF and CDF
plot_components = True # this plots the component distributions. Default is False
plt.figure(figsize=(9, 5))
plt.subplot(121)
mixture_model.PDF(plot_components=plot_components, color='red', linestyle='--')
plt.subplot(122)
mixture_model.CDF(plot_components=plot_components, color='red', linestyle='--')
plt.subplots_adjust(left=0.1, right=0.95)
plt.show()

# extract the mean of the distribution
print('The mean of the distribution is:', mixture_model.mean)

"""
The mean of the distribution is: 74.91607709895453
""
```

Mixture Model



9.3 Fitting a mixture model

Within `reliability.Fitters` is `Fit_Weibull_Mixture`. This function will fit a Weibull Mixture Model consisting of 2 x Weibull_2P distributions (this does not fit the gamma parameter). Just as with all of the other distributions in `reliability.Fitters`, right censoring is supported, though care should be taken to ensure that there still appears to be two groups when plotting only the failure data. A second group cannot be made from a mostly or totally censored set of samples.

Whilst some failure modes may not be fitted as well by a Weibull distribution as they may be by another distribution, it is unlikely that a mixture of data from two distributions (particularly if they are overlapping) will be fitted noticeably better by other types of mixtures than would be achieved by a Weibull mixture. For this reason, other types of mixtures are not implemented.

API Reference

For inputs and outputs see the [API reference](#).

9.3.1 Example 2

In this example, we will create some data using two Weibull distributions and then combine the data using `np.hstack`. We will then fit the Weibull mixture model to the combined data and will print the results and show the plot. As the input data is made up of 40% from the first group, we expect the proportion to be around 0.4.

```
from reliability.Fitters import Fit_Weibull_Mixture
from reliability.Distributions import Weibull_Distribution
from reliability.Other_functions import histogram
import numpy as np
import matplotlib.pyplot as plt

# create some failures from two distributions
group_1 = Weibull_Distribution(alpha=10, beta=3).random_samples(40, seed=2)
group_2 = Weibull_Distribution(alpha=40, beta=4).random_samples(60, seed=2)
all_data = np.hstack([group_1, group_2]) # combine the data
results = Fit_Weibull_Mixture(failures=all_data) #fit the mixture model

# this section is to visualise the histogram with PDF and CDF
# it is not part of the default output from the Fitter
plt.figure(figsize=(9, 5))
plt.subplot(121)
histogram(all_data)
results.distribution.PDF()
plt.subplot(122)
histogram(all_data, cumulative=True)
results.distribution.CDF()

plt.show()

"""

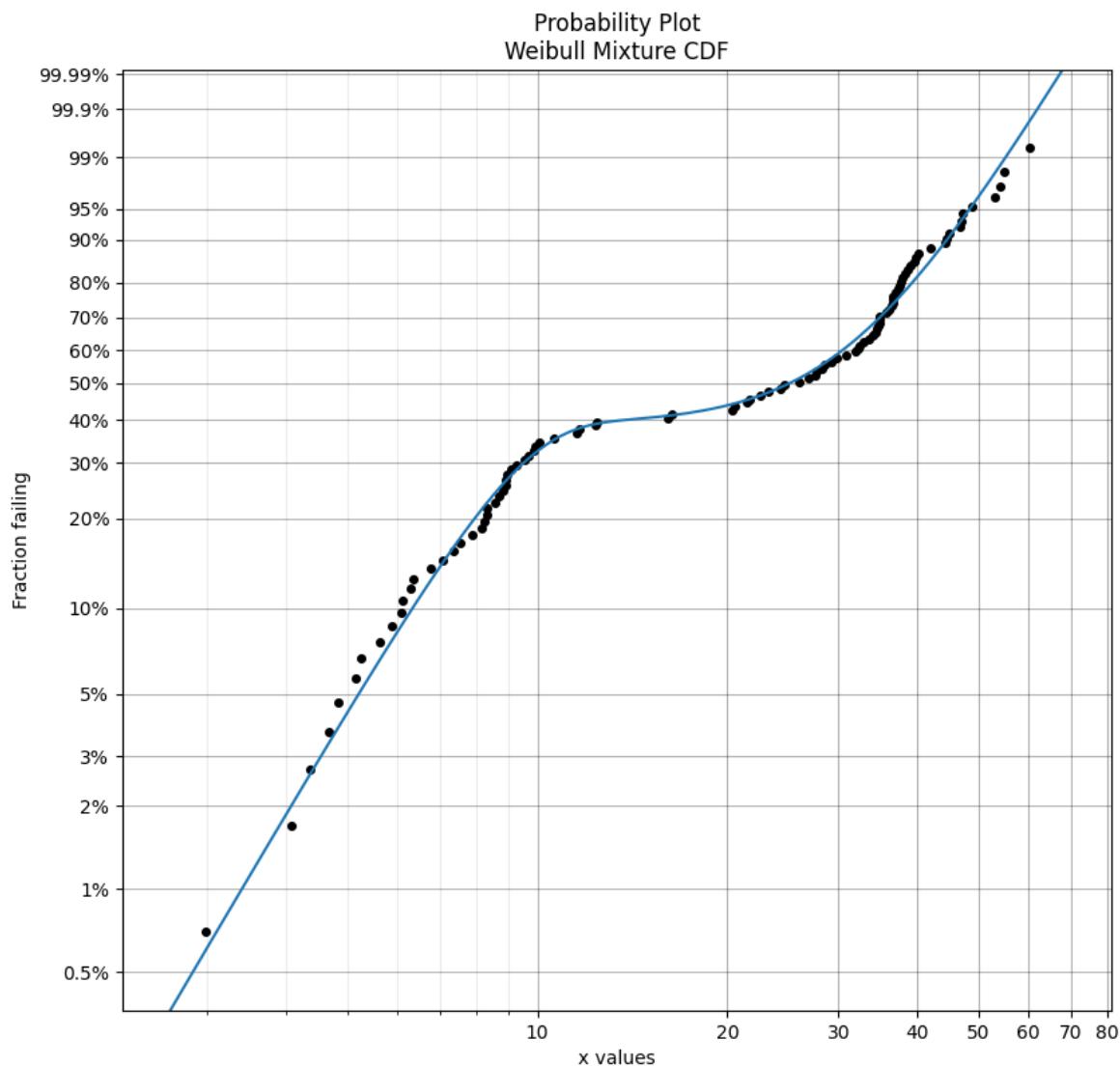
Results from Fit_Weibull_Mixture (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 100/0 (0% right censored)
```

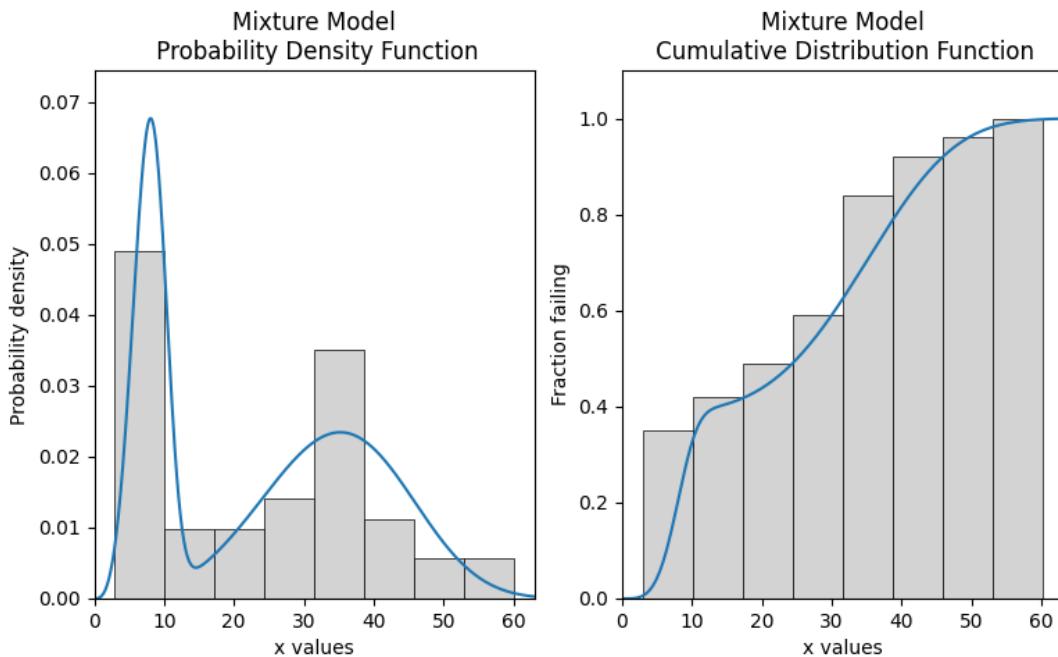
Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
-----------	----------------	----------------	----------	----------

(continues on next page)

(continued from previous page)

Alpha 1	8.65511	0.393835	7.91663	9.46248
Beta 1	3.91197	0.509776	3.03021	5.0503
Alpha 2	38.1103	1.41075	35.4431	40.9781
Beta 2	3.82192	0.421385	3.07916	4.74385
Proportion 1	0.388491	0.0502663	0.295595	0.490263
<i>Goodness of fit</i>				
<i>Log-likelihood</i>				
-375.991				
AICc				
762.619				
BIC				
775.007				
AD				
0.418649				
"				





9.3.2 Example 3

In this example, we will compare how well the Weibull Mixture performs vs a single Weibull_2P. Firstly, we generate some data from two Weibull distributions, combine the data, and right censor it above our chosen threshold. Next, we will fit the Mixture and Weibull_2P distributions. Then we will visualise the histogram and PDF of the fitted mixture model and Weibull_2P distributions. The goodness of fit measure is used to check whether the mixture model is really a much better fit than a single Weibull_2P distribution (which it is due to the lower BIC).

```
from reliability.Fitters import Fit_Weibull_Mixture, Fit_Weibull_2P
from reliability.Distributions import Weibull_Distribution
from reliability.Other_functions import histogram, make_right_censored_data
import numpy as np
import matplotlib.pyplot as plt

# create some failures and right censored data
group_1 = Weibull_Distribution(alpha=10, beta=2).random_samples(700, seed=2)
group_2 = Weibull_Distribution(alpha=30, beta=3).random_samples(300, seed=2)
all_data = np.hstack([group_1, group_2])
data = make_right_censored_data(all_data, threshold=30)

# fit the Weibull Mixture and Weibull_2P
mixture = Fit_Weibull_Mixture(failures=data.failures, right_censored=data.right_censored,
    show_probability_plot=False, print_results=False)
single = Fit_Weibull_2P(failures=data.failures, right_censored=data.right_censored, show_
    probability_plot=False, print_results=False)
print('Weibull_Mixture BIC:', mixture.BIC, '\nWeibull_2P BIC:', single.BIC) # print the_
    goodness of fit measure

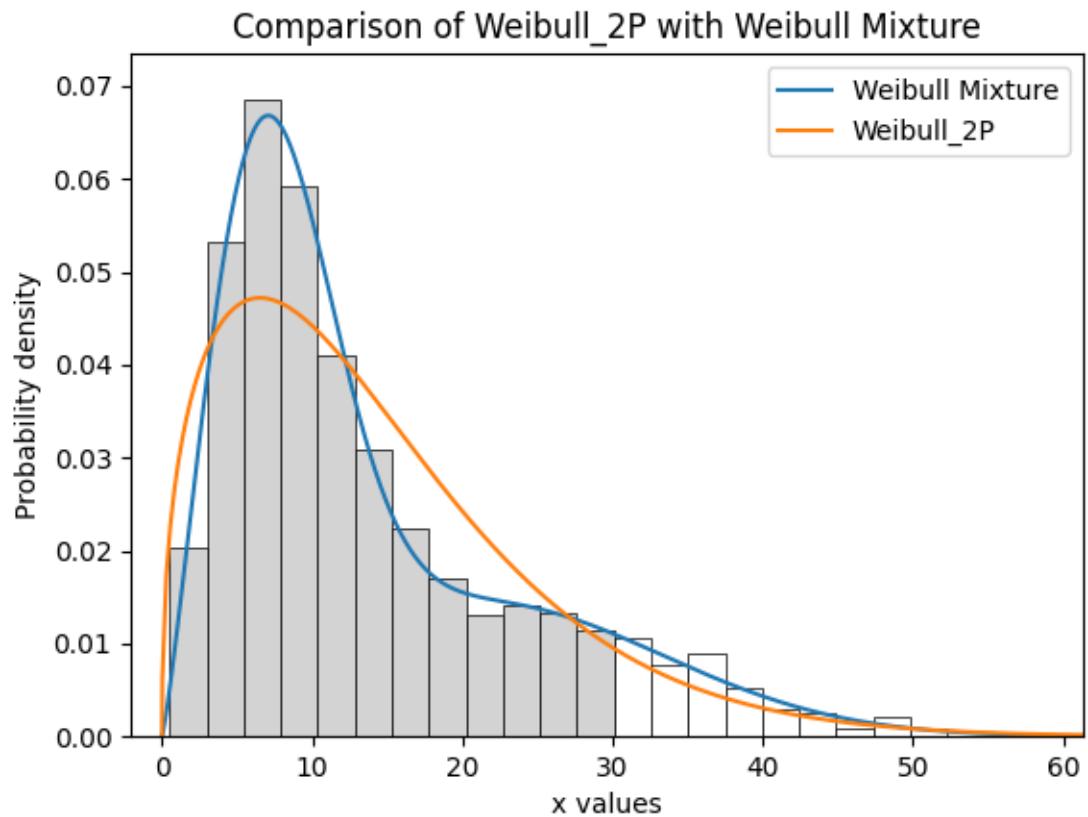
# plot the Mixture and Weibull_2P
histogram(all_data, white_above=30)
mixture.distribution.PDF(label='Weibull Mixture')
```

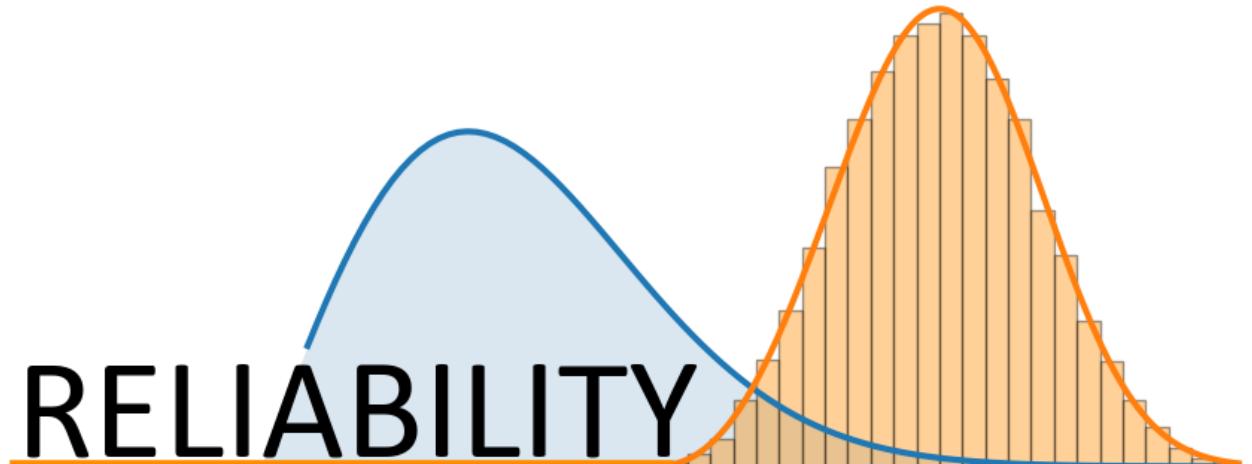
(continues on next page)

(continued from previous page)

```
single.distribution.PDF(label='Weibull_2P')
plt.title('Comparison of Weibull_2P with Weibull Mixture')
plt.legend()
plt.show()

"""
Weibull_Mixture BIC: 6431.578404076784
Weibull_2P BIC: 6511.511759597337
""
```





RELIABILITY
A Python library for reliability engineering

COMPETING RISKS MODELS

10.1 What are competing risks models?

Competing risks models are a combination of two or more distributions that represent failure modes which are “competing” to end the life of the system being modelled. This model is similar to a [mixture model](#) in the sense that it uses multiple distributions to create a new model that has a shape with more flexibility than a single distribution. However, unlike in mixture models, we are not adding proportions of the PDF or CDF, but are instead multiplying the survival functions. The formula for the competing risks model is typically written in terms of the survival function (SF). Since we may consider the system’s reliability to depend on the reliability of all the parts of the system (each with its own failure modes), the equation is written as if the system was in series, using the product of the survival functions for each failure mode. For a competing risks model with 2 distributions, the equations are shown below:

$$SF_{\text{Competing Risks}} = SF_1 \times SF_2$$

$$CDF_{\text{Competing Risks}} = 1 - SF_{\text{Competing Risks}}$$

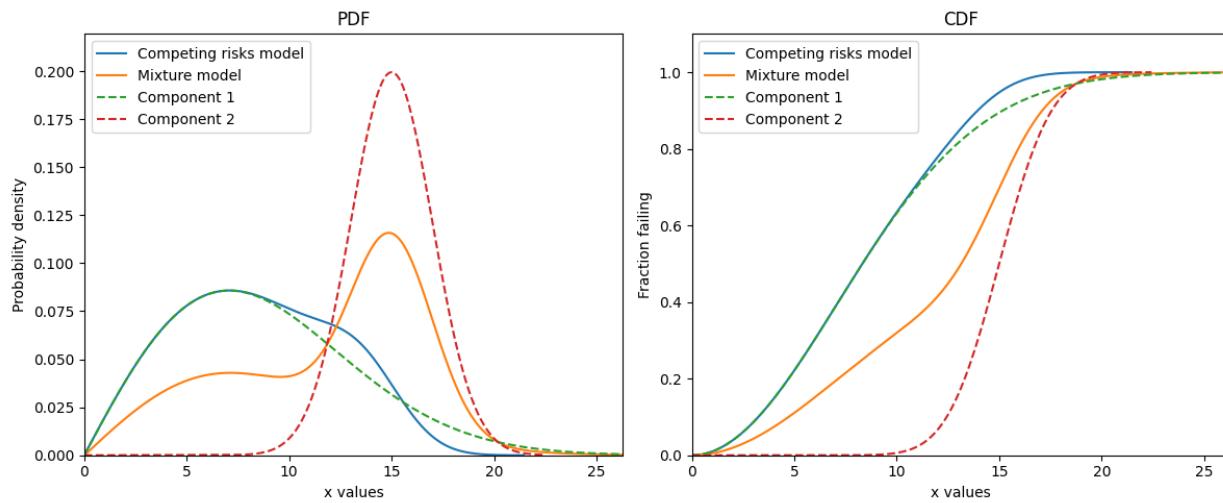
Since $SF = \exp(-CHF)$ we may equivalently write the competing risks model in terms of the hazard or cumulative hazard function as:

$$HF_{\text{Competing Risks}} = HF_1 + HF_2$$

$$CHF_{\text{Competing Risks}} = CHF_1 + CHF_2$$

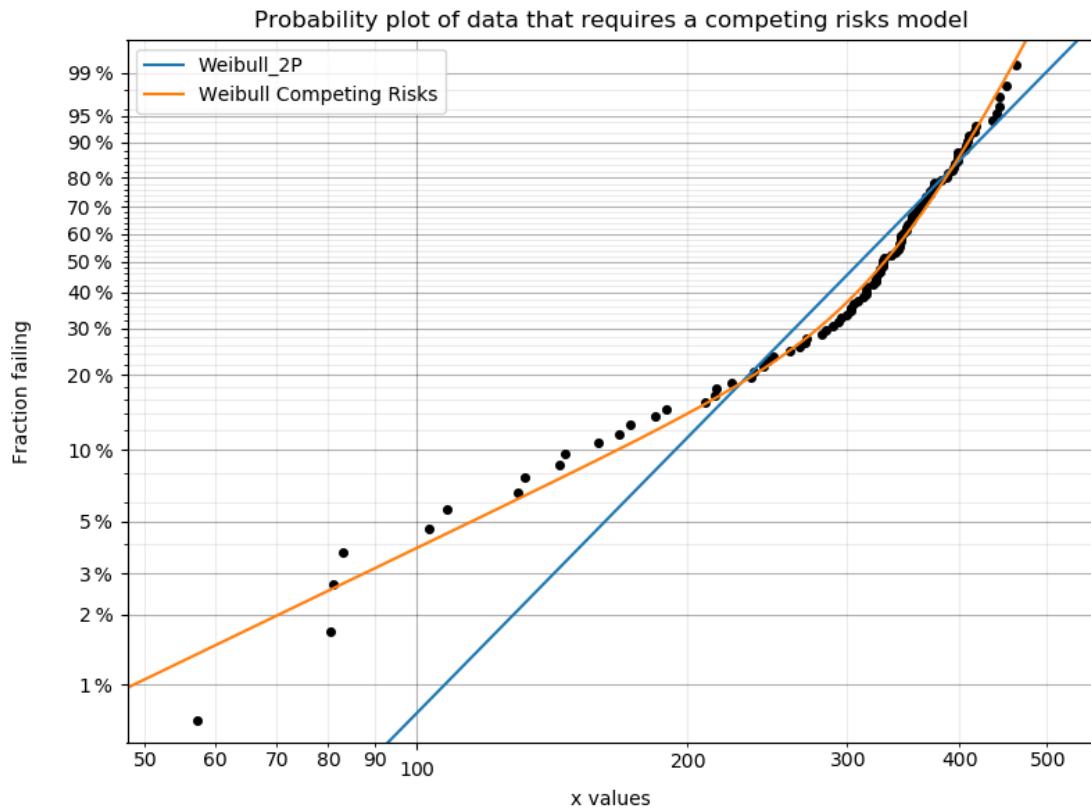
$$PDF_{\text{Competing Risks}} = HF_{\text{Competing Risks}} \times SF_{\text{Competing Risks}}$$

The image below illustrates the difference between the competing risks model and the mixture model, each of which is made up of the same two component distributions. Note that the PDF of the competing risks model is always equal to or to the left of the component distributions, and the CDF is equal to or higher than the component distributions. This shows how a failure mode that occurs earlier in time can end the lives of units under observation before the second failure mode has the chance to. This behaviour is characteristic of real systems which experience multiple failure modes, each of which could cause system failure.



Competing risks models are useful when there is more than one failure mode that is generating the failure data. This can be recognised by the shape of the PDF and CDF being outside of what any single distribution can accurately model. On a probability plot, a combination of failure modes can be identified by bends in the data that you might otherwise expect to be linear. An example of this is shown in the image below. You should not use a competing risks model just because it fits your data better than a single distribution, but you should use a competing risks model if you suspect that there are multiple failure modes contributing to the failure data you are observing. To judge whether a competing risks model is justified, look at the goodness of fit criterion (AICc or BIC) which penalises the score based on the number of parameters in the model. The closer the goodness of fit criterion is to zero, the better the fit. It is not appropriate to use the Log-likelihood or AD goodness of fit criterions as these do not penalise the score based on the number of parameters, therefore making the model susceptible to overfitting.

See also [mixture models](#) for another method of combining distributions using the sum of the CDF rather than the product of the SF.



10.2 Creating a competing risks model

Within `reliability.Distributions` is the `Competing_Risks_Model`. This function accepts an array or list of distribution objects created using the `reliability.Distributions` module (available distributions are Exponential, Weibull, Gumbel, Normal, Lognormal, Loglogistic, Gamma, Beta). There is no limit to the number of components you can add to the model, but it is generally preferable to use as few as are required to fit the data appropriately (typically 2 or 3). Unlike the mixture model, you do not need to specify any proportions.

As this process is multiplicative for the survival function (or additive for the hazard function), and may accept many distributions of different types, the mathematical formulation quickly gets complex. For this reason, the algorithm combines the models numerically rather than empirically so there are no simple formulas for many of the descriptive statistics (mean, median, etc.). Also, the accuracy of the model is dependent on `xvals`. If the `xvals` array is small (<100 values) then the answer will be “blocky” and inaccurate. The variable `xvals` is only accepted for `PDF`, `CDF`, `SF`, `HF`, and `CHF`. The other methods (like random samples) use the default `xvals` for maximum accuracy. The default number of values generated when `xvals` is not given is 1000. Consider this carefully when specifying `xvals` in order to avoid inaccuracies in the results.

API Reference

For inputs and outputs see the [API reference](#).

10.2.1 Example 1

The following example shows how the Competing_Risks_Model object can be created, visualised and used.

```
from reliability.Distributions import Lognormal_Distribution, Gamma_Distribution, Weibull_Distribution, Competing_Risks_Model
import matplotlib.pyplot as plt

# create the competing risks model
d1 = Lognormal_Distribution(mu=4, sigma=0.1)
d2 = Weibull_Distribution(alpha=50, beta=2)
d3 = Gamma_Distribution(alpha=30, beta=1.5)
CR_model = Competing_Risks_Model(distributions=[d1, d2, d3])

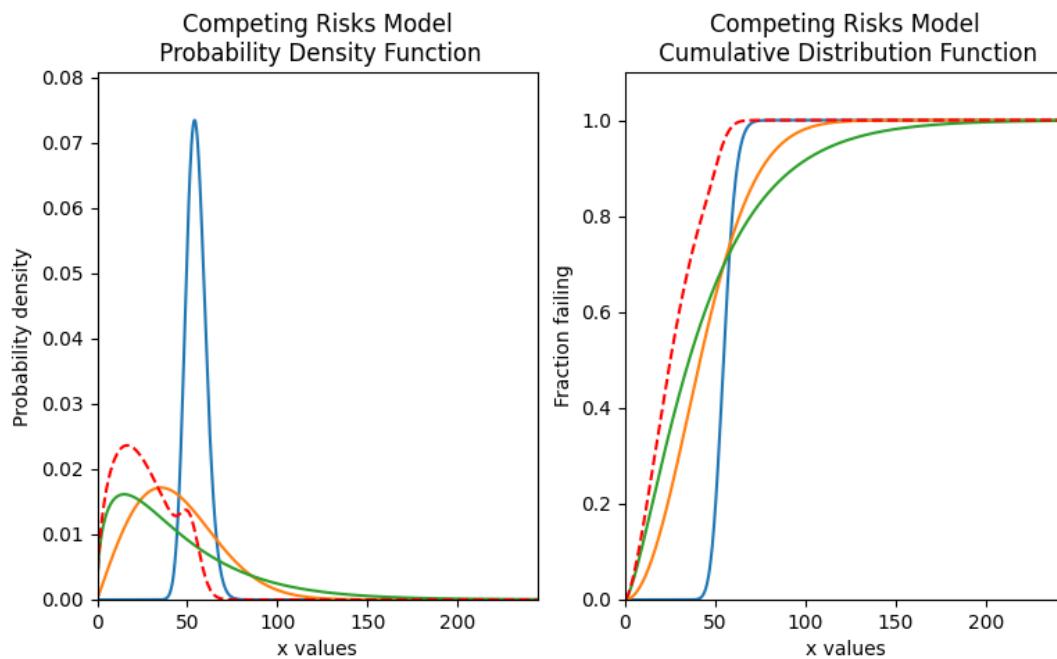
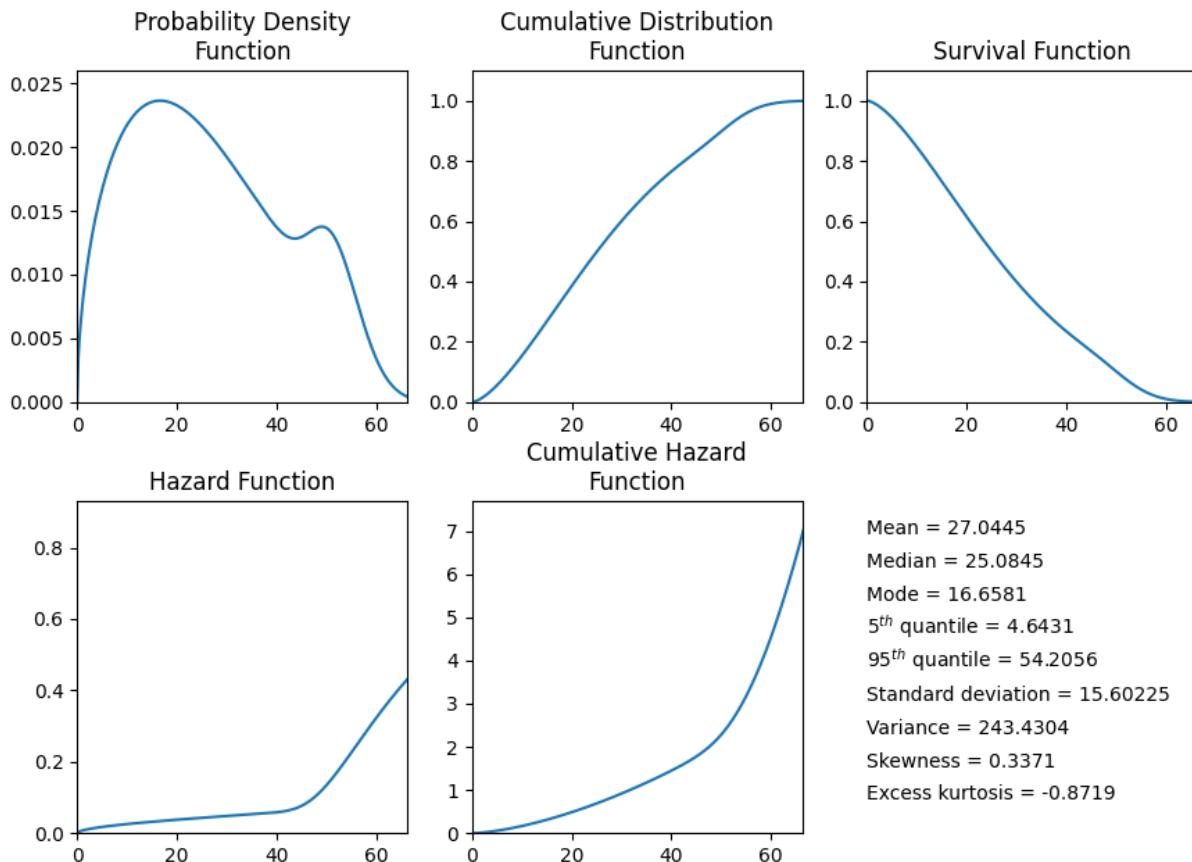
# plot the 5 functions using the plot() function
CR_model.plot()

# plot the PDF and CDF
plot_components = True # this plots the component distributions. Default is False
plt.figure(figsize=(9, 5))
plt.subplot(121)
CR_model.PDF(plot_components=plot_components, color='red', linestyle='--')
plt.subplot(122)
CR_model.CDF(plot_components=plot_components, color='red', linestyle='--')
plt.show()

# extract the mean of the distribution
print('The mean of the distribution is:', CR_model.mean)

"""
The mean of the distribution is: 27.04449126273065
""
```

Competing Risks Model



10.3 Fitting a competing risks model

Within *reliability.Fitters* is *Fit_Weibull_CR*. This function will fit a Weibull Competing Risks Model consisting of 2 x Weibull_2P distributions (this does not fit the gamma parameter). Just as with all of the other distributions in *reliability.Fitters*, right censoring is supported.

Whilst some failure modes may not be fitted as well by a Weibull distribution as they may be by another distribution, it is unlikely that a competing risks model of data from two distributions (particularly if they are overlapping) will be fitted noticeably better by other types of competing risks models than would be achieved by a Weibull Competing Risks Model. For this reason, other types of competing risks models are not implemented.

API Reference

For inputs and outputs see the [API reference](#).

10.3.1 Example 2

In this example, we will create some data using a competing risks model from two Weibull distributions. We will then fit the Weibull mixture model to the data and will print the results and show the plot.

```
from reliability.Distributions import Weibull_Distribution, Competing_Risks_Model
from reliability.Fitters import Fit_Weibull_CR
from reliability.Other_functions import histogram
import matplotlib.pyplot as plt

# create some data that requires a competing risks models
d1 = Weibull_Distribution(alpha=50, beta=2)
d2 = Weibull_Distribution(alpha=40, beta=10)
CR_model = Competing_Risks_Model(distributions=[d1, d2])
data = CR_model.random_samples(100, seed=2)

# fit the Weibull competing risks model
results = Fit_Weibull_CR(failures=data)

# this section is to visualise the histogram with PDF and CDF
# it is not part of the default output from the Fitter
plt.figure(figsize=(9, 5))
plt.subplot(121)
histogram(data)
results.distribution.PDF()
plt.subplot(122)
histogram(data, cumulative=True)
results.distribution.CDF()

plt.show()

"""

Results from Fit_Weibull_CR (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: L-BFGS-B
Failures / Right censored: 100/0 (0% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
```

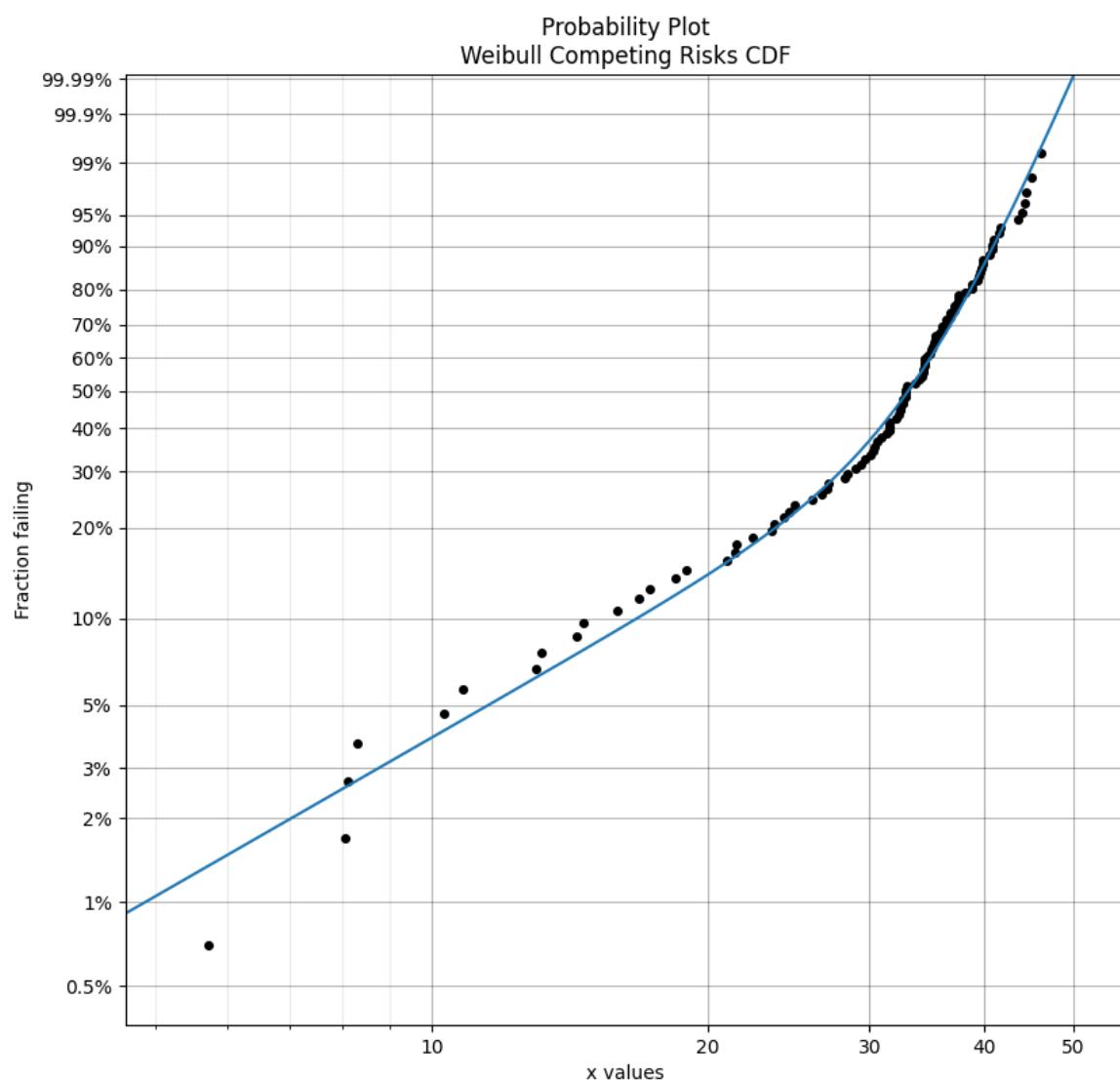
(continues on next page)

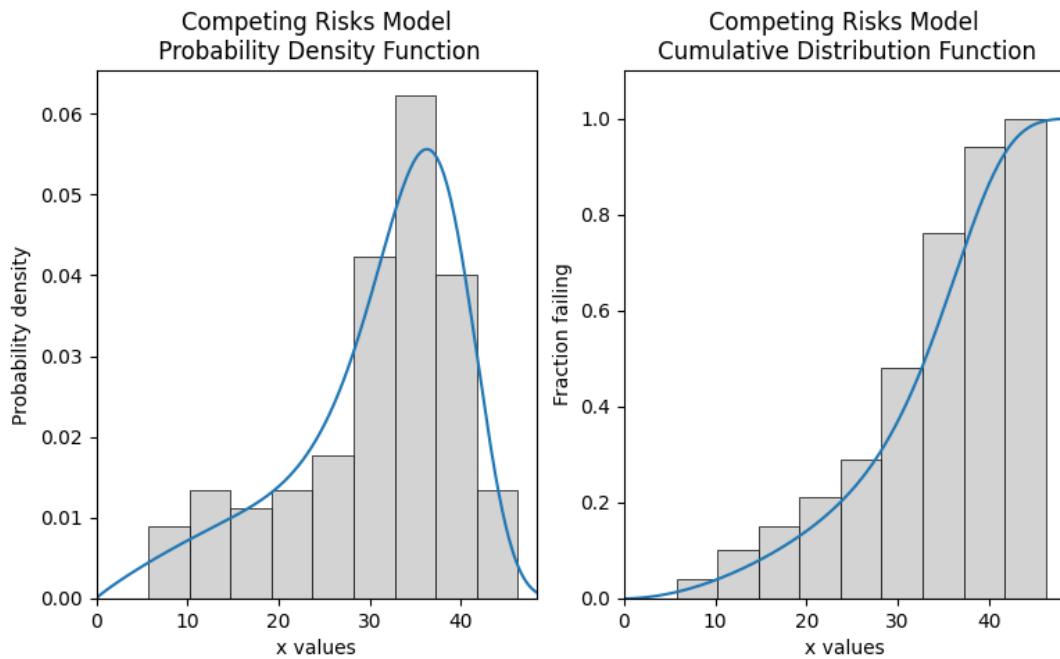
(continued from previous page)

<i>Alpha 1</i>	55.2695	14.3883	33.1812	92.0615
<i>Beta 1</i>	1.89484	0.452994	1.18598	3.02738
<i>Alpha 2</i>	38.175	1.07992	36.116	40.3514
<i>Beta 2</i>	7.97514	1.18035	5.96701	10.6591

Goodness of fit *Value*
Log-likelihood -352.479
AICc 713.38
BIC 723.379
AD 0.390325

"





10.3.2 Example 3

In this example, we will compare the mixture model to the competing risks model. The data is generated from a competing risks model so we expect the Weibull competing risks model to be more appropriate than the Mixture model. Through comparison of the AICc or BIC, we can see which model is more appropriate. Since the AICc and BIC penalise the goodness of fit criterion based on the number of parameters and the mixture model has 5 parameters compared to the competing risk model's 4 parameters, we expect the competing risks model to have a lower (closer to zero) goodness of fit than the Mixture model, and this is what we observe in the results. Notice how the log-likelihood and AD statistics of the mixture model indicates a better fit (because the value is closer to zero), but this does not take into account the number of parameters in the model.

```
from reliability.Distributions import Weibull_Distribution, Competing_Risks_Model
from reliability.Fitters import Fit_Weibull_CR, Fit_Weibull_Mixture
import matplotlib.pyplot as plt
import pandas as pd

# create some data from a competing risks model
d1 = Weibull_Distribution(alpha=250, beta=2)
d2 = Weibull_Distribution(alpha=210, beta=10)
CR_model = Competing_Risks_Model(distributions=[d1, d2])
data = CR_model.random_samples(50, seed=2)

CR_fit = Fit_Weibull_CR(failures=data) # fit the Weibull competing risks model
print('-----')
MM_fit = Fit_Weibull_Mixture(failures=data) # fit the Weibull mixture model
plt.legend()
plt.show()
print('-----')

# create a dataframe to display the goodness of fit criterion as a table
goodness_of_fit = {'Model': ['Competing Risks', 'Mixture'], 'AICc': [CR_fit.AICc, MM_fit.
(continues on next page)
```

(continued from previous page)

```

→AICc], 'BIC': [CR_fit.BIC, MM_fit.BIC], 'AD': [CR_fit.AD, MM_fit.AD]}
df = pd.DataFrame(goodness_of_fit, columns=['Model', 'AICc', 'BIC', 'AD'])
print(df)

```

```
""
```

Results from Fit_Weibull_CR (95% CI):

Analysis method: Maximum Likelihood Estimation (MLE)

Optimizer: L-BFGS-B

Failures / Right censored: 50/0 (0% right censored)

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha 1	229.868	51.2178	148.531	355.744
Beta 1	2.50124	0.747103	1.39286	4.49162
Alpha 2	199.717	8.56554	183.615	217.231
Beta 2	9.20155	2.20135	5.75734	14.7062

Goodness of fit Value

Log-likelihood	-255.444
AICc	519.777
BIC	526.536
AD	0.582534

Results from Fit_Weibull_Mixture (95% CI):

Analysis method: Maximum Likelihood Estimation (MLE)

Optimizer: TNC

Failures / Right censored: 50/0 (0% right censored)

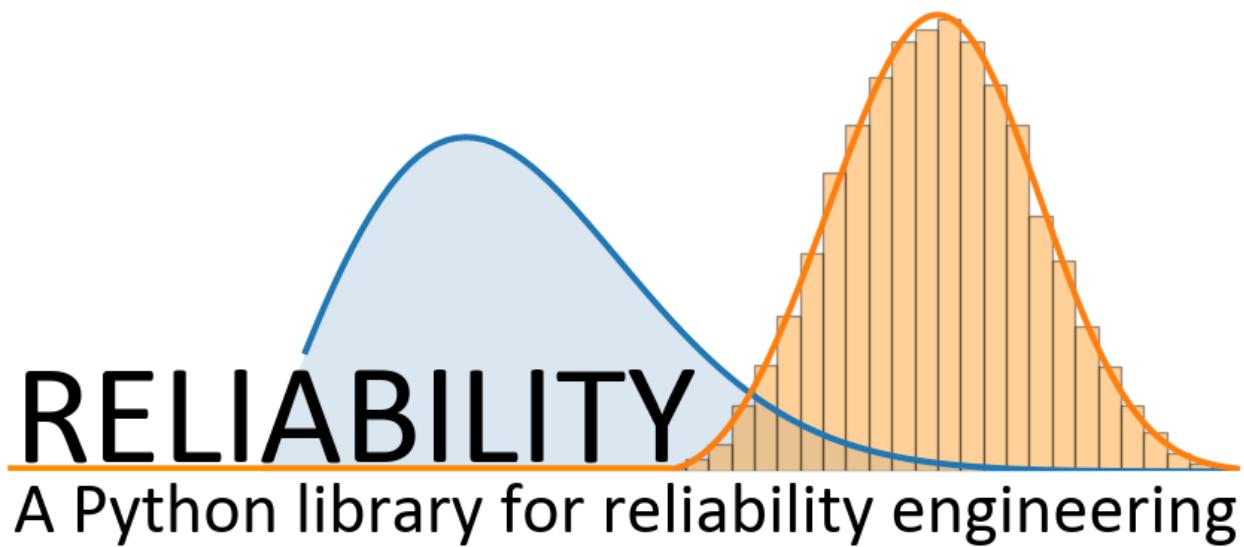
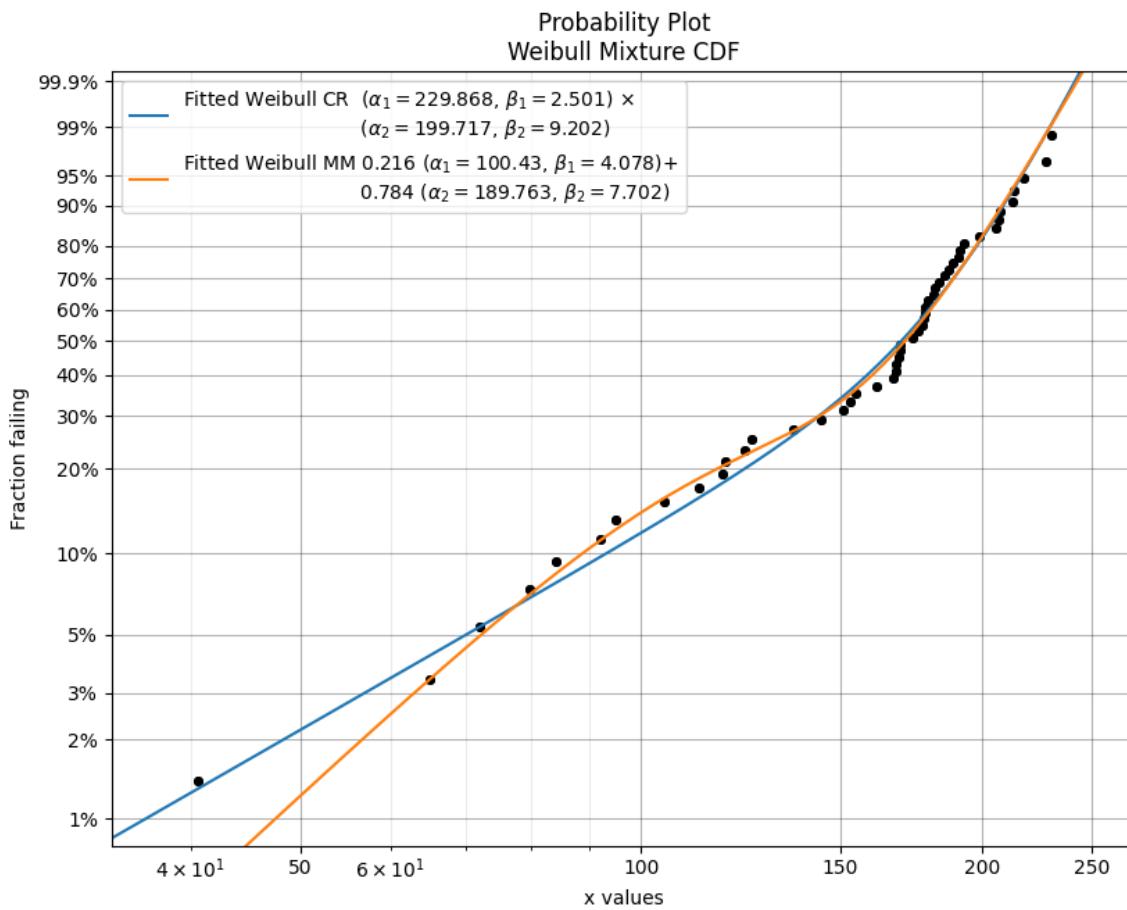
Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha 1	100.43	12.4535	78.761	128.06
Beta 1	4.07764	1.2123	2.27689	7.30257
Alpha 2	189.763	5.13937	179.953	200.108
Beta 2	7.70223	1.35191	5.46024	10.8648
Proportion 1	0.215599	0.0815976	0.0964618	0.414394

Goodness of fit Value

Log-likelihood	-254.471
AICc	520.306
BIC	528.503
AD	0.529294

Model	AICc	BIC	AD
0 Competing Risks	519.777	526.536	0.582534
1 Mixture	520.306	528.503	0.529294

```
""
```



DSZI MODELS

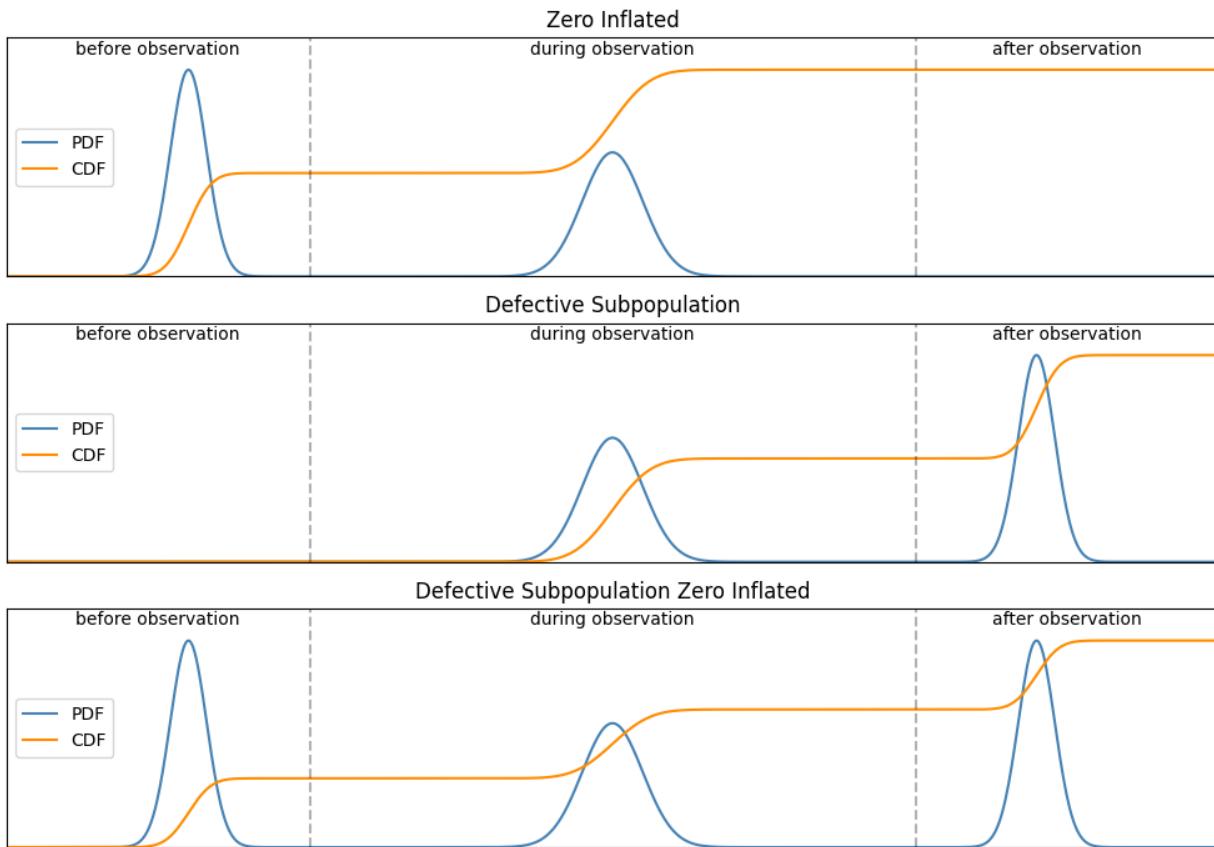
11.1 What are DSZI models?

DSZI is an acronym for “Defective Subpopulation Zero Inflated”. It is a combination of the Defective Subpopulation (DS) model and the Zero Inflated (ZI) model.

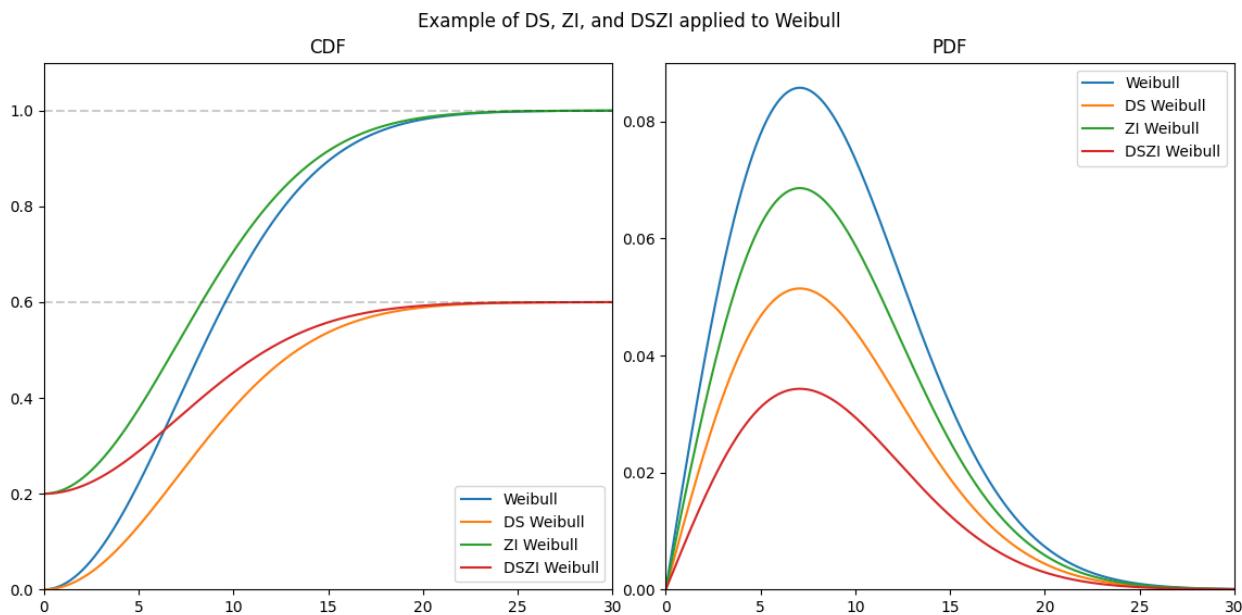
A defective subpopulation model is where the CDF does not reach 1 during the period of observation. This is caused when a portion of the population fails (known as the defective subpopulation) but the remainder of the population does not fail (and is right censored) by the end of the observation period.

A zero inflated model is where the CDF starts above 0 at the start of the observation period. This is caused by many “dead-on-arrival” items from the population, represented by failure times of 0. This is not the same as left censored data since left censored is when the failures occurred between 0 and the observation time. In the zero inflated model, the observation time is considered to start at 0 so the failure times are 0.

In a DSZI model, the CDF (which normally goes from 0 to 1) goes from above 0 to below 1, as shown in the image below. In this image the scale of the PDF and CDF are normalized so they can both be viewed together. In reality the CDF is much larger than the PDF.



A DSZI model may be applied to any distribution (Weibull, Normal, Lognormal, etc.) using the transformations explained in the next section. The plot below shows how a Weibull distribution can become a DS_Weibull, ZI_Weibull and DSZI_Weibull. Note that the PDF of the DS, ZI, and DSZI models appears smaller than that of the original Weibull model since the area under the PDF is no longer 1. This is because the CDF does not range from 0 to 1.



11.2 Equations of DSZI models

A DSZI Model adds a minor modification to the PDF and CDF of any standard distribution (referred to here as the “base distribution”) to transform it into a DSZI Model. The transformations are as follows:

$$PDF_{DSZI} = PDF_{base}(DS - ZI)$$

$$CDF_{DSZI} = CDF_{base}(DS - ZI) + ZI$$

In the above equations the base distribution (represented by PDF_{base} and CDF_{base}) is transformed using the parameters DS and ZI. DS is the maximum of the CDF which represents the fraction of the total population that is defective (the defective subpopulation). ZI is the minimum of the CDF which represents the fraction of the total population that failed at t=0 or equivalently were “dead-on-arrival” (the zero inflated fraction). To create only a DS model we can set ZI as 0. To create only a ZI model we can set DS as 1. The parameters DS and ZI must be between 0 and 1, and DS must be greater than ZI. The above equations can be expanded depending on the equation of the base distribution. For example, if the base distribution is a two parameter Weibull distribution, the DSZI model would be:

$$PDF: \quad f(t) = \frac{\beta}{\alpha} \left(\frac{t}{\alpha}\right)^{(\beta-1)} e^{-(\frac{t}{\alpha})^\beta} (DS - ZI)$$

$$CDF: \quad F(t) = \left(1 - e^{-(\frac{t}{\alpha})^\beta}\right) (DS - ZI) + ZI$$

The SF, HF and CHF can be obtained using transformations from the CDF and PDF using the [relationships between the five functions](#).

11.3 Creating a DSZI model

Within reliability, the DSZI Model is available within the Distributions module. The input requires the base distribution to be specified using a distribution object and the DS and ZI parameters to be specified if required. DS defaults to 1 and ZI defaults to 0. The output API matches the API for the standard distributions.

API Reference

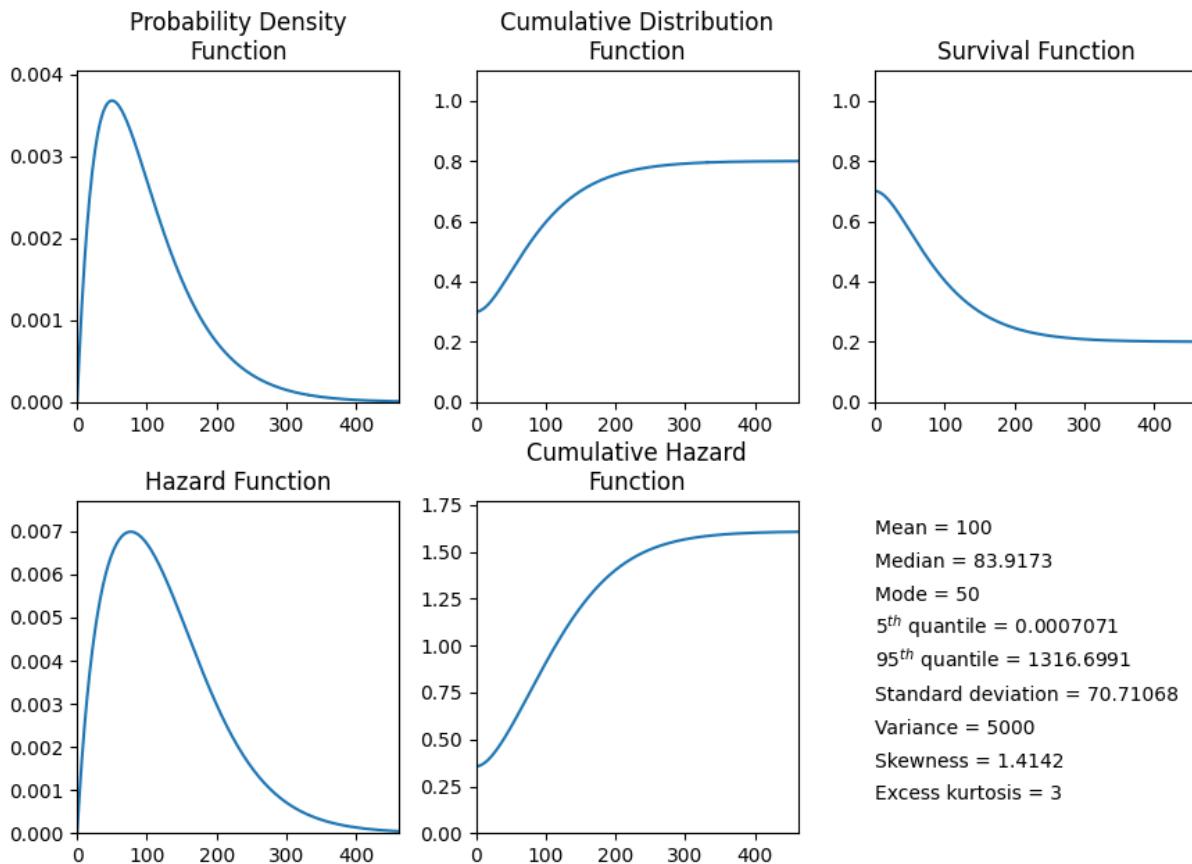
For inputs and outputs see the [API reference](#).

11.3.1 Example 1

In this first example, we will create a Gamma DSZI model and plot the 5 functions.

```
from reliability.Distributions import Gamma_Distribution, DSZI_Model
model = DSZI_Model(distribution = Gamma_Distribution(alpha=50,beta=2), DS= 0.8, ZI=0.3)
model.plot()
```

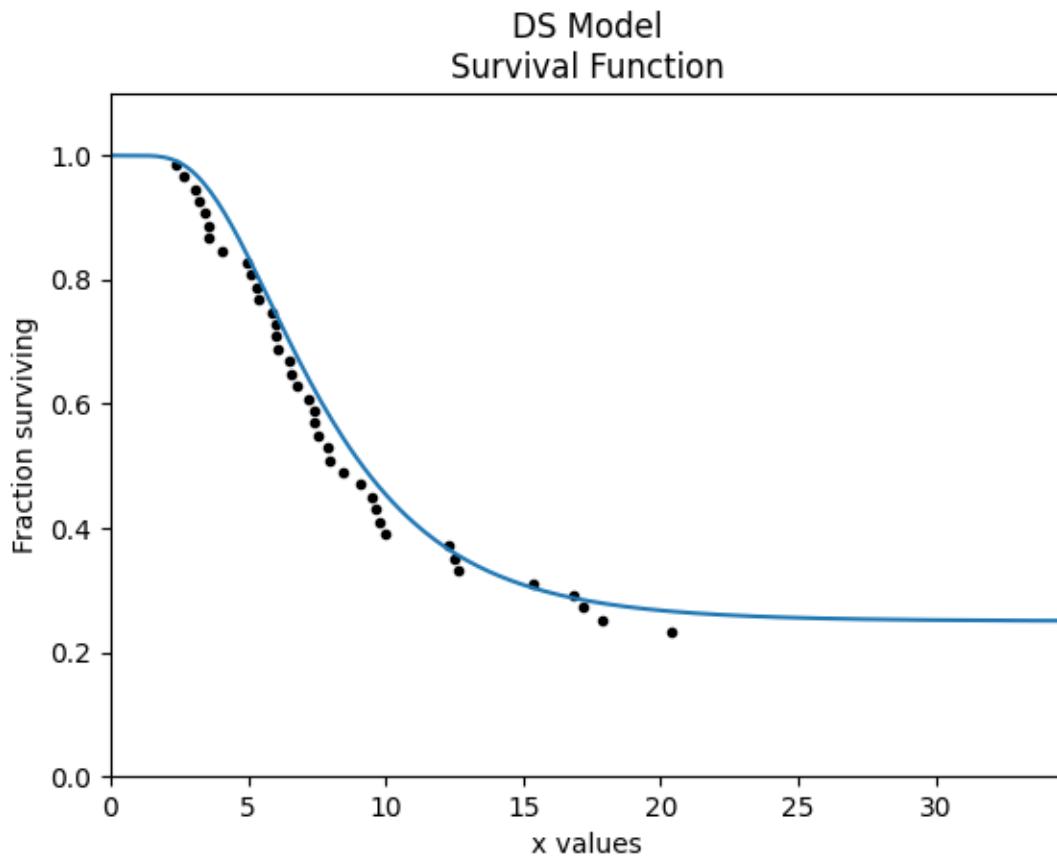
DSZI Model



11.3.2 Example 2

In this second example, we will create a Lognormal_DS model, draw some random samples and plot those samples on the survival function plot.

```
from reliability.Distributions import Lognormal_Distribution, DSZI_Model
from reliability.Probability_plotting import plot_points
import matplotlib.pyplot as plt
model = DSZI_Model(distribution = Lognormal_Distribution(mu=2,sigma=0.5), DS= 0.75)
failures, right_censored = model.random_samples(50,seed=7, right_censored_time = 50)
model.SF()
plot_points(failures = failures, right_censored = right_censored, func="SF")
plt.show()
```



Note that in the above example, the `random_samples` function returns failures and `right_censored` values. This differs from all other Distributions which only return failures. The reason for returning failures and `right_censored` data is that is is essential to have `right_censored` data in order to have a DS Model.

11.4 Fitting a DSZI model

API Reference

For inputs and outputs see the API reference for [Fit_Weibull_DS](#), [Fit_Weibull_ZI](#), and [Fit_Weibull_DSZI](#).

As we saw above, the DSZI_Model can be either DS, ZI, or DSZI depending on the values of the DS and ZI parameters. Within the Fitters module, three functions are offered, one of each of these cases with the Weibull_2P distribution as the base distribution. The three Fitters available are `Fit_Weibull_DS`, `Fit_Weibull_ZI`, and `Fit_Weibull_DSZI`. If your data contains zeros then only the `Fit_Weibull_ZI` and `Fit_Weibull_DSZI` fitters are appropriate. Using anything else will cause the zeros to be automatically removed and a warning to be printed. `Fit_Weibull_ZI` does not mandate that the failures contain zeros, but if failures does not contain zeros then ZI will be 0 and the alpha and beta parameters will be equivalent to the results from `Fit_Weibull_2P`. `Fit_Weibull_DS` does not mandate that `right_censored` data is provided, but if `right_censored` data is not provided then DS will be 1 and the alpha and beta parameters will be equivalent to the results from `Fit_Weibull_2P`. `Fit_Weibull_DSZI` does not mandate that failures contain zeros or that `right_censored` data is provided. If `right_censored` data is not provided then DS will be 1. If failures does not contain zeros then ZI will be 0. If failures does not contain zeros and no `right_censored` data is provided then DS will be 1, ZI will be 0 and the alpha and beta parameters will be equivalent to the results from `Fit_Weibull_2P`.

11.4.1 Example 3

In this example, we will create 70 samples of failure data from a Weibull Distribution, and append 30 zeros to it. We will then use Fit_Weibull_ZI to model the data.

```
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_ZI
from reliability.Probability_plotting import plot_points
import numpy as np
import matplotlib.pyplot as plt

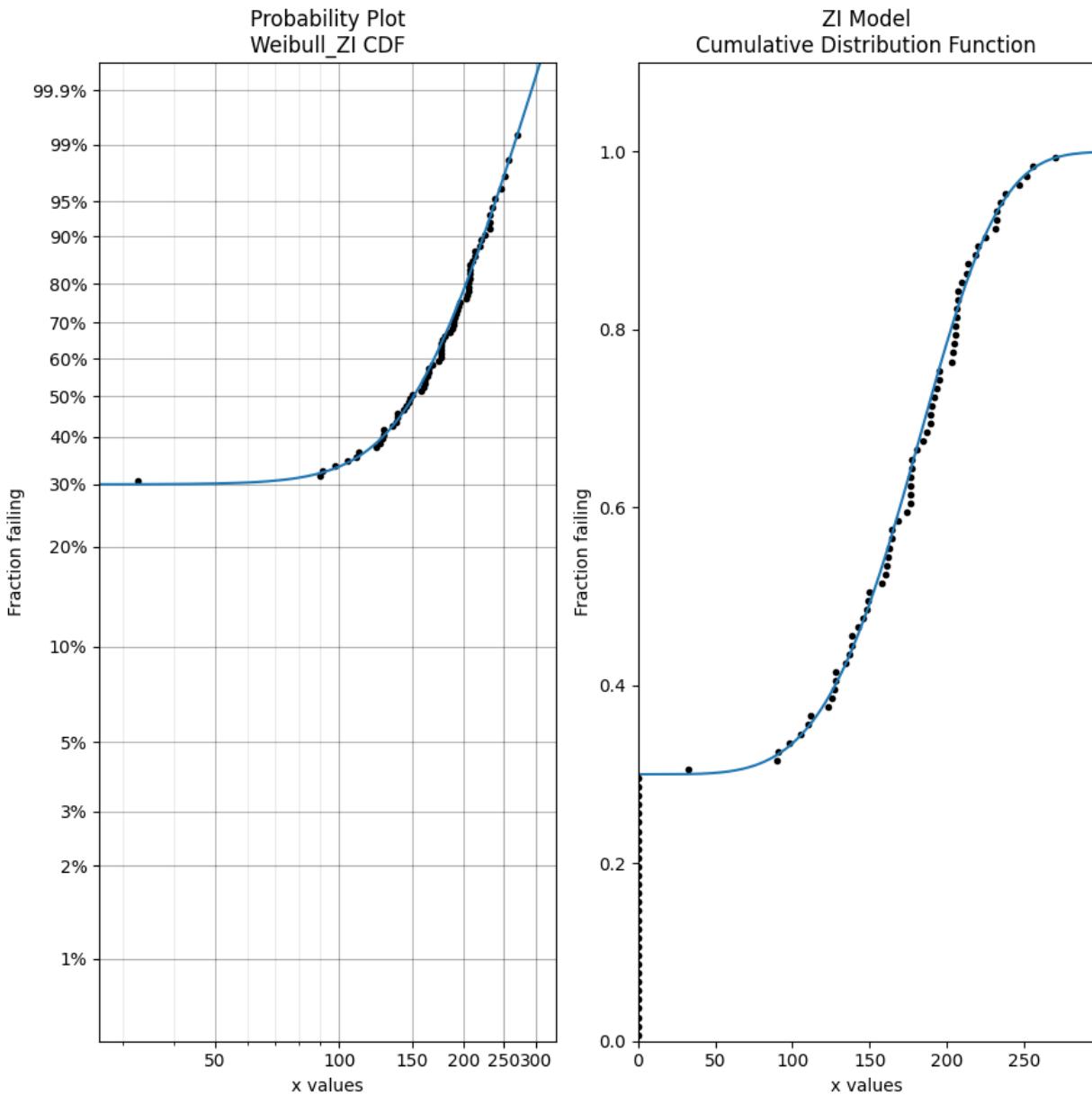
data = Weibull_Distribution(alpha=200, beta=5).random_samples(70, seed=1)
zeros = np.zeros(30)
failures = np.hstack([zeros, data])
plt.subplot(121)
fit = Fit_Weibull_ZI(failures=failures)
plt.subplot(122)
fit.distribution.CDF()
plot_points(failures=failures)
plt.tight_layout()
plt.show()

"""

Results from Fit_Weibull_ZI (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 100/0 (0% right censored)

Parameter  Point Estimate  Standard Error  Lower CI  Upper CI
Alpha        192.931      5.33803     182.747    203.682
Beta         4.53177      0.431272    3.76064    5.46102
ZI            0.3          0.0458258   0.218403   0.396613

Goodness of fit      Value
Log-likelihood -426.504
AICc    859.259
BIC     866.824
AD      5.88831
""
```



We can see above how the fitter correctly identified that the distribution was 30% zero inflated, and it did a reasonable job of finding the alpha and beta parameters of the base distribution.

11.4.2 Example 4

In this example, we will use `Fit_Weibull_DS` to model some data that is heavily right censored. The `DS=0.4` parameter means that only 40% of the data is failure data, with the rest being right censored. The original distribution is overlayed in the plot for comparison of the goodness of fit.

```
from reliability.Distributions import DSZI_Model, Weibull_Distribution
from reliability.Fitters import Fit_Weibull_DS
import matplotlib.pyplot as plt
from reliability.Probability_plotting import plot_points
```

(continues on next page)

(continued from previous page)

```

model = DSZI_Model(distribution=Weibull_Distribution(alpha=70, beta=2.5), DS=0.4)
failures, right_censored = model.random_samples(100, right_censored_time=120, seed=3)
model.CDF(label="true model", xmax=300)
fit_DS = Fit_Weibull_DS(failures=failures, right_censored=right_censored, show_
    probability_plot=False)
fit_DS.distribution.CDF(label="fitted Weibull_DS", xmax=300)
plot_points(failures=failures, right_censored=right_censored)
plt.legend()
plt.show()

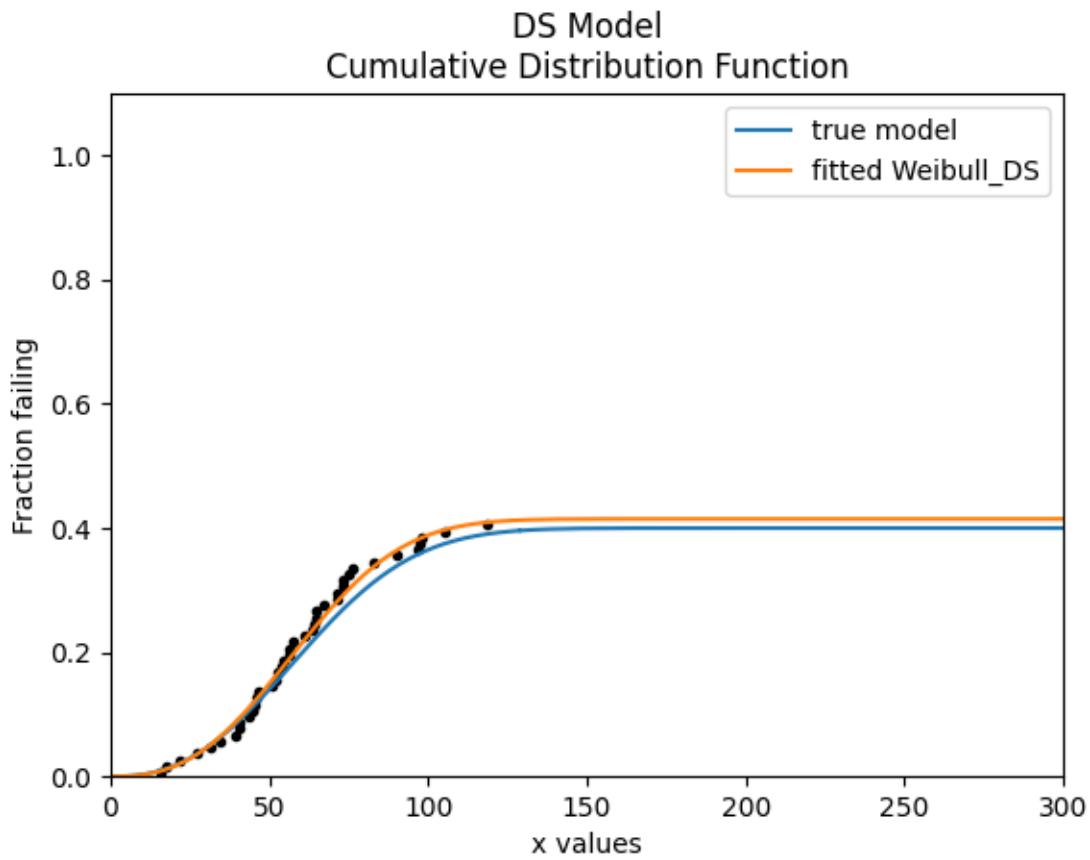
"""

Results from Fit_Weibull_DS (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 41/59 (59% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
Alpha      67.9275      4.61424  59.4599  77.6009
Beta       2.63207     0.357826  2.0164  3.43571
DS         0.414739    0.0500682  0.321106  0.514964

Goodness of fit      Value
Log-likelihood -254.236
AICc      514.721
BIC       522.287
AD        374.746
"""

```



11.4.3 Example 5

In this example, we will use some real world data from a vehicle manufacturer, which is available in the Datasets module. This example shows how the Weibull_2P model can be an inappropriate choice for a dataset that is heavily right censored. In addition to the visual proof provided by the probability plot (left) and the CDF (right), we can see the goodness of fit criterion indicate that Weibull_DS was much better (closer to zero) than Weibull_2P.

```
from reliability.Fitters import Fit_Weibull_DS, Fit_Weibull_2P
import matplotlib.pyplot as plt
from reliability.Probability_plotting import plot_points
from reliability.Datasets import defective_sample

failures = defective_sample().failures
right_censored = defective_sample().right_censored

plt.subplot(121)
fit_DS = Fit_Weibull_DS(failures=failures, right_censored=right_censored)
print('-----')
fit_2P = Fit_Weibull_2P(failures=failures, right_censored=right_censored)

plt.subplot(122)
fit_DS.distribution.CDF(label="fitted Weibull_DS", xmax=1000)
fit_2P.distribution.CDF(label="fitted Weibull_2P", xmax=1000)
```

(continues on next page)

(continued from previous page)

```

plot_points(failures=failures, right_censored=right_censored)
plt.ylim(0,0.25)
plt.legend()
plt.title('Cumulative Distribution Function')
plt.suptitle('Comparison of Weibull_2P with Weibull_DS')
plt.gcf().set_size_inches(12,6)
plt.tight_layout()
plt.show()

"""

Results from Fit_Weibull_DS (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 1350/12295 (90.10627% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
Alpha      170.983      4.61716   162.169   180.276
Beta       1.30109      0.0297713  1.24403   1.36077
DS         0.12482      0.00333709  0.118425  0.131509

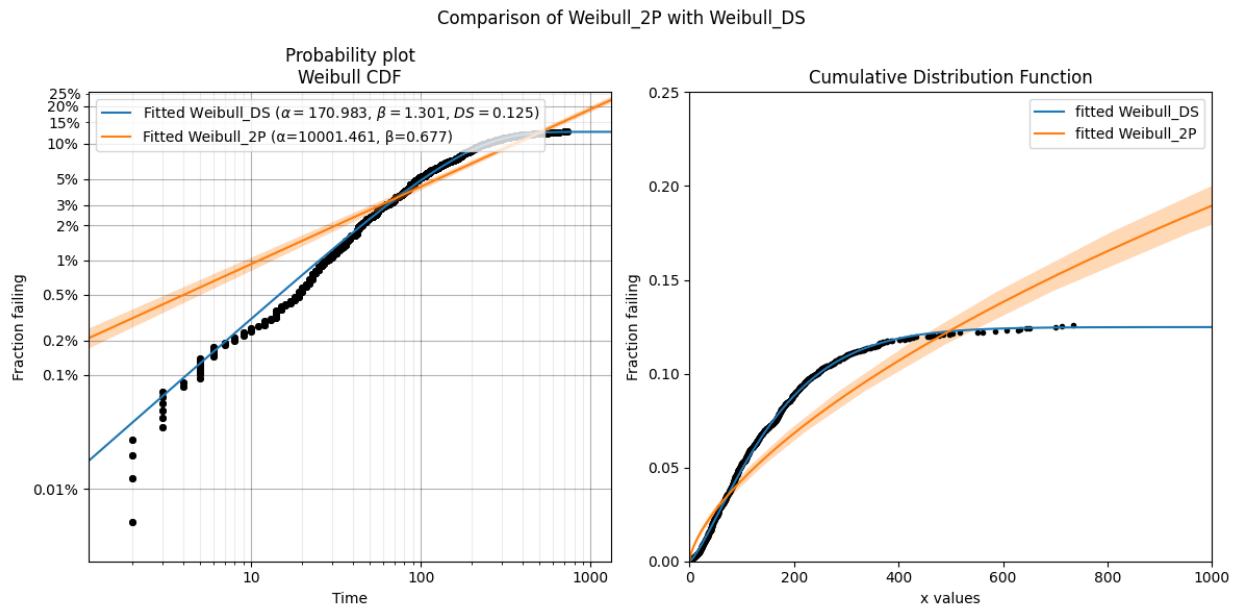
Goodness of fit      Value
Log-likelihood -11977.7
AICc      23961.3
BIC       23983.9
AD        27212.4

-----
Results from Fit_Weibull_2P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 1350/12295 (90.10627% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
Alpha      10001.5      883.952   8410.7   11893.1
Beta       0.677348     0.0166663  0.645463  0.710807

Goodness of fit      Value
Log-likelihood -12273.2
AICc      24550.3
BIC       24565.4
AD        27213
"""

```



11.4.4 Example 6

In this example we will create a DSZI model with $DS=0.7$ and $ZI=0.2$. Based on these parameters, we expect the random samples to be around 70% failures and of those failures 20% of the total samples (failures + right censored) should be zeros due to the zero inflated fraction. We draw the random samples from the model and then fit a Weibull_DSZI model to the data. The result is surprisingly accurate showing $DS=0.700005$ and $ZI=0.22$, with the alpha and beta parameters closely resembling the parameters of the input Weibull Distribution. The plot below shows the CDF on the Weibull probability plot (left) and on linear axes (right) which each provide a different perspective of how the distribution models the failure points.

```
from reliability.Distributions import DSZI_Model, Weibull_Distribution
from reliability.Probability_plotting import plot_points
import matplotlib.pyplot as plt
from reliability.Fitters import Fit_Weibull_DSZI

model = DSZI_Model(distribution=Weibull_Distribution(alpha=1200,beta=3),DS=0.7,ZI=0.2)
failures, right_censored = model.random_samples(100,seed=5,right_censored_time=3000)

plt.subplot(121)
fit = Fit_Weibull_DSZI(failures=failures,right_censored=right_censored,label='fitted_Weibull_DSZI')
model.CDF(label='true model')
plt.legend()

plt.subplot(122)
fit.distribution.CDF(label='fitted Weibull_DSZI')
model.CDF(label='true model')
plot_points(failures=failures,right_censored=right_censored)
plt.legend()
plt.tight_layout()
plt.show()

""
```

(continues on next page)

(continued from previous page)

Results from Fit_Weibull_DSZI (95% CI):

Analysis method: Maximum Likelihood Estimation (MLE)

Optimizer: TNC

Failures / Right censored: 70/30 (30% right censored)

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha	1170.12	68.0933	1043.99	1311.49
Beta	2.60255	0.299069	2.07771	3.25997
DS	0.700005	0.045826	0.603391	0.781602
ZI	0.22	0.0414247	0.149465	0.311627

Goodness of fit *Value*

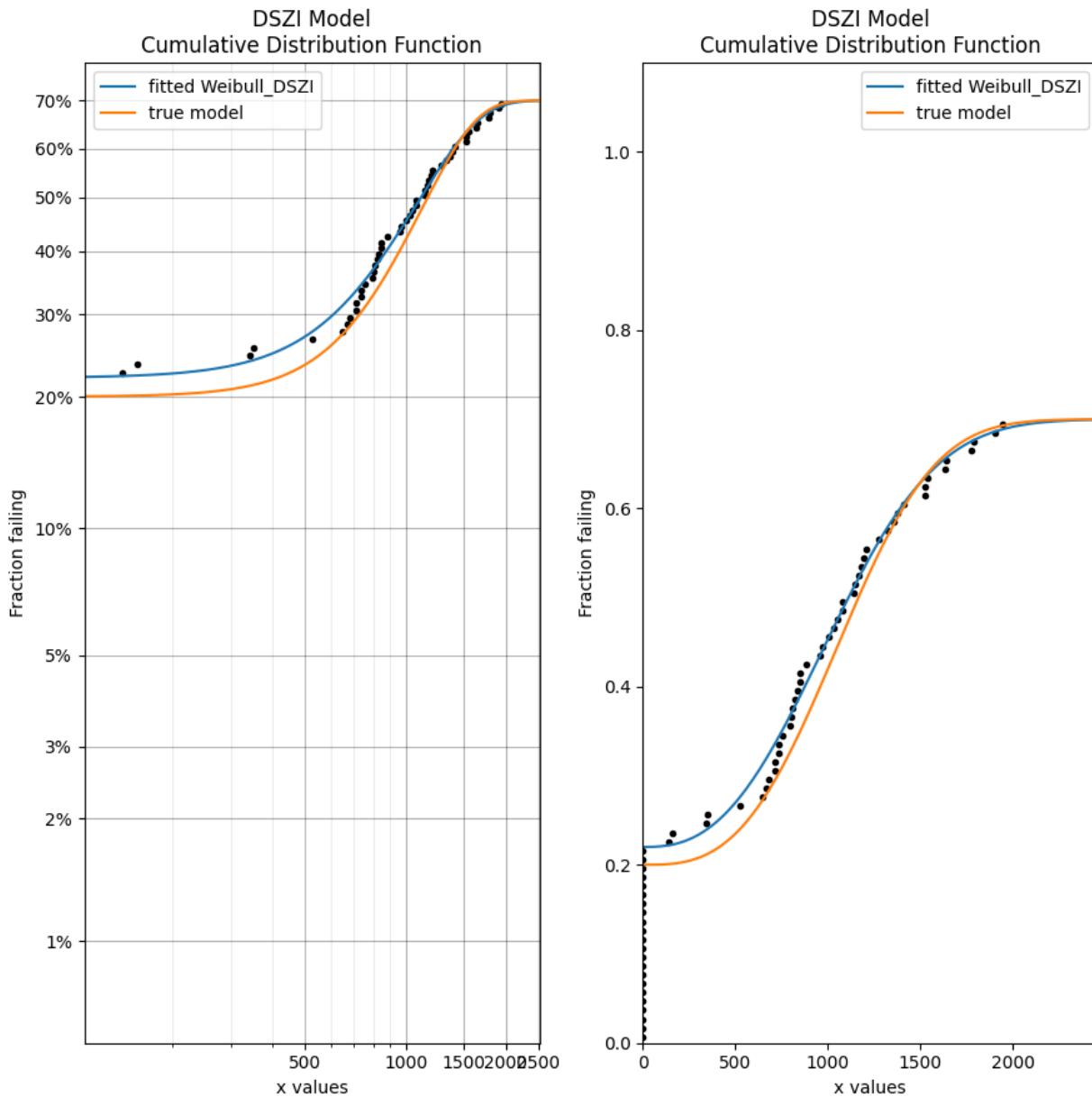
Log-likelihood -463.613

AICc 935.647

BIC 945.646

AD 166.025

""



The DSZI model is a model of my own making. It combines the well established DS and ZI models together for the first time to enable heavily right censored data to be modelled using a DS distribution while also allowing for zero inflation of the failures.



RELIABILITY
A Python library for reliability engineering

OPTIMIZERS

12.1 What is an optimizer?

An optimizer is an algorithm that uses two primary inputs; a target function and an initial guess. The optimizer's job is to figure out which input to the target function will minimise the output of the target function.

Within *reliability*, the *Fitters* and *ALT_Fitters* modules rely heavily on optimizers to find the parameters of the distribution that will minimize the log-likelihood function for the given data set. This process is fundamental to the Maximum Likelihood Estimation (MLE) method of fitting a probability distribution.

There are four optimizers supported by *reliability*. These are “TNC”, “L-BFGS-B”, “nelder-mead”, and “powell”. All of these optimizers are bound constrained, meaning that the functions within *reliability* will specify the bounds of the parameters (such as making the parameters greater than zero) and the optimizer will find the optimal solution that is within these bounds. These four optimizers are provided by [scipy](#).

The optimizer can be specified as a string using the “optimizer” argument. For example:

```
from reliability.Fitters import Fit_Weibull_2P
Fit_Weibull_2P(failures=[1,7,12], optimizer='TNC')
```

The optimizer that was used is always reported by each of the functions in *Fitters* and *ALT_Fitters*. An example of this is shown below on the third line of the output. In the case of *Fit_Everything* and *Fit_Everything_ALT*, the optimizer used for each distribution or model is provided in the table of results.

```
...
Results from Fit_Weibull_2P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 3/0 (0% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
Alpha      7.16845      3.33674    2.8788    17.85
Beta       1.29924      0.650074   0.487295  3.46408

Goodness of fit          Value
Log-likelihood           -8.56608
AICc Insufficient data
BIC                      19.3294
AD                       3.72489
```

12.2 Why do we need different optimizers?

Each optimizer has various strengths and weaknesses because they work in different ways. Often they will arrive at the same result. Sometimes they will arrive at different results, either because of the very shallow gradient near the minimum, or the non-global minimum they have found. Sometimes they will fail entirely.

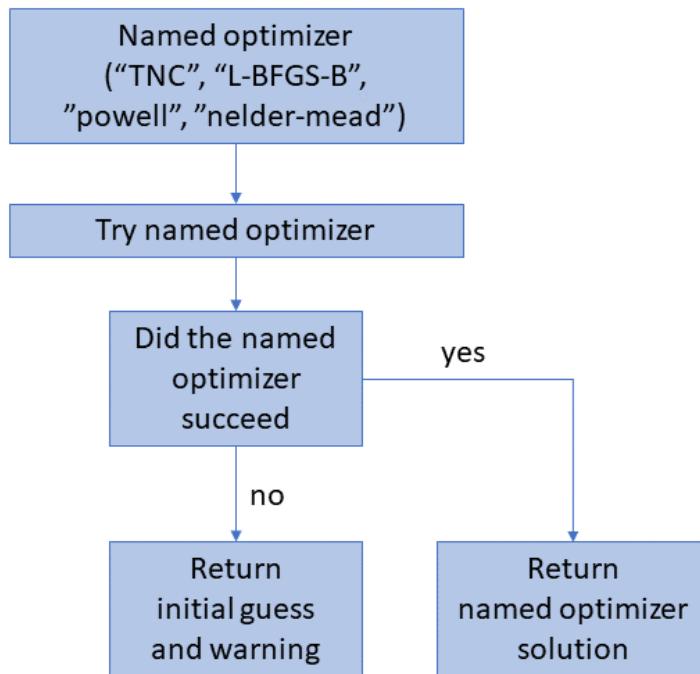
There is no single best optimizer for fitting probability distributions so a few options are provided as described below.

12.3 Which optimizer should I pick?

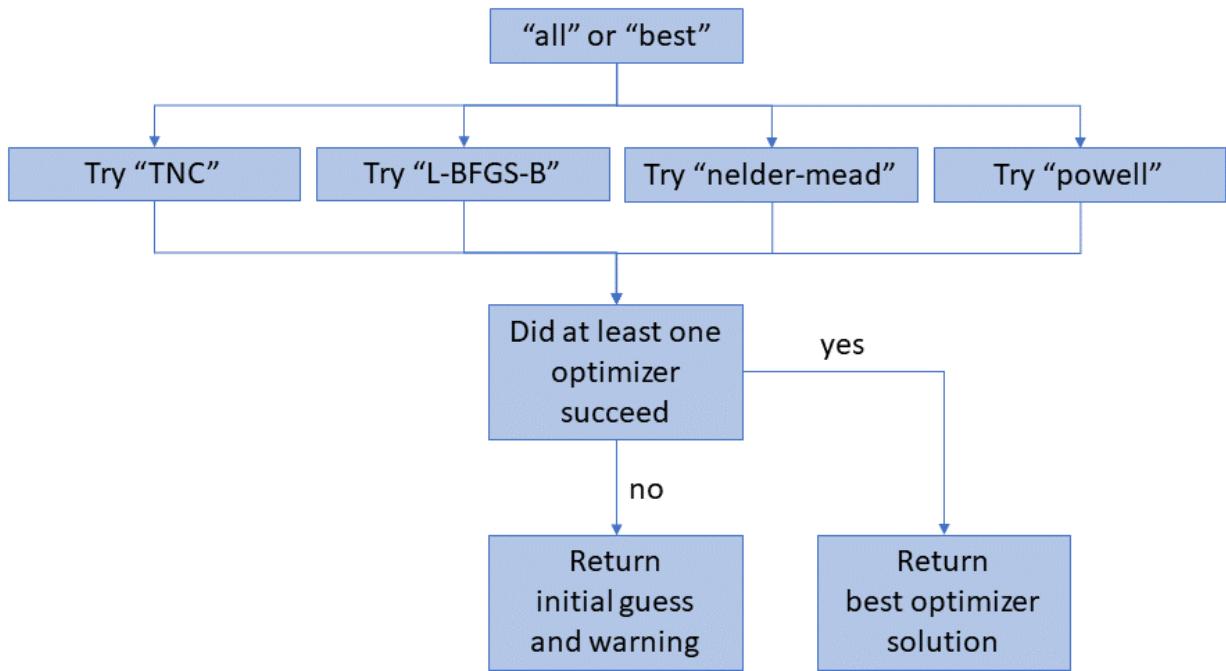
You don't really need to worry about picking an optimizer as the default choice is usually sufficient. If you do want to select your optimizer, you have four to choose from. Most importantly, you should be aware of what the optimizer is doing (minimizing the negative log-likelihood equation by varying the parameters) and understand that optimizers aren't all the same which can cause different results. If you really need to know the best optimizer then select "best", otherwise you can just leave the default as None.

There are three behaviours within reliability with respect to the choice of optimizer. These depend on whether the user has specified a specific optimizer ("TNC", "L-BFGS-B", "nelder-mead", "powell"), specified all optimizers ("all" or "best"), or not specified anything (None).

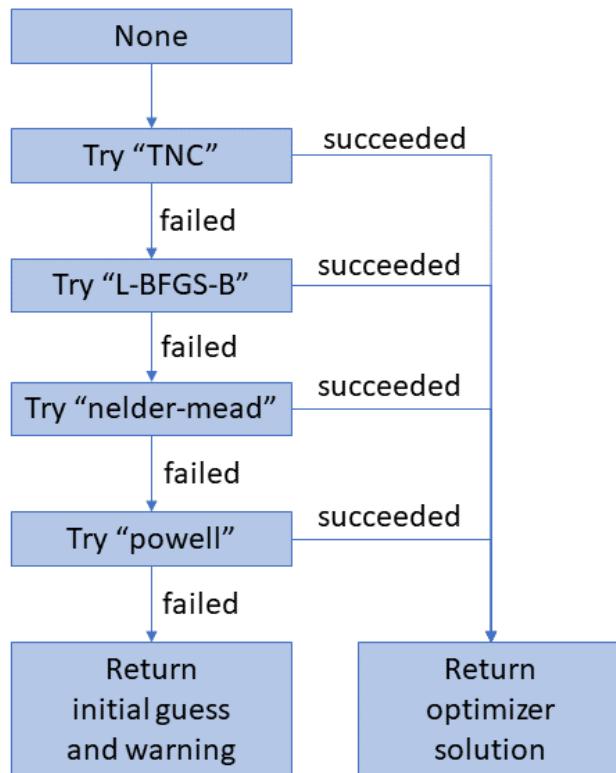
In the case of a specific optimizer being specified, it will be used. If it fails, then the initial guess will be returned with a warning.



In the case of "best" or "all" being specified, all four of the optimizers will be tried. The results from the best one (based on the lowest log-likelihood it finds) will be returned.



In the case of no optimizer being specified, they will be tried in order of "TNC", "L-BFGS-B", "nelder-mead", "powell". Once one of them succeeds, the results will be returned and no further optimizers will be run.

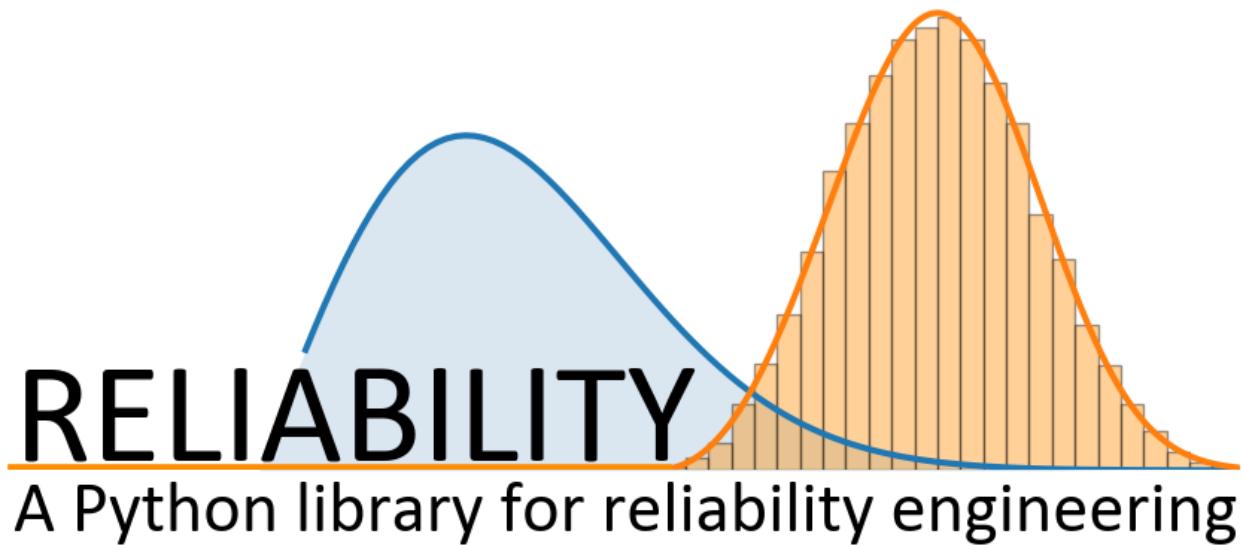


Note

For large sample sizes (above 10000) it will take considerable time to run multiple optimizers. In particular, “nelder-mead” and “powell” are much slower than “TNC” and “L-BFGS-B”. For this reason, *reliability* does not try multiple optimizers unless told to or if the default did not succeed.

Note

There are some rare occasions when the optimizer finds a result (and reports it succeeded) but another optimizer may find a better result. If you always want to be sure that the best result has been found, specify “best” or “all” for the optimizer, and be prepared to wait longer for it to compute if you have a large amount of data. The typical difference between the results of different optimizers which succeeded is very small (around 1e-8 for the log-likelihood) but this is not always the case as the number of parameters in the model increase.



CHAPTER
THIRTEEN

PROBABILITY PLOTS

Probability plots are a general term for several different plotting techniques. One of these techniques is a graphical method for comparing two data sets and includes [probability-probability](#) (PP) plots and [quantile-quantile](#) (QQ) plots. The second plotting technique is used for assessing the goodness of fit of a distribution by plotting the empirical CDF of the failures against their failure time and scaling the axes in such a way that the distribution appears linear. This method allows the reliability analyst to fit the distribution parameters using a simple “least squares” fitting method for a straight line and was popular before computers were capable of calculating the MLE estimates of the parameters. While we do not typically favour the use of least squares as a fitting method, we can still use probability plots to assess the goodness of fit. The module [reliability.Probability_plotting](#) contains functions for each of the standard distributions supported in [reliability](#). These functions are:

- `Weibull_probability_plot`
- `Normal_probability_plot`
- `Lognormal_probability_plot`
- `Gamma_probability_plot`
- `Beta_probability_plot`
- `Exponential_probability_plot`
- `Exponential_probability_plot_Weibull_Scale`
- `Loglogistic_probability_plot`
- `Gumbel_probability_plot`

There is also a function to obtain the plotting positions called `plotting_positions`. This function is mainly used by other functions and is not discussed further here. For more detail, consult the help file of the function. To obtain a scatter plot of the plotting positions in the form of the PDF, CDF, SF, HF, or CHF, you can use the function `plot_points`. This is explained [here](#).

Within each of the above probability plotting functions you may enter failure data as well as right censored data. For those distributions that have a function in [reliability.Fitters](#) for fitting location shifted distributions (Weibull_3P, Gamma_3P, Lognormal_3P, Exponential_2P, Loglogistic_3P), you can explicitly tell the probability plotting function to fit the gamma parameter using `fit_gamma=True`. By default the gamma parameter is not fitted. Fitting the gamma parameter will also change the x-axis to time-gamma such that everything will appear linear. An example of this is shown in the [second example](#) below.

 **Note**

Beta and Gamma probability plots have their y-axes scaled based on the distribution’s parameters so you will find that when you overlay two Gamma or two Beta distributions on the same Gamma or Beta probability paper, one will be a curved line if they have different shape parameters. This is unavoidable due to the nature of Gamma and

Beta probability paper and is the reason why you will never find a hardcopy of such paper and also the reason why these distributions are not used in ALT probability plotting.

API Reference

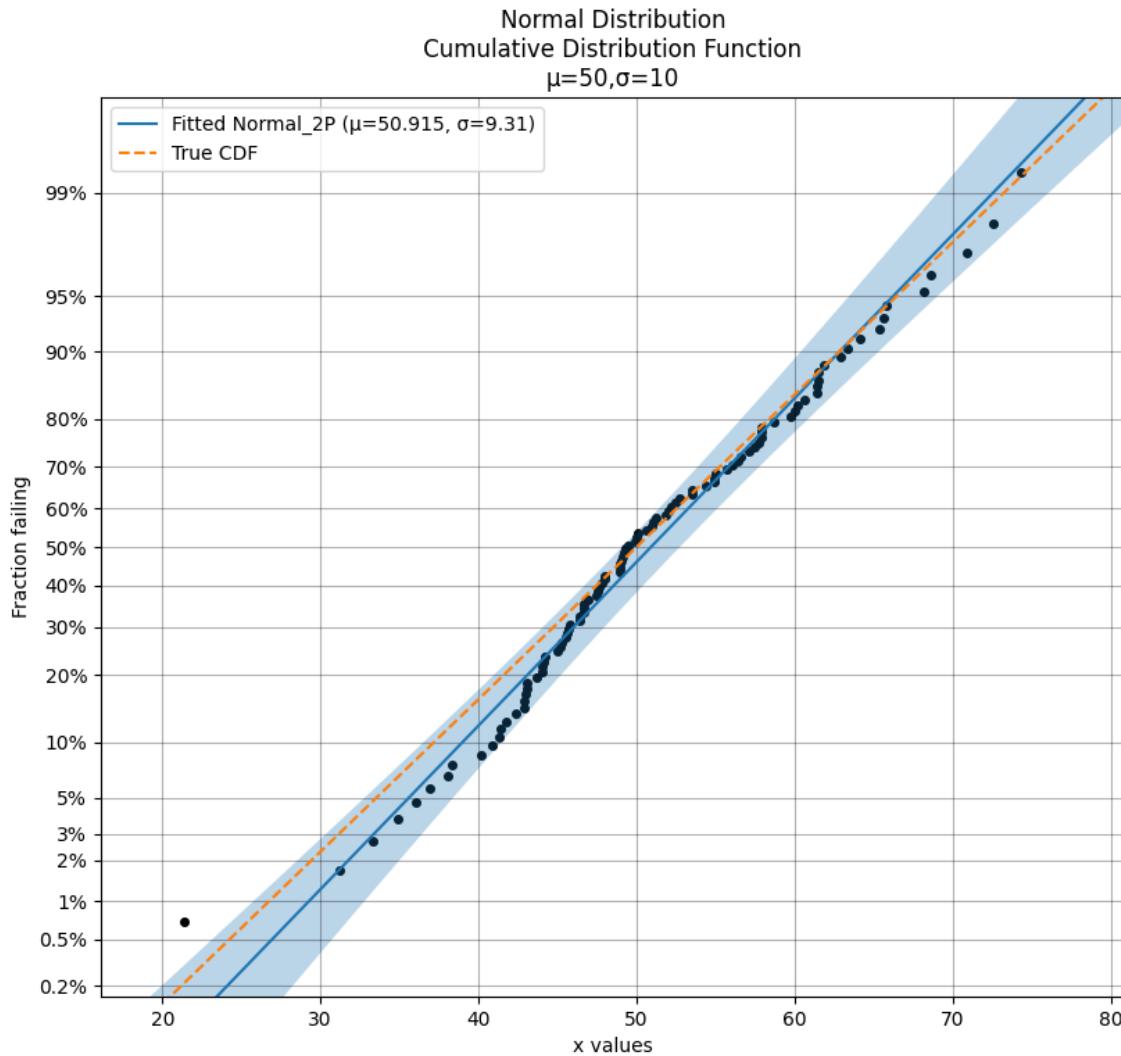
For inputs and outputs see the [API reference](#).

13.1 Example 1

In the example below we generate some samples from a Normal Distribution and provide these to the probability plotting function. It is also possible to overlay other plots of the CDF as is shown by the dashed line.

```
from reliability.Distributions import Normal_Distribution
from reliability.Probability_plotting import Normal_probability_plot
import matplotlib.pyplot as plt

dist = Normal_Distribution(mu=50, sigma=10)
failures = dist.random_samples(100, seed=5)
Normal_probability_plot(failures=failures) #generates the probability plot
dist.CDF(linestyle='--', label='True CDF') #this is the actual distribution provided for
#comparison
plt.legend()
plt.show()
```



13.2 Example 2

In this second example, we will fit an Exponential distribution to some right censored data. To create this data, we will generate the random samples from an Exponential distribution that has a location shift of 12. Once again, the true CDF has also been plotted to provide the comparison. Note that the x-axis is time-gamma as it is necessary to subtract gamma from the x-plotting positions if we want the plot to appear linear.

```
from reliability.Distributions import Exponential_Distribution
from reliability.Probability_plotting import Exponential_probability_plot
import matplotlib.pyplot as plt
from reliability.Other_functions import make_right_censored_data
```

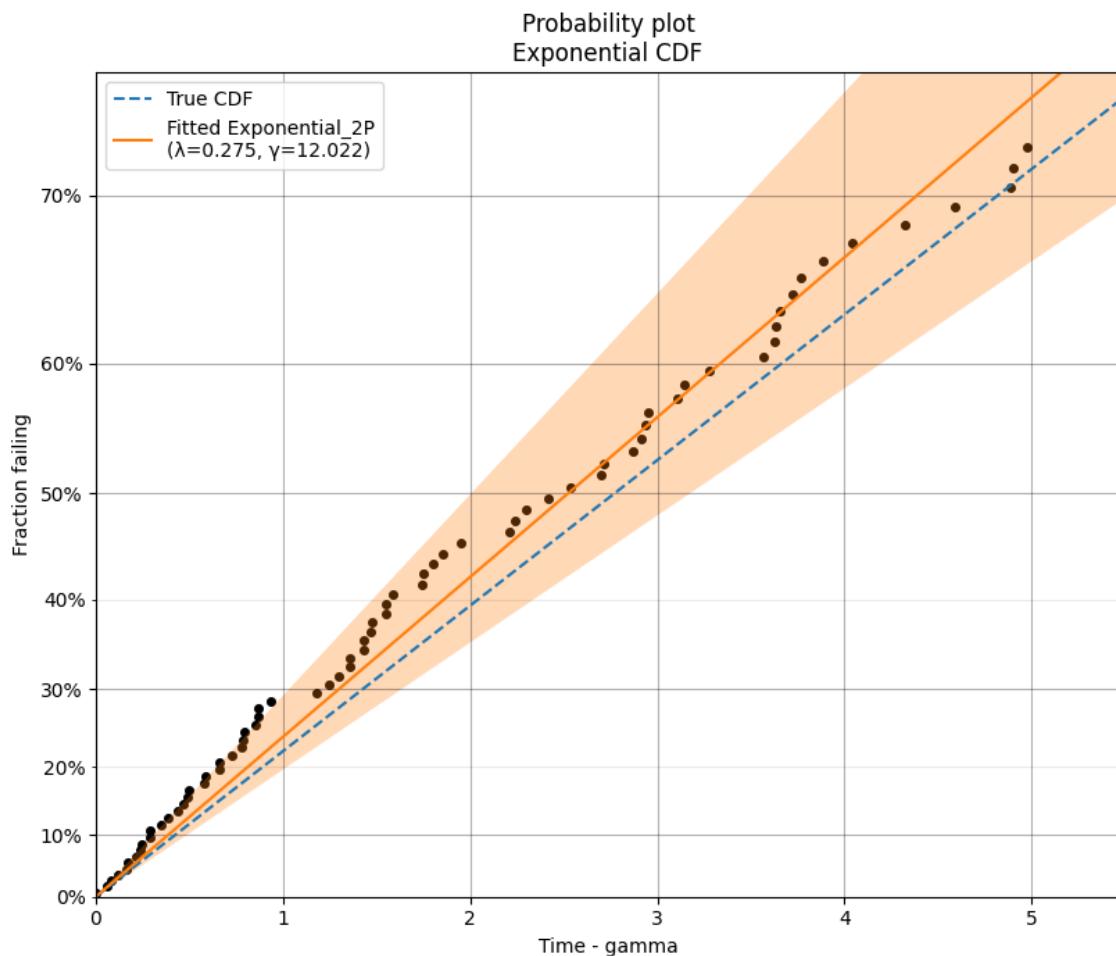
(continues on next page)

(continued from previous page)

```

dist = Exponential_Distribution(Lambda=0.25, gamma=12)
raw_data = dist.random_samples(100, seed=42) # draw some random data from an
# exponential distribution
data = make_right_censored_data(raw_data, threshold=17) # right censor the data at 17
Exponential_Distribution(Lambda=0.25).CDF(linestyle='--', label='True CDF') # we can't
# plot dist because it will be location shifted
Exponential_probability_plot(failures=data.failures, right_censored=data.right_censored,
# fit_gamma=True) # do the probability plot. Note that we have specified to fit gamma
plt.legend()
plt.show()

```



13.3 Example 3

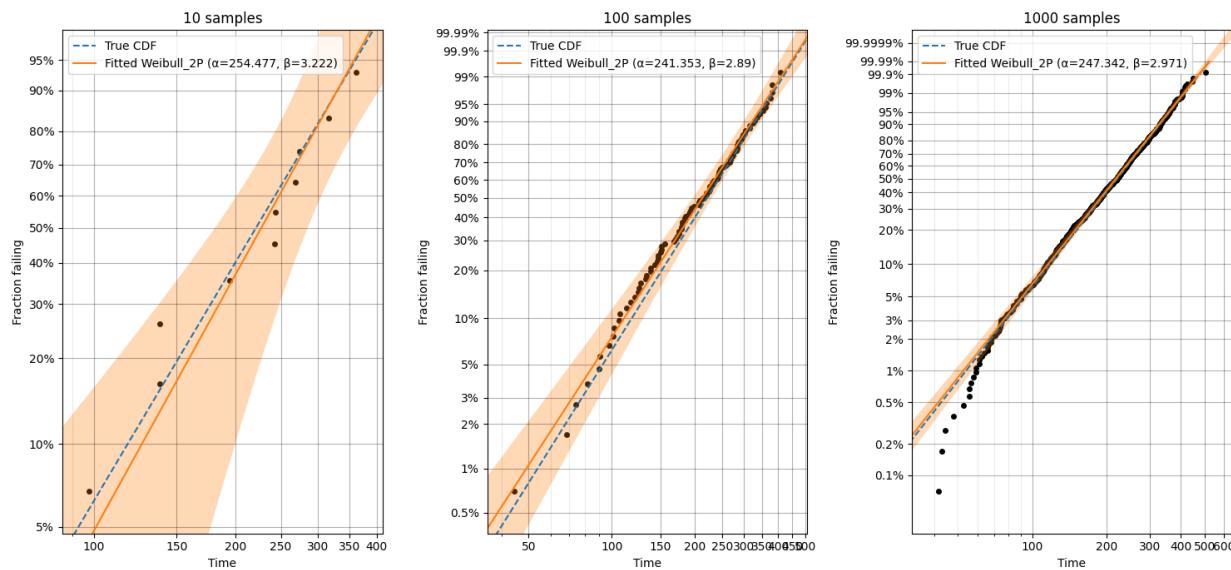
In this third example, we will see how probability plotting can be used to highlight the importance of getting as much data as possible. This code performs a loop in which increasing numbers of samples are used for fitting a Weibull distribution and the accuracy of the results (shown both in the legend and by comparison with the True CDF) increases with the number of samples. We can also see the width of the confidence intervals decreasing as the number of samples increases.

```

from reliability.Distributions import Weibull_Distribution
from reliability.Probability_plotting import Weibull_probability_plot
import matplotlib.pyplot as plt

dist = Weibull_Distribution(alpha=250, beta=3)
for i, x in enumerate([10,100,1000]):
    plt.subplot(131 + i)
    dist.CDF(linestyle='--', label='True CDF')
    failures = dist.random_samples(x, seed=42) # take 10, 100, 1000 samples
    Weibull_probability_plot(failures=failures) # this is the probability plot
    plt.title(str(str(x) + ' samples'))
plt.gcf().set_size_inches(15, 7) # adjust the figure size after creation. Necessary to
# do it after as it is automatically adjusted within probability_plot
plt.tight_layout()
plt.show()

```



13.4 Example 4

In this fourth example, we will take a look at the special case of the Exponential probability plot using the Weibull Scale. This plot is essentially a Weibull probability plot, but the fitting and plotting functions are Exponential. The reason for plotting an Exponential distribution on Weibull probability paper is to achieve parallel lines for different Lambda parameters rather than having the lines radiating from the origin as we see in the Exponential probability plot on Exponential probability paper. This has applications in ALT probability plotting and is the default plot provided from Fit_Exponential_1P and Fit_Exponential_2P. An example of the differences between the plots are shown below. Remember that the Alpha parameter from the Weibull distribution is equivalent to 1/Lambda from the Exponential distribution and a Weibull distribution with Beta = 1 is the same as an Exponential distribution.

```

from reliability.Distributions import Exponential_Distribution
from reliability.Probability_plotting import Exponential_probability_plot, Weibull_
#probability_plot, Exponential_probability_plot_Weibull_Scale
import matplotlib.pyplot as plt

data1 = Exponential_Distribution(Lambda=1 / 10).random_samples(50, seed=42) # should
#(continues on next page)

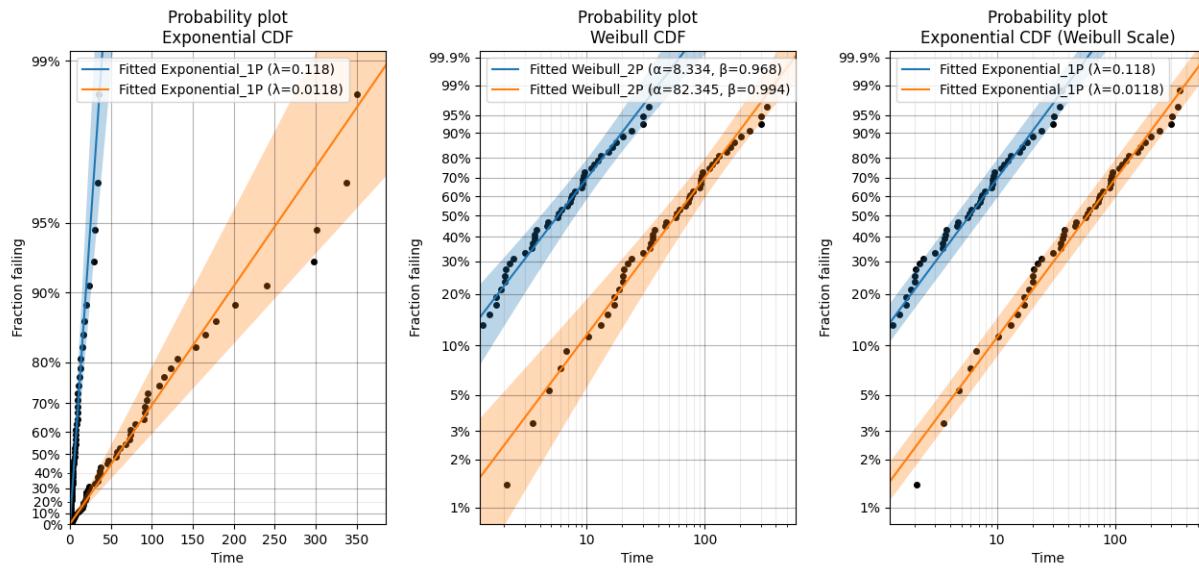
```

(continued from previous page)

```

→give Exponential Lambda = 0.01 OR Weibull alpha = 10
data2 = Exponential_Distribution(Lambda=1 / 100).random_samples(50, seed=42) # should_
→give Exponential Lambda = 0.001 OR Weibull alpha = 100
plt.subplot(131)
Exponential_probability_plot(failures=data1)
Exponential_probability_plot(failures=data2)
plt.subplot(132)
Weibull_probability_plot(failures=data1)
Weibull_probability_plot(failures=data2)
plt.subplot(133)
Exponential_probability_plot_Weibull_Scale(failures=data1)
Exponential_probability_plot_Weibull_Scale(failures=data2)
plt.gcf().set_size_inches(13, 6)
plt.subplots_adjust(left=0.06, right=0.97, top=0.91, wspace=0.30) # format the plot
plt.show()

```



13.5 Example 5

In this example we will look at how to create a probability plot that has different colors representing different groups which are being analysed together. Consider corrosion failure data from an oil pipeline where we know the location of the corrosion (either the Bend, Valve, or Joint of the pipe). To show the location of the corrosion in different colors we need to hide the default scatter plot from the probability plot and then replot the scatter plot using the function `plot_points`. The function `plot_points` passes keyword arguments (like `color`) directly to matplotlib's `plt.scatter()` whereas the `probability_plot` does some preprocessing of keyword arguments before passing them on. This means that it is only possible to provide a list of colors for the scatter plot to `plot_points`.

```

from reliability.Probability_plotting import Weibull_probability_plot, plot_points,_
→plotting_positions
import matplotlib.pyplot as plt
import numpy as np

# failure data from oil pipe corrosion

```

(continues on next page)

(continued from previous page)

```

bend = [74, 52, 32, 76, 46, 35, 65, 54, 56, 20, 71, 72, 38, 61, 29]
valve = [78, 83, 94, 76, 86, 39, 54, 82, 96, 66, 63, 57, 82, 70, 72, 61, 84, 73, 69, 97]
joint = [74, 52, 32, 76, 46, 35, 65, 54, 56, 25, 71, 72, 37, 61, 29]

# combine the data into a single array
data = np.hstack([bend, valve, joint])
color = np.hstack(['red'] * len(bend), ['green'] * len(valve), ['blue'] * len(joint))

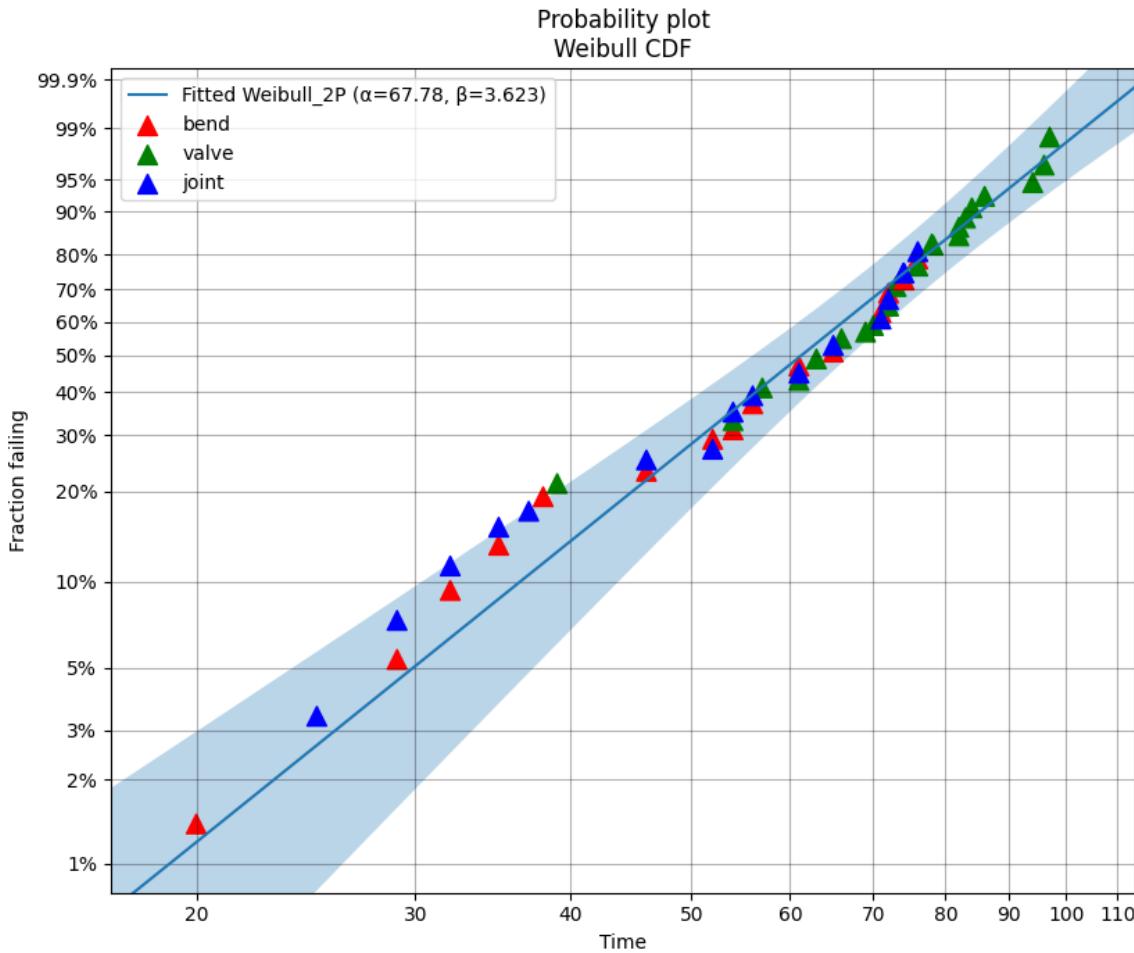
# create the probability plot and hide the scatter points
Weibull_probability_plot(failures=data, show_scatter_points=False)

# redraw the scatter points. kwargs are passed to plt.scatter so a list of color is accepted
plot_points(failures=data, color=color, marker='^', s=100)

# To show the legend correctly, we need to replot some points in separate scatter plots to create different legend entries
x, y = plotting_positions(failures=data)
plt.scatter(x[0], y[0], color=color[0], marker='^', s=100, label='bend')
plt.scatter(x[len(bend)], y[len(bend)], color=color[len(bend)], marker='^', s=100, label='valve')
plt.scatter(x[len(bend) + len(valve)], y[len(bend) + len(valve)], color=color[len(bend) + len(valve)], marker='^', s=100, label='joint')
plt.legend()

plt.show()

```



13.6 Example 6

In this final example, we take a look at how a probability plot can show us that there's something wrong with our assumption of a single distribution. To generate the data, the random samples are drawn from two different distributions which are shown in the left image. In the right image, the scatterplot of failure times is clearly non-linear. The green line is the attempt to fit a single Weibull_2P distribution and this will do a poor job of modelling the data. Also note that the points of the scatterplot do not fall on the True CDF of each distribution. This is because the median rank method of obtaining the plotting positions does not work well if the failure times come from more than one distribution. If you see a pattern like this, try a [mixture model](#) or a [competing risks model](#). Always remember that cusps, corners, and doglegs indicate a mixture of failure modes.

```
from reliability.Probability_plotting import Weibull_probability_plot
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt
import numpy as np

dist_1 = Weibull_Distribution(alpha=200, beta=3)
dist_2 = Weibull_Distribution(alpha=900, beta=4)
plt.subplot(121) # this is for the PDFs of the 2 individual distributions
dist_1.PDF(label=dist_1.param_title_long)
```

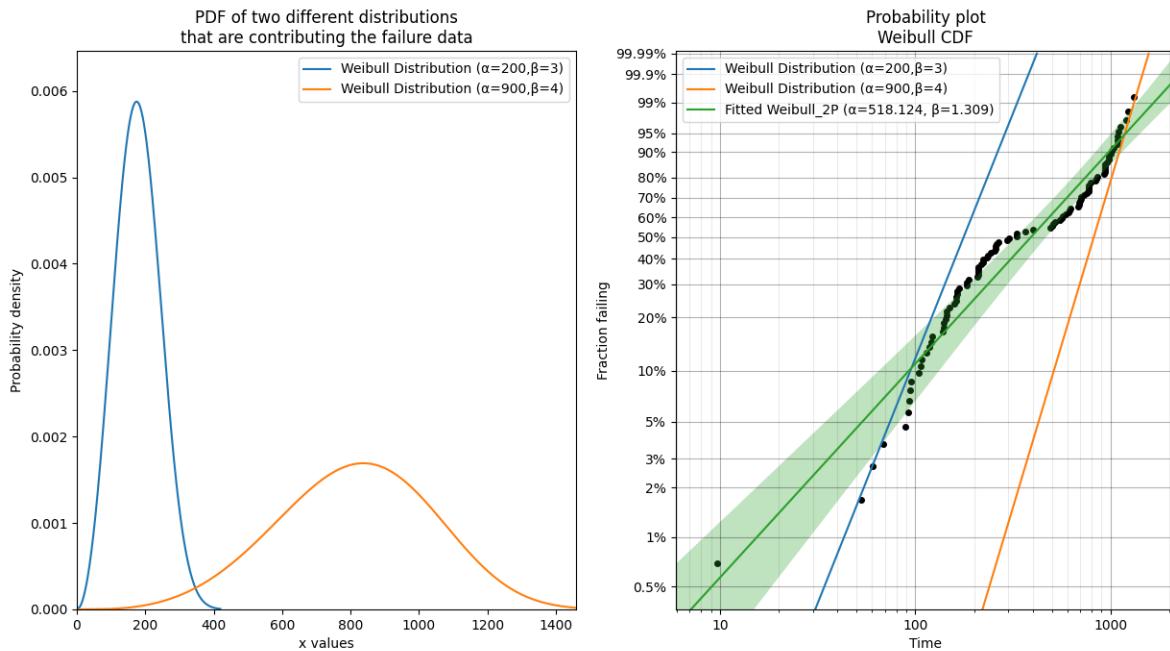
(continues on next page)

(continued from previous page)

```

dist_2.PDF(label=dist_2.param_title_long)
plt.legend()
plt.title('PDF of two different distributions\nthat are contributing the failure data')
plt.subplot(122) # this will be the probability plot
dist_1_data = dist_1.random_samples(50, seed=1)
dist_2_data = dist_2.random_samples(50, seed=1)
all_data = np.hstack([dist_1_data, dist_2_data]) # combine the failure data into one
# array
dist_1.CDF(label=dist_1.param_title_long) # plot each individual distribution for
# comparison
dist_2.CDF(label=dist_2.param_title_long)
Weibull_probability_plot(failures=all_data) # do the probability plot
plt.gcf().set_size_inches(13, 7) # adjust the figure size after creation. Necessary to
# do it after as it is automatically adjusted within probability_plot
plt.subplots_adjust(left=0.08, right=0.96) # formatting the layout
plt.legend()
plt.show()

```



13.7 What does a probability plot show me?

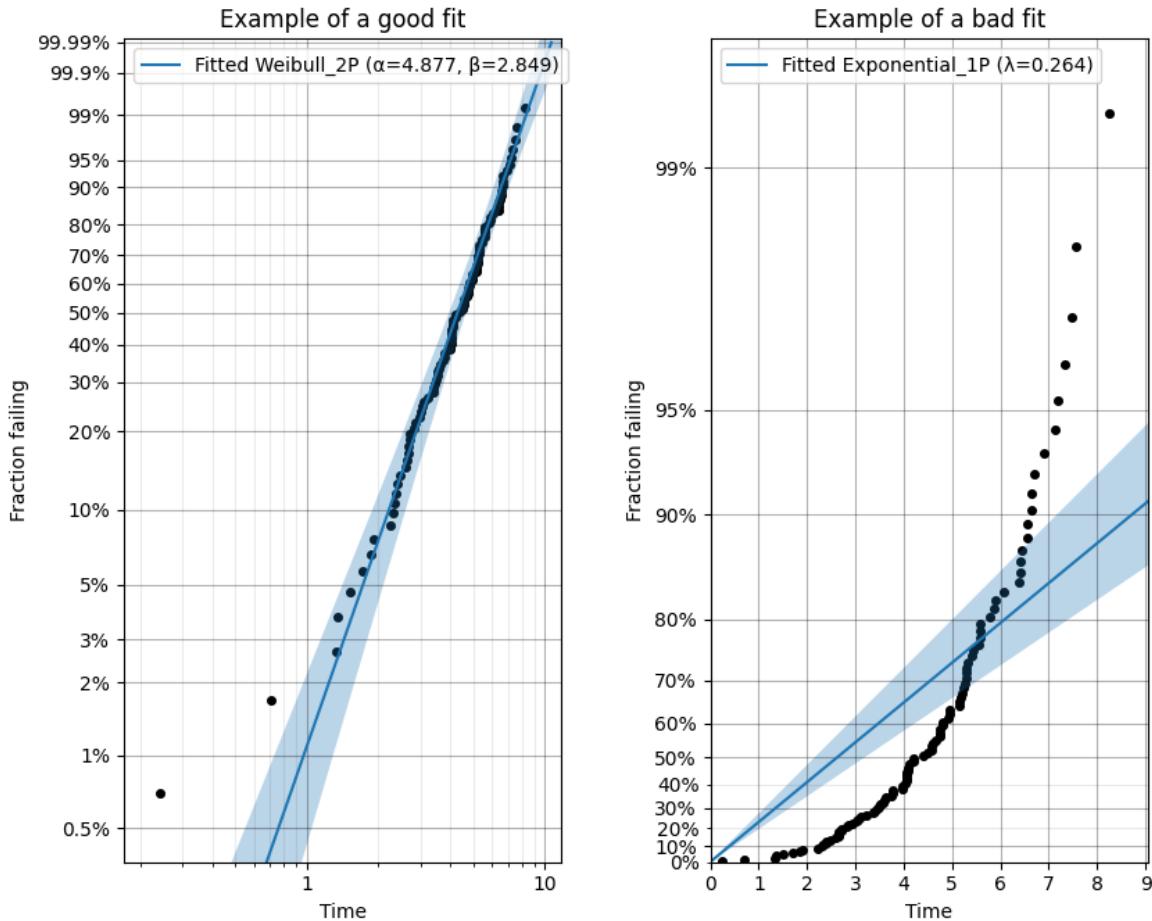
A probability plot shows how well your data is modelled by a particular distribution. By scaling the axes in such a way that the fitted distribution's CDF appears to be a straight line, we can judge whether the empirical CDF of the failure data (the black dots) are in agreement with the CDF of the fitted distribution. Ideally we would see that all of the black dots would lie on the straight line but most of the time this is not the case. A bad fit is evident when the line or curve formed by the black dots is deviating significantly from the straight line. We can usually tolerate a little bit of deviation at the tails of the distribution but the majority of the black dots should follow the line. A historically popular test was the 'fat pencil test' which suggested that if a fat pencil could cover the majority of the data points then the fit was probably suitable. Such a method makes no mention of the size of the plot window which could easily affect the result so it is best to use your own judgement and experience. This approach is not a substitute for statistical inference so it is often preferred to use quantitative measures for goodness of fit such as AICc and BIC. Despite being an imprecise measure,

probability plots remain popular among reliability engineers and in reliability engineering software as they can reveal many features that are not accurately captured in a single goodness of fit statistic.

13.8 Example 7

```
from reliability.Probability_plotting import Weibull_probability_plot, Exponential_
    ↪probability_plot
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt

data = Weibull_Distribution(alpha=5,beta=3).random_samples(100,seed=1)
plt.subplot(121)
Weibull_probability_plot(failures=data)
plt.title('Example of a good fit')
plt.subplot(122)
Exponential_probability_plot(failures=data)
plt.title('Example of a bad fit')
plt.subplots_adjust(bottom=0.1, right=0.94, top=0.93, wspace=0.34) # adjust the
    ↪formatting
plt.show()
```



13.9 Downsampling the scatterplot

When matplotlib is asked to plot large datasets (thousands of items), it can become very slow to generate the plot. To show probability plots when fitting distributions to large datasets, *reliability* allows for the scatterplot to be downsampled.

Downsampling only affects the scatterplot, not the calculations. It is applied automatically for all probability plots (including when these plots are generated as an output from the Fitters), but can be controlled using the “`downsample_scatterplot`” keyword.

If “`downsample_scatterplot`” is True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. For example, if 2 is specified then every 2nd point will be displayed, whereas if 3 is specified then every 3rd point will be displayed.

The min and max points will always be displayed in the downsampled scatterplot which preserves the plotting range.

See the [API](#) documentation for more detail on the default in each function.

13.10 Example 8

In this example, we will see how downsampling affects the probability plot for a dataset with 100000 datapoints.

```
from reliability.Fitters import Fit_Weibull_2P
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt

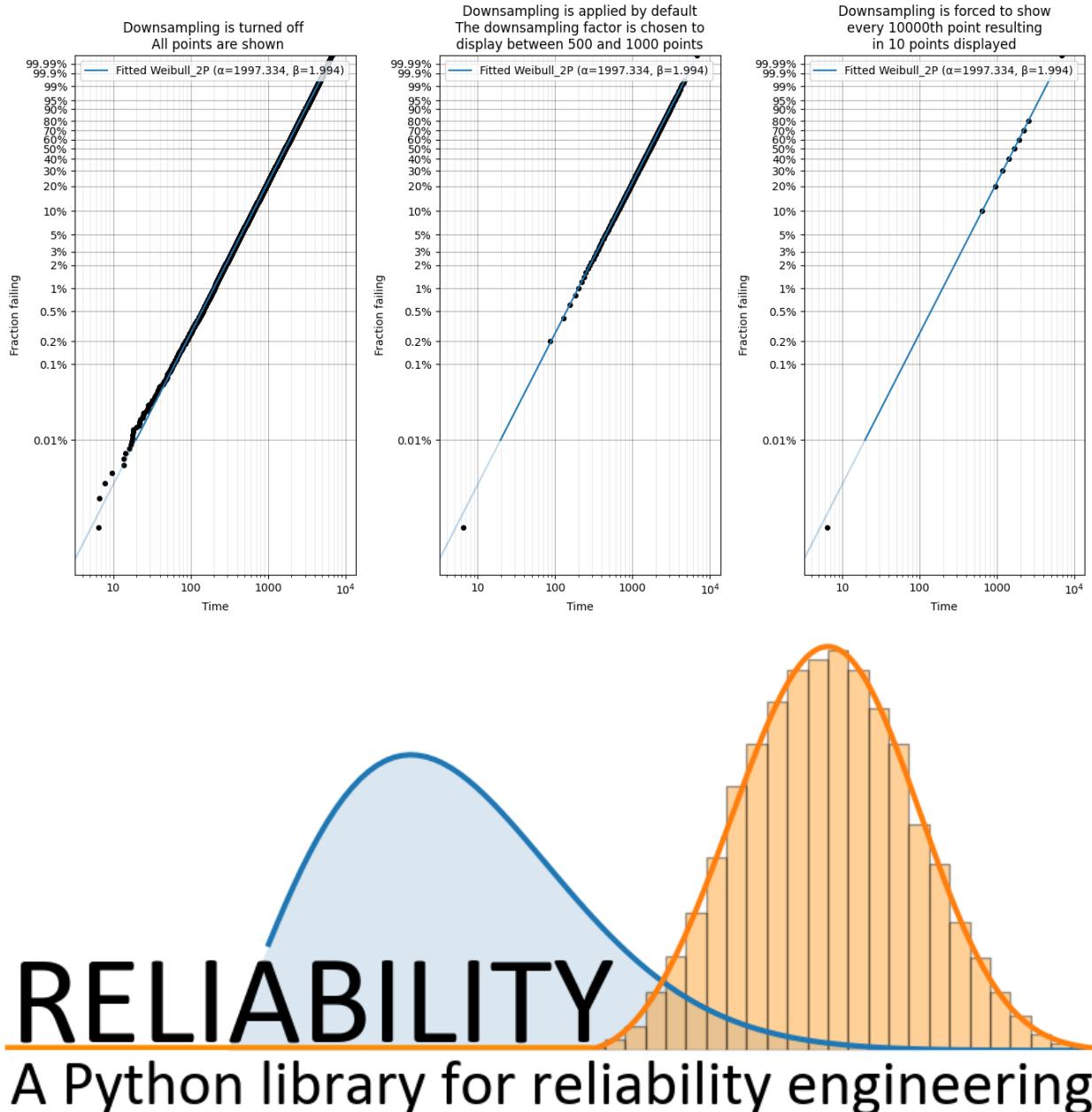
data = Weibull_Distribution(alpha=2000,beta=2).random_samples(100000,seed=1)

plt.subplot(131)
Fit_Weibull_2P(failures=data,print_results=False,downsample_scatterplot=False)
plt.title('Downsampling is turned off \n All points are shown')

plt.subplot(132)
Fit_Weibull_2P(failures=data,print_results=False)
plt.title('Downsampling is applied by default\nThe downsampling factor is chosen to\n->ndisplay between 500 and 1000 points')

plt.subplot(133)
Fit_Weibull_2P(failures=data,print_results=False,downsample_scatterplot=10000)
plt.title('Downsampling is forced to show\nevery 10000th point resulting\nin 10 points\n->displayed')

plt.gcf().set_size_inches(14,8)
plt.tight_layout()
plt.show()
```



QUANTILE-QUANTILE PLOTS

This section contains two different styles of quantile-quantile plots. These are the fully parametric quantile-quantile plot (`reliability.Probability_plotting.QQ_plot_parametric`) and the semi-parametric quantile-quantile plot (`reliability.Probability_plotting.QQ_plot_sempiparametric`). These will be described separately below. A quantile-quantile (QQ) plot is made by plotting time vs time for shared quantiles. A quantile is the time at which a given fraction (from 0 to 1) has failed. In other words we are asking what fraction has failed after a certain time and comparing that fraction for each distribution. If the two distributions are identical then the QQ plot would form a straight line at 45 degrees (assuming the axes are scaled identically). Anything other than a 45 degree line tells us that one distribution leads or lags the other in the fraction failing for a given period of time. Everywhere we say ‘time’ we may equivalently say any other life unit (e.g. cycles, miles, landings, rounds, etc.).

14.1 Parametric Quantile-Quantile plot

To generate this plot we calculate the failure units (these may be units of time, strength, cycles, landings, rounds fired, etc.) at which a certain fraction has failed (0.01,0.02,0.03...0.99). We do this for each distribution so we have an array of failure units and then we plot these failure units against each other. The time (or any other failure unit) at which a given fraction has failed is found using the inverse survival function. If the distributions are identical then the QQ plot will be a straight line at 45 degrees. If the distributions are similar in shape, then the QQ plot should be a reasonably straight line (but not necessarily a 45 degree line) indicating the failure rates are proportional but not identical. By plotting the failure times at equal quantiles for each distribution (and finding the gradient of the line) we can obtain a conversion between the two distributions. Such conversions are useful for accelerated life testing (ALT) to easily convert field time to test time.

 **API Reference**

For inputs and outputs see the [API reference](#).

14.2 Example 1

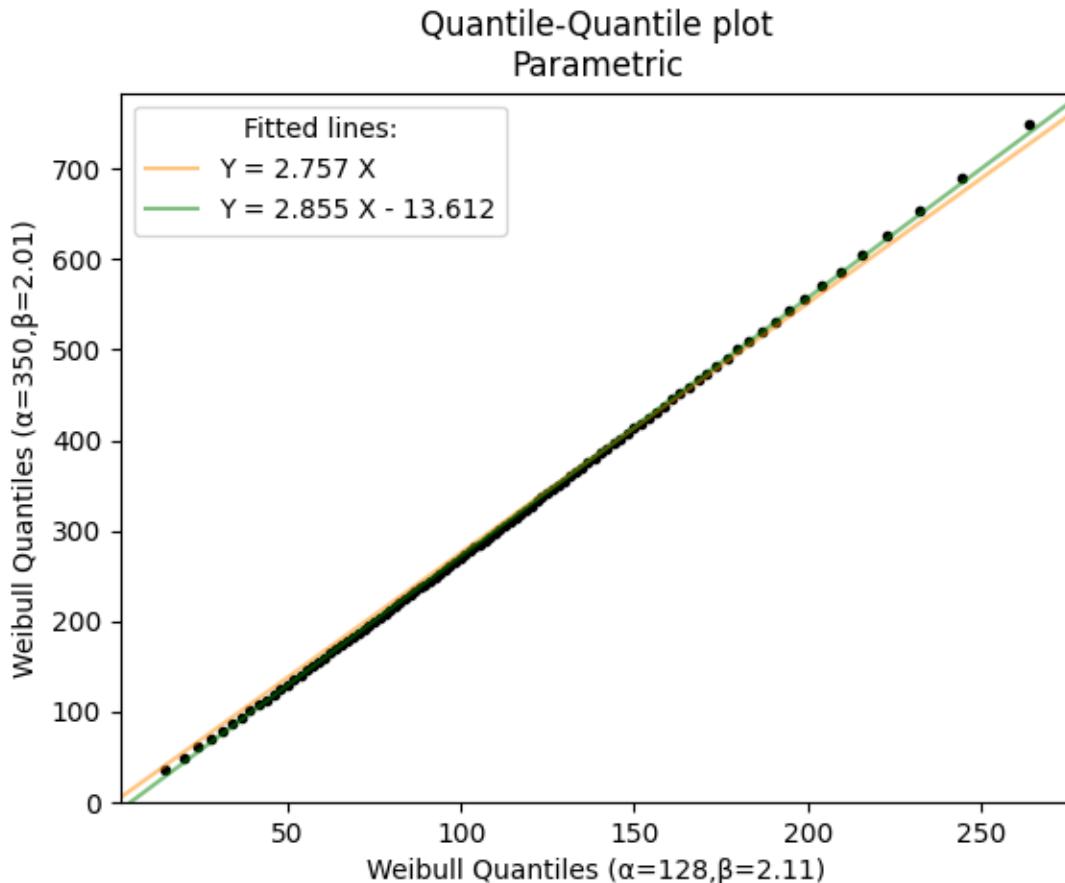
In the example below, we have determined that the field failures follow a Weibull distribution ($=350$, $=2.01$) with time represented in months. By using an accelerated life test we have replicated the failure mode and Weibull shape parameter reasonably closely and the Lab failures follow a Weibull distribution ($=128$, $=2.11$) with time measured in hours. We would like to obtain a simple Field-to-Lab conversion for time so we know how much lab time is required to simulate 10 years of field time. The QQ plot will automatically provide the equations for the lines of best fit. If we use the $Y=m.X$ equation we see that $Field(\text{months})=2.757 \times \text{Lab}(\text{hours})$. Therefore, to simulate 10 years of field time (120 months) we need to run the accelerated life test for approximately 43.53 hours in the Lab.

```
from reliability.Probability_plotting import QQ_plot_parametric
from reliability.Distributions import Weibull_Distribution
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt
Field = Weibull_Distribution(alpha=350,beta=2.01)
Lab = Weibull_Distribution(alpha=128,beta=2.11)
QQ_plot_parametric(X_dist=Lab, Y_dist=Field)
plt.show()
```



14.3 Semiparametric Quantile-Quantile plot

This plot is still a Quantile-Quantile plot (plotting failure units vs failure units for shared quantiles), but instead of using two parametric distributions, we use the failure data directly as one set of quantiles. We then estimate what the quantiles of the parametric distribution would be and plot the parametric (theoretical) failure units against the actual failure units. To generate this plot we begin with the failure units (these may be units of time, strength, cycles, landings, etc.). We then obtain an empirical CDF using either Kaplan-Meier, Nelson-Aalen, or Rank Adjustment. The empirical CDF gives us the quantiles we will use to equate the actual and theoretical failure times. Once we have the empirical CDF, we use the inverse survival function of the specified distribution to obtain the theoretical failure units and then plot the actual and theoretical failure units together. The primary purpose of this plot is as a graphical goodness of fit test. If the specified distribution is a good fit to the data then the QQ plot should be a reasonably straight line along the diagonal.

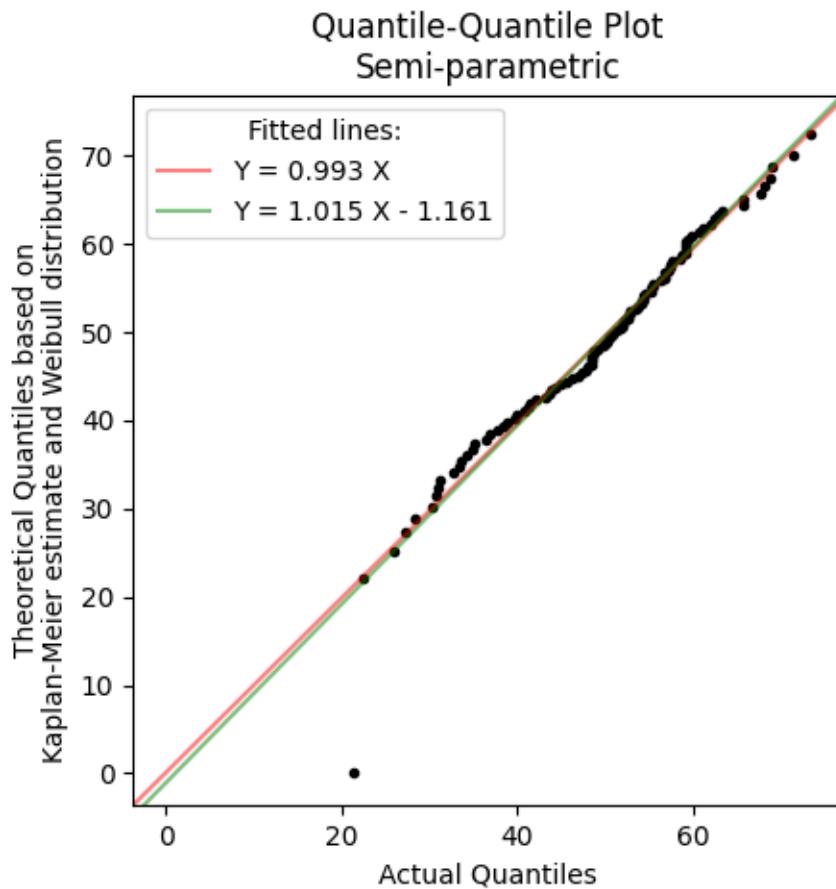
[API Reference](#)

For inputs and outputs see the [API reference](#).

14.4 Example 2

In the example below, we generate 100 random samples from a Normal distribution. We then fit a Weibull_2P distribution to this data and using `QQ_plot_semiparametric` we compare the actual quantile (the original data) with the theoretical quantiles (from the fitted distribution). The lines of best fit are automatically provided and the $Y=0.992X$ shows the relationship is very close to perfect with only some deviation around the tails of the distribution. The final example on this page compares a `QQ_plot_semiparametric` with a `PP_plot_semiparametric` for the same dataset to show the differences between the two.

```
from reliability.Probability_plotting import QQ_plot_semiparametric
from reliability.Fitters import Fit_Weibull_2P
from reliability.Distributions import Normal_Distribution
import matplotlib.pyplot as plt
data = Normal_Distribution(mu=50, sigma=12).random_samples(100)
fitted_dist = Fit_Weibull_2P(failures=data, print_results=False, show_probability_
    ↪plot=False).distribution
QQ_plot_semiparametric(X_data_failures=data, Y_dist=fitted_dist)
plt.show()
```



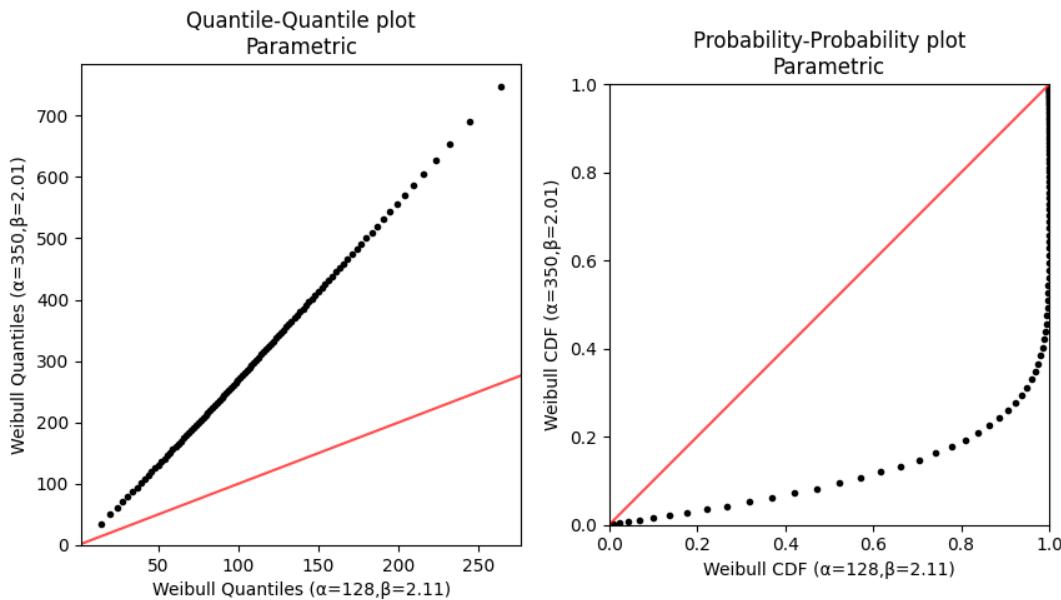
14.5 Comparing PP plots with QQ plots

Normally, it is not practical to compare the output of the two plots as they are so vastly different and are used for different purposes, but the comparison below is provided for the reader's understanding. The differences between these plots are so significant because one is the time at which the fraction has failed (the Quantile) and the other is the fraction failing at a given time (the CDF). Parametric PP plots are not very common as their only use is in providing a graphical understanding of the differences between the CDFs of two distributions, such as how one lags or leads the other at various times. See [Probability-Probability plots](#) for more detail on the uses of parametric PP plots.

14.6 Example 3

In this example we compare a QQ_plot_parametric with a PP_plot_parametric for the same pair of distributions.

```
from reliability.Probability_plotting import QQ_plot_parametric, PP_plot_parametric
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt
Field = Weibull_Distribution(alpha=350,beta=2.01)
Lab = Weibull_Distribution(alpha=128,beta=2.11)
plt.figure(figsize=(10,5))
plt.subplot(121)
QQ_plot_parametric(X_dist=Lab, Y_dist=Field, show_diagonal_line=True, show_fitted_
lines=False)
plt.subplot(122)
PP_plot_parametric(X_dist=Lab, Y_dist=Field, show_diagonal_line=True)
plt.show()
```

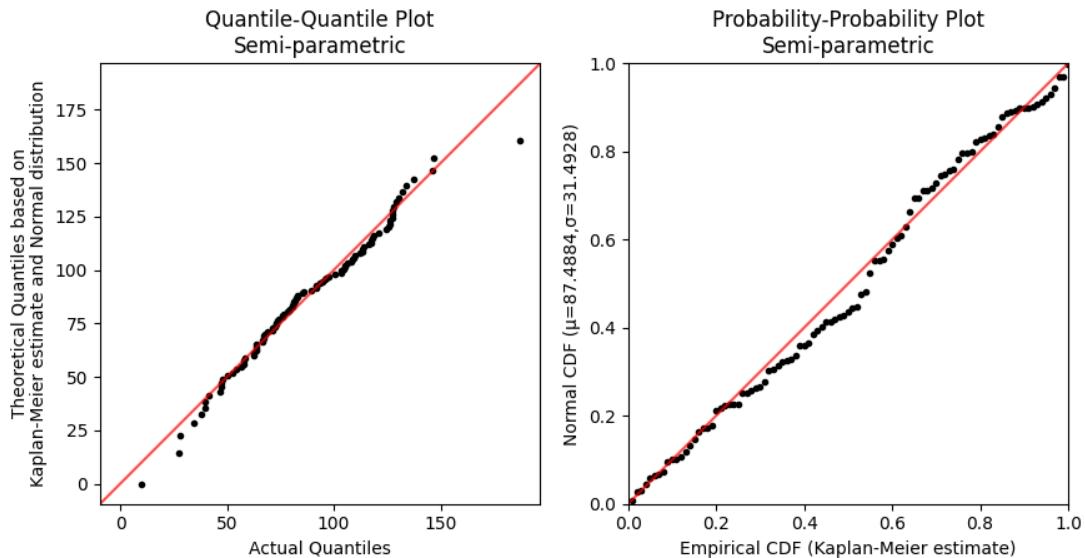


14.7 Example 4

In this example we compare a QQ_plot_semi parametric with a PP_plot_semi parametric for the same dataset. Both plots are intended to be used as graphical goodness of fit tests. In a PP plot we get a lot of resolution in the center of the distributions, but less at the tails, whereas the QQ plot gives very good resolution at the tails, but less in the center. Because most data analysts are more concerned about the extremes (tails) of a distribution, QQ plots are the

more commonly used plot between the two.

```
from reliability.Probability_plotting import PP_plot_semiparametric, QQ_plot_
    _semiparametric
from reliability.Fitters import Fit_Normal_2P
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt
data = Weibull_Distribution(alpha=100,beta=3).random_samples(100) #create some data
dist = Fit_Normal_2P(failures=data,print_results=False,show_probability_plot=False).
    distribution #fit a normal distribution
plt.figure(figsize=(10,5))
plt.subplot(121)
QQ_plot_semiparametric(X_data_failures=data,Y_dist=dist,show_fitted_lines=False,show_
    _diagonal_line=True)
plt.subplot(122)
PP_plot_semiparametric(X_data_failures=data,Y_dist=dist)
plt.show()
```





RELIABILITY
A Python library for reliability engineering

PROBABILITY-PROBABILITY PLOTS

This section contains two different styles of probability-probability (PP) plots. These are the fully parametric probability-probability plot (*reliability.Probability_plotting.PP_plot_parametric*) and the semi-parametric probability-probability plot (*reliability.Probability_plotting.PP_plot_semiparametric*). These will be described separately below. A PP plot is made by plotting the fraction failing (CDF) of one distribution vs the fraction failing (CDF) of another distribution. In the semiparametric form, when we only have the failure data and one hypothesised distribution, the CDF for the data can be obtained non-parametrically to generate an empirical CDF.

15.1 Parametric Probability-Probability plot

To generate this plot we simply plot the CDF of one distribution vs the CDF of another distribution. If the distributions are very similar, the points will lie on the 45 degree diagonal. Any deviation from this diagonal indicates that one distribution is leading or lagging the other. Fully parametric PP plots are rarely used as their utility is limited to providing a graphical comparison of the similarity between two CDFs. To aide this comparison, the `PP_plot_parametric` function accepts `x` and `y` quantile lines that will be traced across to the other distribution.

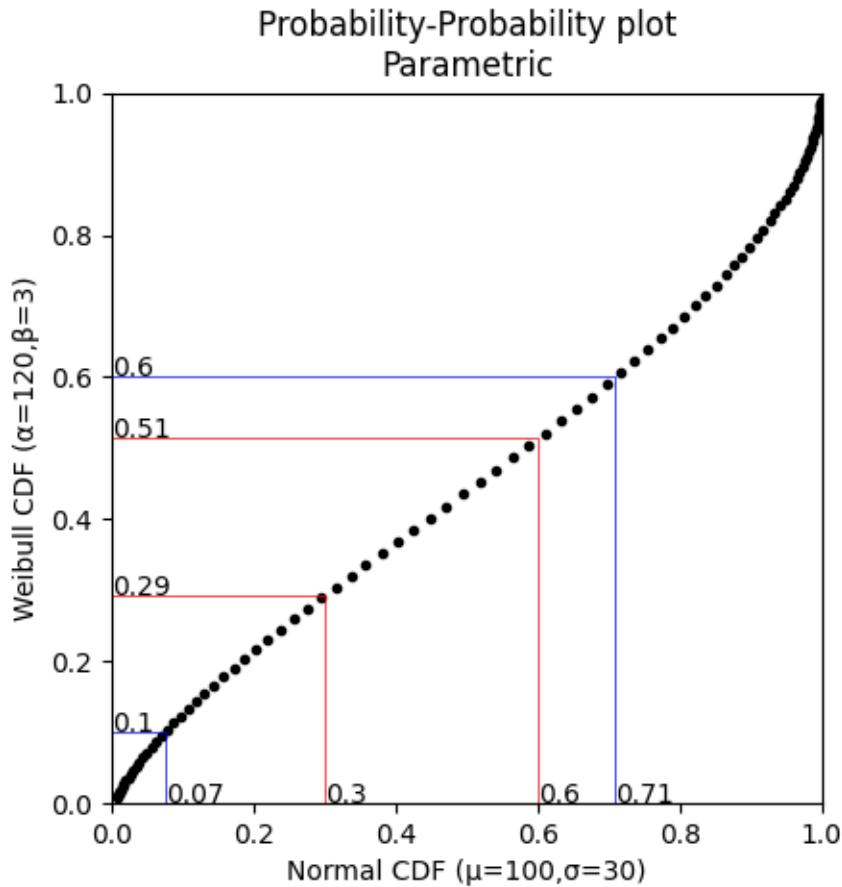
ⓘ API Reference

For inputs and outputs see the [API reference](#).

15.2 Example 1

In the example below, we generate two parametric distributions and compare them using a PP plot. We are interested in the differences at specific quantiles so these are specified and the plot traces them across to the opposing distribution.

```
from reliability.Probability_plotting import PP_plot_parametric
from reliability.Distributions import Weibull_Distribution,Normal_Distribution
import matplotlib.pyplot as plt
Field = Normal_Distribution(mu=100,sigma=30)
Lab = Weibull_Distribution(alpha=120,beta=3)
PP_plot_parametric(X_dist=Field, Y_dist=Lab, x_quantile_lines=[0.3, 0.6], y_quantile_
↳lines=[0.1, 0.6])
plt.show()
```



15.3 Semiparametric Probability-Probability plot

A semiparametric PP plot is still a probability-probability plot, but since we only have one parametric distribution to give us the CDF, we must use the failure data to obtain the non-parametric estimate of the empirical CDF. To create a semiparametric PP plot, we must provide the failure data and the non-parametric method ('KM', 'NA', 'RA' for Kaplan-Meier, Nelson-Aalen, and Rank Adjustment respectively) to estimate the empirical CDF, and we must also provide the parametric distribution for the parametric CDF. The failure units (times, cycles, rounds fired, strength units, etc.) are the limiting values here so the parametric CDF is only calculated at the failure units since that is the result we get from the empirical CDF. Note that the empirical CDF also accepts `X_data_right_censored` just as `KaplanMeier`, `NelsonAalen` and `RankAdjustment` will also accept right censored data. If the fitted distribution is a good fit the PP plot will follow the 45 degree diagonal line. Assessing goodness of fit in a graphical way is the main purpose of this type of plot. The `Fit_everything` function also uses a semiparametric PP plot to show the goodness of fit in a graphical way.

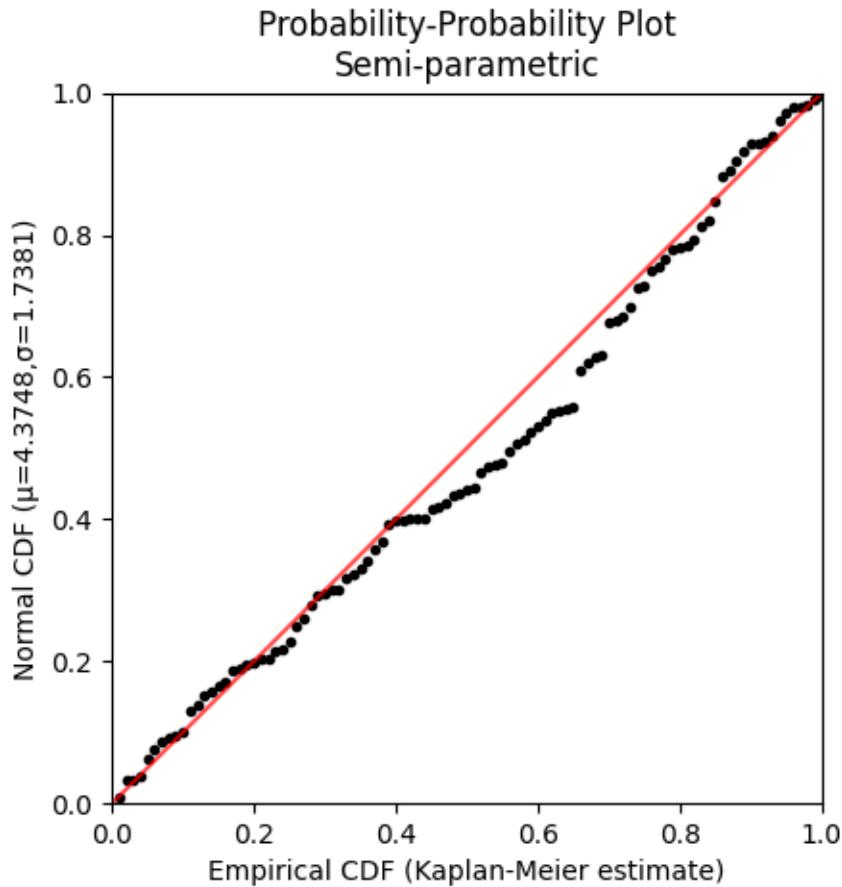
API Reference

For inputs and outputs see the [API reference](#).

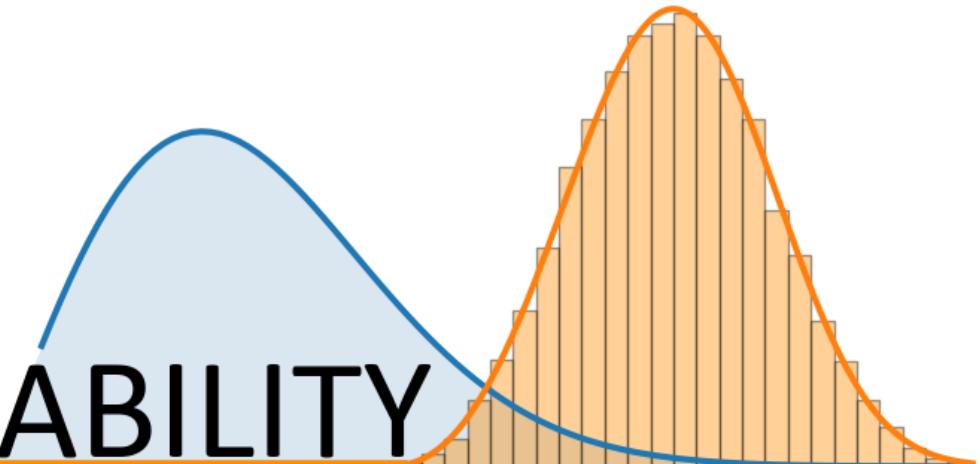
15.4 Example 2

In the example below, we create 100 random samples from a Weibull distribution. We hypothesise that a Normal distribution may fit this data well so we fit the Normal distribution and then plot the CDF of the fitted distribution against the empirical CDF (obtained using the Kaplan-Meier estimate). We see that the plot follows the 45 degree diagonal quite well so we may consider that the fitted Normal distribution is reasonably good at describing this data. Ideally, this comparison should be made against other distributions as well and the graphical results are often hard to tell apart which is why we often use quantitative goodness of fit measures like AICc and BIC.

```
from reliability.Probability_plotting import PP_plot_semiparametric
from reliability.Fitters import Fit_Normal_2P
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt
data = Weibull_Distribution(alpha=5,beta=3).random_samples(100)
dist = Fit_Normal_2P(failures=data,show_probability_plot=False,print_results=False).
       distribution
PP_plot_semiparametric(X_data_failures=data,Y_dist=dist)
plt.show()
```



To see how semiparametric PP plots compare with semiparametric QQ plots as a graphical goodness of fit test, please see the second example in the section on [comparing PP plots with QQ plots](#).



RELIABILITY
A Python library for reliability engineering

KAPLAN-MEIER

API Reference

For inputs and outputs see the [API reference](#).

The Kaplan-Meier estimator provides a method by which to estimate the survival function (reliability function) of a population without assuming that the data comes from a particular distribution. Due to the lack of parameters required in this model, it is a non-parametric method of obtaining the survival function. With a few simple transformations, the survival function (SF) can be used to obtain the cumulative hazard function (CHF) and the cumulative distribution function (CDF). It is not possible to obtain a useful version of the probability density function (PDF) or hazard function (HF) as this would require the differentiation of the CDF and CHF respectively, which results in a very spiky plot due to the non-continuous nature of these plots.

The Kaplan-Meier estimator is very similar in result (but quite different in method) to the [Nelson-Aalen estimator](#) and [Rank Adjustment estimator](#). While none of the three has been proven to be more accurate than the others, the Kaplan-Meier estimator is generally more popular as a non-parametric means of estimating the SF. Confidence intervals are provided using the [Greenwood method](#) with Normal approximation.

The Kaplan-Meier estimator can be used with both complete and right censored data. This function can be accessed from *reliability.Nonparametric.KaplanMeier*.

16.1 Example 1

In this first example, we will provide Kaplan-Meier with a list of failure times and right censored times. By leaving everything else unspecified, the plot will be shown with the confidence intervals shaded. We will layer this first Kaplan-Meier plot with a second one using just the failure data. As can be seen in the example below, the importance of including censored data is paramount to obtain an accurate estimate of the reliability, because without it the population's survivors are not included so the reliability will appear much lower than it truly is.

```
from reliability.Nonparametric import KaplanMeier
import matplotlib.pyplot as plt
f = [5248, 7454, 16890, 17200, 38700, 45000, 49390, 69040, 72280, 131900]
rc = [3961, 4007, 4734, 6054, 7298, 10190, 23060, 27160, 28690, 37100, 40060, 45670, 53000, 67000, 69630, 77350, 78470, 91680, 105700, 106300, 150400]
KaplanMeier(failures=f, right_censored=rc, label='Failures + right censored')
KaplanMeier(failures=f, label='Failures only')
plt.title('Kaplan-Meier estimates showing the\nimportance of including censored data')
plt.xlabel('Miles to failure')
plt.legend()
plt.show()
```

(continues on next page)

(continued from previous page)

""

Results from KaplanMeier (95% CI):

Failure times	Censoring code (censored=0)	Items remaining	Kaplan-Meier Estimate	Lower CI bound	Upper CI bound
3961		0	1	3961	
1	1	0	1	1	
4007		0	1	4007	
1	1	0	1	1	
4734		0	1	4734	
1	1	1	1	1	
5248		1	0.964286	5248	
0.895548	1	0	0.964286	0.895548	
6054		0	0.964286	6054	
0.895548	1	0	0.964286	0.895548	
7298		0	0.964286	7298	
0.895548	1	0	0.964286	0.895548	
7454		1	0.925714	7454	
0.826513	1	1	0.925714	0.826513	
10190		0	0.925714	10190	
0.826513	1	0	0.925714	0.826513	
16890		1	0.885466	16890	
0.76317	1	1	0.885466	0.76317	
17200		1	0.845217	17200	
0.705334	0.985101	0	0.845217	0.705334	
23060		0	0.845217	23060	
0.705334	0.985101	0	0.845217	0.705334	
27160		0	0.845217	27160	
0.705334	0.985101	0	0.845217	0.705334	
28690		0	0.845217	28690	
0.705334	0.985101	0	0.845217	0.705334	
37100		0	0.845217	37100	
0.705334	0.985101	0	0.845217	0.705334	
38700		1	0.795499	38700	
0.633417	0.95758	0	0.795499	0.633417	
40060		0	0.795499	40060	
0.633417	0.95758	0	0.795499	0.633417	
45000		1	0.742465	45000	
0.560893	0.924037	1	0.742465	0.560893	
45670		0	0.742465	45670	
0.560893	0.924037	0	0.742465	0.560893	
49390		1	0.685353	49390	
0.48621	0.884496	0	0.685353	0.48621	
53000		0	0.685353	53000	
0.48621	0.884496	0	0.685353	0.48621	
67000		0	0.685353	67000	
0.48621	0.884496	0	0.685353	0.48621	
69040		1	0.616817	69040	
0.396904	0.836731	0	0.616817	0.396904	
69630		0	0.616817	69630	
0.396904	0.836731	0	0.616817	0.396904	
72280		1	0.539715	72280	

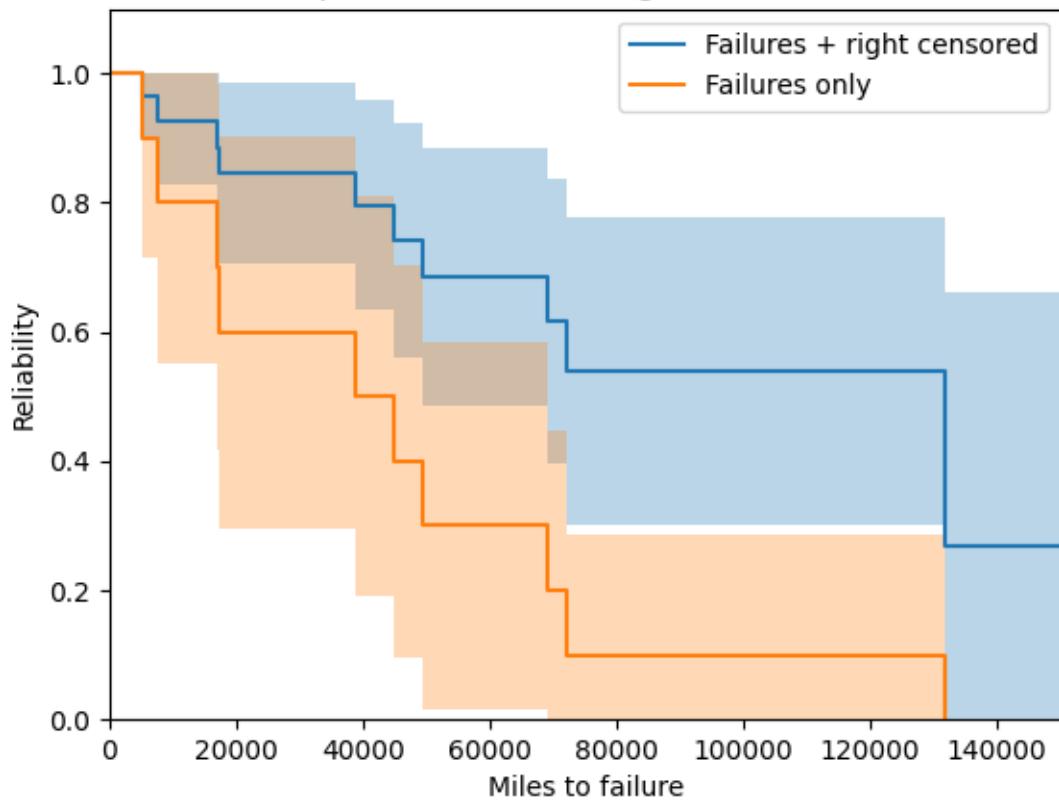
(continues on next page)

(continued from previous page)

↪ 0.300949	0.778481	0	7	0.539715	█
77350					
↪ 0.300949	0.778481	0	6	0.539715	█
78470					
↪ 0.300949	0.778481	0	5	0.539715	█
91680					
↪ 0.300949	0.778481	0	4	0.539715	█
105700					
↪ 0.300949	0.778481	0	3	0.539715	█
106300					
↪ 0.300949	0.778481	1	2	0.269858	█
131900					
↪ 0	0.662446	0	1	0.269858	█
150400					
↪ 0	0.662446				

Results from KaplanMeier (95% CI):					
Failure times	Censoring code (censored=0)	Items remaining	Kaplan-Meier Estimate	█	
↪ Lower CI bound	Upper CI bound				
5248		1	10	0.9	█
↪ 0.714061	1	1	9	0.8	█
7454					
↪ 0.552082	1	1	8	0.7	█
16890					
↪ 0.415974	0.984026	1	7	0.6	█
17200					
↪ 0.296364	0.903636	1	6	0.5	█
38700					
↪ 0.190102	0.809898	1	5	0.4	█
45000					
↪ 0.0963637	0.703636	1	4	0.3	█
49390					
↪ 0.0159742	0.584026	1	3	0.2	█
69040					
↪ 0	0.447918	1	2	0.1	█
72280					
↪ 0	0.285939	1	1	0	█
131900					
↪ 0	0				
""					

Kaplan-Meier estimates showing the importance of including censored data



16.2 Example 2

In this second example, we will create some data from a Weibull distribution, and then right censor the data above our chosen threshold. We will then fit a Weibull_2P distribution to the censored data, and also obtain the Kaplan-Meier estimate of this data. Using the results from the Fit_Weibull_2P and the Kaplan-Meier estimate, we will plot the CDF, SF, and CHF, for both the Weibull and Kaplan-Meier results. Note that the default plot from KaplanMeier will only give you the SF, but the results object provides everything you need to reconstruct the SF plot yourself, as well as what we need to plot the CDF and CHF.

```
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_2P
from reliability.Nonparametric import KaplanMeier
from reliability.Other_functions import make_right_censored_data
import matplotlib.pyplot as plt

dist = Weibull_Distribution(alpha=5, beta=2) # create a distribution
raw_data = dist.random_samples(100, seed=2) # get some data from the distribution.
# Seeded for repeatability
data = make_right_censored_data(raw_data, threshold=9)
wbf = Fit_Weibull_2P(failures=data.failures, right_censored=data.right_censored, show_
#probability_plot=False, print_results=False) # Fit the Weibull_2P

# Create the subplots and in each subplot we will plot the parametric distribution and
```

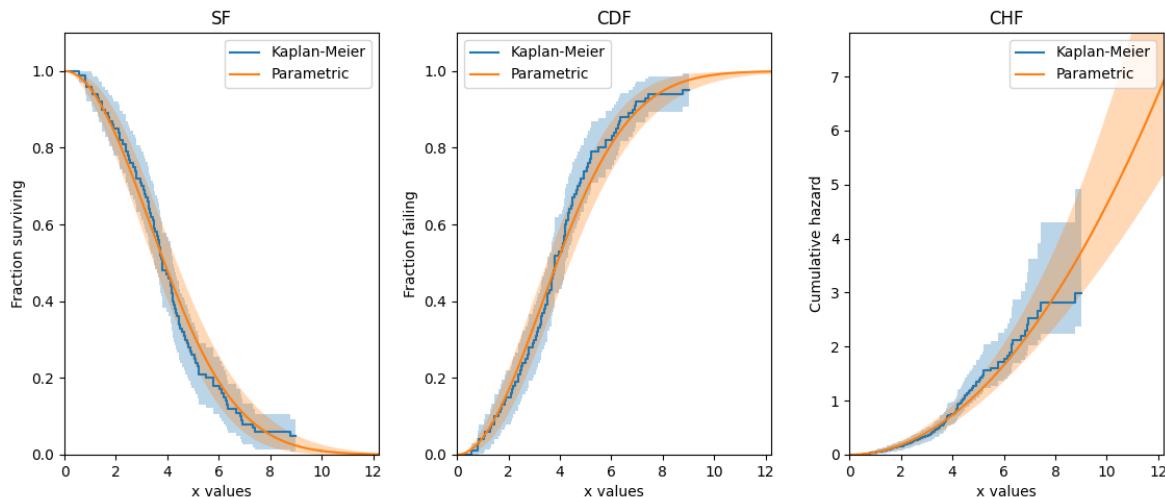
(continues on next page)

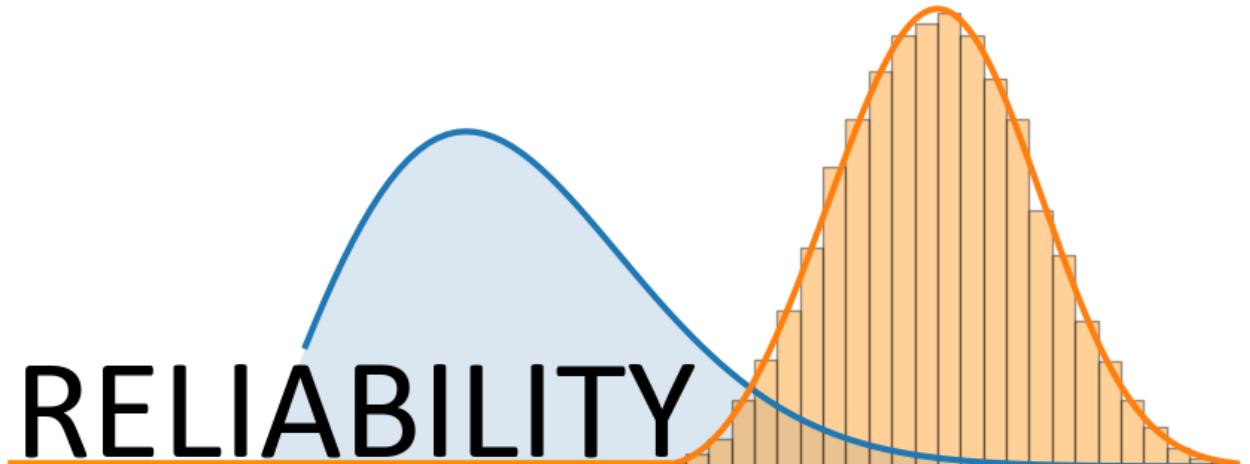
(continued from previous page)

```

→ obtain the Kaplan Meier fit.
# Note that the plot_type is being changed each time
plt.figure(figsize=(12, 5))
plt.subplot(131)
KaplanMeier(failures=data.failures, right_censored=data.right_censored, plot_type='SF', ↴
→ print_results=False, label='Kaplan-Meier')
wbf.distribution.SF(label='Parametric')
plt.legend()
plt.title('SF')
plt.subplot(132)
KaplanMeier(failures=data.failures, right_censored=data.right_censored, plot_type='CDF', ↴
→ print_results=False, label='Kaplan-Meier')
wbf.distribution.CDF(label='Parametric')
plt.legend()
plt.title('CDF')
plt.subplot(133)
KaplanMeier(failures=data.failures, right_censored=data.right_censored, plot_type='CHF', ↴
→ print_results=False, label='Kaplan-Meier')
wbf.distribution.CHF(label='Parametric')
plt.legend()
plt.title('CHF')
plt.subplots_adjust(left=0.07, right=0.95, top=0.92, wspace=0.25) # format the plot
→ layout
plt.show()

```





RELIABILITY
A Python library for reliability engineering

NELSON-AALEN

API Reference

For inputs and outputs see the [API reference](#).

The Nelson-Aalen estimator provides a method to estimate the hazard function of a population without assuming that the data comes from a particular distribution. From the hazard function, the Nelson-Aalen method obtains the cumulative hazard function, which is then used to obtain the survival function. Due to the lack of parameters required in this model, it is a non-parametric method of obtaining the survival function. As with the Kaplan-Meier estimator, once we have the survival function (SF), then we also have the cumulative hazard function (CHF) and the cumulative distribution function (CDF). It is not possible to obtain a useful version of the probability density function (PDF) or hazard function (HF). While the hazard function is obtained directly by the Nelson-Aalen method, it is a useless function on its own as it is a very spiky plot due to the non-continuous nature of the hazard. It is only when we smooth the results out using the cumulative hazard function that we obtain some utility from the results.

The Nelson-Aalen estimator is very similar in result (but quite different in method) to the [Kaplan-Meier estimator](#) and [Rank Adjustment estimator](#). While none of the three have been proven to be more accurate than the others, the Kaplan-Meier estimator is generally more popular as a non-parametric means of estimating the SF. Confidence intervals are provided using the [Greenwood method](#) with Normal approximation.

The Nelson-Aalen estimator can be used with both complete and right censored data. This function can be accessed from *reliability.Nonparametric.NelsonAalen* as shown in the examples below.

17.1 Example 1

In the example below, we will compare the results from the Nelson-Aalen estimator with the results from the Kaplan-Meier estimator and Rank Adjustment estimator. We will also extract the column of point estimates from the results and print these for each method in a dataframe.

```
from reliability.Nonparametric import KaplanMeier, NelsonAalen, RankAdjustment
import matplotlib.pyplot as plt
import pandas as pd

failures = [5248, 7454, 16890, 17200, 38700, 45000, 49390, 69040, 72280, 131900]
censored = [3961, 4007, 4734, 6054, 7298, 10190, 23060, 27160, 28690, 37100, 40060, 45670, 53000, 67000, 69630, 77350, 78470, 91680, 105700, 106300, 150400]
KM = KaplanMeier(failures=failures, right_censored=censored, label='Kaplan-Meier', print_results=False)
NA = NelsonAalen(failures=failures, right_censored=censored, label='Nelson-Aalen', print_results=False)
```

(continues on next page)

(continued from previous page)

```

RA = RankAdjustment(failures=failures, right_censored=censored, label='Rank Adjustment
', print_results=False)

plt.title('Comparison of Kaplan-Meier, Nelson-Aalen, and Rank Adjustment\nwith 95% CI
bounds')
plt.legend()

# print a table of the SF estimates for each method
data = {'Kaplan-Meier': KM.KM, 'Nelson-Aalen': NA.NA, 'Rank Adjustment': RA.RA}
df = pd.DataFrame(data, columns=['Kaplan-Meier', 'Nelson-Aalen', 'Rank Adjustment'])
print(df)

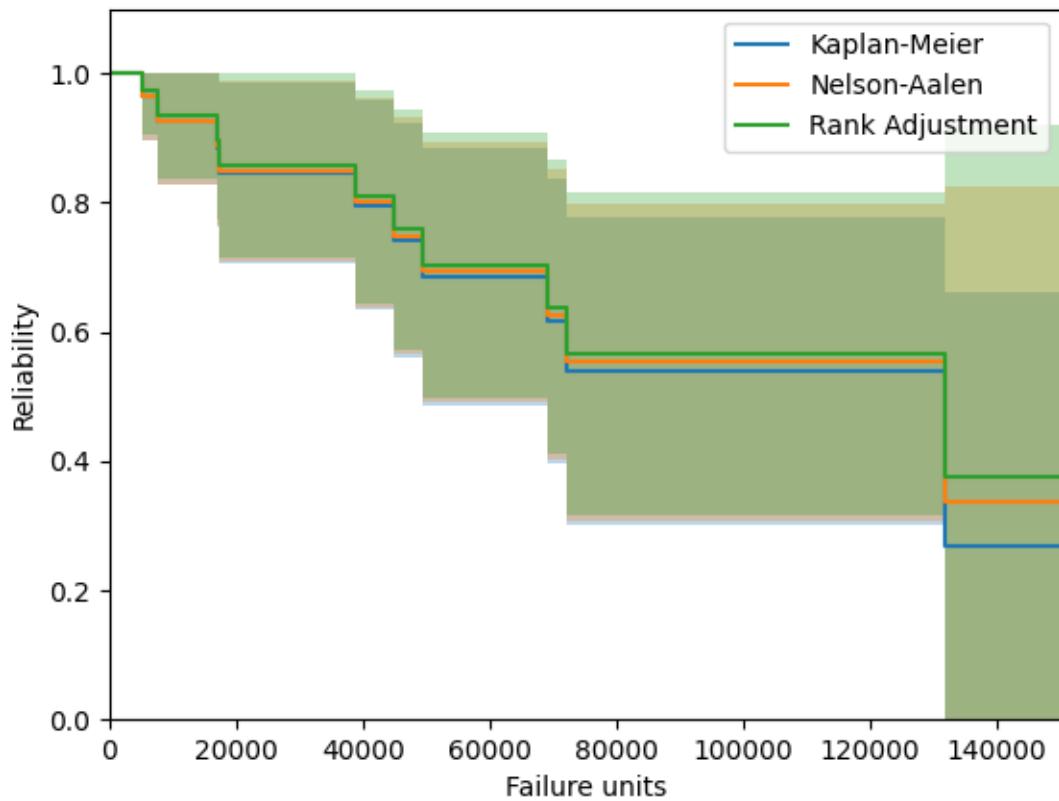
plt.show()

"""

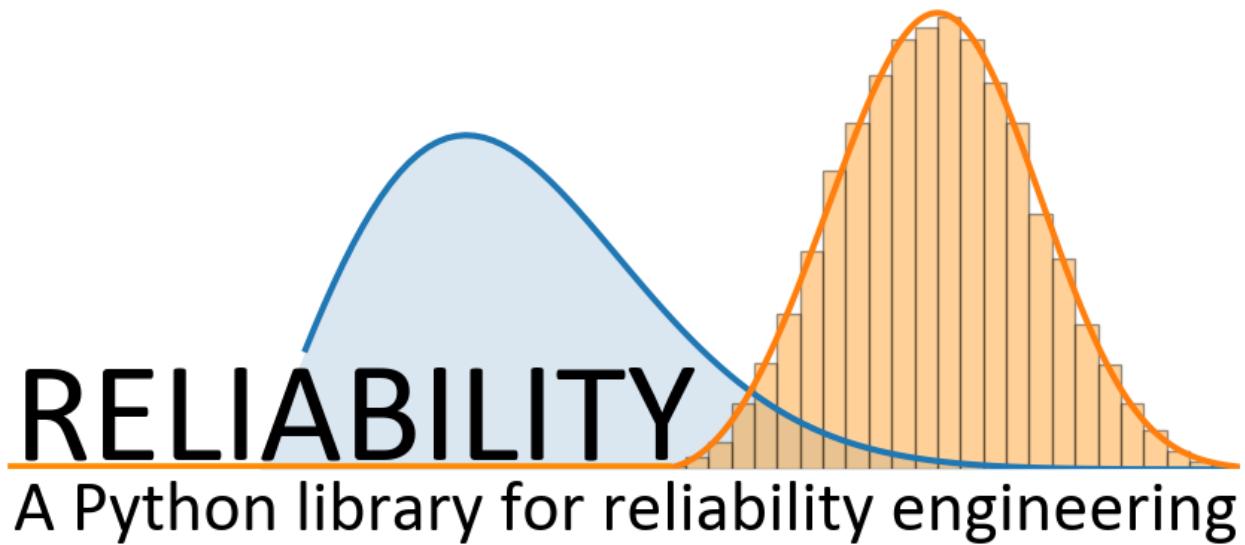
    Kaplan-Meier    Nelson-Aalen    Rank Adjustment
0      1.000000      1.000000      1.000000
1      1.000000      1.000000      1.000000
2      1.000000      1.000000      1.000000
3      0.964286      0.964916      0.974412
4      0.964286      0.964916      0.974412
5      0.964286      0.964916      0.974412
6      0.925714      0.927081      0.936568
7      0.925714      0.927081      0.936568
8      0.885466      0.887637      0.897146
9      0.845217      0.848193      0.857724
10     0.845217      0.848193      0.857724
11     0.845217      0.848193      0.857724
12     0.845217      0.848193      0.857724
13     0.845217      0.848193      0.857724
14     0.795499      0.799738      0.809542
15     0.795499      0.799738      0.809542
16     0.742465      0.748161      0.758348
17     0.742465      0.748161      0.758348
18     0.685353      0.692768      0.703498
19     0.685353      0.692768      0.703498
20     0.685353      0.692768      0.703498
21     0.616817      0.626842      0.638675
22     0.616817      0.626842      0.638675
23     0.539715      0.553186      0.566650
24     0.539715      0.553186      0.566650
25     0.539715      0.553186      0.566650
26     0.539715      0.553186      0.566650
27     0.539715      0.553186      0.566650
28     0.539715      0.553186      0.566650
29     0.269858      0.335524      0.374582
30     0.269858      0.335524      0.374582
"""

```

Comparison of Kaplan-Meier, Nelson-Aalen, and Rank Adjustment with 95% CI bounds



Further examples are provided in the documentation for the [Kaplan-Meier estimator](#) as this function is written to work exactly the same way as the Nelson-Aalen estimator.



RANK ADJUSTMENT

API Reference

For inputs and outputs see the [API reference](#).

The Rank Adjustment estimator provides a method by which to estimate the survival function (reliability function) of a population without assuming that the data comes from a particular distribution. Due to the lack of parameters required in this model, it is a non-parametric method of obtaining the survival function. With a few simple transformations, the survival function (SF) can be used to obtain the cumulative hazard function (CHF) and the cumulative distribution function (CDF). It is not possible to obtain a useful version of the probability density function (PDF) or hazard function (HF) as this would require the differentiation of the CDF and CHF respectively, which results in a very spiky plot due to the non-continuous nature of these plots.

The Rank Adjustment estimator is very similar in result (but quite different in method) to the [Kaplan-Meier estimator](#) and [Nelson-Aalen estimator](#). While none of the three has been proven to be more accurate than the others, the Kaplan-Meier estimator is generally more popular as a non-parametric means of estimating the SF. Confidence intervals are provided using the [Greenwood method](#) with Normal approximation.

The Rank Adjustment estimator can be used with both complete and right censored data. This function can be accessed from `reliability.Nonparametric.RankAdjustment`. The Rank-adjustment algorithm is the same as is used in `Probability_plotting.plotting_positions` to obtain the y-values for the scatter plot. As with `plotting_positions`, the heuristic constant “a” is accepted, with the default being 0.3 for median ranks.

18.1 Example 1

In this first example, we will see how Rank Adjustment compares with Kaplan-Meier and Nelson-Aalen for a large censored dataset. The plots show these three methods arrive at a similar result, with Kaplan-Meier giving the lowest estimate of the survival function, followed by Nelson-Aalen, and finally Rank-Adjustment. Note that this is when the median ranks are used in the Rank Adjustment heuristic. As sample size is increased, the differences between the three methods reduces.

```
import matplotlib.pyplot as plt
from reliability.Other_functions import make_right_censored_data
from reliability.Nonparametric import KaplanMeier, NelsonAalen, RankAdjustment
from reliability.Distributions import Weibull_Distribution

dist = Weibull_Distribution(alpha=500, beta=2)

plt.figure(figsize=(12, 7))
samples = [10, 100, 1000]
```

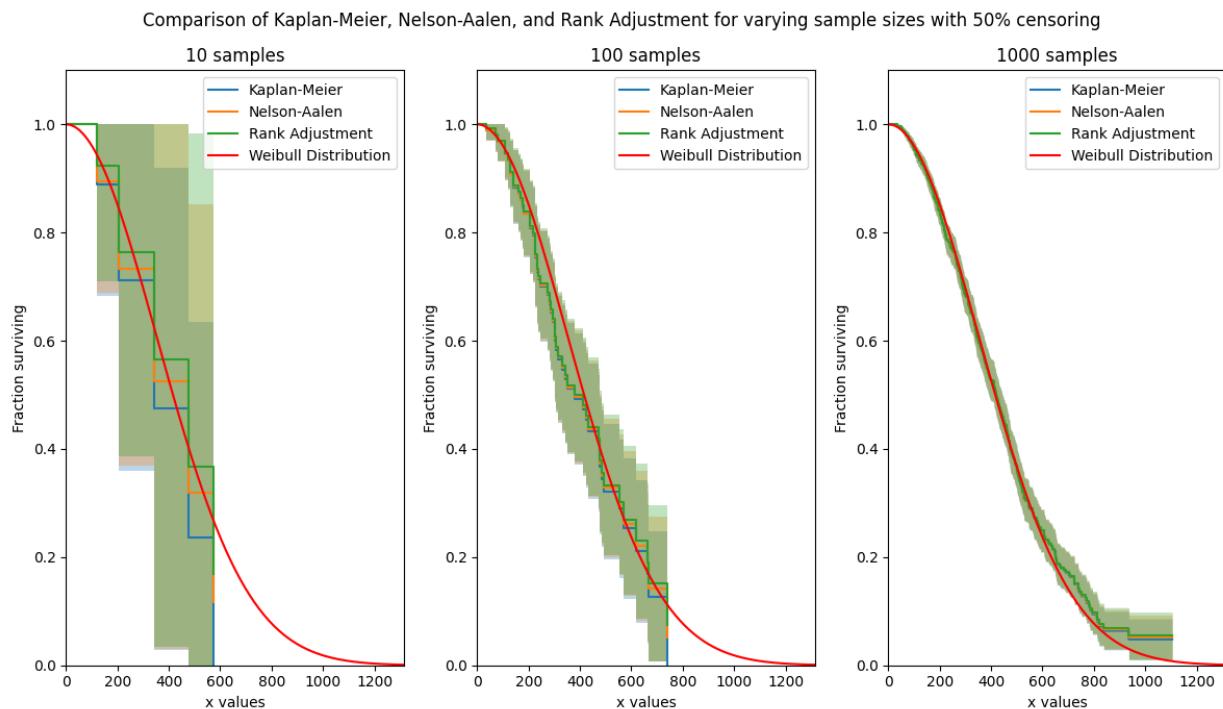
(continues on next page)

(continued from previous page)

```

for i, s in enumerate(samples):
    raw_data = dist.random_samples(number_of_samples=s, seed=42)
    data = make_right_censored_data(data=raw_data, fraction_censored=0.5, seed=42) #_
    # this will multiply-censor 50% of the data
    plt.subplot(131 + i)
    KaplanMeier(failures=data.failures, right_censored=data.right_censored, print_
    results=False, show_plot=True, label='Kaplan-Meier')
    NelsonAalen(failures=data.failures, right_censored=data.right_censored, print_
    results=False, show_plot=True, label='Nelson-Aalen')
    RankAdjustment(failures=data.failures, right_censored=data.right_censored, print_
    results=False, show_plot=True, label='Rank Adjustment')
    dist.SF(label='Weibull Distribution', color='red')
    plt.title(str(s) + ' samples')
    plt.legend()
plt.suptitle('Comparison of Kaplan-Meier, Nelson-Aalen, and Rank Adjustment for varying_
    sample sizes with 50% censoring')
plt.tight_layout()
plt.show()

```



18.2 Example 2

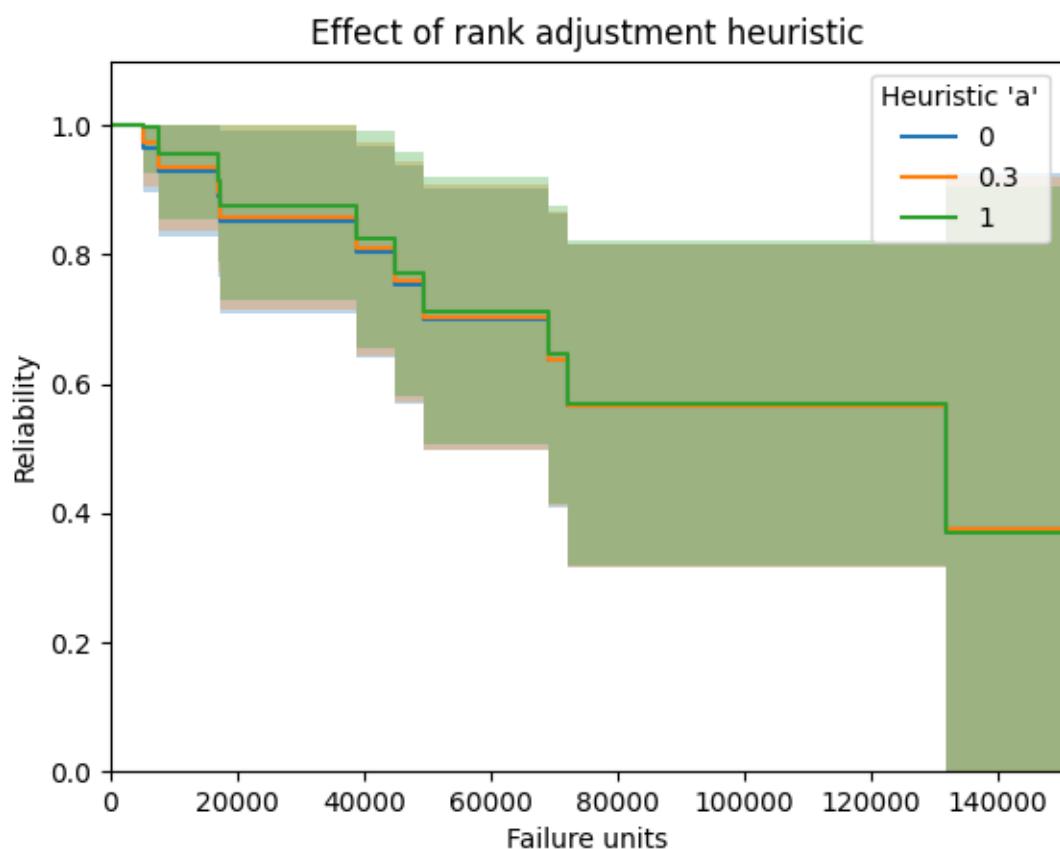
In this second example we will look at the effect of the plotting heuristic “a”. The default heuristic used is 0.3 which gives the median ranks, but there are many [other heuristics](#) available by varying a from 0 to 1. Here we will look at the effect of setting “a” to be 0, 0.3, and 1. The effect is fairly minimal, though there is a small difference (which reduces as sample size increases) leading to the use of different heuristics. The median ranks (a=0.3) is most popular and is the default in most reliability engineering software for obtaining the plotting positions.

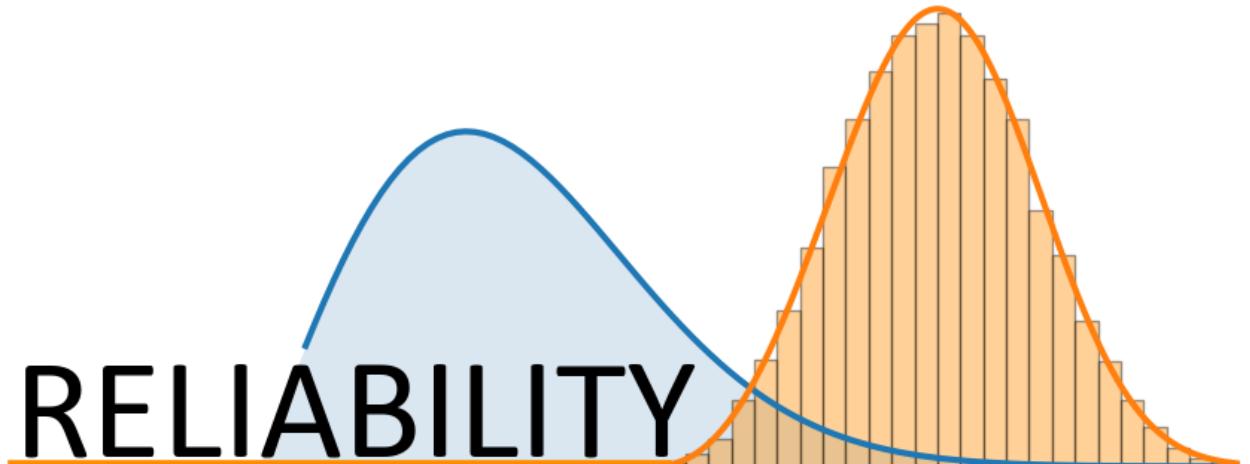
```

from reliability.Nonparametric import RankAdjustment
import matplotlib.pyplot as plt

f = [5248, 7454, 16890, 17200, 38700, 45000, 49390, 69040, 72280, 131900]
rc = [3961, 4007, 4734, 6054, 7298, 10190, 23060, 27160, 28690, 37100, 40060, 45670, 53000, 67000, 69630, 77350, 78470, 91680, 105700, 106300, 150400]
a_trials = [0, 0.3, 1]
for a in a_trials:
    RankAdjustment(failures=f, right_censored=rc, print_results=False, a=a, label=str(a))
plt.legend(title="Heuristic 'a'")
plt.title('Effect of rank adjustment heuristic')
plt.show()

```





RELIABILITY
A Python library for reliability engineering

WHAT IS ACCELERATED LIFE TESTING

Accelerated life testing (ALT) is a method of test and analysis to determine how failures would likely occur in the future. ALT is a popular method of testing because of its ability to “speed up time”. ALT is often used when we can not afford to wait for failures to occur at their normal rate but we need to know how failures are likely to occur in the future.

Consider an electronics manufacturer who wants to know how many failures will occur in 10 years (possibly for warranty purposes). If the component being tested has a mean life of 30 years, the manufacturer cannot reasonably spend several years performing a reliability test as they are ready to release their product on the market soon. By increasing the stress on the component, failure will be induced more rapidly. Done correctly, this is equivalent to speeding up the passage of time. The electronics manufacturer can collect failure data at a variety of stresses, fit the appropriate life-stress model, and then enter the “use stress” into the life-stress model to determine the failure distribution that is expected to occur at the use stress.

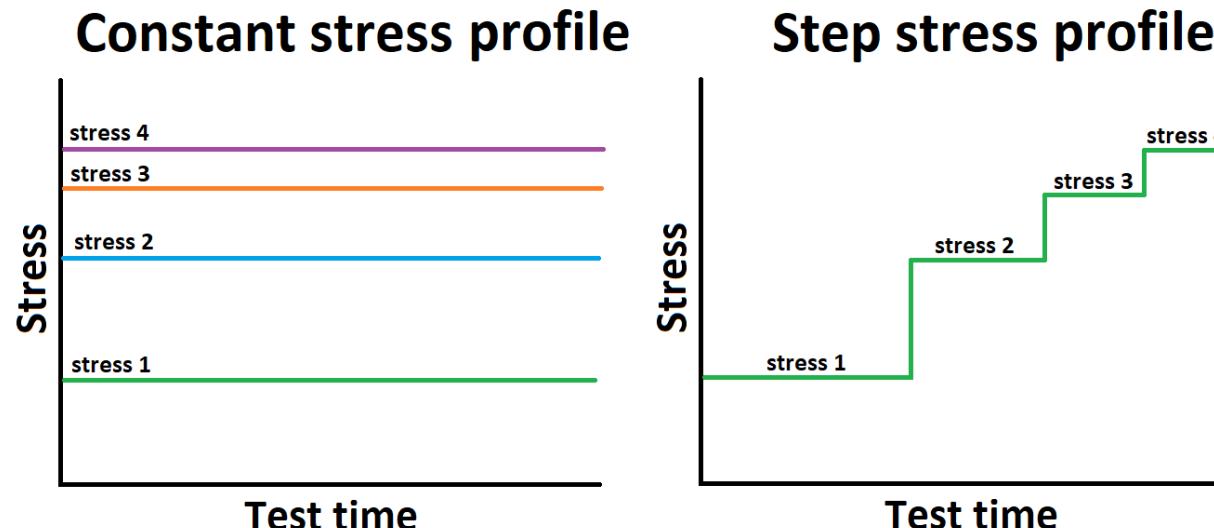
ALT testing is also a very useful way to determine the effectiveness of derating. Derating is the process of reducing the load (typically voltage or current) on a component below its “rated” load, or equivalently selecting a component that is rated above the design load. How much the component life will be extended can be quantitatively measured using an ALT test to find the life-stress model.

To ensure the ALT test is performed correctly, the analyst must ensure that the failure modes are the same at each stress. This will be evidenced by the shape parameter of the distribution as a changing shape parameter will show the failure mode is changing, though it is desirable that each failed component be examined to ensure that the failure mode being studied was the failure mode experienced. As with any model fitting the analyst must ensure there is sufficient data to fit the model such that the results are meaningful. This means the ALT test needs sufficient stresses (usually 3 or more) and sufficient failures (as many as you can afford to test) at each stress.

ALT tests may either be single stress or dual stress. In dual stress models, there are two stresses being tested, such as temperature and humidity. The testing process is largely the same though users should note that with an additional variable in the model it is highly desirable to have more failure data to fit the model accurately. Additionally, it is important that both stresses are varied sufficiently (relative to the design load) so that the life-stress curve (or life-stress surface in the case of dual stress models) has enough data over enough range to be fitted accurately.

19.1 Types of ALT

The way an ALT test is performed depends on the stress profile. There are two popular methods to perform an ALT test; using a constant stress profile, and using a step stress profile. In a constant stress profile, each item under test only ever experiences a single stress level. In a step stress profile each item begins at the lowest stress which is held for a period of time before being stepped up to higher and higher levels. Constant stress profiles are mathematically easier to fit and understand and therefore are more popular. Step stress profiles are useful when you only have a limited number of items and you do not know at what stress you should test them. Selecting a stress that is too low may result in no failures so the opportunity to use the same components (which have not yet failed) from the first test in subsequent tests at higher levels is advantageous.



Within *reliability* there are 24 constant stress ALT models currently implemented (12 single stress and 12 dual stress). Step stress models are not yet implemented within *reliability* though this feature is planned for a future release. Users seeking to fit a step stress profile may want to consider using Reliasoft's [ALTA](#).

The mathematical formulation of ALT models is explained further in the section on [Equations of ALT models](#).

19.2 ALT vs HALT vs ESS vs Burn-in

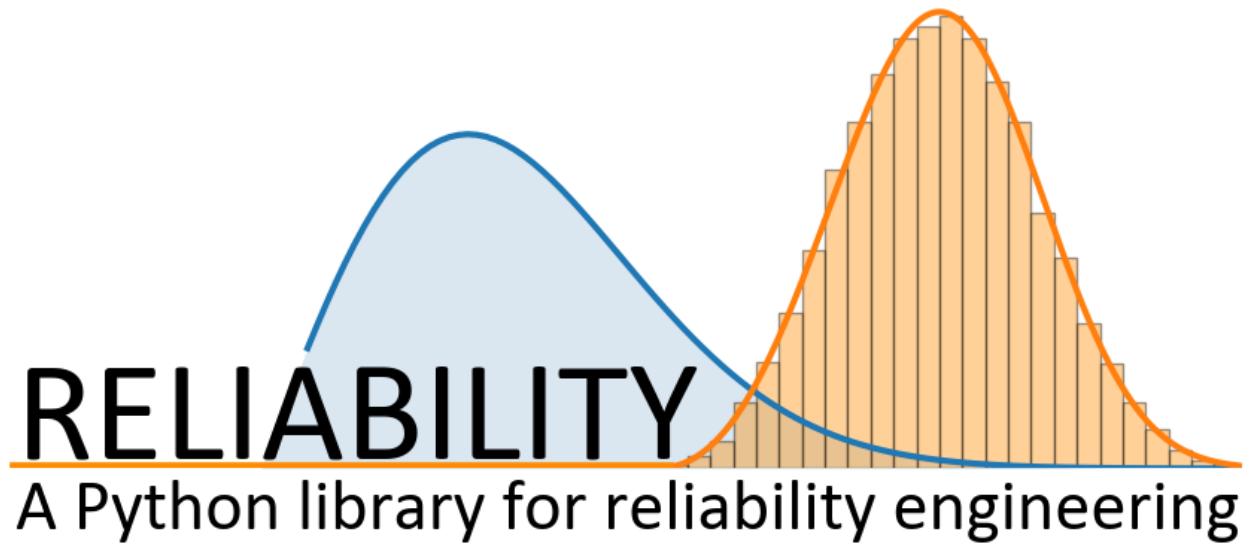
Highly Accelerated Life Testing (HALT) is a type of testing to determine how things fail, rather than when things will fail. HALT has no survivors as the primary goal is to record the way in which items fail (their failure mode) so that design improvements can be made to make the design more resistant to those failure modes. HALT is mostly qualitative while ALT is quantitative. Since HALT is qualitative, there are no models required for fitting failure data.

Environmental Stress Screening (ESS) is a process of exposing components to a series of stresses which they are likely to experience throughout their lifetime such as rapid thermal cycling, vibration, and shock loads. These stresses precipitate latent manufacturing defects as early failures. ESS is often confused with burn-in since both are a screening process to remove weak items from a batch, effectively removing the infant mortality failures from the customer's experience. Unlike burn-in, ESS uses a range of loads, more than just thermal and voltage as is seen in burn-in. ESS does not simulate the component's design environment or usage profile, though it should use a range of stresses (or combinations of stresses) which are on the upper or lower limits of the component's design limit. It is important that the applied stress does not approach the mechanical, electrical, or thermal stress limits of any component as ESS is not intended to cause damage or fatigue. Ideally, components that pass ESS will not have had any of their life consumed during the ESS process. Each screening profile must be tailored specifically for the component/product on which it is applied.

Burn-in involves stressing components with a higher load than their design load such that the "weak" items are screened out through failure. Often confused with ESS, burn-in can be thought of as a subset of ESS with a focus on thermal or electrical loads generally used for screening electrical components. The strength of a population of components will always have some variability (that can be modeled using a probability distribution). By "burning-in" the population of components, manufacturers can screen out (through failure) the lower part of the distribution (of strengths) to be left with only the stronger components from the batch. Burn-in is only appropriate for stresses which cause wear-in or random failure modes (not wear out failure modes which accumulate damage). If the stress causes cumulative damage then the burn-in process would consume some of the component's life. MIL-STD-883C defines a burn-in test as *Burn-in is a test performed for the purpose of screening or eliminating marginal devices, those with inherent defects or defects resulting from manufacturing aberrations which cause time and stress dependent failures.*

References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 136-168
- Accelerated Life Testing Data Analysis Reference - ReliaWiki, Reliawiki.com, 2019. [Online].



EQUATIONS OF ALT MODELS

20.1 Constructing an ALT model

ALT models are probability distributions with a stress dependent model replacing their scale (or rate) parameter. For example, the Weibull-Exponential model is obtained by replacing the α parameter with the equation for the exponential life-stress model as follows:

Weibull PDF:
$$f(t) = \frac{\beta t^{\beta-1}}{\alpha^\beta} \cdot \exp\left(-\left(\frac{t}{\alpha}\right)^\beta\right)$$

Exponential Life-Stress model:
$$L(S) = b \cdot \exp\left(\frac{a}{S}\right)$$

Replacing α with $L(S)$ gives the PDF of the Weibull-Exponential model:

Weibull-Exponential:
$$f(t, S) = \frac{\beta t^{\beta-1}}{\left(b \cdot \exp\left(\frac{a}{S}\right)\right)^\beta} \cdot \exp\left(-\left(\frac{t}{b \cdot \exp\left(\frac{a}{S}\right)}\right)^\beta\right)$$

By replacing the scale parameter with a stress dependent model, the scale parameter of the distribution can be varied as the stress varies. The shape parameter (β in the above example) is kept constant. On a probability plot (which is scaled appropriately such that the distribution appears as a straight line), the process of changing the scale parameter has the effect of moving the line to the left or right.

ALT models can use any probability distribution which does not scale the axes based on the shape or scale parameters. The Gamma and Beta distributions do scale their axes based on their parameters (which is why you'll never find Gamma or Beta probability paper) so these probability distributions could not be used for ALT models. Within *reliability* the Weibull_2P, Exponential_1P, Lognormal_2P, and Normal_2P [distributions](#) are used.

In the above example we saw that α was replaced with the life model $L(S)$. A direct substitution is not always the case. The correct substitutions for each of the four models used in *reliability* are as follows:

Weibull:
$$\alpha = L(S)$$

Normal:
$$\mu = L(S)$$

Lognormal:
$$\mu = \ln(L(S))$$

Exponential:
$$\lambda = \frac{1}{L(S)}$$

The life-stress models available within *reliability* are:

Exponential:
$$L(S) = b \cdot \exp\left(\frac{a}{S}\right)$$

(also known as Arrhenius) limits: $(-\infty < a < \infty)$, $(b > 0)$

Eyring:
$$L(S) = \frac{1}{S} \cdot \exp\left(-\left(c - \frac{a}{S}\right)\right)$$

limits: $(-\infty < a < \infty)$, $(-\infty < c < \infty)$

Power:
$$L(S) = a \cdot S^n$$

(also known as Inverse Power Law) limits: $(a > 0)$, $(-\infty < n < \infty)$

Dual-Exponential:

$$L(S_1, S_2) = c \cdot \exp \left(\frac{a}{S_1} + \frac{b}{S_2} \right)$$

(also known as Temperature-Humidity) limits: $(-\infty < a < \infty)$, $(-\infty < b < \infty)$, $(c > 0)$

Dual-Power:

$$L(S_1, S_2) = c \cdot S_1^m \cdot S_2^n$$

limits: $(c > 0)$, $(-\infty < m < \infty)$, $(-\infty < n < \infty)$

Power-Exponential:

$$L(S_1, S_2) = c \cdot \exp \left(\frac{a}{S_1} \right) \cdot S_2^n$$

(also known as Thermal-Nonthermal) limits: $(-\infty < a < \infty)$, $(c > 0)$, $(-\infty < n < \infty)$

Note that while this last model is named “Power-Exponential” (keeping in line with academic literature), it would be more appropriate to call it the Exponential-Power model since the stresses are modelled in the “Thermal-Nonthermal” stress order. This means that the first stress (S_1) is modelled by the Exponential model (typically used for thermal stresses) and the second stress (S_2) is modelled by the Power model (typically used for nonthermal stresses). The model may perform quite differently if given S_1 and S_2 in the opposite order.

Since each ALT model is a combination of a life model (Weibull, Exponential, Lognormal, Normal) and a life-stress model (Exponential, Eyring, Power, Dual-Exponential, Dual-Power, Power-Exponential), there are 24 possible models (12 for single stress and 12 for dual stress).

20.2 Weibull ALT models

Weibull-Exponential:

$$f(t, S) = \frac{\beta t^{\beta-1}}{\left(b \cdot \exp \left(\frac{a}{S} \right) \right)^\beta} \cdot \exp \left(- \left(\frac{t}{b \cdot \exp \left(\frac{a}{S} \right)} \right)^\beta \right)$$

Weibull-Eyring:

$$f(t, S) = \frac{\beta t^{\beta-1}}{\left(\frac{1}{S} \cdot \exp \left(- \left(c - \frac{a}{S} \right) \right) \right)^\beta} \cdot \exp \left(- \left(\frac{t}{\frac{1}{S} \cdot \exp \left(- \left(c - \frac{a}{S} \right) \right)} \right)^\beta \right)$$

Weibull-Power:

$$f(t, S) = \frac{\beta t^{\beta-1}}{\left(a \cdot S^n \right)^\beta} \cdot \exp \left(- \left(\frac{t}{a \cdot S^n} \right)^\beta \right)$$

Weibull-Dual-Exponential:

$$f(t, S_1, S_2) = \frac{\beta t^{\beta-1}}{\left(c \cdot \exp \left(\frac{a}{S_1} + \frac{b}{S_2} \right) \right)^\beta} \cdot \exp \left(- \left(\frac{t}{c \cdot \exp \left(\frac{a}{S_1} + \frac{b}{S_2} \right)} \right)^\beta \right)$$

Weibull-Dual-Power:

$$f(t, S_1, S_2) = \frac{\beta t^{\beta-1}}{\left(c \cdot S_1^m \cdot S_2^n \right)^\beta} \cdot \exp \left(- \left(\frac{t}{c \cdot S_1^m \cdot S_2^n} \right)^\beta \right)$$

Weibull-Power-Exponential:

$$f(t, S_1, S_2) = \frac{\beta t^{\beta-1}}{\left(c \cdot \exp \left(\frac{a}{S_1} \right) \cdot S_2^n \right)^\beta} \cdot \exp \left(- \left(\frac{t}{c \cdot \exp \left(\frac{a}{S_1} \right) \cdot S_2^n} \right)^\beta \right)$$

20.3 Lognormal ALT models

Lognormal-Exponential:

$$f(t, S) = \frac{1}{\sigma t \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{\ln(t) - \ln(b \cdot \exp(\frac{a}{S}))}{\sigma} \right)^2 \right)$$

Lognormal-Eyring:

$$f(t, S) = \frac{1}{\sigma t \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{\ln(t) - \ln(\frac{1}{S} \cdot \exp(-\left(c - \frac{a}{S} \right)))}{\sigma} \right)^2 \right)$$

Lognormal-Power:

$$f(t, S) = \frac{1}{\sigma t \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{\ln(t) - \ln(a \cdot S^n)}{\sigma} \right)^2 \right)$$

Lognormal-Dual-Exponential:

$$f(t, S_1, S_2) = \frac{1}{\sigma t \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{\ln(t) - \ln(c \cdot \exp(\frac{a}{S_1} + \frac{b}{S_2}))}{\sigma} \right)^2 \right)$$

Lognormal-Dual-Power:
$$f(t, S_1, S_2) = \frac{1}{\sigma t \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{\ln(t) - \ln(c \cdot S_1^m \cdot S_2^n)}{\sigma} \right)^2 \right)$$

Lognormal-Power-Exponential:
$$f(t, S_1, S_2) = \frac{1}{\sigma t \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{\ln(t) - \ln(c \cdot S_2^n \cdot \exp(\frac{a}{S_1}))}{\sigma} \right)^2 \right)$$

20.4 Normal ALT models

Normal-Exponential:
$$f(t, S) = \frac{1}{\sigma \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{t - b \cdot \exp(\frac{a}{S})}{\sigma} \right)^2 \right)$$

Normal-Eyring:
$$f(t, S) = \frac{1}{\sigma \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{t - \frac{1}{S} \cdot \exp(-c - \frac{a}{S})}{\sigma} \right)^2 \right)$$

Normal-Power:
$$f(t, S) = \frac{1}{\sigma \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{t - a \cdot S^n}{\sigma} \right)^2 \right)$$

Normal-Dual-Exponential:
$$f(t, S_1, S_2) = \frac{1}{\sigma \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{t - c \cdot \exp(\frac{a}{S_1} + \frac{b}{S_2})}{\sigma} \right)^2 \right)$$

Normal-Dual-Power:
$$f(t, S_1, S_2) = \frac{1}{\sigma \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{t - c \cdot S_1^m \cdot S_2^n}{\sigma} \right)^2 \right)$$

Normal-Power-Exponential:
$$f(t, S_1, S_2) = \frac{1}{\sigma \sqrt{2\pi}} \cdot \exp \left(-\frac{1}{2} \left(\frac{t - c \cdot S_2^n \cdot \exp(\frac{a}{S_1})}{\sigma} \right)^2 \right)$$

20.5 Exponential ALT models

Exponential-Exponential:
$$f(t, S) = b \cdot \exp(\frac{a}{S}) \cdot \exp \left(\frac{-t}{b \cdot \exp(\frac{a}{S})} \right)$$

Exponential-Eyring:
$$f(t, S) = \frac{1}{S} \cdot \exp \left(- \left(c - \frac{a}{S} \right) \right) \cdot \exp \left(\frac{-t}{\frac{1}{S} \cdot \exp \left(- \left(c - \frac{a}{S} \right) \right)} \right)$$

Exponential-Power:
$$f(t, S) = a \cdot S^n \cdot \exp \left(\frac{-t}{a \cdot S^n} \right)$$

Exponential-Dual-Exponential:
$$f(t, S_1, S_2) = c \cdot \exp \left(\frac{a}{S_1} + \frac{b}{S_2} \right) \cdot \exp \left(\frac{-t}{c \cdot \exp \left(\frac{a}{S_1} + \frac{b}{S_2} \right)} \right)$$

Exponential-Dual-Power:
$$f(t, S_1, S_2) = c \cdot S_1^m \cdot S_2^n \cdot \exp \left(\frac{-t}{c \cdot S_1^m \cdot S_2^n} \right)$$

Exponential-Power-Exponential:
$$f(t, S_1, S_2) = c \cdot S_2^n \cdot \exp \left(\frac{a}{S_1} \right) \cdot \exp \left(\frac{-t}{c \cdot S_2^n \cdot \exp \left(\frac{a}{S_1} \right)} \right)$$

20.6 Acceleration factor

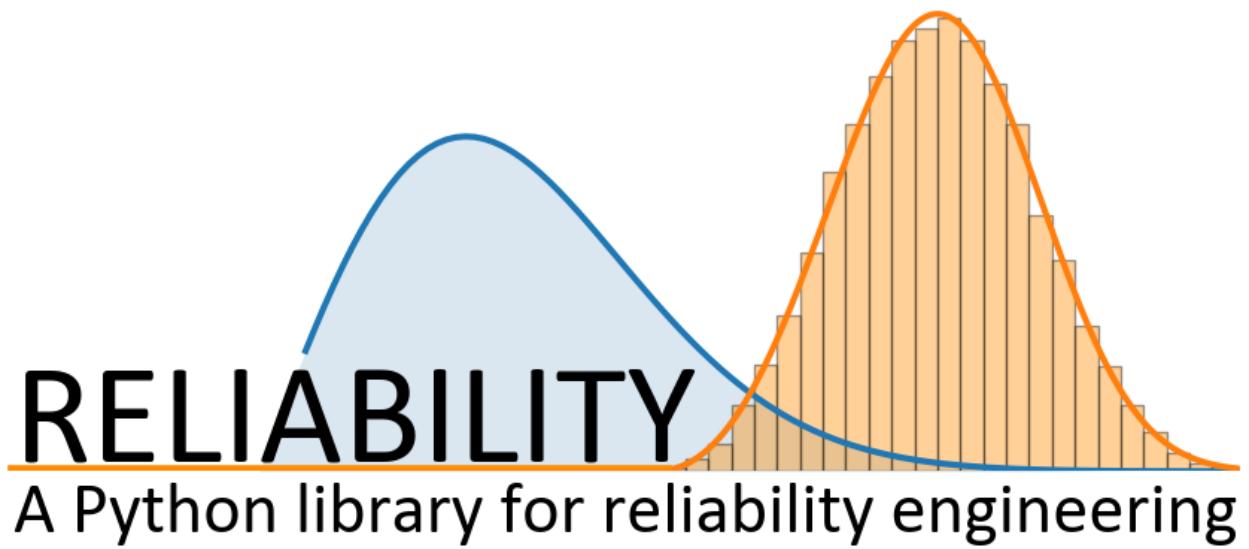
The acceleration factor is a value used to show by how much the life is being accelerated. The acceleration factor is given by the equation:

$$AF = \frac{L_{USE}}{L_{ACCELERATED}}$$

References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 136-168

- Accelerated Life Testing Data Analysis Reference - ReliaWiki, Reliawiki.com, 2019. [Online].



CHAPTER
TWENTYONE

GETTING YOUR ALT DATA IN THE RIGHT FORMAT

Because the ALT probability models failures and right censored data from many stress levels, it was not practical to make an input for each stress level. Instead, the failure times are combined in a single input and the failure_stress input provides a list of the corresponding stresses at which each failure occurred. The same is true of the right_censored and right_censored_stress inputs.

To get your data in the correct format, ensure you have combined all your failure times into a single list or numpy array and there is a corresponding list or array of the same length that provides all of the stresses. The following example illustrates one method to do this if you do not have the list already imported from Excel or another source. This is done for failures only but if you have right_censored data then you would do the same thing, but keep it separate to the failure data. There is no need to sort the data in any particular order as this is all done automatically. The only requirement is that the length of failures matches the length of the failure_stress, and that there are no new stresses in right_censored_stress that are not present in failure_stress.

```
import numpy as np
#create the data
failure_times_at_stress_1 = [800,850,910,940]
failure_stress_1 = [40,40,40,40]
failure_times_at_stress_2 = [650,670,715,740]
failure_stress_2 = [50,50,50,50]
failure_times_at_stress_3 = [300,320,350,380]
failure_stress_3 = [60,60,60,60]
# combine the data
failures = np.hstack([failure_times_at_stress_1,failure_times_at_stress_2,failure_times_
-at_stress_3])
failure_stresses = np.hstack([failure_stress_1,failure_stress_2,failure_stress_3])
# print for inspection
print(failures)
print(failure_stresses)

"""
[800 850 910 940 650 670 715 740 300 320 350 380]
[40 40 40 40 50 50 50 50 60 60 60 60]
""
```



RELIABILITY
A Python library for reliability engineering

CHAPTER
TWENTYTWO

FITTING A SINGLE STRESS MODEL TO ALT DATA

Before reading this section it is recommended that readers are familiar with the concepts of [fitting probability distributions](#), [probability plotting](#), and have an understanding of [what accelerated life testing \(ALT\) involves](#).

The module `reliability. ALT_fitters` contains 24 [ALT models](#); 12 of these models are for single stress and 12 are for dual stress. This section details the single stress models, though the process for [fitting dual-stress models](#) is similar. The decision to use a single stress or dual stress model depends entirely on your data. If your data only has one stress that is being changed then you will use a single stress model.

The following single stress models are available within `ALT_fitters`:

- `Fit_Weibull_Exponential`
- `Fit_Weibull_Eyring`
- `Fit_Weibull_Power`
- `Fit_Lognormal_Exponential`
- `Fit_Lognormal_Eyring`
- `Fit_Lognormal_Power`
- `Fit_Normal_Exponential`
- `Fit_Normal_Eyring`
- `Fit_Normal_Power`
- `Fit_Exponential_Exponential`
- `Fit_Exponential_Eyring`
- `Fit_Exponential_Power`

 **API Reference**

For inputs and outputs see the [API reference](#).

22.1 Example 1

In the following example, we will fit the Weibull-Power model to an ALT dataset obtained from a fatigue test. This dataset can be found in `reliability.Datasets`. We want to know the mean life at the use level stress of 60 so the parameter `use_level_stress` is specified. All other values are left as defaults and the results and plot are shown.

```

from reliability. ALT_fitters import Fit_Weibull_Power
from reliability. Datasets import ALT_load2
import matplotlib.pyplot as plt

Fit_Weibull_Power(failures=ALT_load2().failures, failure_stress=ALT_load2().failure_
stresses, right_censored=ALT_load2().right_censored, right_censored_stress=ALT_load2().
right_censored_stresses, use_level_stress=60)
plt.show()

"""

Results from Fit_Weibull_Power (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 13/5 (27.77778% right censored)

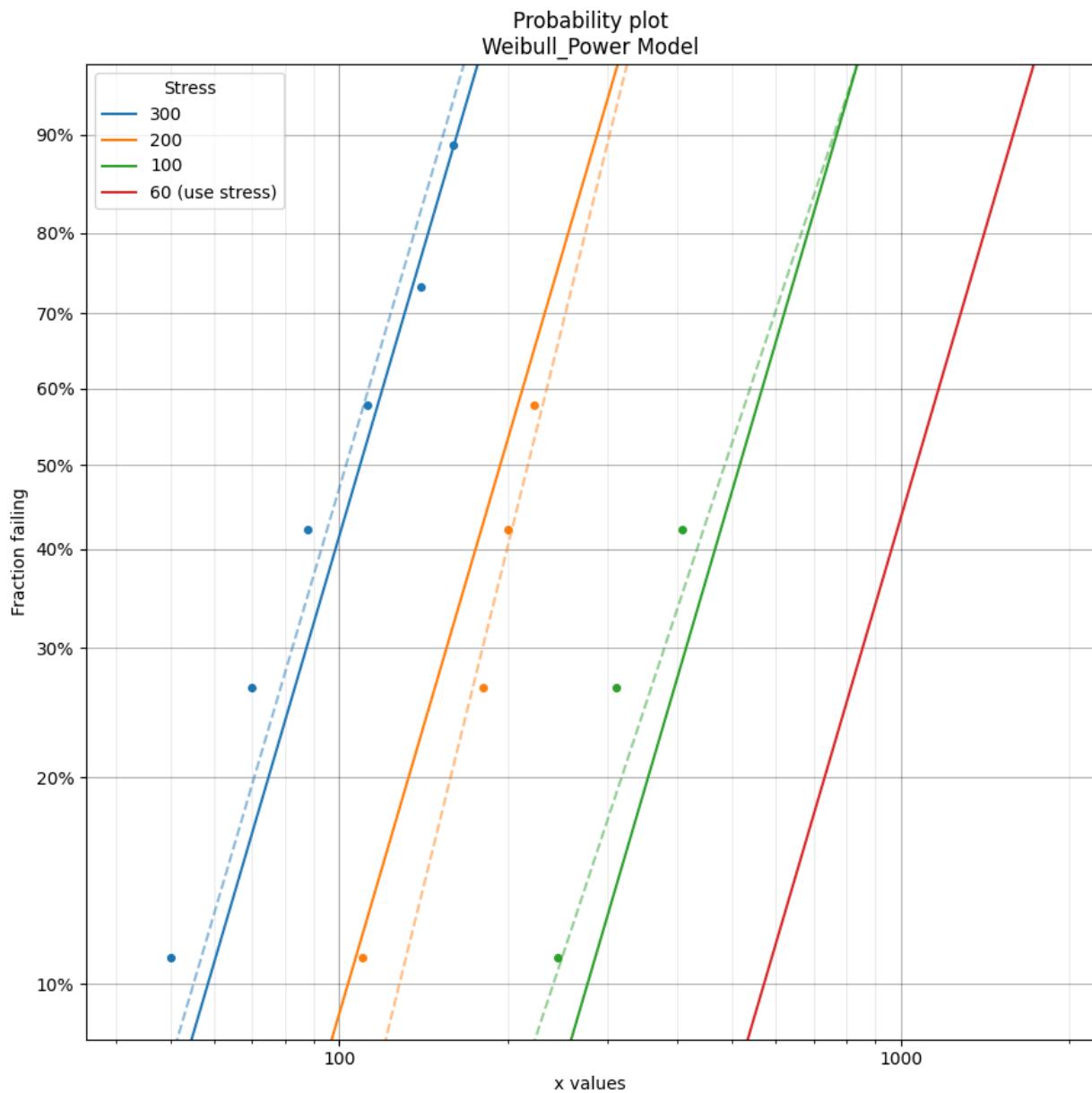
Parameter Point Estimate Standard Error Lower CI Upper CI
a 393440 508989 31166.6 4.9667e+06
n -1.41476 0.242371 -1.8898 -0.939725
beta 3.01934 0.716268 1.89664 4.80662

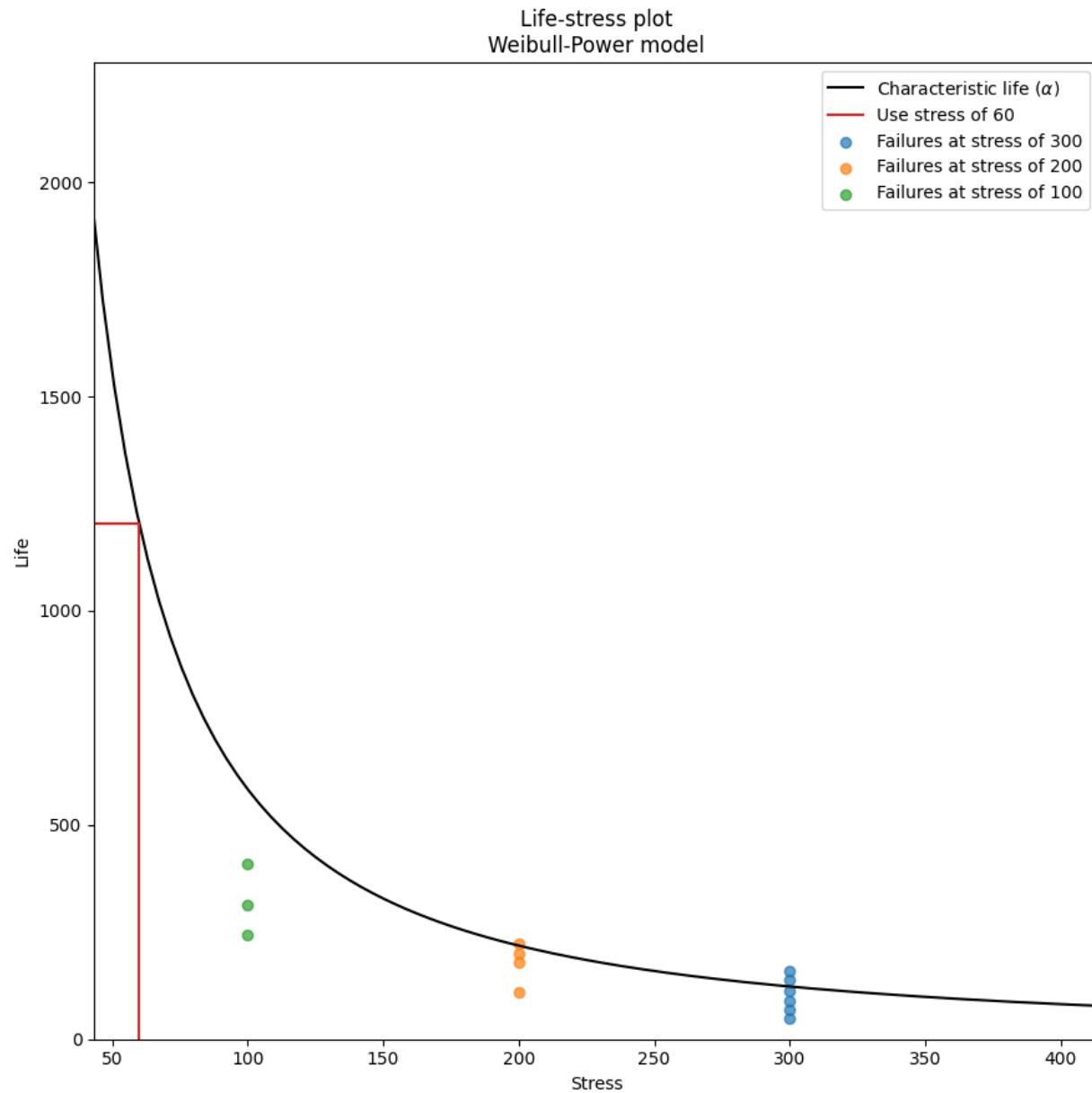
stress original alpha original beta new alpha common beta beta change acceleration_
factor
300 116.174 3.01009 123.123 3.01934 +0.31%
9.74714
200 240.182 3.57635 218.507 3.01934 -15.57%
5.49224
100 557.42 2.6792 582.575 3.01934 +12.7%
2.05998

Goodness of fit Value
Log-likelihood -76.8542
AICc 161.423
BIC 162.379

At the use level stress of 60, the mean life is 1071.96438
"""

```





In the results above we see 3 tables of results; the fitted parameters (along with their confidence bounds) dataframe, the change of parameters dataframe, and the goodness of fit dataframe. For the change of parameters dataframe the “original alpha” and “original beta” are the fitted values for the Weibull_2P distribution that is fitted to the data at each stress (shown on the probability plot by the dashed lines). The “new alpha” and “new beta” are from the Weibull_Power model. The beta change is extremely important as it allows us to identify whether the fitted ALT model is appropriate at each stress level. A beta change of over 50% will trigger a warning to be printed informing the user that the failure mode may be changing across different stresses, or that the model is inappropriate for the data. The acceleration factor column will only be returned if the use level stress is provided since acceleration factor is a comparison of the life at the higher stress vs the use stress.

22.2 Example 2

In this second example we will fit the Exponential-Eyring model. Instead of using an existing dataset we will create our own data using the function `make_ALT_data`. Since the Exponential_1P distribution has only 1 parameter (Lambda), the function fits a Weibull_2P distribution and then compares the change of parameters of the Weibull alpha and beta with the Exponential 1/Lambda (obtained from the life-stress model) and the shape parameter of 1 (since a Weibull distribution with beta=1 is equivalent to the Exponential distribution). This provides similar functionality for examining the change of parameters as we find with the models for all the other distributions (Weibull, Lognormal, and Normal).

The results show that the fitted parameters agree well with the parameters we used to generate the data, as does the mean life at the use stress. This accuracy improves with more data.

```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Exponential_Eyring
import matplotlib.pyplot as plt

use_level_stress = 300
ALT_data = make_ALT_data(distribution='Exponential', life_stress_model='Eyring', a=1500, c=-10, stress_1=[500, 400, 350], number_of_samples=100, fraction_censored=0.2, seed=1, use_level_stress=use_level_stress)
Fit_Exponential_Eyring(failures=ALT_data.failures, failure_stress=ALT_data.failure_stresses, right_censored=ALT_data.right_censored, right_censored_stress=ALT_data.right_censored_stresses, use_level_stress=use_level_stress)
print('The mean life at use stress of the true model is:', ALT_data.mean_life_at_use_stress)
plt.show()

"""

Results from Fit_Exponential_Eyring (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 240/60 (20% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
a 1428.47 178.875 1077.88 1779.06
c -10.2599 0.443394 -11.1289 -9.39085

stress weibull alpha weibull beta new 1/Lambda common shape shape change
acceleration factor
500 1034.22 0.981495 994.473 1 +1.89%
11.1948
400 2149.92 0.877218 2539.17 1 +14.0%
4.38449
350 5251.88 1.07081 4833.32 1 -6.61%
2.30337

Goodness of fit Value
Log-likelihood -2098.01
AICc 4200.06
BIC 4207.42

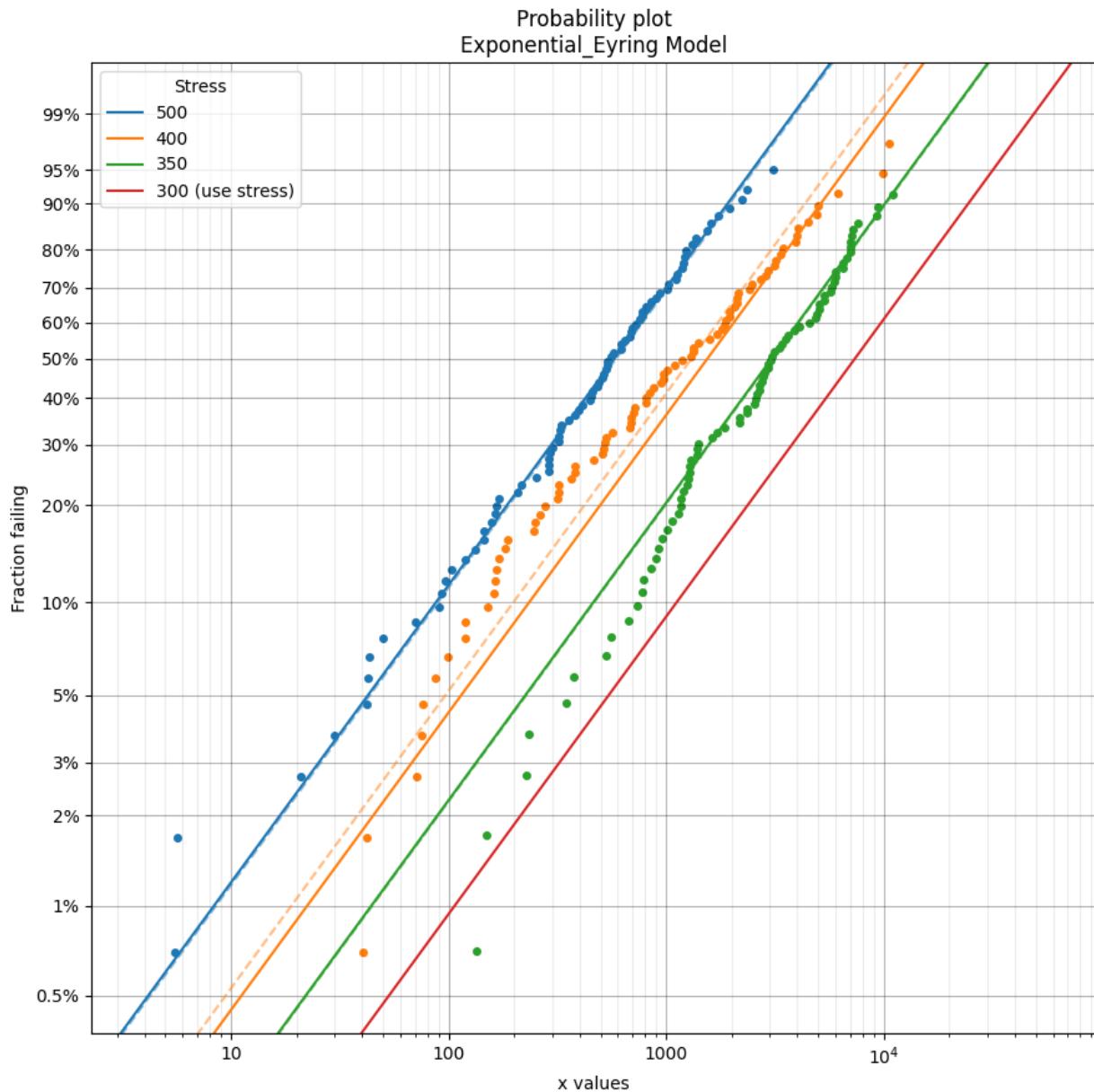
At the use level stress of 300, the mean life is 11132.94095

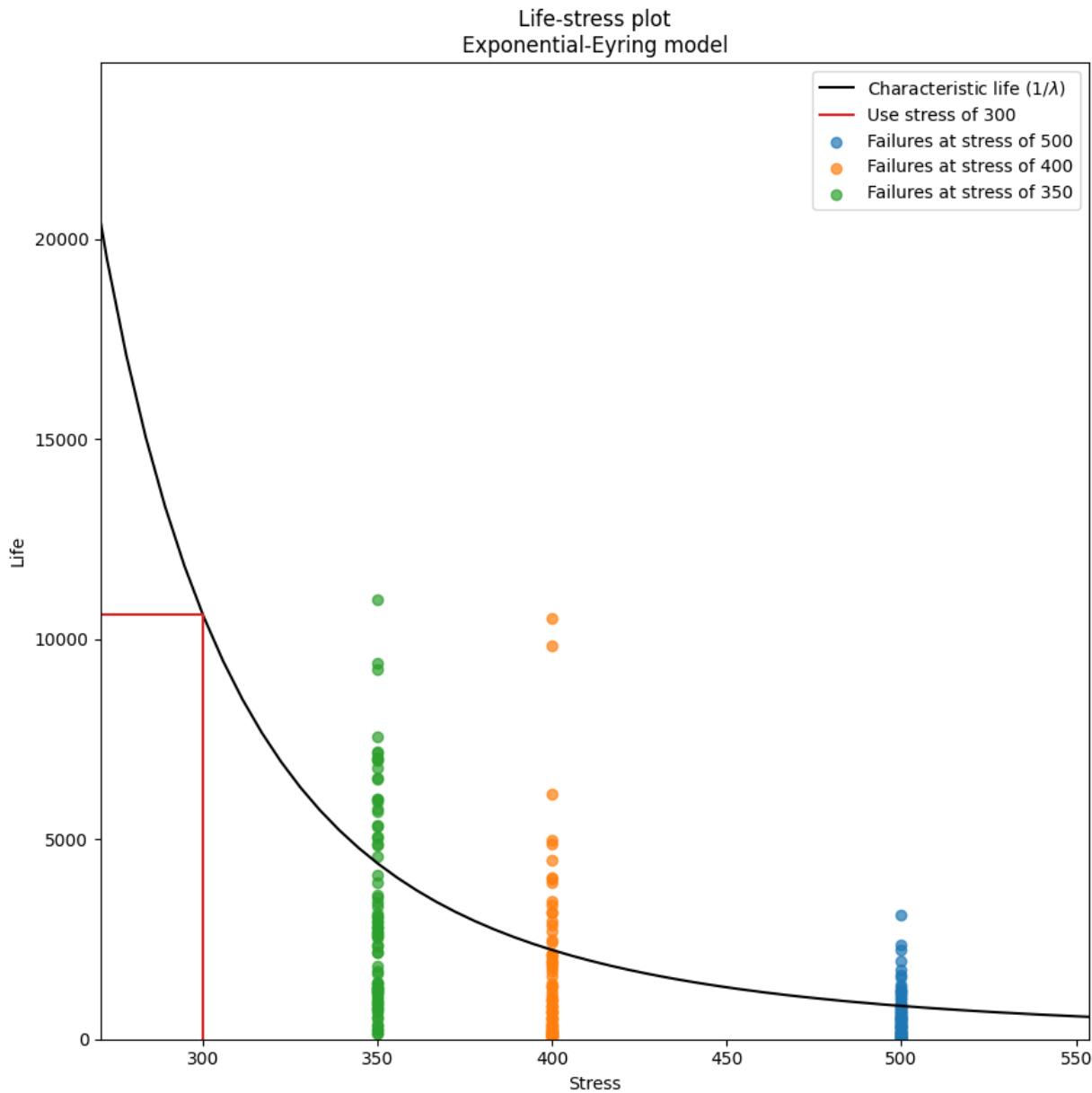
The mean life at use stress of the true model is: 10896.724574907037
```

(continues on next page)

(continued from previous page)

...





22.3 Example 3

In this third example, we will look at how to customise the labels on the plots. Two of the outputs returned are the axes handles for the probability plot and the life-stress plot. These handles can be used to set certain values such as xlabel, ylabel, title, legend title, etc. For simplicity in this example the printing of results and the probability plot are turned off so the only output is the life-stress plot.

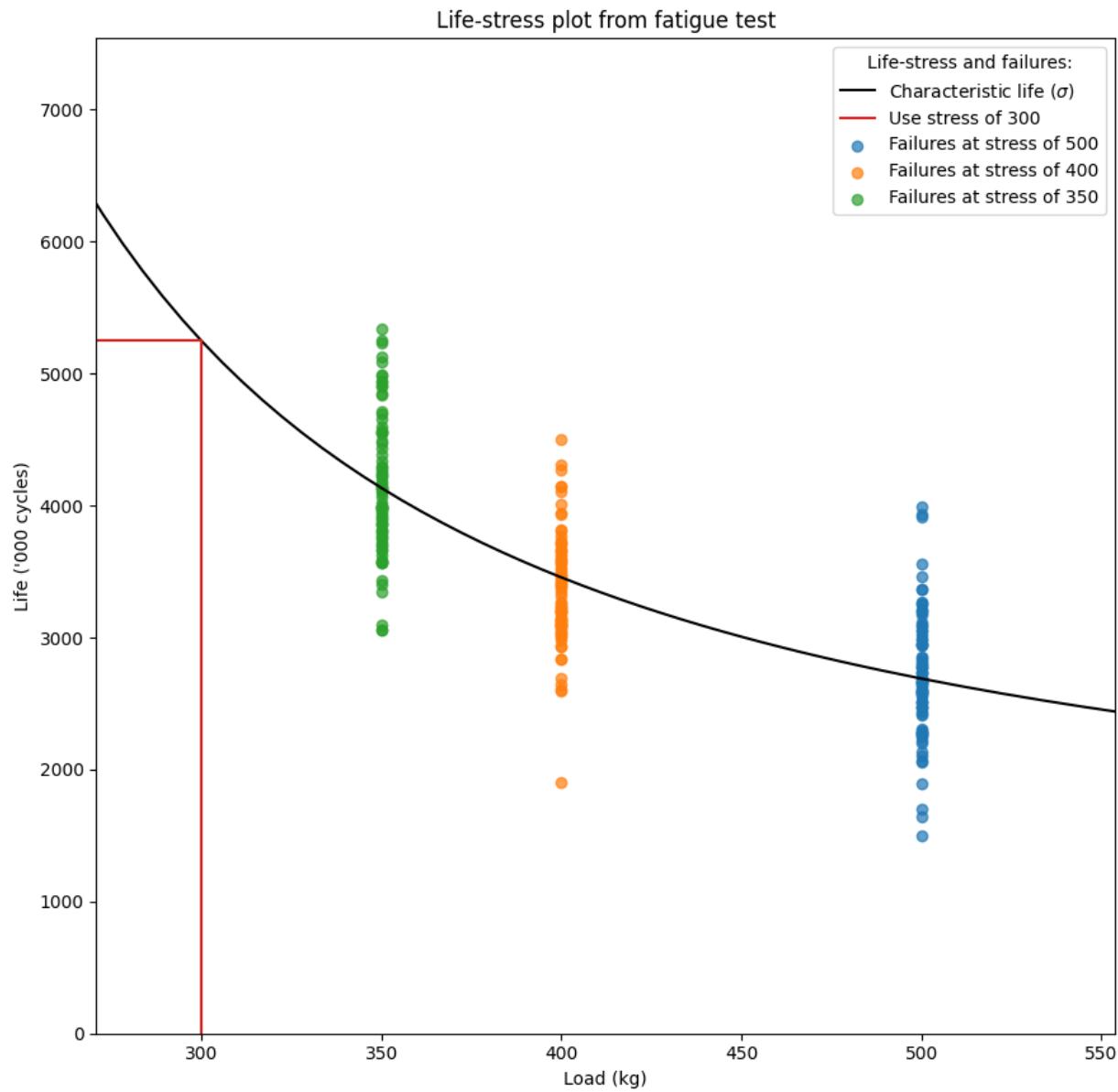
```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Normal_Exponential
import matplotlib.pyplot as plt

ALT_data = make_ALT_data(distribution='Normal', life_stress_model='Exponential', a=500,
```

(continues on next page)

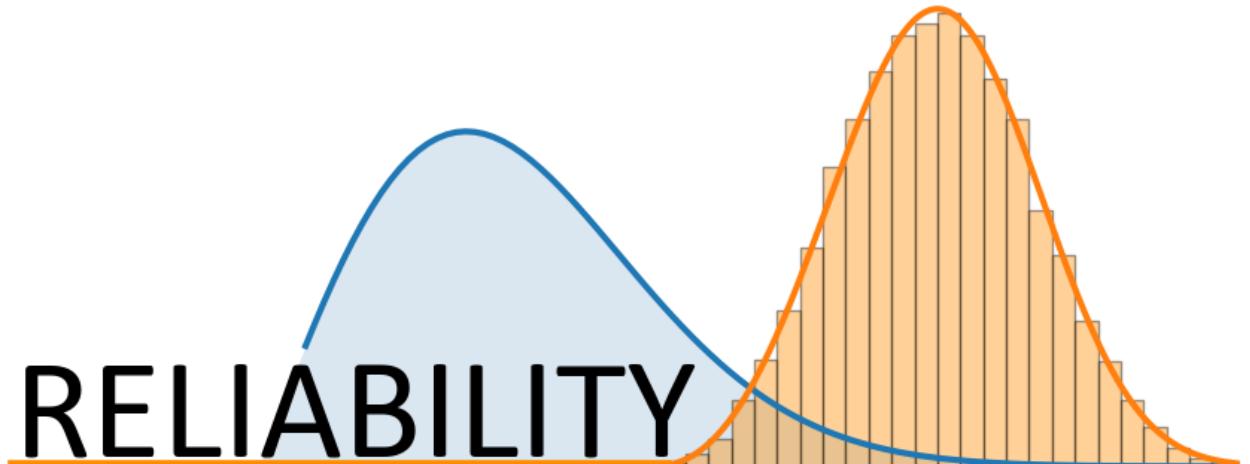
(continued from previous page)

```
↳b=1000,sigma=500,stress_1=[500,400,350],number_of_samples=100,fraction_censored=0.2,
↳seed=1)
# the results and probability plot have been turned off so we just get the life-stress
↳plot
model = Fit_Normal_Exponential(failures=ALT_data.failures, failure_stress=ALT_data.
↳failure_stresses, right_censored=ALT_data.right_censored, right_censored_stress=ALT_
↳data.right_censored_stresses, use_level_stress=300, print_results=False, show_
↳probability_plot=False)
# customize the life-stress plot labels
model.life_stress_plot.set_xlabel('Load (kg)')
model.life_stress_plot.set_ylabel("Life ('000 cycles')")
model.life_stress_plot.set_title('Life-stress plot from fatigue test')
model.life_stress_plot.legend(title='Life-stress and failures:')
plt.show()
```



References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 136-168
- Accelerated Life Testing Data Analysis Reference - ReliaWiki, Reliawiki.com, 2019. [Online].



RELIABILITY
A Python library for reliability engineering

CHAPTER
TWENTYTHREE

FITTING A DUAL STRESS MODEL TO ALT DATA

Before reading this section it is recommended that readers are familiar with the concepts of [fitting probability distributions](#), [probability plotting](#), and have an understanding of [what accelerated life testing \(ALT\) involves](#).

The module `reliability. ALT_fitters` contains 24 [ALT models](#); 12 of these models are for single stress and 12 are for dual stress. This section details the dual stress models, though the process for [fitting single stress models](#) is similar. The decision to use a single stress or dual stress model depends entirely on your data. If your data has two stresses that are being changed then you will use a dual stress model.

The following dual stress models are available within `ALT_fitters`:

- `Fit_Weibull_Dual_Exponential`
- `Fit_Weibull_Power_Exponential`
- `Fit_Weibull_Dual_Power`
- `Fit_Lognormal_Dual_Exponential`
- `Fit_Lognormal_Power_Exponential`
- `Fit_Lognormal_Dual_Power`
- `Fit_Normal_Dual_Exponential`
- `Fit_Normal_Power_Exponential`
- `Fit_Normal_Dual_Power`
- `Fit_Exponential_Dual_Exponential`
- `Fit_Exponential_Power_Exponential`
- `Fit_Exponential_Dual_Power`

API Reference

For inputs and outputs see the [API reference](#).

23.1 Example 1

In the following example, we will fit the Normal-Dual-Exponential model to an ALT dataset obtained from a temperature-voltage dual stress test. This dataset can be found in `reliability.Datasets`. We want to know the mean life at the use level stress of 330 Kelvin, 2.5 Volts so the parameter `use_level_stress` is specified. All other values are left as defaults and the results and plot are shown.

```

from reliability.Datasets import ALT_temperature_voltage
from reliability. ALT_fitters import Fit_Normal_Dual_Exponential
import matplotlib.pyplot as plt
data = ALT_temperature_voltage()
Fit_Normal_Dual_Exponential(failures=data.failures, failure_stress_1=data.failure_stress_
    ↪temp, failure_stress_2=data.failure_stress_voltage, use_level_stress=[330, 2.5])
plt.show()

"""
Results from Fit_Normal_Dual_Exponential (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 12/0 (0% right censored)

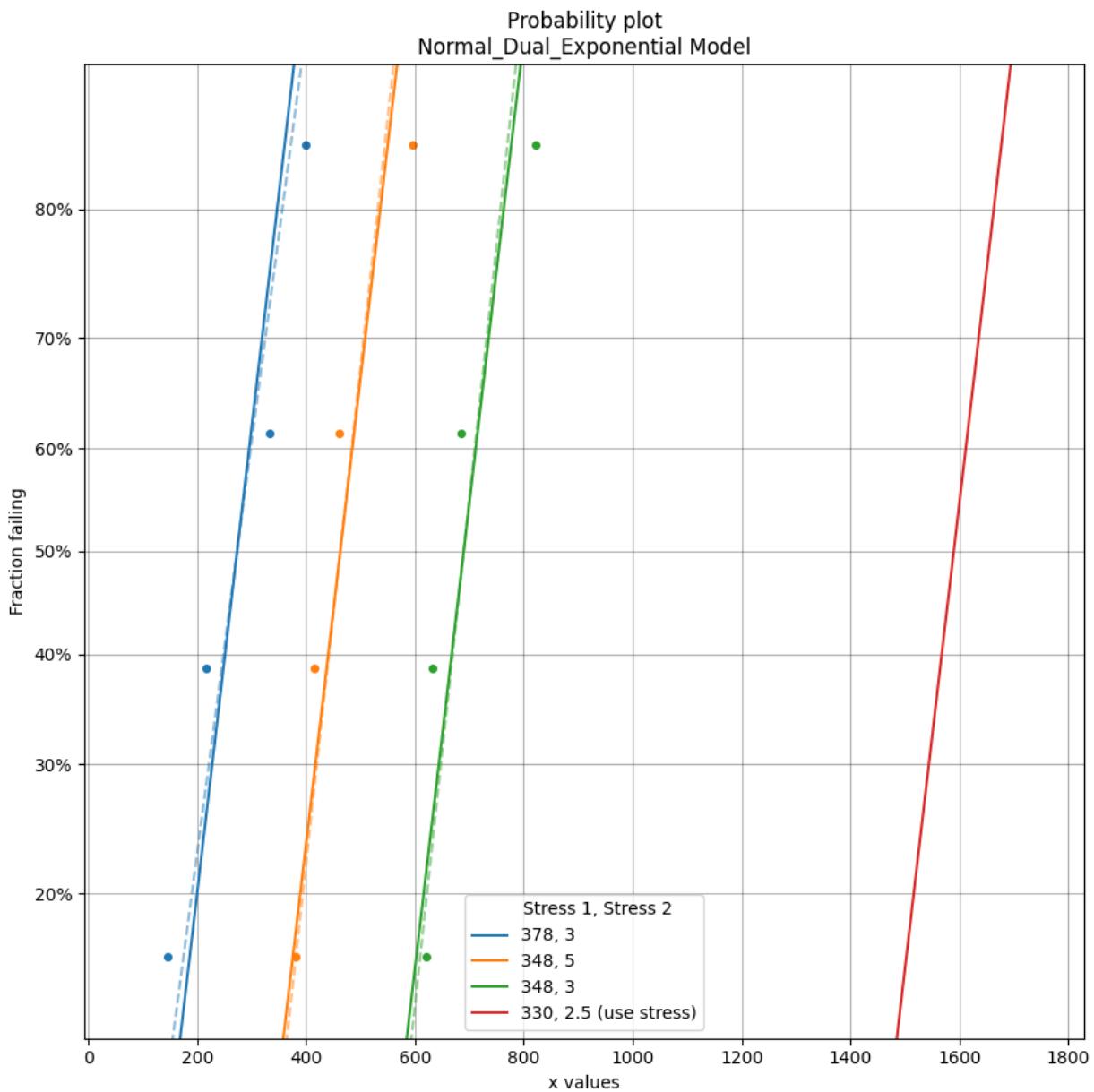
Parameter Point Estimate Standard Error Lower CI Upper CI
    a        4056.06      752.936   2580.33   5531.78
    b        2.98949     0.851782   1.32002   4.65895
    c      0.00220837  0.00488704 2.88663e-05 0.168947
  sigma      87.3192      17.824    58.5274   130.275

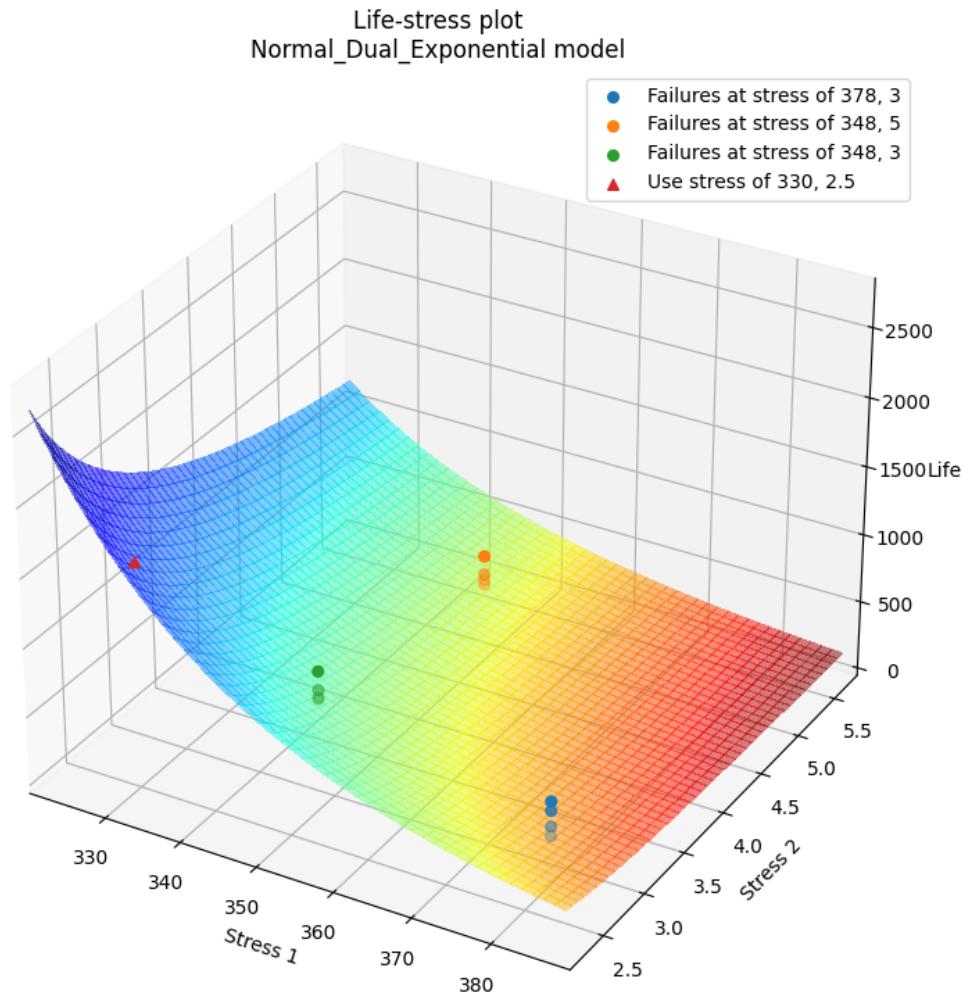
stress original mu original sigma new mu common sigma sigma change acceleration
    ↪factor
378, 3        273.5      98.7256   273.5      87.3192    -11.55%      5.
    ↪81285
348, 5        463       81.8474 463.001      87.3192    +6.69%      3.
    ↪43371
348, 3       689.75      80.176 689.749      87.3192    +8.91%      2.
    ↪30492

Goodness of fit Value
Log-likelihood -70.6621
    AICc 155.039
    BIC 151.264

At the use level stress of 330, 2.5, the mean life is 1589.81428
"""

```





In the results above we see 3 tables of results; the fitted parameters (along with their confidence bounds) data frame, the change of parameters data frame, and the goodness of fit data frame. For the change of parameters data frame the “original mu” and “original sigma” are the fitted values for the Normal_2P distribution that is fitted to the data at each stress (shown on the probability plot by the dashed lines). The “new mu” and “new sigma” are from the Normal_Dual_Exponential model. The sigma change is extremely important as it allows us to identify whether the fitted ALT model is appropriate at each stress level. A sigma change of over 50% will trigger a warning to be printed informing the user that the failure mode may be changing across different stresses, or that the model is inappropriate for the data. The acceleration factor column will only be returned if the use level stress is provided since acceleration factor is a comparison of the life at the higher stress vs the use stress.

23.2 Example 2

In this second example we will fit the Lognormal_Power_Exponential model. Instead of using an existing dataset we will create our own data using the function `make_ALT_data`. The results show that the fitted parameters agree well with the parameters we used to generate the data, as does the mean life at the use stress. This accuracy improves with more data.

Two of the outputs returned are the axes handles for the probability plot and the life-stress plot. These handles can be used to set certain values. In the example below we see the axes labels being set to custom values after the plots have been generated but before the plots have been displayed.

```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Lognormal_Power_Exponential
import matplotlib.pyplot as plt
use_level_stress = [150, 3]
ALT_data = make_ALT_data(distribution='Lognormal', life_stress_model='Power_Exponential',
a=200, c=400, n=-0.5, sigma=0.5, stress_1=[500, 400, 350, 420, 245], stress_2=[12, 8, 6, 9, 10],
number_of_samples=100, fraction_censored=0.5, seed=1, use_level_stress=use_level_stress)
model = Fit_Lognormal_Power_Exponential(failures=ALT_data.failures, failure_stress_1=ALT_
data.failure_stresses_1, failure_stress_2=ALT_data.failure_stresses_2, right_
censored=ALT_data.right_censored, right_censored_stress_1=ALT_data.right_censored_
stresses_1, right_censored_stress_2=ALT_data.right_censored_stresses_2, use_level_
stress=use_level_stress)
# this will change the xlabel on the probability plot
model.probability_plot.set_xlabel('Time (hours)')
# this will change the axes labels on the life-stress plot
model.life_stress_plot.set_xlabel('Temperature (^OK$)')
model.life_stress_plot.set_ylabel('Voltage (kV)')
model.life_stress_plot.set_zlabel('Life (hours)')

print('The mean life at use stress of the true model is:', ALT_data.mean_life_at_use_
stress)
plt.show()

"""

Results from Fit_Lognormal_Power_Exponential (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 250/250 (50% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
a 192.105 39.4889 114.708 269.502
c 451.448 134.274 252.02 808.687
n -0.49196 0.12119 -0.729488 -0.254433
sigma 0.491052 0.0212103 0.451191 0.534433

stress original mu original sigma new mu common sigma sigma change acceleration_
factor
500, 12 5.29465 0.496646 5.2742 0.491052 -1.13% 4.
84765
420, 9 5.54536 0.525041 5.48891 0.491052 -6.47% 3.
91096
400, 8 5.42988 0.392672 5.56972 0.491052 +25.05% 3.
60733
```

(continues on next page)

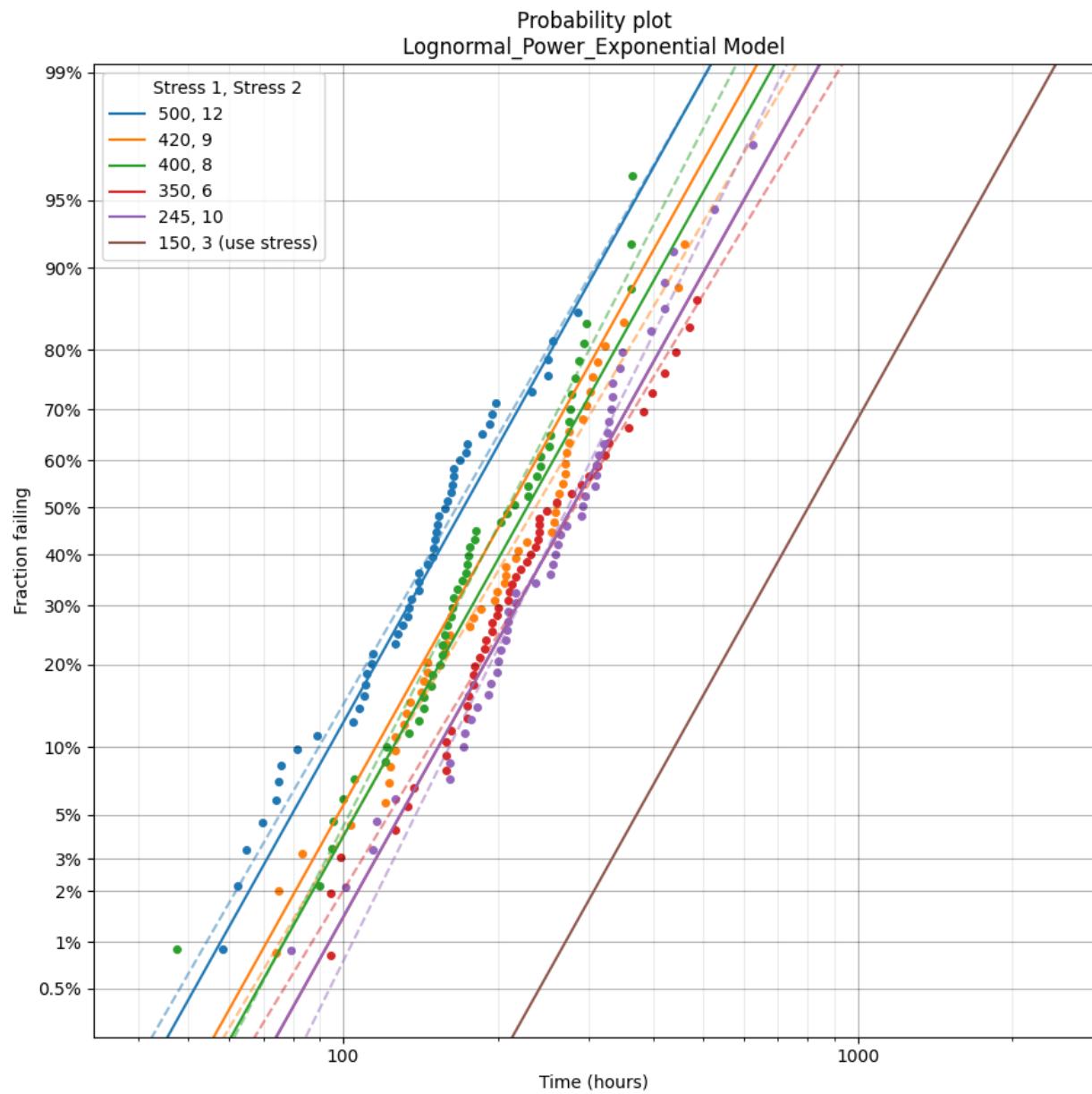
(continued from previous page)

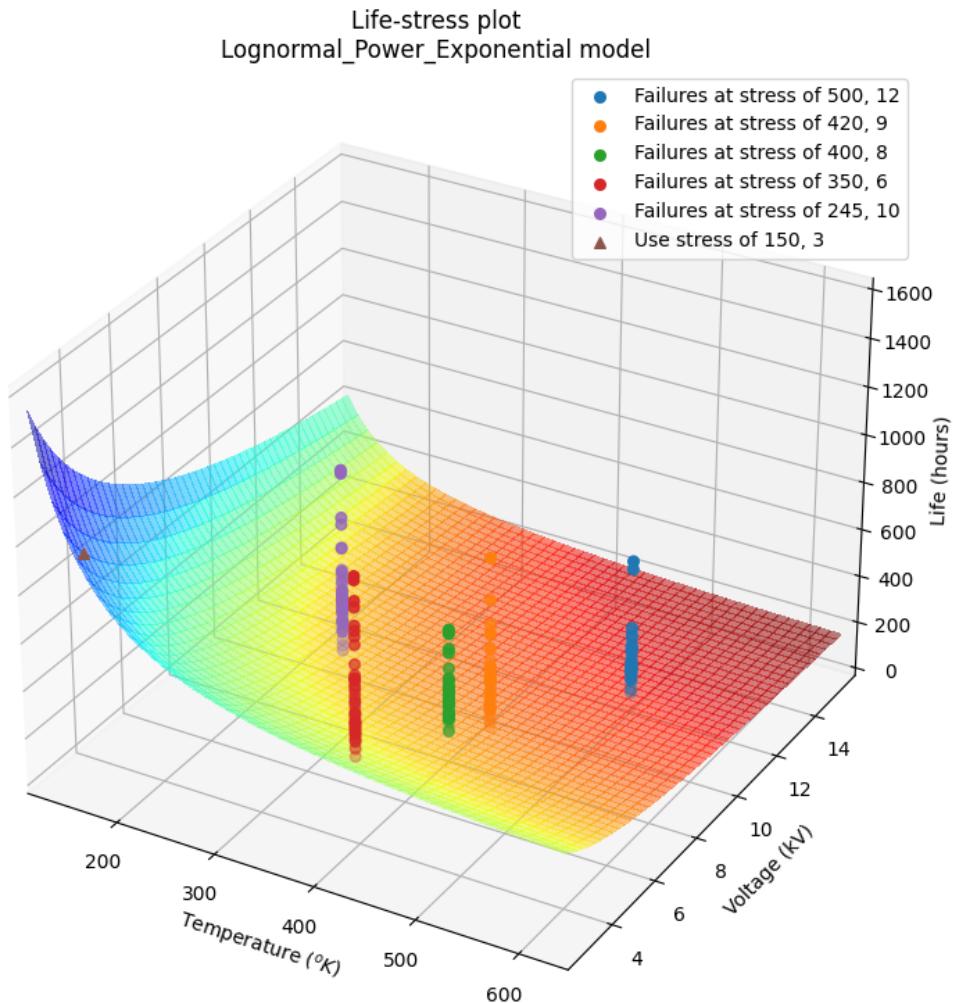
350, 6 ↳ 92364	5.84254	0.550746 5.77986	0.491052	-10.84%	2.
245, 10 ↳ 97102	5.75338	0.457948 5.76378	0.491052	+7.23%	2.

Goodness of fit *Value*
Log-likelihood -1596.29
AICc 3200.66
BIC 3217.44

At the use level stress of 150, 3, the mean life is 1067.69246

The mean life at use stress of the true model is: 992.7627728988726
 ""





Note

In the dual-stress life stress plots, there is a known visibility issue inherent in matplotlib where the 3D surface plot and the scatter plots are drawn in layers (relative to the observer). This results in the scatter plot always appearing in front of the 3D surface, even when some of the points should actually be occluded by the surface. The layering was chosen to show the scatter plot above the 3D surface plot as this provides better visibility than the alternative.

References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 136-168
- Accelerated Life Testing Data Analysis Reference - ReliaWiki, Reliawiki.com, 2019. [Online].



RELIABILITY
A Python library for reliability engineering

FITTING ALL AVAILABLE MODELS TO ALT DATA

Just as the function *Fitters.Fit_Everything* provides users with a quick way to fit all available distributions to their dataset, we can do a similar thing using *ALT_fitters.Fit_Everything_ALT* to fit all of the ALT models to an ALT dataset.

There are 24 ALT models available within *reliability*; 12 single stress models and 12 dual stress models. *Fit_Everything_ALT* will automatically fit the single stress or dual stress models based on whether the input includes single or dual stress data. Manual exclusion of certain models is also possible using the *exclude* argument. From the results, the models are sorted based on their goodness of fit test results, where the smaller the goodness of fit value, the better the fit of the model to the data.

API Reference

For inputs and outputs see the [API reference](#).

24.1 Example 1

In this first example, we will use *Fit_Everything_ALT* on some data that is generated using the function *make_ALT_data*. We can then compare the fitted results to the input parameters used to create the data. *Fit_Everything_ALT* produces two plots; a grid of all the fitted models (usually 12 models unless you have excluded some) and a larger plot of the best fitting model's probability plot. These are shown by default, so using *plt.show()* is not required to display the plots.

```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Everything_ALT

ALT_data = make_ALT_data(distribution='Normal', life_stress_model='Exponential', a=500,
                           b=1000, sigma=500, stress_1=[500, 400, 350], number_of_samples=100, fraction_censored=0.2,
                           seed=1)
Fit_Everything_ALT(failures=ALT_data.failures, failure_stress_1=ALT_data.failure_
                     _stresses, right_censored=ALT_data.right_censored, right_censored_stress_1=ALT_data.
                     right_censored_stresses, use_level_stress=300)

"
Results from Fit_Everything_ALT:
Analysis method: Maximum Likelihood Estimation (MLE)
Failures / Right censored: 240/60 (20% right censored)

      ALT_model      a      b      c      n      beta      sigma  Log-
      likelihood    AICc    BIC optimizer
Normal_Exponential  510.328 973.822                                486.137  -

```

(continues on next page)

(continued from previous page)

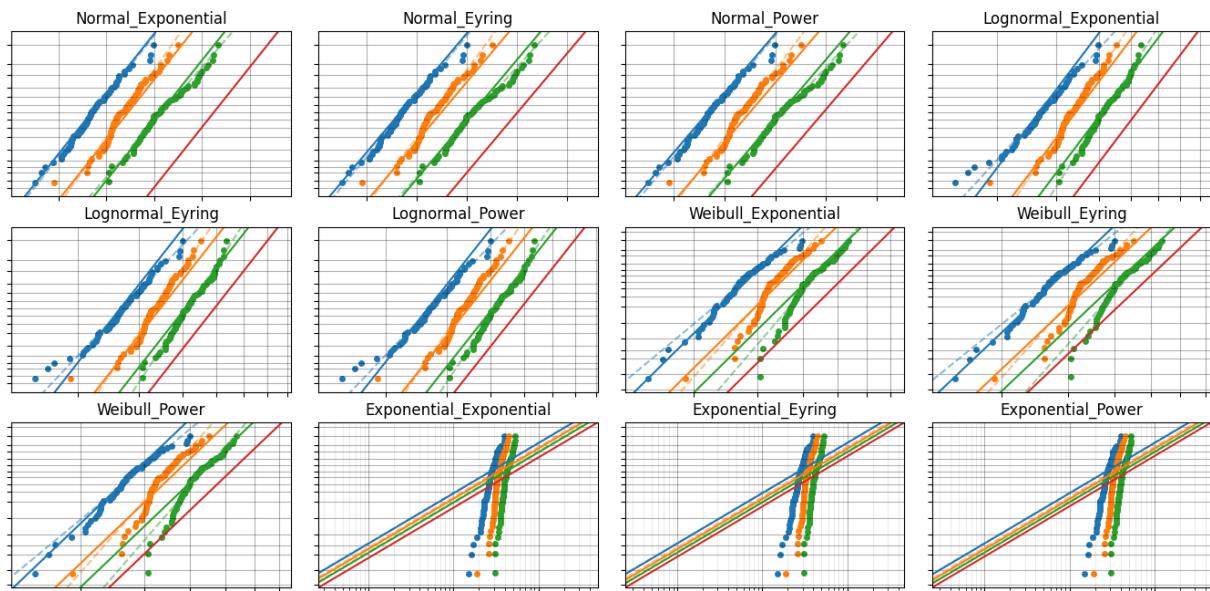
→1832.21	3670.51	3681.54	TNC			
	Normal_Eyring	97.5819		-13.9143	489.775	-
→1834.07	3674.22	3685.25	TNC			
	Normal_Power	5.47181e+06		-1.22605	490.81	-
→1834.59	3675.26	3686.29	TNC			
	Lognormal_Exponential	511.939	960.359		0.148961	-
→1838.87	3683.82	3694.85	TNC			
	Lognormal_Eyring	93.8622		-13.914	0.149853	-
→1840.35	3686.78	3697.81	TNC			
	Lognormal_Power	5.12115e+06		-1.21668	0.150092	-
→1840.74	3687.57	3698.6	TNC			
	Weibull_Exponential	452.154	1196.63		7.06525	-
→1852.39	3710.87	3721.9	TNC			
	Weibull_Eyring	35.0269		-14.1323	6.9927	-
→1854.7	3715.47	3726.5	TNC			
	Weibull_Power	5.56179e+06		-1.21792	6.85258	-
→1857.75	3721.58	3732.61	L-BFGS-B			
	Exponential_Exponential	503.344	1098.06			-
→2216.57	4437.17	4444.54	TNC			
	Exponential_Eyring	85.4578		-14.0477		-
→2216.62	4437.27	4444.64	TNC			
	Exponential_Power	5.56179e+06		-1.21159		-
→2216.63	4437.3	4444.66	L-BFGS-B			

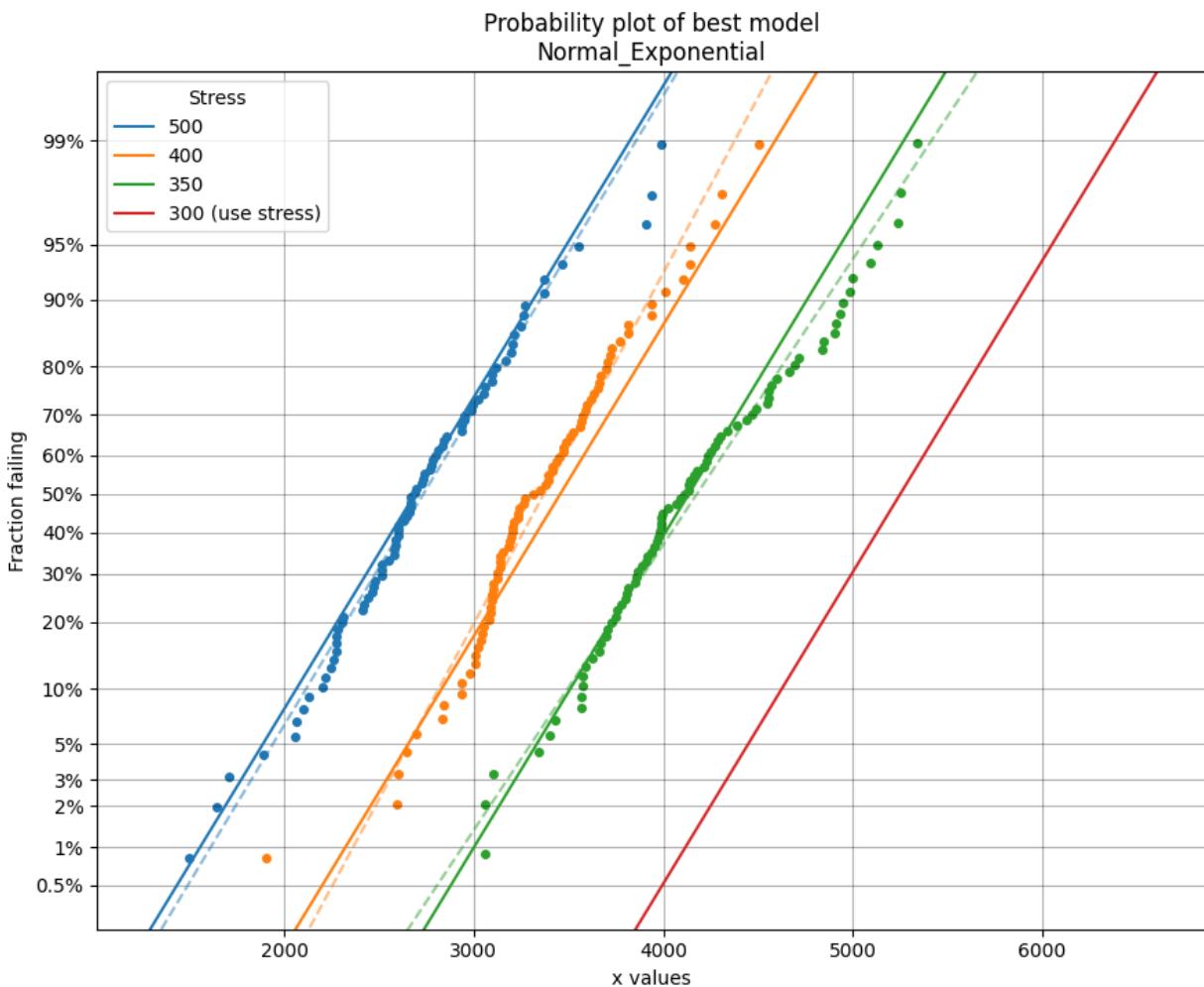
At the use level stress of 300, the Normal_Exponential model has a mean life of 5336.

→48499

""

Probability plots of each fitted ALT model





24.2 Example 2

In this second example, we will repeat what we saw in Example 1, but this time we will use a dual stress dataset generated using a Weibull_Dual_Power model.

```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Everything_ALT

ALT_data = make_ALT_data(distribution='Weibull', life_stress_model='Dual_Power', c=1e15, m=-4, n=-2, beta=2.5, stress_1=[500, 400, 350, 420, 245], stress_2=[12, 8, 6, 9, 10], number_of_samples=100, fraction_censored=0.2, seed=1)
Fit_Everything_ALT(failures=ALT_data.failures, failure_stress_1=ALT_data.failure_stresses_1, failure_stress_2=ALT_data.failure_stresses_2, right_censored=ALT_data.right_censored, right_censored_stress_1=ALT_data.right_censored_stresses_1,right_censored_stress_2=ALT_data.right_censored_stresses_2, use_level_stress=[250,7])

"""
Results from Fit_Everything_ALT:
Analysis method: Maximum Likelihood Estimation (MLE)
Failures / Right censored: 400/100 (20% right censored)

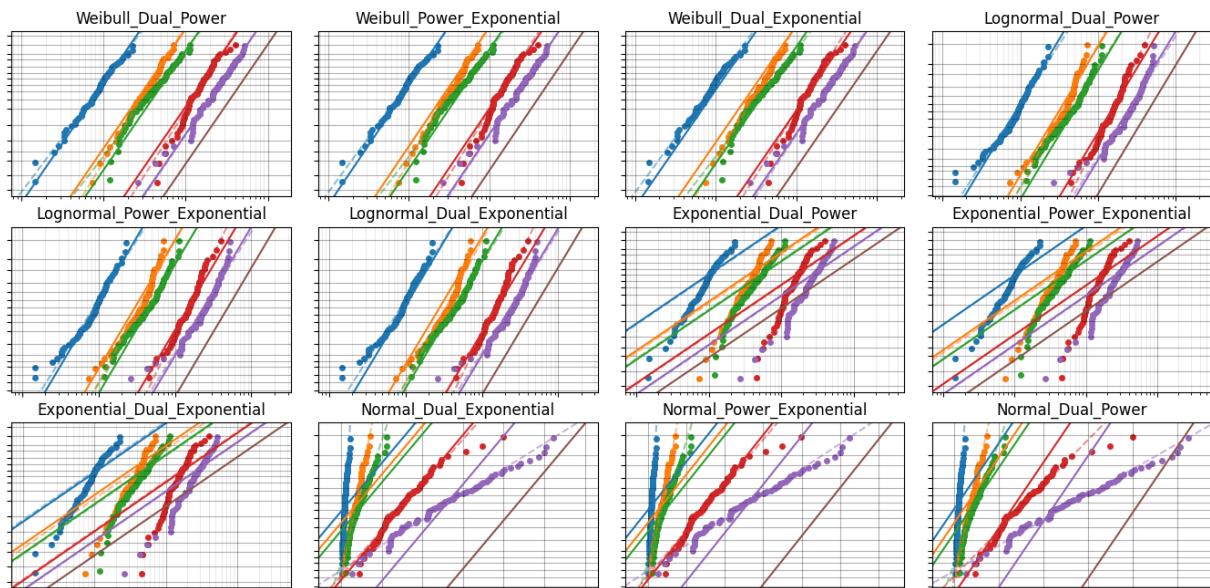
```

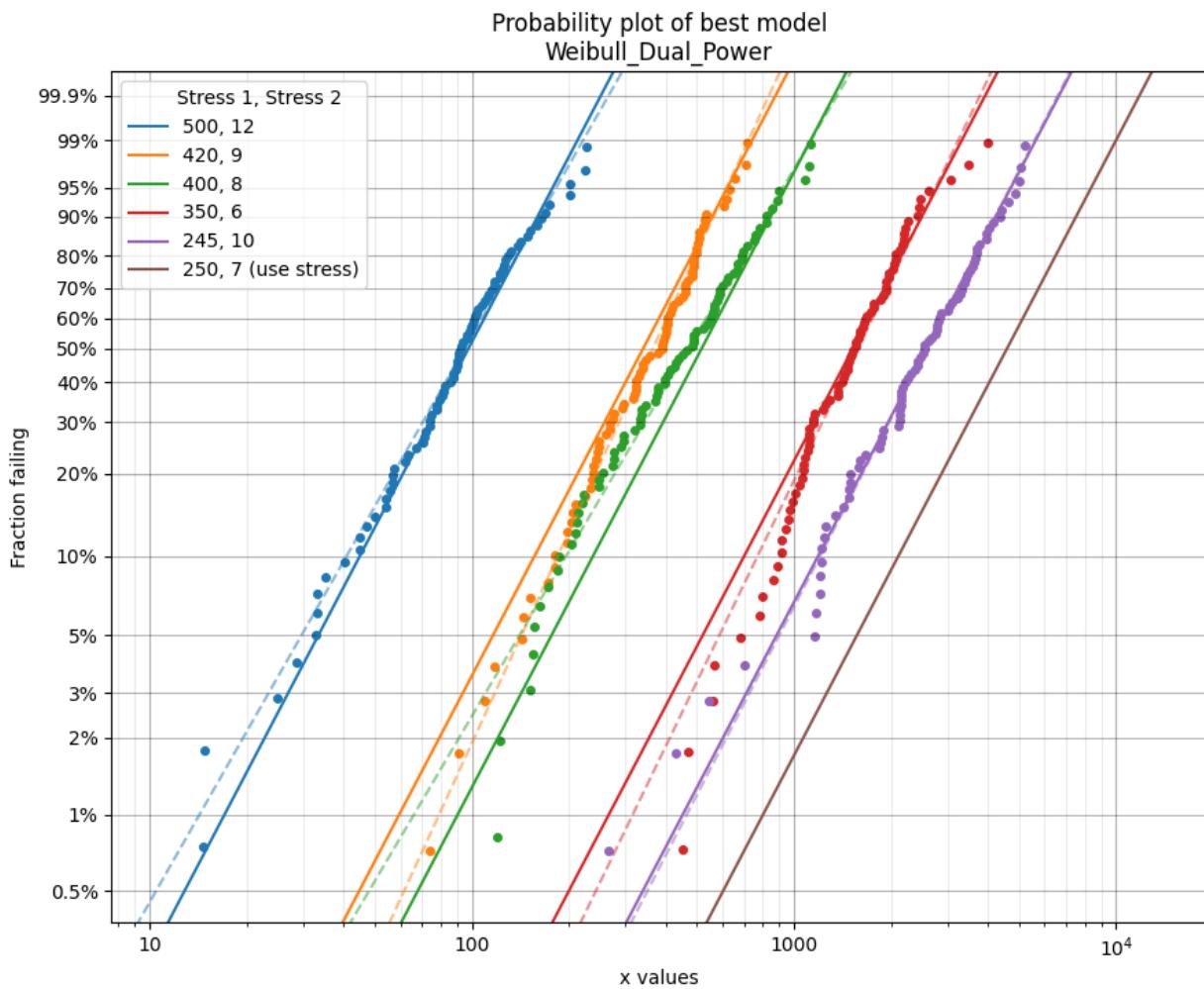
(continues on next page)

(continued from previous page)

ALT_model	a	b	c	m	n	beta	l
sigma Log-likelihood AICc		BIC optimizer					
Weibull_Dual_Power			1.46794e+15	-4.09472	-1.89863	2.49644	l
-2815.87 5639.82 5656.59			TNC				l
Weibull_Power_Exponential	1348.44			2675.53		-2.33209	2.49569
-2816.33 5640.73 5657.51			TNC				l
Weibull_Dual_Exponential	1362.97	18.8418		1.79306		2.45183	l
-2822.88 5653.84 5670.62			TNC				l
Lognormal_Dual_Power			1.47649e+15	-4.12473	-1.92408	0.	l
510416 -2841.82 5691.72 5708.5			TNC				0.
Lognormal_Power_Exponential	1358.86			2217		-2.36348	0.
510473 -2841.93 5691.93 5708.71			TNC				l
Lognormal_Dual_Exponential	1378.59	19.2034		1.30704		0.	l
514815 -2845.56 5699.2 5715.98			TNC				l
Exponential_Dual_Power			1.76808e+15	-4.13272	-1.89779	0.	l
-3007.17 6020.4 6032.99			TNC				l
Exponential_Power_Exponential	1361.03			2510.92		-2.33605	l
-3007.29 6020.62 6033.22			TNC				l
Exponential_Dual_Exponential	1378.66	18.9113		1.63804		0.	l
-3008.35 6022.75 6035.34			TNC				l
Normal_Dual_Exponential	1174.43	14.8298		5.00316		604.	l
207 -3172.69 6353.45 6370.23			TNC				l
Normal_Power_Exponential	1203.19			1954.03		-1.99859	604.
571 -3173.15 6354.38 6371.16			TNC				l
Normal_Dual_Power			1.13695e+15	-4.08012	-1.94369	450.	l
658 -3256.12 6520.32 6537.09	L-BFGS-B						
At the use level stress of 250, 7, the Weibull_Dual_Power model has a mean life of 4912.							
81077							
"							

Probability plots of each fitted ALT model





24.3 Example 3

In this third example, we will look at how to extract specific parameters from the output. This example uses a dataset from *reliability.Datasets*. The plots are turned off for this example.

```
from reliability.Datasets import ALT_temperature
from reliability. ALT_fitters import Fit_Everything_ALT

model = Fit_Everything_ALT(failures=ALT_temperature().failures, failure_stress_1=ALT_
    .temperature().failure_stresses, right_censored=ALT_temperature().right_censored, right_
    .censored_stress_1=ALT_temperature().right_censored_stresses, show_probability_
    .plot=False, show_best_distribution_probability_plot=False)
print('The Lognormal_Power model parameters are:\n a:',model.Lognormal_Power_a,' \n n:',_
    .model.Lognormal_Power_n, '\n sigma:',model.Lognormal_Power_sigma)

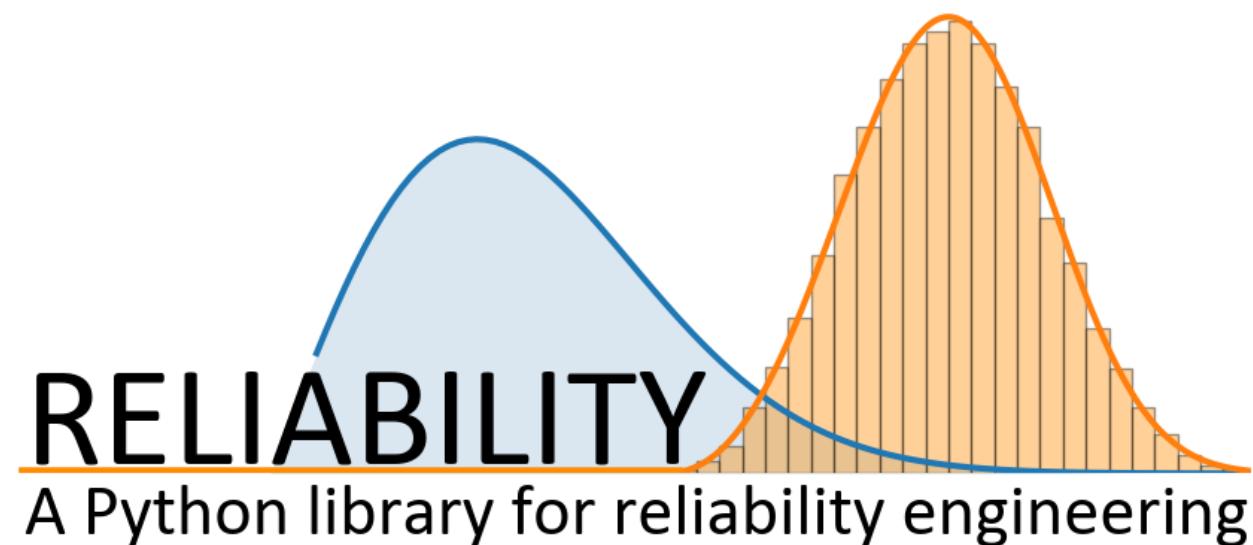
"""

Results from Fit_Everything_ALT:
Analysis method: Maximum Likelihood Estimation (MLE)
Failures / Right censored: 35/102 (74.45255474452554% right censored)
```

(continues on next page)

(continued from previous page)

ALT_model	a	b	c	n	beta	sigma	Log-
↪ likelihood	AICc	BIC	optimizer				
↪ 339.183	684.546	693.126	TNC		-3.64018	0.961968	-
↪ 339.835	685.851	694.43	L-BFGS-B		-9.94803	0.976603	-
↪ 340.144	686.468	695.047	TNC	197.357	134.732	0.986886	-
↪ 340.39	686.96	695.54	TNC		-3.73732	1.44776	-
↪ 343.274	690.639	696.389	TNC		-4.85419		-
↪ 343.795	691.679	697.43	Weibull_Eyring	151.091	-10.1367	1.42117	-
↪ 341.206	688.592	697.171	TNC		-9.31393		-
↪ 343.991	692.071	697.821	Exponential_Eyring	211.096			-
↪ 343.919	714.018	722.598	TNC	266.147	71.2215		-
↪ 341.591	689.363	697.942	Weibull_Exponential	208.334	157.573	1.39983	-
↪ 353.905	714.018	722.598	TNC		-11.7653	2464.82	-
↪ 354.496	715.172	723.751	L-BFGS-B	37.0335		2439.04	-
↪ 465.464	937.109	945.688	Normal_Power	89.9058	855.015		-
↪ 465.464	937.109	945.688	L-BFGS-B	772496	-1.48137	2464.97	-
							"
The Lognormal_Power model parameters are:							
a: 12103627516.445246							
n: -3.6401834647746396							
sigma: 0.9619680090329055							



WHAT DOES AN ALT PROBABILITY PLOT SHOW ME

An ALT probability plot shows us how well our dataset can be modeled by the chosen distribution. This is more than just a goodness of fit at each stress level, because the distribution needs to be a good fit at all stress levels and be able to fit well with a common shape parameter. If you find the shape parameter changes significantly as the stress increases then it is likely that your accelerated life test is experiencing a different failure mode at higher stresses. When examining an ALT probability plot, the main things we are looking for are:

- Does the model appear to fit the data well at all stress levels (ie. the dashed lines pass reasonably well through all the data points)
- Examine the AICc and BIC values when comparing multiple models. A lower value suggests a better fit.
- Is the amount of change to the shape parameter within the acceptable limits (generally less than 50% for each distribution).

The following example fits 2 models to ALT data that is generated from a Normal_Exponential model. The first plot is an example of a good fit. The second plot is an example of a very bad fit. Notice how a warning is printed in the output telling the user that the shape parameter is changing too much, indicating the model may be a poor fit for the data. Also note that the total AIC and total BIC for the Exponential_Power model is higher (worse) than for the Normal_Exponential model.

If you are uncertain about which model you should fit, try [fitting everything](#) and select the best fitting model.

If you find that none of the models work without large changes to the shape parameter at the higher stresses, then you can conclude that there must be a change in the failure mode for higher stresses and you may need to look at changing the design of your accelerated test to keep the failure mode consistent across tests.

25.1 Example 1

```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Normal_Exponential, Fit_Exponential_Power
import matplotlib.pyplot as plt

ALT_data = make_ALT_data(distribution='Normal', life_stress_model='Exponential', a=500,
                         b=1000, sigma=500, stress_1=[500, 400, 350], number_of_samples=100, fraction_censored=0.2,
                         seed=1)
Fit_Normal_Exponential(failures=ALT_data.failures, failure_stress=ALT_data.failure_
                        _stresses, right_censored=ALT_data.right_censored, right_censored_stress=ALT_data.right_
                        _censored_stresses, show_life_stress_plot=False)
print('-----')
Fit_Exponential_Power(failures=ALT_data.failures, failure_stress=ALT_data.failure_
                        _stresses, right_censored=ALT_data.right_censored, right_censored_stress=ALT_data.right_
                        _censored_stresses, show_life_stress_plot=False)
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

```
""
```

```
Results from Fit_Normal_Exponential (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Failures / Right censored: 240/60 (20% right censored)
```

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
a	501.729	27.5897	447.654	555.804
b	985.894	70.4156	857.107	1134.03
sigma	487.321	22.1255	445.829	532.674

stress	original mu	original sigma	new mu	common sigma	sigma	change
500	2733.7	482.409	2689.22	487.321		+1.02%
400	3369.57	432.749	3456.02	487.321		+12.61%
350	4176.89	531.769	4134.25	487.321		-8.36%

```
Goodness of fit      Value
Log-likelihood -1833.41
      AICc   3672.89
      BIC   3683.93
```

If this model is being used for the Arrhenius Model, $a = Ea/K_B \Rightarrow Ea = 0.04324$ eV

```
Results from Fit_Exponential_Power (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Failures / Right censored: 240/60 (20% right censored)
```

Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
a	4.23394e+06	1.10593e+07	25314.2	7.0815e+08
n	-1.16747	0.433666	-2.01744	-0.317497

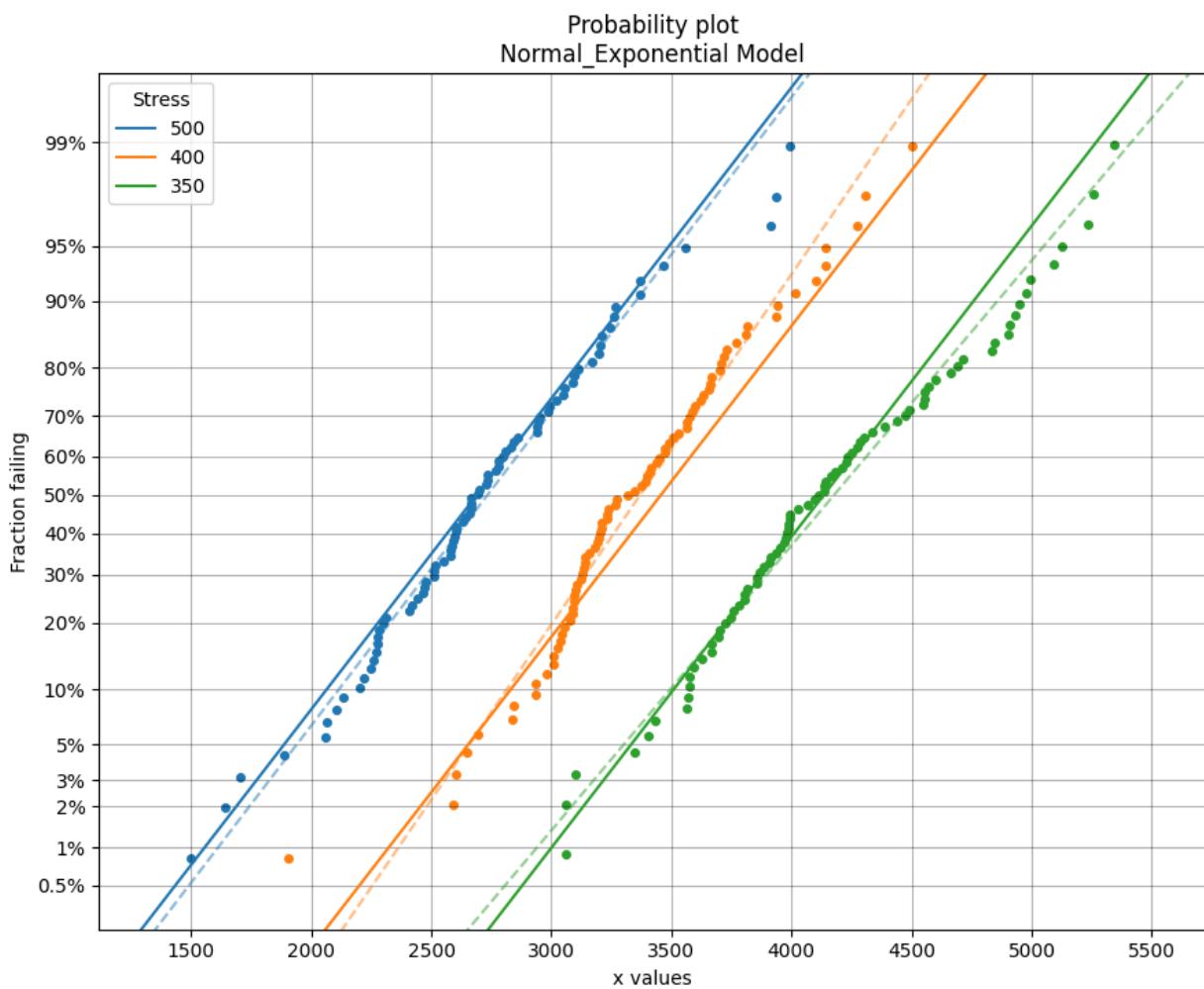
stress	weibull alpha	weibull beta	new 1/Lambda	common shape	shape	change
500	2937.9	5.99874	2990.79	1		-83.33%
400	3561	8.20311	3880.84	1		-87.81%
350	4415.41	8.3864	4535.54	1		-88.08%

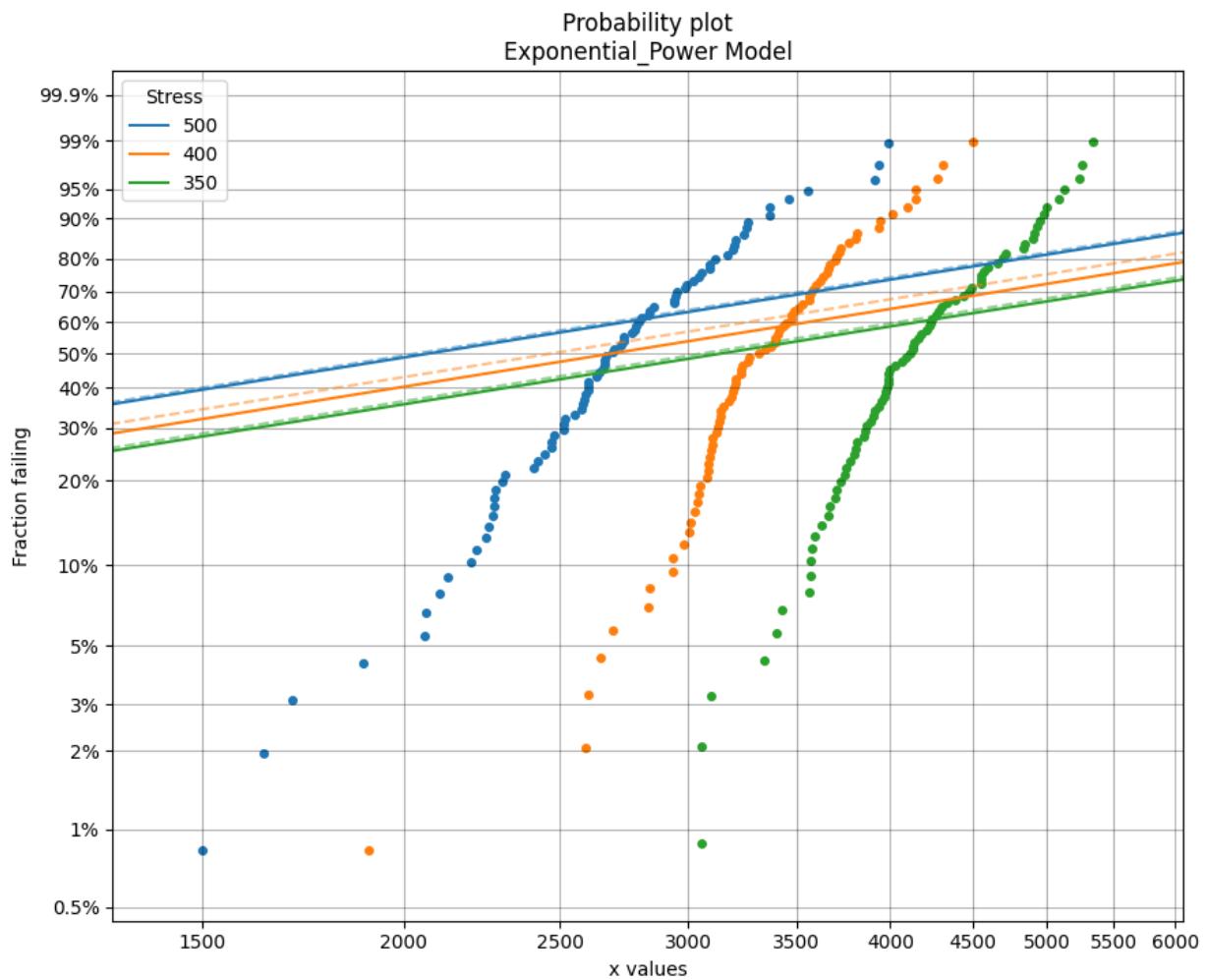
The shape parameter has been found to change significantly (>50%) when fitting the ALT model.

This may indicate that a different failure mode is acting at different stress levels or that the Exponential distribution may not be appropriate.

```
Goodness of fit      Value
Log-likelihood -2214.94
      AICc   4433.93
      BIC   4441.3
```

```
""
```





References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 136-168
- Accelerated Life Testing Data Analysis Reference - ReliaWiki, Reliawiki.com, 2019. [Online].



RELIABILITY
A Python library for reliability engineering

DATASETS

API Reference

For inputs and outputs see the [API reference](#).

There are a few datasets that have been included with reliability that users may find useful for testing and experimenting. Within *reliability.Datasets* the following datasets are available:

Standard datasets

- automotive - 10 failures, 21 right censored. It is used in [this example](#)
- mileage - 100 failures with no right censoring. It is used in the examples for `KTest` and `chi2test`.
- system_growth - 22 failures with no right censoring. It is used in the example for `reliability_growth`.
- defective_sample - 1350 failures, 12296 right censored. It exhibits the behavior of a defective sample (also known as Limited failure population or Defective subpopulation).
- mixture - 71 failures, 3320 right censored. This is best modelled using a mixture model.
- electronics - 10 failures, 4072 right censored. It is used in [this example](#).

ALT Datasets

- ALT_temperature - conducted at 3 temperatures. 35 failures, 102 right censored. For example usage of many of the ALT Datasets see the [examples here](#).
- ALT_temperature2 - conducted at 4 temperatures. 40 failures, 20 right censored.
- ALT_temperature3 - conducted at 3 temperatures. 30 failures, 0 right censored.
- ALT_temperature4 - conducted at 3 temperatures. 20 failures, 0 right censored.
- ALT_load - conducted at 3 loads. 20 failures, 0 censored.
- ALT_load2 - conducted at 3 loads. 13 failures, 5 right censored.
- ALT_temperature_voltage - conducted at 2 different temperatures and 2 different voltages. 12 failures, 0 right censored.
- ALT_temperature_voltage2 - conducted at 3 different temperatures and 2 different voltages. 18 failures, 8 right censored.
- ALT_temperature_humidity - conducted at 2 different temperatures and 2 different humidities. 12 failures, 0 right censored.

MCF Datasets

- MCF_1 - this dataset contains failure and retirement times for 5 repairable systems. Exhibits a worsening repair rate.
- MCF_2 - this dataset contains failure and retirement times for 56 repairable systems. Exhibits a worsening then improving repair rate. Difficult to fit this dataset.

All datasets are functions which create objects and every dataset object has several attributes.

For the standard datasets, these attributes are:

- info - a dataframe of statistics about the dataset
- failures - a list of the failure data
- right_censored - a list of the right_censored data
- right_censored_stress - a list of the right_censored stresses (ALT datasets only)

For the ALT datasets, these attributes are similar to the above standard attributes, just with some variation for the specific dataset. These include things like:

- failure_stress_humidity
- right_censored_stress_voltage
- failure_stress_temp
- other similarly named attributes based on the dataset

For the MCF datasets these attributes are:

- times
- number_of_systems

If you would like more information on a dataset, you can type the name of the dataset in the help function (after importing it).

```
from reliability.Datasets import automotive
print(help(automotive))
```

If you would like the statistics about a dataset you can access the info dataframe as shown below.

```
from reliability.Datasets import defective_sample
print(defective_sample().info)

"""
      Stat          Value
      Name  defective_sample
  Total  Values          13645
  Failures          1350 (9.89%)
Right Censored      12295 (90.11%)
"""
```

The following example shows how to import a dataset and use it. Note that we must use brackets () to call the dataset (since it is a class) before accessing the failures and right_censored values.

```
from reliability.Datasets import automotive
from reliability.Fitters import Fit_Weibull_2P
Fit_Weibull_2P(failures=automotive().failures,right_censored=automotive().right_censored,
               show_probability_plot=False)
```

(continues on next page)

(continued from previous page)

""

*Results from Fit_Weibull_2P (95% CI):**Analysis method: Maximum Likelihood Estimation (MLE)**Failures / Right censored: 10/21 (67.74194% right censored)*

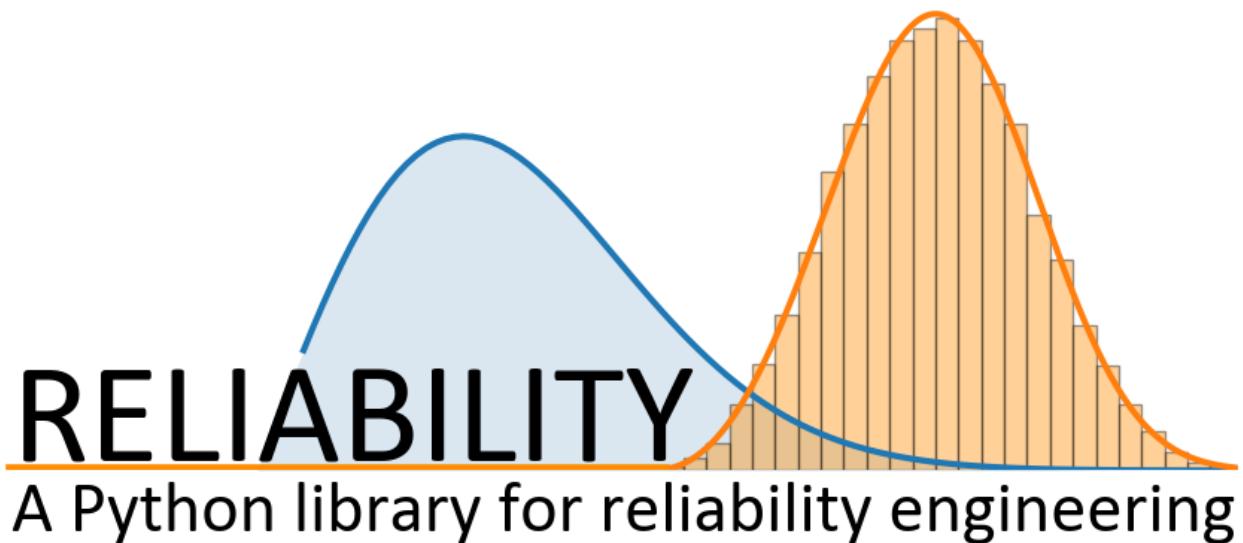
Parameter	Point Estimate	Standard Error	Lower CI	Upper CI
Alpha	134243	42371.1	72314.7	249204
Beta	1.15586	0.295842	0.699905	1.90884

Goodness of fit *Value**Log-likelihood* -128.974*AICc* 262.376*BIC* 264.816*AD* 35.6075

""

If you have an interesting dataset, please email me (alpha.reliability@gmail.com) and I may include it in this database.

If you would like to use any of these datasets in your own work, you are permitted to do so under the [LGPLv3](#) license. Under this license you must [acknowledge the source](#) of the datasets.



CHAPTER
TWENTYSEVEN

IMPORTING DATA FROM EXCEL

The module Convert_data contains three functions for importing data from Microsoft Excel (.xlsx files) into Python and converting that data into the structure of a Python object for further use. These functions are:

- xlsx_to_XCN
- xlsx_to_FNRM
- xlsx_to_FR

Each of the three data formats has an acceptable reduced form as follows:

- XCN reduced form is XC and all rows are assumed to have a quantity of 1
- FNRM reduced form is FN and it is assumed that there is no right censored data
- FR reduced form is F and it is assumed that there is no right censored data

API Reference

For inputs and outputs see the [API reference](#).

These three functions should be used only for data in their respective format (XCN, FNRM, FR). This means that the columns of the xlsx file should be in the same order as the name of the data format. For example, xlsx_to_XCN is expecting to receive an xlsx file with 3 columns corresponding to X, C, and N. If these are in a different order they may be misinterpreted or trigger an error. You should correct the column order in the xlsx file before importing into Python.

All of the three conversion functions contain the following methods:

- print() - this will print a dataframe of the data in the output format to the console
- write_to_xlsx() - this will export the data in the output format to an xlsx file at the specified path. Ensure you specify the path string preceded by r to indicate raw text. For example: write_to_xlsx(path='C:/Users/Current User/Desktop/mydata.xlsx'). If the file already exists in the destination folder, the user will be asked (Y/N) whether they want to overwrite the existing file. If they input N then specified filename will have (new) added to the end.

Why use different formats

There are advantages and disadvantages of each of these formats depending on the data being represented. Most importantly, we need an easy way of converting data between these formats as different software may store and receive data in different formats.

- XCN - This format is the default in most commercial software including Reliasoft and Minitab. The sequence of XCN and the names may change between different software, but the format is essentially the same. Within reliability the XCN format may be reduced to XC (and all items are assumed to have quantity of 1). Some other software accepts the further reduced form of X (where there are no censored items and all items have a quantity

of 1). If you have only failure data that is not grouped, then you should use the FR format as FR has a reduced form of F which is equivalent to X from XCN.

- FNRN - This format is not used as a data entry format for *reliability* or any commercial software (that the author has used), but is still a valid format which combines aspects of the XCN and FR formats together. FNRN is used internally within reliability as part of the MLE algorithms.
- FR - This is the standard data entry format for *reliability*. The FR format is the most simple, but for data with many repeated values it is not as efficient at representing the data in a table as FNRN or XCN. Python has no problems with long arrays so the FR format is chosen as the data entry format for its simplicity.

For more information on these three data formats as well as how to convert data between the different formats, please see the section on [Converting data between different formats](#).

27.1 Example 1

In the example below, a screenshot of the data from Excel is shown along with the import process and an example of the print method. The censoring codes are automatically recognised. See [Example 3](#) for how to modify the censor code recognition process.

	A	B	C
1	Event time	State	Number in state
2	13	failure	2
3	45	failure	3
4	78	failure	1
5	89	suspension	4
6	102	suspension	1
7	105	suspension	2

```
from reliability.Convert_data import xlsx_to_XCN
data = xlsx_to_XCN(path=r'C:\Users\Current User\Desktop\XCN.xlsx')
print(data.X)
print(data.C)
print(data.N)
data.print()

"""
[ 13.  45.  78.  89.  102.  105.]
['F' 'F' 'F' 'C' 'C' 'C']
[2 3 1 4 1 2]
Data (XCN format)
event time censor code  number of events
    13      F          2
    45      F          3
    78      F          1
    89      C          4
   102      C          1
   105      C          2
""
```

27.2 Example 2

The use of `xlsx_to_FNRR` and `xlsx_to_FR` are very similar to that shown above. This example shows the use of `xlsx_to_FR`.

	A	B
1	failures	suspensions
2	37	200
3	67	200
4	120	200
5		300
6		300

```
from reliability.Convert_data import xlsx_to_FR
data = xlsx_to_FR(path=r'C:\Users\Current User\Desktop\FR.xlsx')
print(data.failures)
print(data.right_censored)
data.print()

"""
[ 37.  67. 120.]
[200 200 200 300 300]
Data (FR format)
failures  right censored
      37          200
      67          200
     120          200
            300
            300
"""

```

27.3 Example 3

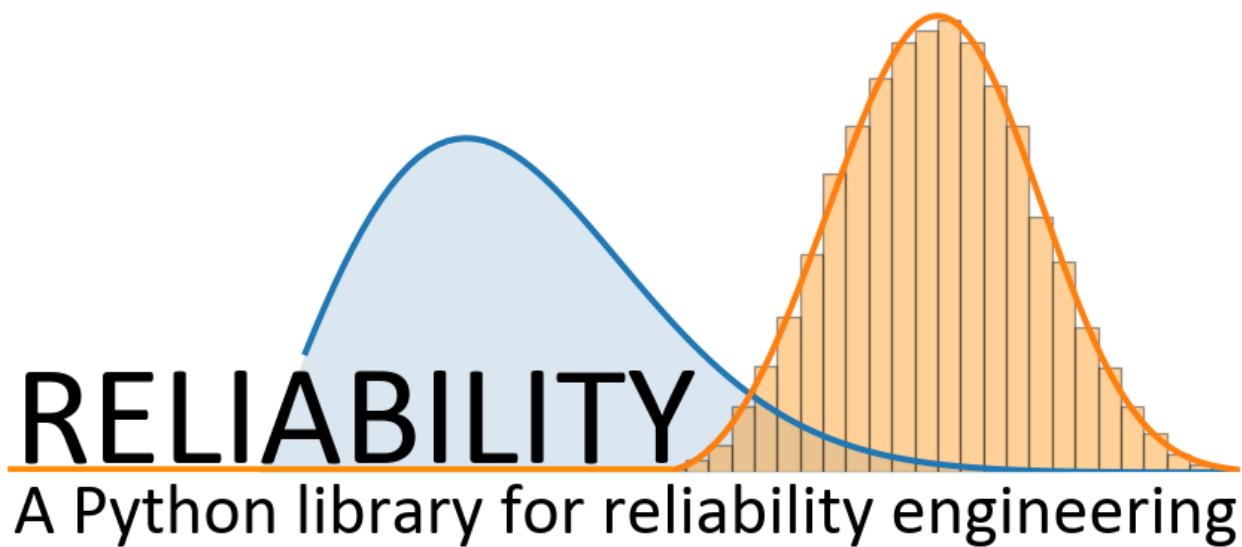
In this example we will again use `xlsx_to_XCN`, however this time the censoring codes need customizing. The `xlsx` file uses 1 in the second column for failures (note that the defaults recognise 0 as failure and 1 as right censored) and ‘still alive’ for the right censored items (‘still alive’ is not part of the recognised defaults). If we do not specify these custom censoring codes, the failures will be misinterpreted as right censored items and the ‘still alive’ items will return an error as this code is not recognised. To resolve this we must set the `censor_code_in_xlsx` and `failure_code_in_xlsx` arguments. Furthermore we want the `XCN` data object in Python to use ‘S’ instead of ‘C’ for the censored items. We do this by setting the `censor_code_in_XCN` argument. If we wanted to change the failure code from ‘F’ to something else we could similarly use the argument `failure_code_in_XCN`.

	A	B	C
1	Event time	State	Number in state
2	13	1	2
3	45	1	3
4	78	1	1
5	89	still alive	4
6	102	still alive	1
7	105	still alive	2

```
from reliability.Convert_data import xlsx_to_XCN
data = xlsx_to_XCN(path=r'C:\Users\Current User\Desktop\XCN.xlsx', censor_code_in_xlsx=
    'still alive', failure_code_in_xlsx=1, censor_code_in_XCN='S')
print(data.X)
print(data.C)
print(data.N)
data.print()

"""
[ 13.  45.  78.  89.  102.  105.]
[F' F' F' 'S' 'S' 'S']
[2 3 1 4 1 2]
Data (XCN format)
event time censor code  number of events
    13          F          2
    45          F          3
    78          F          1
    89          S          4
   102          S          1
   105          S          2
"""


```



CHAPTER
TWENTYEIGHT

CONVERTING DATA BETWEEN DIFFERENT FORMATS

The module Convert_data contains six functions for converting data between the three formats (XCN, FNRRN, FR). These functions are:

- XCN_to_FNRRN
- XCN_to_FR
- FNRRN_to_XCN
- FNRRN_to_FR
- FR_to_XCN
- FR_to_FNRRN

The three data formats are different ways of representing the same information. The following image shows the same data expressed in each format.

Data (XCN format)

event	time	censor	code	number of events
13			F	2
45			F	3
78			F	1
89			C	4
102			C	1
105			C	2

Data (FNRRN format)

failures	number of failures	right censored	number of right censored
13	2	89	4
45	3	102	1
78	1	105	2

Data (FR format)

failures	right censored
13	89
13	89
45	89
45	89
45	102
78	105
	105

XCN - event time, censoring code, number of events - This format is the default in most commercial software including Reliasoft and Minitab. The sequence of XCN and the names may change between different software, but the format is essentially the same. Within reliability the XCN format may be reduced to XC (where all items are assumed to have quantity of 1). Some other software accepts the further reduced form of X (where there are no censored items and all items have a quantity of 1). If you have only failure data that is not grouped, then you should use the FR format as FR has a reduced form of F which is equivalent to X from XCN.

FNRM - failures, number of failures, right censored, number of right censored - This format is not used as a data entry format for reliability or any commercial software (that the author has used), but is still a valid format which combines aspects of the XCN and FR formats together. FNRM is used internally within reliability as part of the MLE algorithms.

FR - failures, right censored - This is the standard data entry format for reliability. The FR format is the most simple, but for data with many repeated values it is not as efficient at representing the data in a table as FNRM or XCN. Python has no problems performing calculations with long arrays so the FR format is chosen as the data entry format for its simplicity.

Each of the three data formats has an acceptable reduced form as follows:

- XCN reduced form is XC and all rows are assumed to have a quantity of 1
- FNRM reduced form is FN and it is assumed that there is no right censored data
- FR reduced form is F and it is assumed that there is no right censored data

For more information on these three data formats as well as how to import data from Microsoft Excel (.xlsx files) into Python, please see the section on [Importing data from Excel](#).

API Reference

For inputs and outputs see the [API reference](#).

All of the six conversion functions contain the following methods:

- `print()` - this will print a dataframe of the data in the output format to the console
- `write_to_xlsx()` - this will export the data in the output format to an xlsx file at the specified path. Ensure you specify the path string preceeded by `r` to indicate raw text. For example: `write_to_xlsx(path='C:/Users/Current User/Desktop/mydata.xlsx')`. If the file already exists in the destination folder, the user will be asked (Y/N) whether they want to overwrite the existing file. If they input N then specified filename will have (new) added to the end.

28.1 Example 1

In the example below we are converting FR to FNRM format and then printing each of the available outputs. Using the `print()` method will print a dataframe to the console.

```
from reliability.Convert_data import FR_to_FNRM
FNRM = FR_to_FNRM(failures=[8,15,15,20,25,30,30,30,30,32,32,32], right_censored=[17,17,
˓→50,50,50,50,78,78,78,78,90])
print(FNRM.failures)
print(FNRM.num_failures)
print(FNRM.right_censored)
print(FNRM.num_right_censored)
FNRM.print()
```

(continues on next page)

(continued from previous page)

```

"""
[ 8 15 20 25 30 32]
[1 2 1 1 4 3]
[17 50 78 90]
[2 4 4 1]
Data (FNRN format)
failures number of failures right censored number of right censored
    8           1           17           2
    15          2           50           4
    20          1           78           4
    25          1           90           1
    30          4
    32          3
"""

```

28.2 Example 2

In the example below we are converting XCN to FR format. The XCN data uses censor code 1 for failures and 0 for right censored. Within *reliability* the default censor code for failures is 0 and for right censored is 1. If we do not correct this, the converter will interpret the censor codes the wrong way around. This is resolved by specifying the arguments `censor_code` and `failure_code`.

```

from reliability.Convert_data import XCN_to_FR
FR = XCN_to_FR(X=[12,15,18,32,35,38,60], C=[1,1,1,0,0,0,0], N=[1,1,1,2,2,1,3], failure_
    ↪code=1, censor_code=0)
print(FR.failures)
print(FR.right_censored)
FR.print()

"""
[12. 15. 18.]
[32. 32. 35. 35. 38. 60. 60. 60.]
Data (FR format)
failures right censored
    12           32
    15           32
    18           35
                35
                38
                60
                60
                60
"""

```



RELIABILITY
A Python library for reliability engineering

CHAPTER
TWENTYNINE

RELIABILITY GROWTH

The reliability of a non-repairable component always decreases with time, but for repairable systems the term “reliability growth” refers to the process of gradual product improvement through the elimination of design deficiencies. In repairable systems, reliability growth is observable through an increase in the interarrival times of failures. Reliability growth is applicable to all levels of design decomposition from complete systems down to components. The maximum achievable reliability is locked in by design, so reliability growth above the design reliability is only possible through design changes. It may be possible to achieve some reliability growth through other improvements (such as optimizing the maintenance program) though these improvements will only help the system to achieve its design reliability.

The function *Repairable_systems.reliability_growth* fits a model to the failure times so that the growth (or deterioration) of the mean time between failures (MTBF) can be predicted. The two models available within *reliability* are the Duane model and the Crow-AMSAA model.

In both the Duane and Crow-AMSAA models, the input is a list of failure times (actual times not interarrival times). The point estimate for the cumulative MTBF is calculated as $\frac{\text{failure times}}{\text{failure numbers}}$. This produces the scatter plot shown in the plots below.

The smooth curve shows the model (Duane or Crow-AMSAA) that is fitted to the data. The formulation of these models is explained below.

29.1 Duane model

The algorithm to fit the model is as follows:

1. accept an input of the failures times and sort the times in ascending order. Let the largest time be T .
2. create an array of failure numbers from 1 to n .
3. calculate $MTBF_{cumulative} = \frac{\text{failure times}}{\text{failure numbers}}$. This is the scatter points seen on the plot.
4. Convert to log space: $x = \ln(\text{failure times})$ and $y = \ln(MTBF_{cumulative})$
5. fit a straight line ($y = m.x + c$) to the data to get the model parameters.
6. extract the model parameters from the parameters of the straight line, such that $\alpha = m$ and $b = \exp(c)$

This gives us the model parameters of b and α . The formulas for the other reported values are:

$DMTBF_C = b.T^\alpha$. This is the demonstrated MTBF (cumulative) and is reported in the results as *DMTBF_C*.

$DFI_C = \frac{1}{DMTBF_C}$. This is the demonstrated failure intensity (cumulative) and is reported in the results as *DFI_C*.

$DFI_I = DFI_C.(1 - \alpha)$. This is the demonstrated failure intensity (instantaneous) and is reported in the results as *DFI_I*.

$DMTBF_I = \frac{1}{DFI_I}$. This is the demonstrated MTBF (instantaneous) and is reported in the results as *DMTBF_I*.

$A = \frac{1}{b}$. This is reported in the results as *A*.

The time to reach the target MTBF is calculated as $t_{target} = \left(\frac{\text{target MTBF}}{b} \right)^{\frac{1}{\alpha}}$

For more information see [reliawiki](#).

29.2 Crow-AMSAA model

The algorithm to fit the model is as follows:

1. accept an input of the failures times and sort the times in ascending order. Let the largest time be T .
2. create an array of failure numbers from 1 to n .
3. calculate $MTBF_{cumulative} = \frac{\text{failure times}}{\text{failure numbers}}$. This is the scatter points seen on the plot.
4. calculate $\beta = \frac{n}{n \ln(T) - \sum \ln(\text{failure times})}$
5. calculate $\lambda = \frac{n}{T^\beta}$

This gives us the model parameters β and λ . The formulas for the other reported values are:

$growthrate = 1 - \beta$. This is reported in the results as `growth_rate`.

$DMTBF_I = \frac{1}{\lambda \cdot \beta \cdot T^{\beta-1}}$. This is the demonstrated MTBF (instantaneous) and is reported in the results as `DMTBF_I`.

$DFI_I = \frac{1}{DMTBF_I}$. This is the demonstrated failure intensity (instantaneous) and is reported in the results as `DFI_I`.

$DMTBF_C = \frac{T^{1-\beta}}{\lambda}$. This is the demonstrated MTBF (cumulative) and is reported in the results as `DMTBF_C`.

$DFI_C = \frac{1}{DMTBF_C}$. This is the demonstrated failure intensity (cumulative) and is reported in the results as `DFI_C`.

The time to reach the target MTBF is calculated as $t_{target} = \left(\frac{1}{\lambda \cdot (\text{target MTBF})} \right)^{\frac{1}{\beta-1}}$

For more information see [reliawiki](#).

API Reference

For inputs and outputs see the [API reference](#).

29.3 Example 1

In this first example, we import a dataset and fit the Duane model. For the plot `log_scale` is set to `True`. The target MTBF is 35 which will give us the time to reach the target MTBF based on the model.

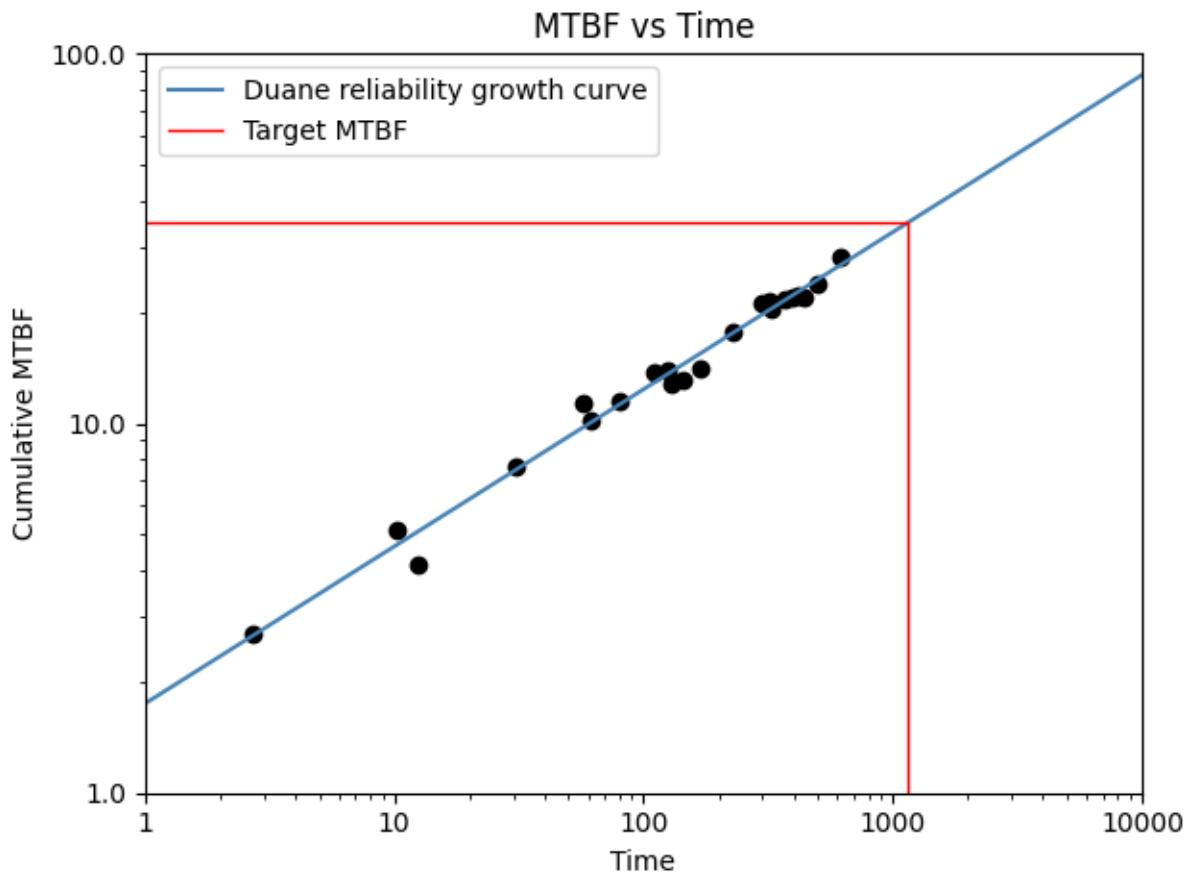
```
from reliability.Repairable_systems import reliability_growth
from reliability.Datasets import system_growth
import matplotlib.pyplot as plt
reliability_growth(times=system_growth().failures, model="Duane", target_MTBF=35, log_
    ↴scale=True)
plt.show()

"""
Duane reliability growth model parameters:
Alpha: 0.42531
A: 0.57338
Demonstrated MTBF (cumulative): 26.86511
Demonstrated MTBF (instantaneous): 46.74719
```

(continues on next page)

(continued from previous page)

Demonstrated failure intensity (cumulative): 0.037223
 Demonstrated failure intensity (instantaneous): 0.021392
 Time to reach target MTBF: 1154.77862
 ""



29.4 Example 2

In this second example, we are using the same failure times as the example above, but now we are fitting the Crow-AMSAA model. The MTBF plot is in linear scale since `log_scale` has not been specified and it defaults to False. Once again, the target MTBF of 35 is specified and the results tell us the time to reach this target.

```
from reliability.Repairable_systems import reliability_growth
from reliability.Datasets import system_growth
import matplotlib.pyplot as plt

reliability_growth(times=system_growth().failures, model="Crow-AMSAA", target_MTBF=35)
plt.show()

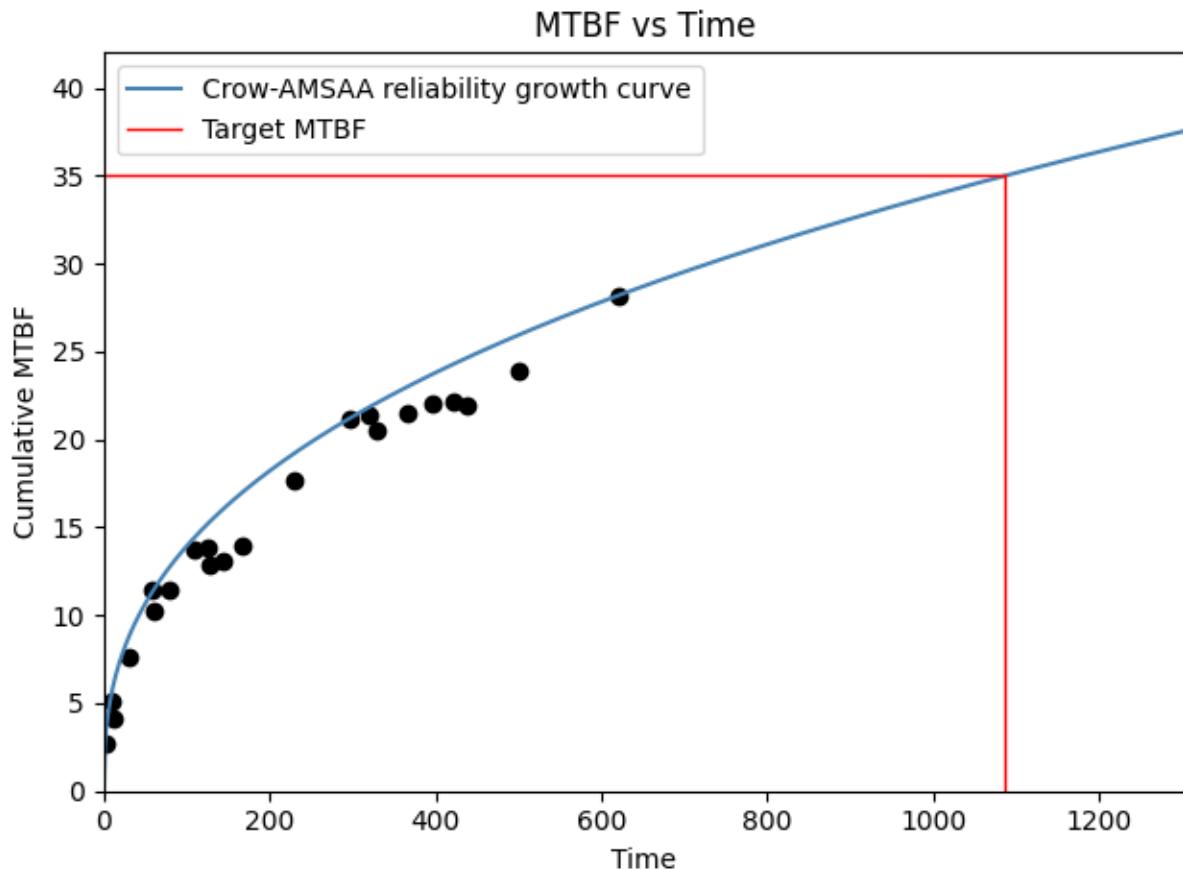
"""
Crow-AMSAA reliability growth model parameters:
Beta: 0.61421
Lambda: 0.42394
```

(continues on next page)

(continued from previous page)

```

Growth rate: 0.38579
Demonstrated MTBF (cumulative): 28.18182
Demonstrated MTBF (instantaneous): 45.883
Demonstrated failure intensity (cumulative): 0.035484
Demonstrated failure intensity (instantaneous): 0.021795
Time to reach target MTBF: 1087.18769
"
```



29.5 Example 3

In this third example, we will compare the two models in both linear space (left plot) and log space (right plot). The fit of the Duane model through the points seems much better than is achieved by the Crow-AMSAA model, though this depends on the dataset. The Crow-AMSAA model places a strong emphasis on the last data point and will always ensure the model passes through this point. Depending on whether the last data point sits above or below the average will affect whether the Crow-AMSAA model is more optimistic (higher) or pessimistic (lower) in its prediction of the achieved MTBF than that which is predicted by the Duane model.

```

from reliability.Repairable_systems import reliability_growth
from reliability.Datasets import automotive
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))

```

(continues on next page)

(continued from previous page)

```

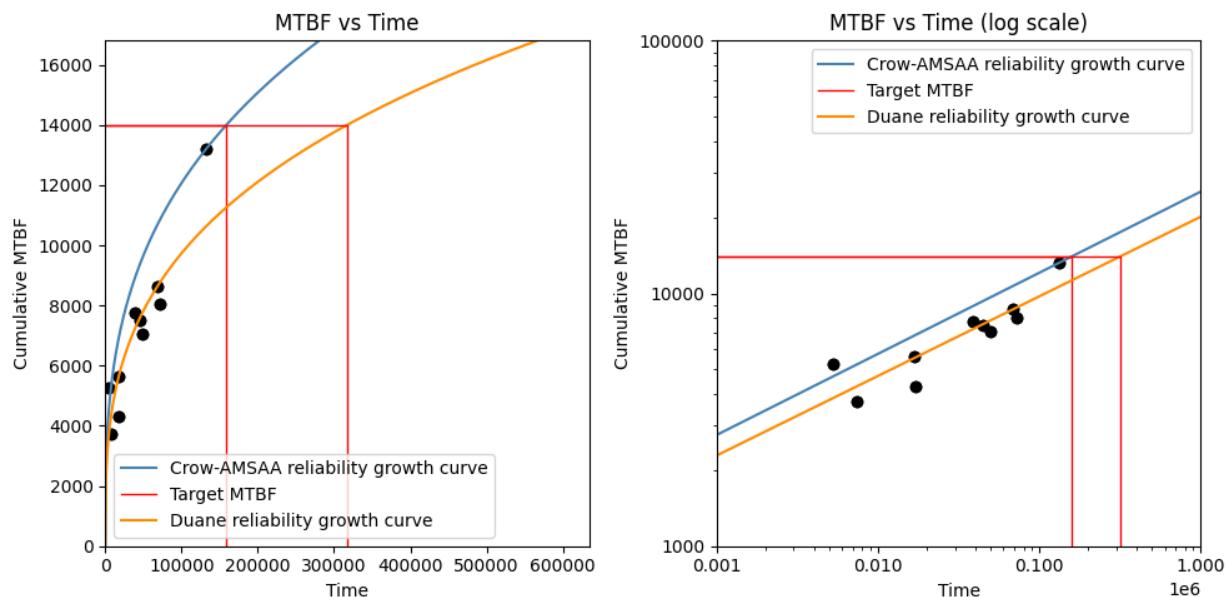
plt.subplot(121)
reliability_growth(times=automotive().failures, model="Crow-AMSAA", target_MTBF=14000)
reliability_growth(times=automotive().failures, model="Duane", target_MTBF=14000,color=
    ↪'darkorange')
plt.subplot(122)
reliability_growth(times=automotive().failures, model="Crow-AMSAA", target_MTBF=14000,
    ↪print_results=False,log_scale=True)
reliability_growth(times=automotive().failures, model="Duane", target_MTBF=14000,color=
    ↪'darkorange',print_results=False,log_scale=True)
plt.title('MTBF vs Time (log scale)')
plt.show()

"""

Crow-AMSAA reliability growth model parameters:
Beta: 0.67922
Lambda: 0.0033282
Growth rate: 0.32078
Demonstrated MTBF (cumulative): 13190
Demonstrated MTBF (instantaneous): 19419.22019
Demonstrated failure intensity (cumulative): 7.5815e-05
Demonstrated failure intensity (instantaneous): 5.1495e-05
Time to reach target MTBF: 158830.62457

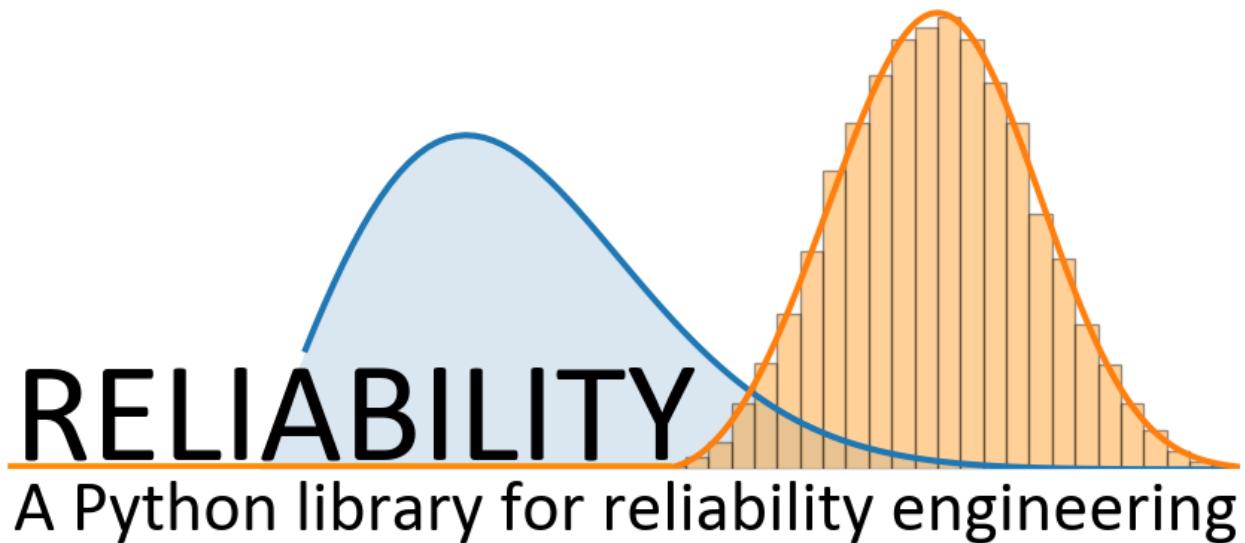
Duane reliability growth model parameters:
Alpha: 0.3148
A: 0.0038522
Demonstrated MTBF (cumulative): 10620.71841
Demonstrated MTBF (instantaneous): 15500.20608
Demonstrated failure intensity (cumulative): 9.4156e-05
Demonstrated failure intensity (instantaneous): 6.4515e-05
Time to reach target MTBF: 317216.14347
"""

```



 Note

The function `reliability_growth` was completely rewritten in v0.8.0 to match the method used by Reliasoft. Prior to v0.8.0, only the Duane model was available, and the values returned were for a model with a completely different parameterisation.



OPTIMAL REPLACEMENT TIME

When conducting maintenance planning, we must optimise the frequency of preventative maintenance (PM) for the minimum overall cost. If PM is conducted too frequently then we will have high costs, but if not conducted often enough then failures will result and we incur the higher cost of corrective maintenance (CM). Depending on the underlying failure distribution, it is possible to model these costs for a range of PM intervals, with the lowest cost per unit time resulting from the optimal replacement time. This function calculates the cost per unit time to determine how cost varies with replacement time. The cost model can be used for HPP (ie. the maintenance makes the system “as good as new”) or Power Law NHPP (ie. the maintenance makes the system “as good as old”). The default is for “as good as new”.

Cost in the above context should include all associated costs of PM and CM. These are not just the costs associated with parts and labor but may also include other costs such as system downtime (which may vary between PM and CM), loss of production output, customer satisfaction, etc. Some of these costs are difficult to quantify but organisations should strive to quantify all the costs of PM and system failure (requiring CM) if they want to accurately optimise their maintenance schedule.

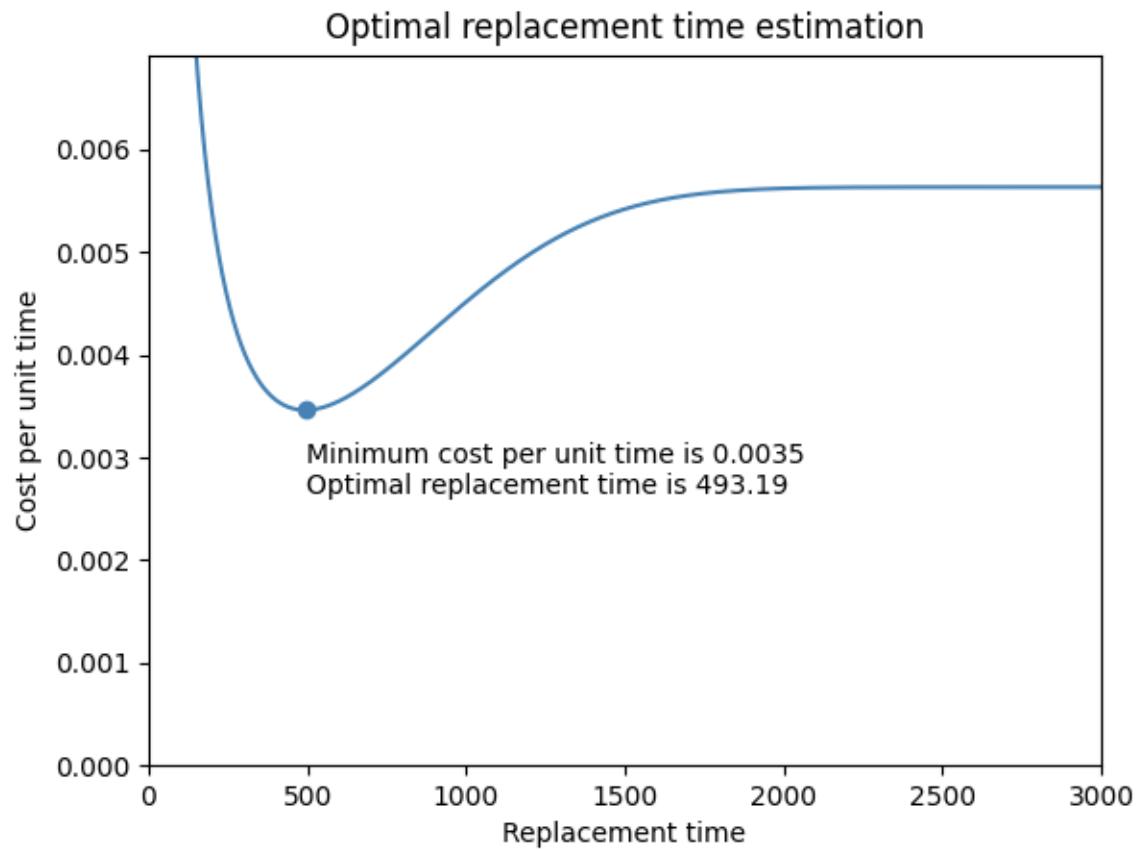
API Reference

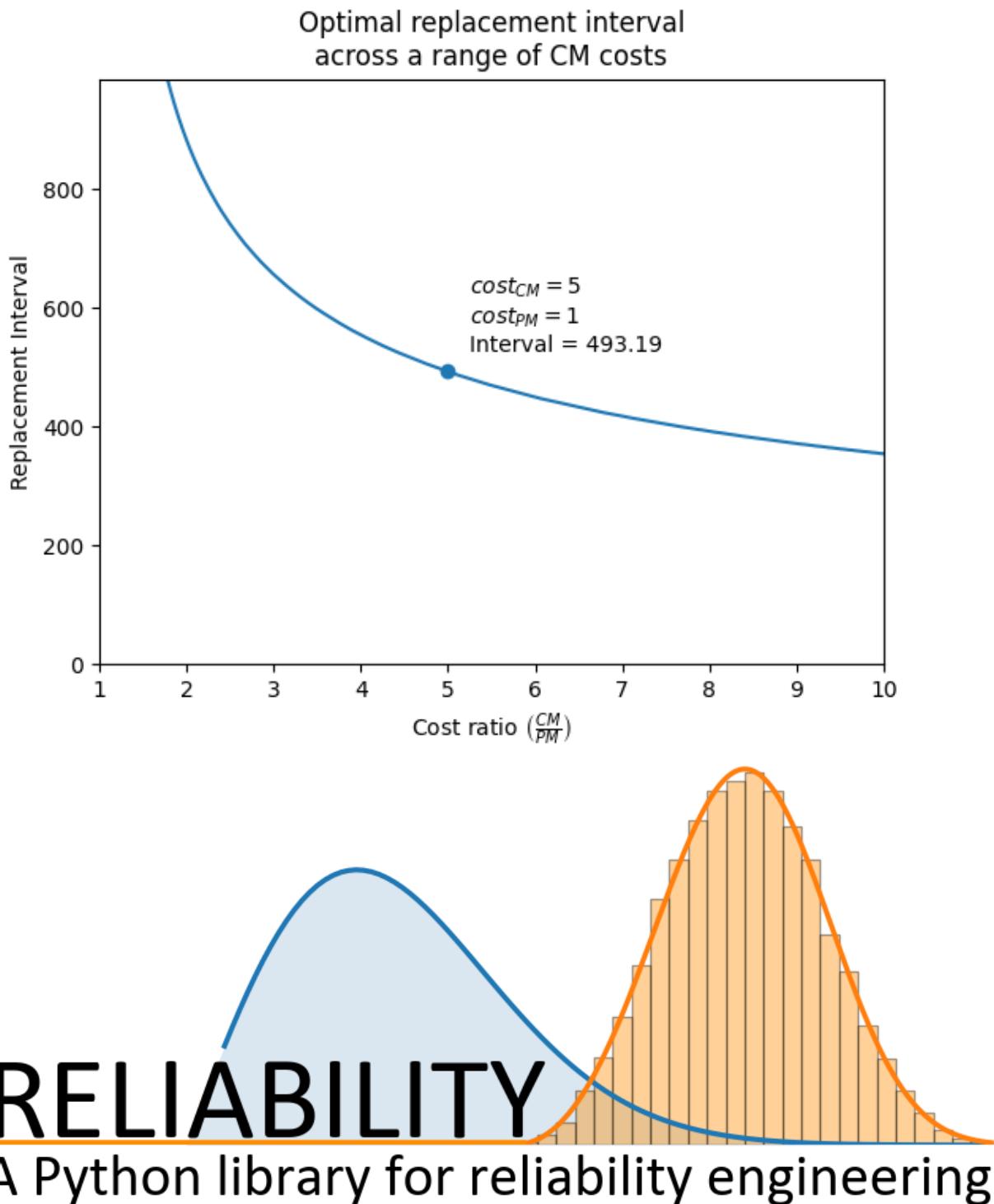
For inputs and outputs see the [API reference](#).

In the example below, we provide the cost of preventative maintenance (cost_PM), and the cost of corrective maintenance (cost_CM), as well as the Weibull parameters of the failure distribution. Leaving the default outputs, we obtain a plot of the cost per unit time and the printed results. This example is based of the example provided on the [reliasoft](#) article.

```
from reliability.Repairable_systems import optimal_replacement_time
import matplotlib.pyplot as plt
optimal_replacement_time(cost_PM=1, cost_CM=5, weibull_alpha=1000, weibull_beta=2.5, q=0)
plt.show()

"""
Results from optimal_replacement_time:
Cost model assuming as good as new replacement (q=0):
The minimum cost per unit time is 0.0035
The optimal replacement time is 493.19
""
```





CHAPTER
THIRTYONE

ROCOF

Rate of occurrence of failures (ROCOF) is used to model the trend (constant, increasing, decreasing) in the failure interarrival times. For a repairable system, we want the ROCOF to be improving (failure interarrival times to be increasing). As failure times can often appear quite random, it is necessary to conduct a statistical test to determine if there is a statistically significant trend, and if there is a trend we can then model that trend using a Power Law NHPP. The test for statistical significance is the Laplace test which compares the Laplace test statistic (U) with the z value (z_crit) from the standard Normal Distribution. If there is a statistically significant trend, the parameters of the model (Lambda_hat and Beta_hat) are calculated. By default the results are printed and a plot of the failure interarrival times and MTBF is plotted.

API Reference

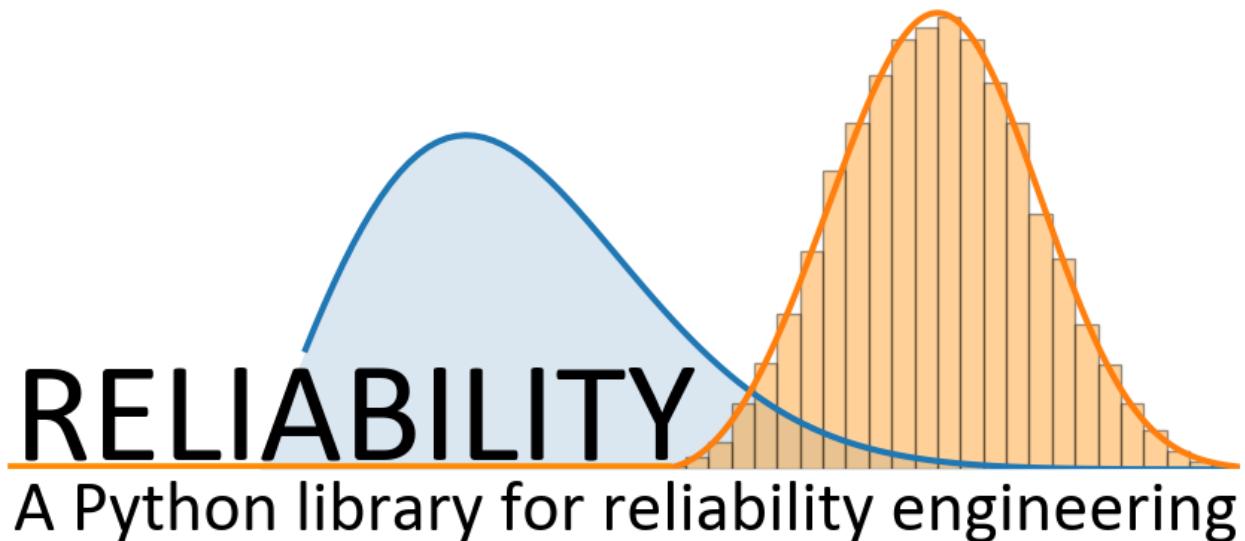
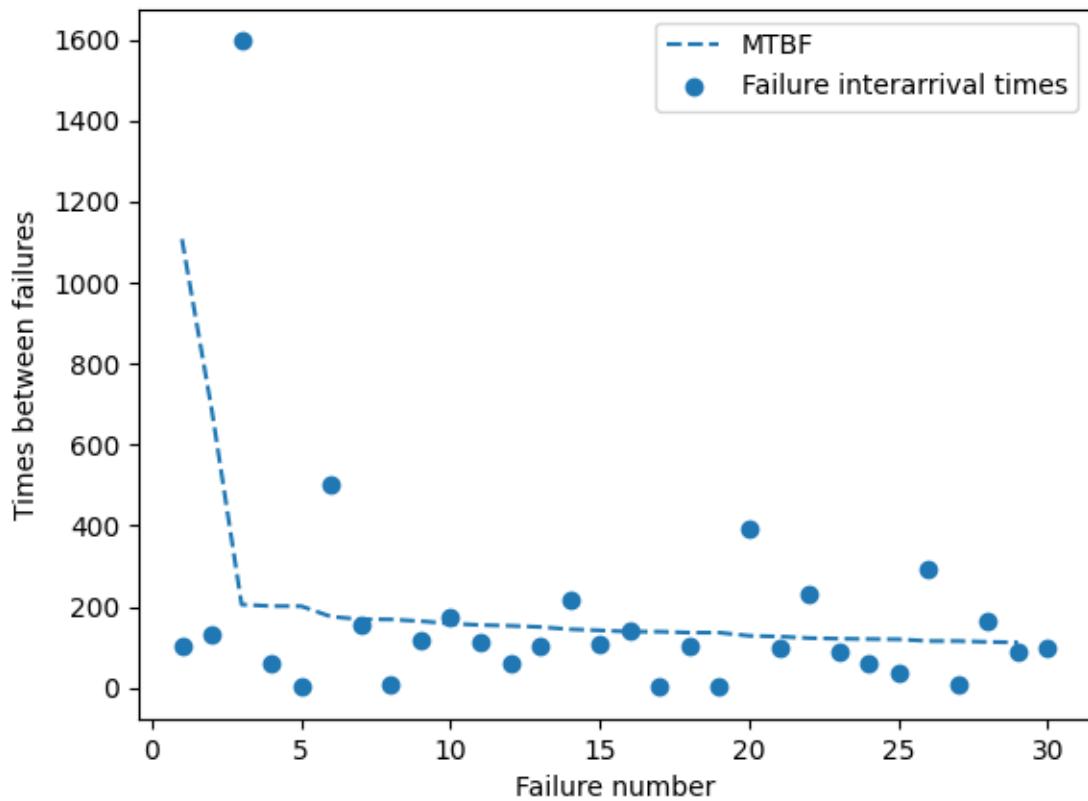
For inputs and outputs see the [API reference](#).

In the example below, we provide the failure interarrival times. The function will run the Laplace test using the default 95% confidence interval and then, when a trend is found, it will plot the MTBF based on the calculated NHPP Power Law model. $MTBF = 1/ROCOF$. This example is based on Example 5.11 (p275) from Reliability Engineering and Risk analysis listed in the [recommended resources](#).

```
from reliability.Repairable_systems import ROCOF
import matplotlib.pyplot as plt
t = [104, 131, 1597, 59, 4, 503, 157, 6, 118, 173, 114, 62, 101, 216, 106, 140, 1, 102, 3, 393, 96, 232, 89, 61,
     37, 293, 7, 165, 87, 99]
ROCOF(times_between_failures=t)
plt.show()

"""
Results from ROCOF analysis:
Laplace test results: U = 2.409, z_crit = (-1.96,+1.96)
At 95% confidence level the ROCOF is WORSENING. Assume NHPP.
ROCOF assuming NHPP has parameters: Beta_hat = 1.588 , Lambda_hat = 3.703e-05
"""
```

Failure interarrival times vs failure number
At 95% confidence level the ROCOF is WORSENING



MEAN CUMULATIVE FUNCTION

The Mean Cumulative Function (MCF) is a cumulative history function that shows the cumulative number of recurrences of an event, such as repairs over time. In the context of repairs over time, the value of the MCF can be thought of as the average number of repairs that each system will have undergone after a certain time. It is only applicable to repairable systems and assumes that each event (repair) is identical. For the non-parametric MCF it does not assume that each system's MCF is identical, but this assumption is made for the parametric MCF.

The shape of the MCF is a key indicator that shows whether the systems are improving, worsening, or staying the same over time. If the MCF is concave down (appearing to level out) then the system is improving. A straight line (constant increase) indicates it is staying the same. Concave up (getting steeper) shows the system is worsening as repairs are required more frequently as time progresses.

Obtaining the MCF from failure times is an inherently non-parametric process (similar to Kaplan-Meier), but once the values are obtained, a model can be fitted to obtain the parametric estimate of the MCF. Each of these two approaches is described below as they are performed by separate functions within reliabilityRepairable_systems.

Note that in some textbooks and academic papers the Mean Cumulative Function is also referred to as the Cumulative Intensity Function (CIF). These are two names for the same thing. If the shape of your MCF is more of an S than a single smooth curve, you may have a change in operating condition or in the repair effectiveness factor. This can be dealt with by splitting the MCF into segments, however, such models are more complex and are generally only found in academic literature.

32.1 Non-parametric MCF

The non-parametric estimate of the MCF provides both the estimate of the MCF and the confidence bounds at a particular time. The procedure to obtain the non-parametric MCF is outlined [here](#). The confidence bounds are the one-sided bounds as this was chosen to align with the method used by Reliasoft.

API Reference

For inputs and outputs see the [API reference](#).

32.2 Example 1

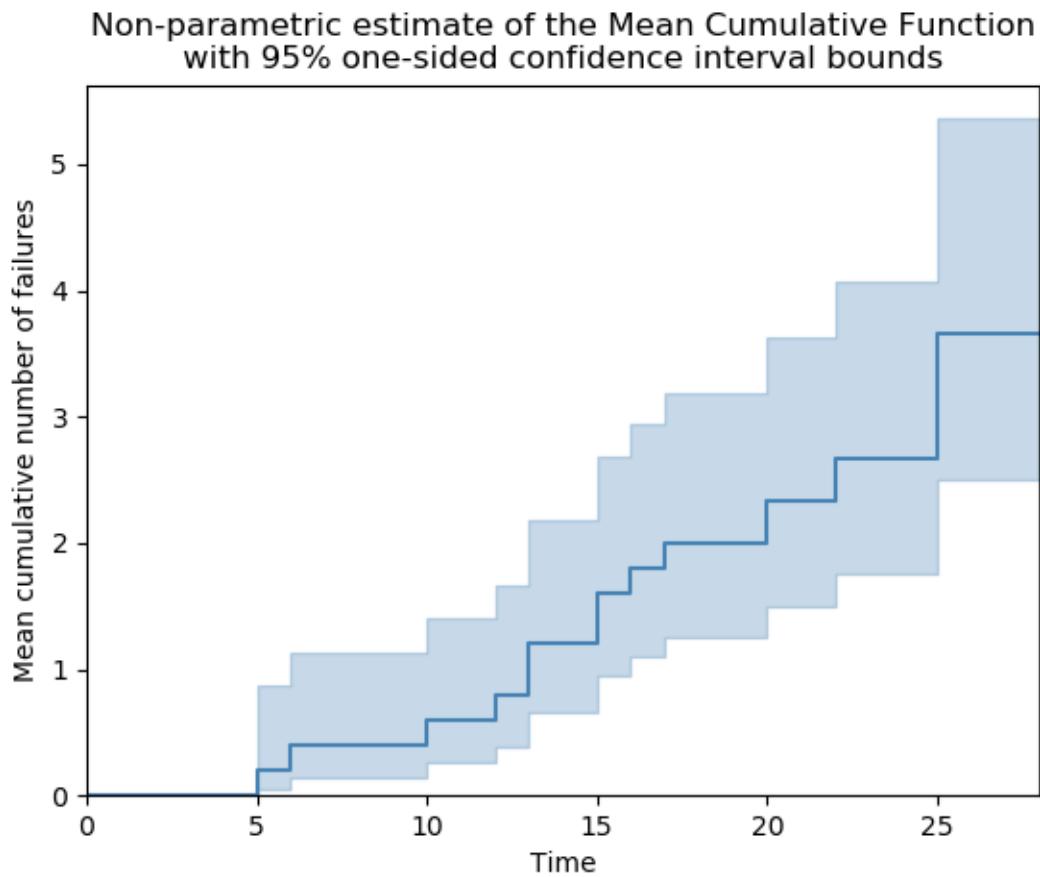
The following example is taken from an [example](#) provided by Reliasoft. The failure times and retirement times (retirement time is indicated by +) of 5 systems are:

Equipment ID	Months
1	5, 10, 15, 17+
2	6, 13, 17, 19+
3	12, 20, 25, 26+
4	13, 15, 24+
5	16, 22, 25, 28+

```
from reliability.Repairable_systems import MCF_nonparametric
import matplotlib.pyplot as plt
times = [[5, 10, 15, 17], [6, 13, 17, 19], [12, 20, 25, 26], [13, 15, 24], [16, 22, 25, 28]]
MCF_nonparametric(data=times)
plt.show()
```

```
"""
Mean Cumulative Function results (95% CI):
state  time  MCF_lower    MCF    MCF_upper  variance
F      5     0.0459299    0.2    0.870893    0.032
F      6     0.14134      0.4    1.13202     0.064
F      10    0.256603     0.6    1.40294     0.096
F      12    0.383374     0.8    1.66939     0.128
F      13    0.517916     1      1.93081     0.16
F      13    0.658169     1.2    2.18789     0.192
F      15    0.802848     1.4    2.44131     0.224
F      15    0.951092     1.6    2.69164     0.256
F      16    1.10229      1.8    2.93935     0.288
F      17    1.25598      2      3.18478     0.32
C      17
C      19
F      20    1.49896  2.33333  3.63215  0.394074
F      22    1.74856  2.66667  4.06684  0.468148
C      24
F      25    2.12259  3.16667  4.72431  0.593148
F      25    2.5071   3.66667  5.36255  0.718148
C      26
C      28
"""


```



32.3 Parametric MCF

The estimates of the parametric MCF are obtained using `MCF_nonparametric` as this is the procedure required to obtain the points for the plot. We use these points to fit a Non-Homogeneous Poisson Process (NHPP) parametric model of the form:

$$MCF(t) = \left(\frac{t}{\alpha}\right)^\beta$$

You may notice that this looks identical to the [Weibull CHF](#), but despite this similarity, they are entirely different functions and the alpha and beta parameters from the MCF cannot be applied to a Weibull distribution for fitting the repair times or repair interarrival times.

The purpose of fitting a parametric model is to obtain the shape parameter (β) which indicates the long term health of the system/s. If the MCF is concave down (<1) then the system is improving. A straight line ($=1$) indicates it is staying the same. Concave up (>1) shows the system is worsening as repairs are required more frequently as time progresses.

Many methods exist for fitting the model to the data. Within reliability, `scipy.optimize.curve_fit` is used which returns the covariance matrix and allows for the confidence intervals to be calculated using the appropriate [formulas](#).

API Reference

For inputs and outputs see the [API reference](#).

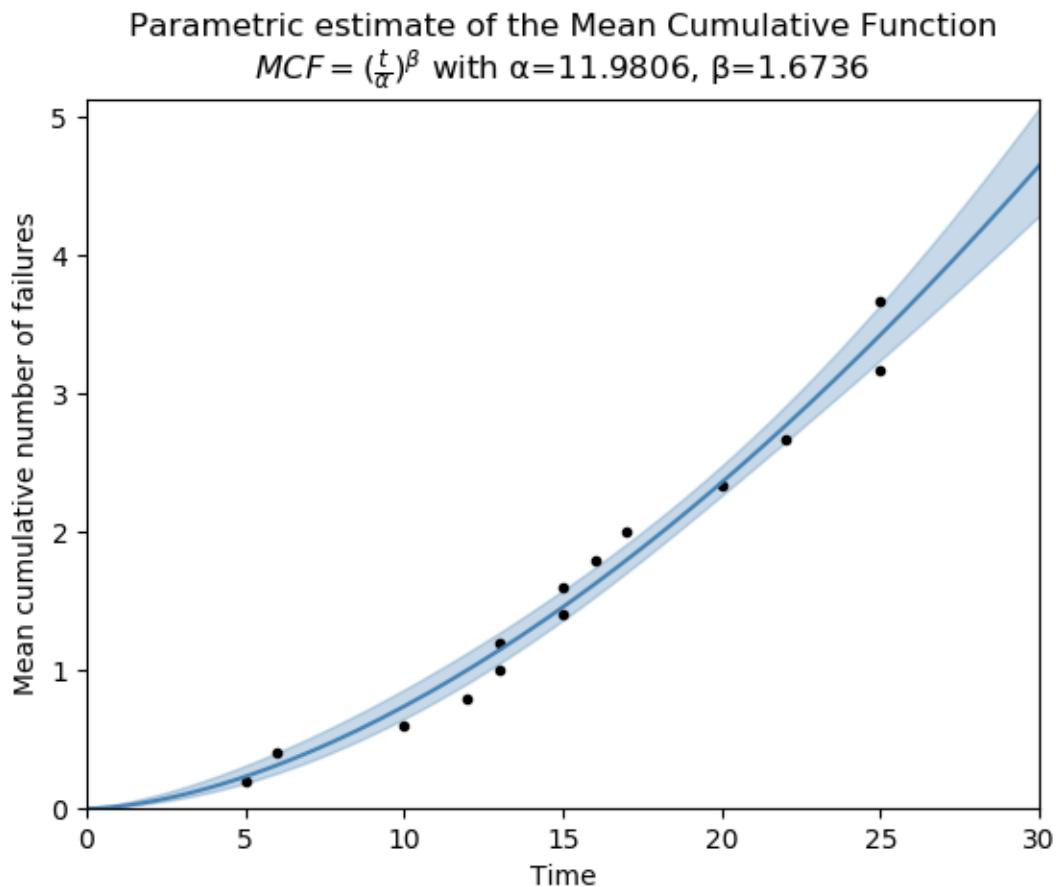
32.4 Example 2

The following example uses the same data as the MCF_nonparametric example provided above. From the output we can clearly see that the system is degrading over time as repairs are needed more frequently.

```
from reliability.Repairable_systems import MCF_parametric
import matplotlib.pyplot as plt
times = [[5, 10, 15, 17], [6, 13, 17, 19], [12, 20, 25, 26], [13, 15, 24], [16, 22, 25, 28]]
MCF_parametric(data=times)
plt.show()

"""
Mean Cumulative Function Parametric Model (95% CI):
MCF = (t/)^
Parameter Point Estimate Standard Error Lower CI Upper CI
Alpha      11.9806      0.401372    11.2192    12.7937
Beta       1.67362     0.0946537   1.49802    1.86981

Since Beta is greater than 1, the system repair rate is WORSENING over time.
""
```

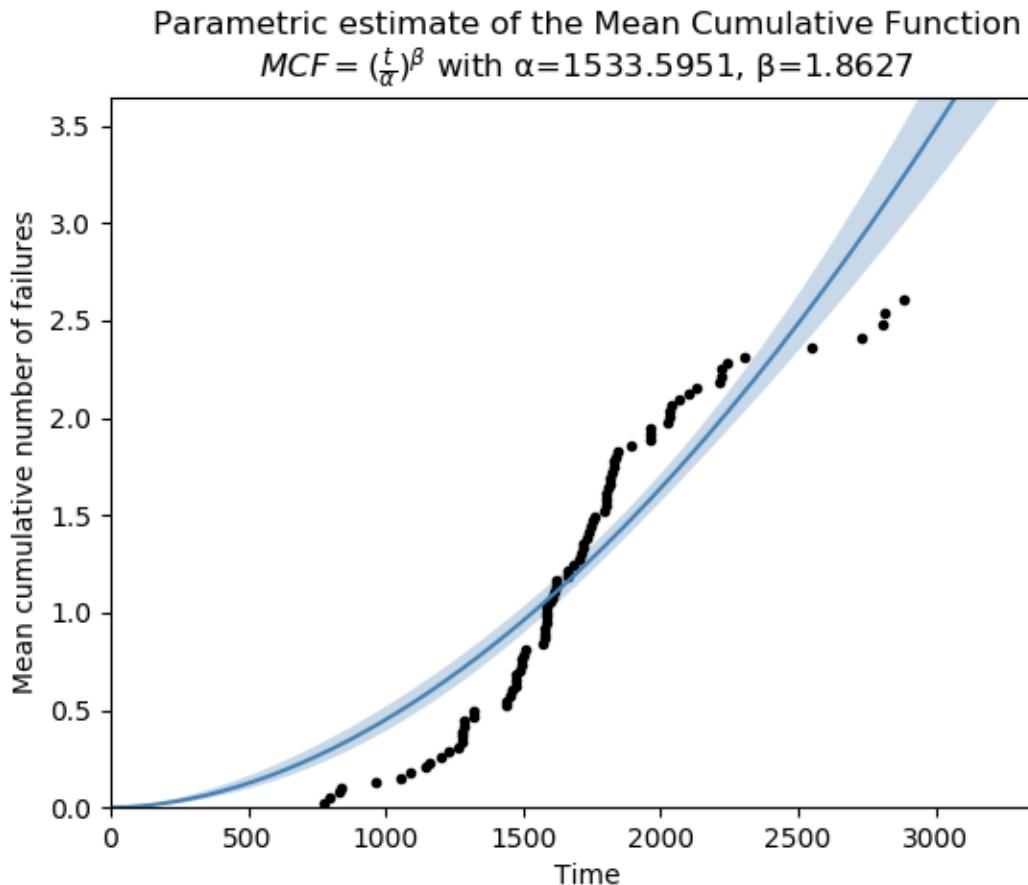


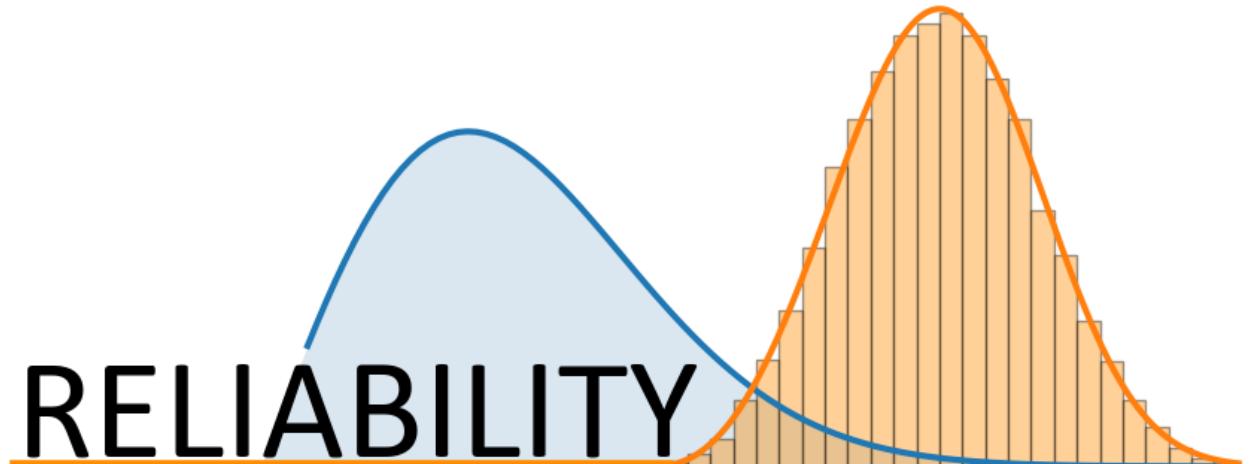
32.5 Example 3

The parametric model that is fitted to the MCF is not always an appropriate model. The example below shows data from a collection of systems, some of which are improving and some are worsening. The net effect is an S-shaped MCF. The power model used by MCF_parametric is not able to accurately follow an S-shaped dataset. In this case, the MCF_nonparametric model is more appropriate, though there are some other parametric models (discussed in the first paragraph) which may be useful to model this dataset.

```
from reliability.Repairable_systems import MCF_parametric
from reliability.Datasets import MCF_2
import matplotlib.pyplot as plt

times = MCF_2().times
MCF_parametric(data=times, print_results=False)
plt.show()
```





RELIABILITY
A Python library for reliability engineering

CHAPTER
THIRTYTHREE

ONE SAMPLE PROPORTION

This function calculates the upper and lower bounds of reliability for a given number of trials and successes. It is most applicable to analysis of test results in which there are only success/failure results and the analyst wants to know the reliability of the batch given those sample results.

API Reference

For inputs and outputs see the [API reference](#).

In this example, consider a scenario in which we have a large batch of items that we need to test for their reliability. The batch is large and testing is expensive so we will conduct the test on 30 samples. From those 30 samples, 29 passed the test. If the batch needs at least 85% reliability with a 95% confidence, then should we accept or reject the batch?

```
from reliability.Reliability_testing import one_sample_proportion
one_sample_proportion(trials=30, successes=29)

"""
Results from one_sample_proportion:
For a test with 30 trials of which there were 29 successes and 1 failures, the bounds on
reliability are:
Lower 95% confidence bound: 0.8278305443665873
Upper 95% confidence bound: 0.9991564290733695
""
```

The lower bound (with 95% confidence interval) on the reliability was 82.78%. Since this is below our requirement of 85%, then we should reject the batch.



RELIABILITY
A Python library for reliability engineering

CHAPTER
THIRTYFOUR

TWO PROPORTION TEST

This function determines if there is a statistically significant difference in the results from two different tests. Similar to the [One_sample_proportion](#), we are interested in using results from a success/failure test, but we are now interested in whether the difference in results is significant when comparing results between two tests.

API Reference

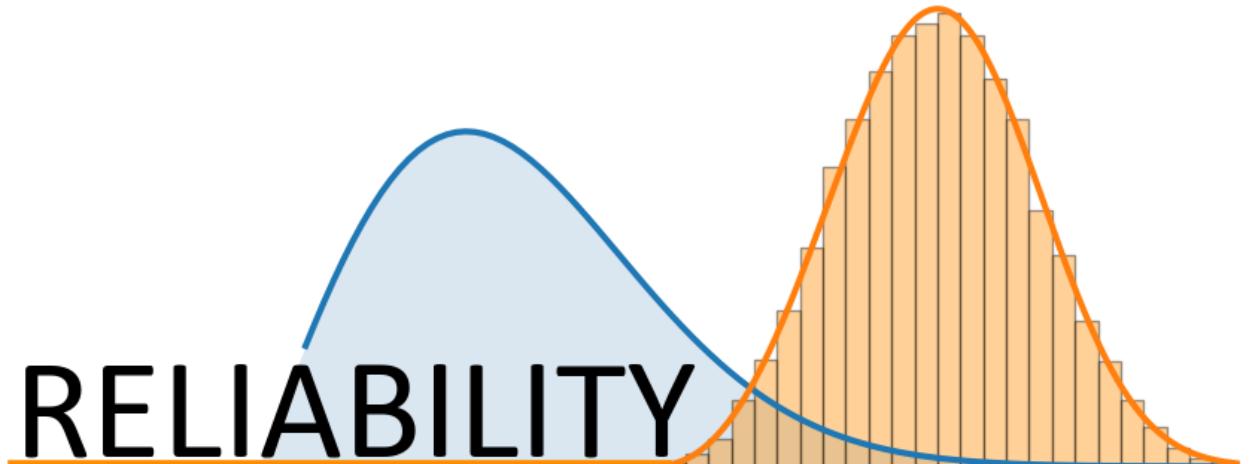
For inputs and outputs see the [API reference](#).

In this example, consider that sample 1 and sample 2 are batches of items that two suppliers sent you as part of their contract bidding process. You test everything each supplier sent you and need to know whether the reliability difference between suppliers is significant. At first glance, the reliability for sample 1 is $490/500 = 98\%$, and for sample 2 is $770/800 = 96.25\%$. Without considering the confidence intervals, we might be inclined to think that sample 1 is almost 2% better than sample 2. Lets run the two proportion test with the 95% confidence interval.

```
from reliability.Reliability_testing import two_proportion_test
two_proportion_test(sample_1_trials=500, sample_1_successes=490, sample_2_trials=800,
                     sample_2_successes=770)

"""
Results from two_proportion_test:
Sample 1 test results (successes/tests): 490/500
Sample 2 test results (successes/tests): 770/800
The 95% confidence bounds on the difference in these results is: -0.0004972498915250083
                     to 0.03549724989152493
Since the confidence bounds contain 0 the result is statistically non-significant.
""
```

Because the lower and upper bounds on the confidence interval includes 0, we can say with 95% confidence that there is no statistically significant difference between the suppliers based on the results from the batches supplied.



RELIABILITY
A Python library for reliability engineering

SAMPLE SIZE REQUIRED FOR NO FAILURES

The function `sample_size_no_failures` is used to determine the minimum sample size required for a test in which no failures are expected, and the desired outcome is the lower bound on the reliability based on the sample size and desired confidence interval.

API Reference

For inputs and outputs see the [API reference](#).

As an example, consider a scenario in which we want to be sure that a batch of LEDs meets the reliability target for on/off cycles. Testing is for the planned lifetime (1 million cycles) and tested items will have most or all of their lifetime used up during testing so we can't test everything. How many items from the batch do we need to test to ensure we achieve 99.9% reliability with a 95% confidence interval?

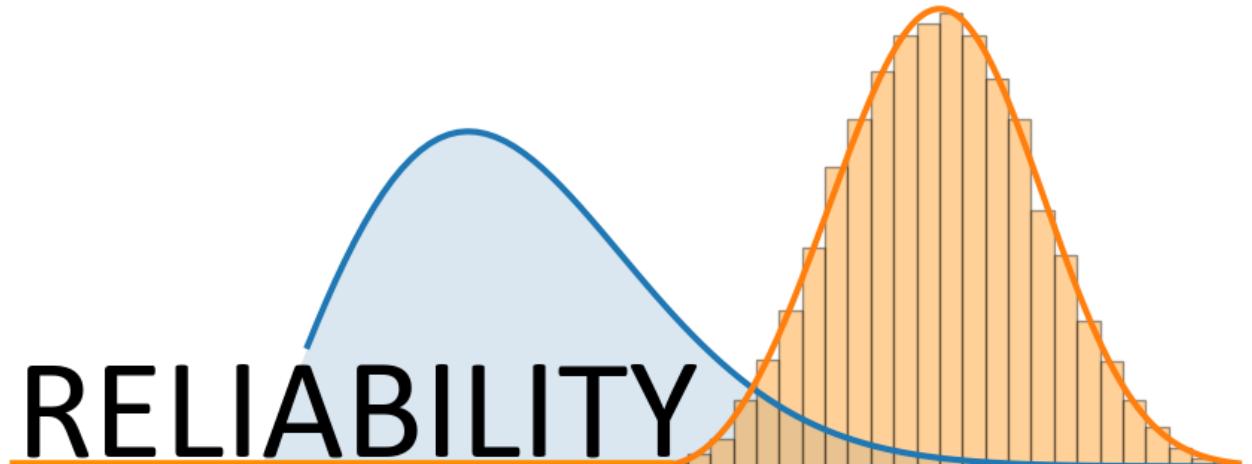
```
from reliability.Reliability_testing import sample_size_no_failures
sample_size_no_failures(reliability=0.999)

"""
Results from sample_size_no_failures:
To achieve the desired reliability of 0.999 with a 95% lower confidence bound, the
required sample size to test is 2995 items.

This result is based on a specified weibull shape parameter of 1 and an equivalent test
duration of 1 lifetime.
If there are any failures during this test, then the desired lower confidence bound will
not be achieved.
If this occurs, use the function Reliability_testing.one_sample_proportion to determine
the lower and upper bounds on reliability.
""
```

Based on this result, we need to test 2995 items from the batch and not have a single failure in order to be 95% confident that the reliability of the batch meets or exceeds 99.9%. If we tested each LED for more on/off cycles (lets say 3 million which is 3 lifetimes), then the number of successful results would only need to be 999. In this way, we can design our qualification test based on the desired reliability, confidence interval, and number of lifetimes that are tested to.

In the event that we suffer a single failure during this test, then we will need to adjust the testing method, either by finishing the testing and calculating the lower bound on reliability using the `one_sample_proportion` test, or by using a `sequential_sampling_chart`.



RELIABILITY
A Python library for reliability engineering

SEQUENTIAL SAMPLING CHART

A sequential sampling chart provides decision boundaries so that a success/failure test may be stopped as soon as there have been enough successes or enough failures to exceed the decision boundary. The decision boundary is calculated based on four parameters; producer's quality, consumer's quality, producer's risk, and consumer's risk. Producer's risk is the chance that the consumer rejects a batch when they should have accepted it. Consumer's risk is the chance that the consumer accepts a batch when they should have rejected it. We can also consider the producer's and consumer's quality to be the desired reliability of the sample, and the producer's and consumer's risk to be 1-confidence interval that the sample test result matches the population test result.

API Reference

For inputs and outputs see the [API reference](#).

In the example below, we use the inputs $p1=0.01$, $p2=0.10$, $\alpha=0.05$, $\beta=0.10$. The resulting decision boundaries are plotted, and the test results that we have supplied are also plotted as a stepped line. The plot shows that after our 3rd failure, the test should be stopped as the batch can be rejected. The dataframe of results is also printed by default. In this dataframe, the value of x is used to replace impossible numbers, ie. we cannot reject 2 failures if we have only conducted 1 inspection. This example is based on an example in the [Engineering statistics handbook](#) published online by NIST.

(continues on next page)

(continued from previous page)

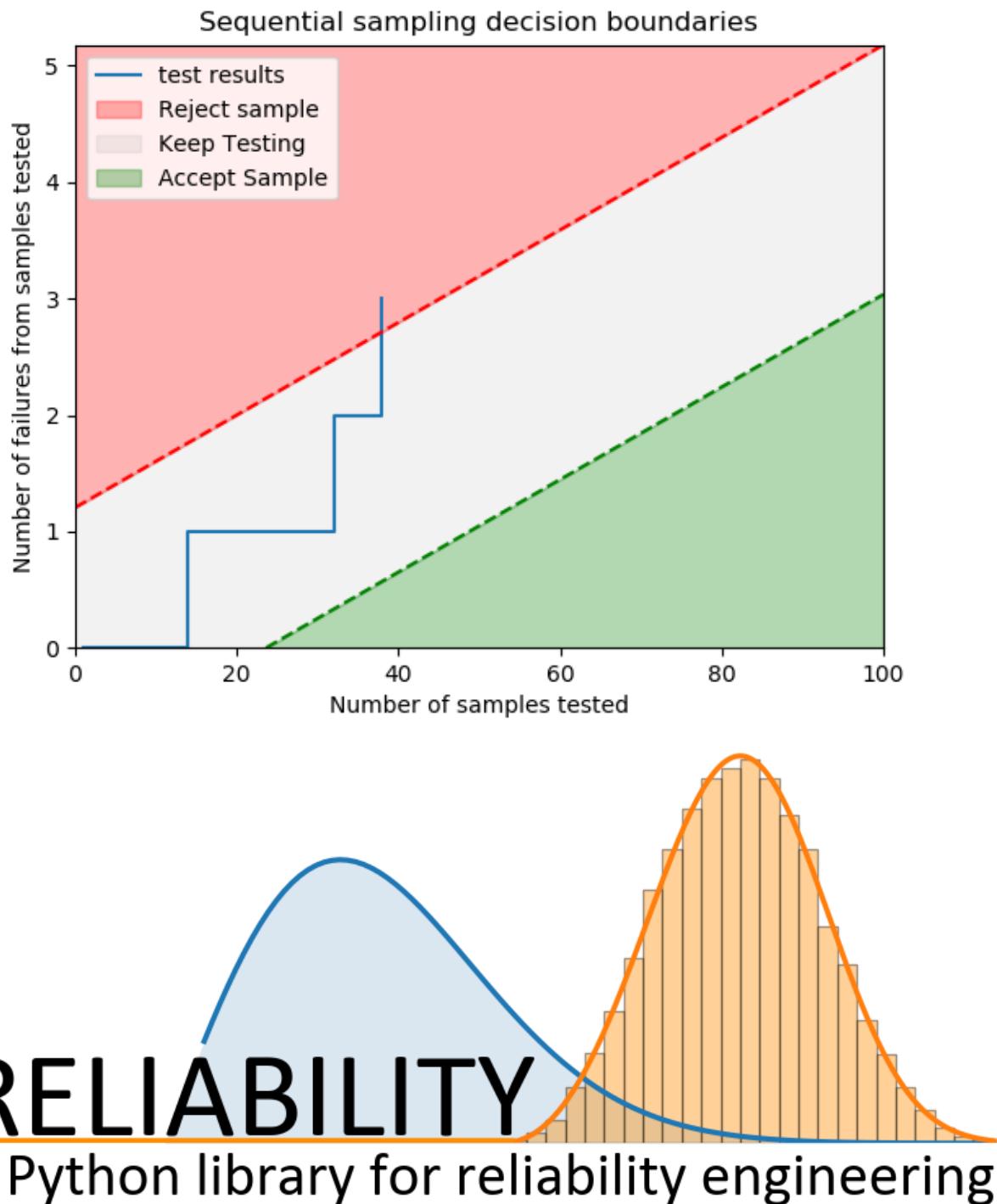
12	x	2
13	x	2
14	x	2
15	x	2
16	x	2
17	x	2
18	x	2
19	x	2
20	x	3
21	x	3
22	x	3
23	x	3
24	0	3
25	0	3
26	0	3
27	0	3
28	0	3
29	0	3
30	0	3
31	0	3
32	0	3
33	0	3
34	0	3
35	0	3
36	0	3
37	0	3
38	0	3
39	0	3
40	0	3
41	0	3
42	0	3
43	0	3
44	0	3
45	0	3
46	0	4
47	0	4
48	0	4
49	1	4
50	1	4
51	1	4
52	1	4
53	1	4
54	1	4
55	1	4
56	1	4
57	1	4
58	1	4
59	1	4
60	1	4
61	1	4
62	1	4
63	1	4

(continues on next page)

(continued from previous page)

64	1	4
65	1	4
66	1	4
67	1	4
68	1	4
69	1	4
70	1	4
71	1	5
72	1	5
73	1	5
74	2	5
75	2	5
76	2	5
77	2	5
78	2	5
79	2	5
80	2	5
81	2	5
82	2	5
83	2	5
84	2	5
85	2	5
86	2	5
87	2	5
88	2	5
89	2	5
90	2	5
91	2	5
92	2	5
93	2	5
94	2	5
95	2	5
96	2	6
97	2	6
98	2	6
99	2	6
100	3	6

"



CHAPTER
THIRTYSEVEN

RELIABILITY TEST PLANNER

This function provides a solver to determine the parameters of a reliability test when given 3 out of the 4 unknowns (lower confidence bound on MTBF, test duration, number of failures, confidence interval).

The underlying assumption is that the failures follow an exponential distribution (ie. failures occur randomly and the hazard rate does not change with age). Using this assumption, the Chi-squared distribution is used to find the lower confidence bound on MTBF for a given test duration, number of failures, and specified confidence interval using the formula:

$$MTBF = \frac{2T}{\chi^2\left(\frac{(1-CI)}{n}, 2F+p\right)}$$

Where:

- MTBF = Mean time between failures (same as mean time to failure (MTTF) when the hazard rate is constant as it is here). Note that this is the lower confidence interval on MTBF. If you want the point estimate then specify CI=0.5 and one_sided=False.
- T = Test duration (this is the total time on test across all units being tested)
- CI = Confidence interval (the confidence interval to be used for the lower bound on the MTBF)
- F = number of failures during the test
- n = adjustment for one sided (n=1) or two sided (n=2) test
- p = adjustment for time terminated (p=2) or failure terminated (p=0) test

The above formula can be rearranged, or solved iteratively to determine any of these 4 parameters when given the other 3. The user must specify any 3 out of the 4 variables (not including one_sided, print_results, or time_terminated) and the remaining variable will be calculated. Note the difference between the one-sided and two-sided confidence intervals which are specified using the input one_sided=True/False described below. A description of the difference between one-sided and two-sided confidence intervals is provided at the end of this page. The formula above can be used for either a time terminated test (where the test was stopped at a particular time which was not related to the number of failures) or a failure terminated test (where the test was stopped after a particular number of failures such as all items failing). This setting is controlled using the argument time_terminated which defaults to True.

A similar calculator is available in the [reliability analytics toolkit](#).

 **API Reference**

For inputs and outputs see the [API reference](#).

37.1 Example 1

In the example below, we have done a time-terminated reliability test for 19520 hours (units are not important here as it may be days, cycles, rounds, etc.). During the test there were 7 failures. We want to know the MTBF that was achieved during the test within an 80% confidence (two-sided).

37.2 Example 2

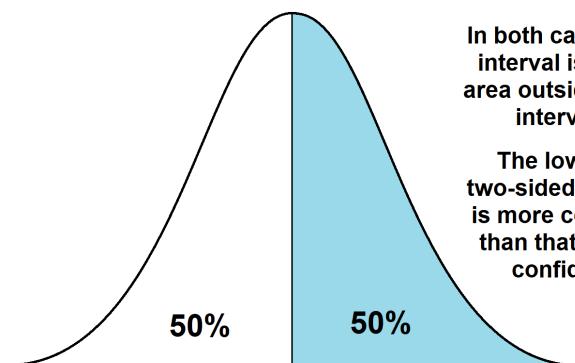
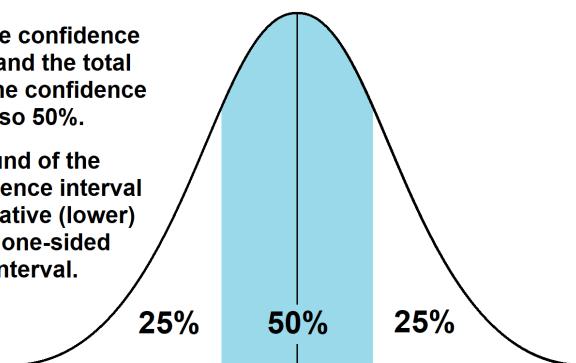
In this second example we want to know the lower confidence bound on the MTBF for a reliability test that has just been conducted. During this test there were 6 failures observed over the 10000km test track and our desired confidence interval is 80%. Note that the output is suppressed by setting `print_results=False` and then the MTBF is accessed by name from the output object.

```
from reliability.Reliability_testing import reliability_test_planner
output = reliability_test_planner(number_of_failures=6, test_duration=10000, CI=0.8, ↴
    ↴print_results=False)
print(output.MTBF)

"""
1101.8815940201118
""
```

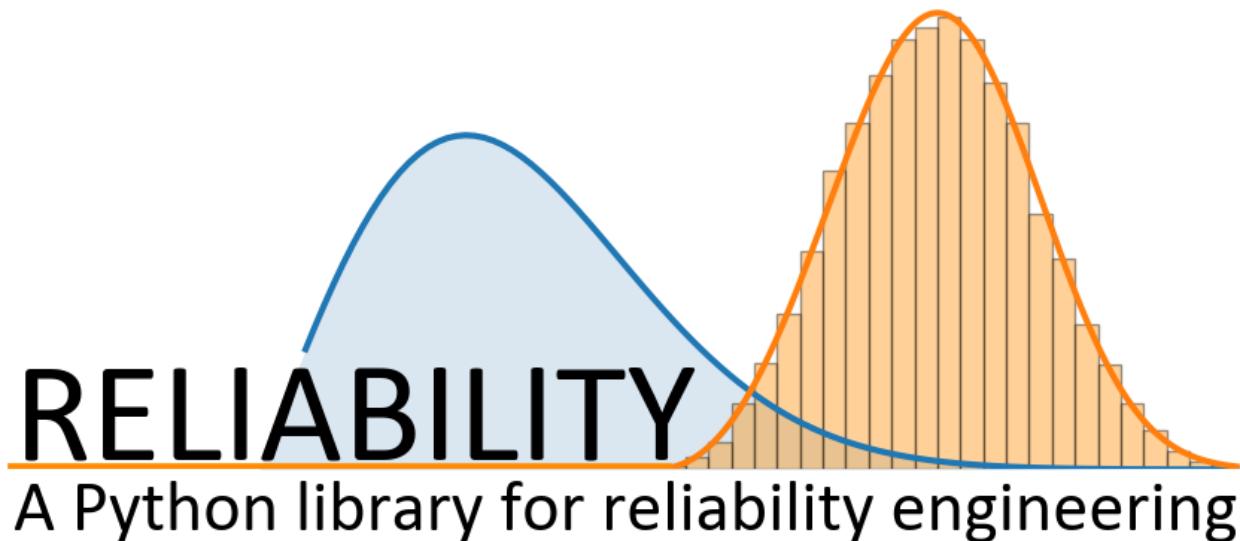
37.3 One-sided vs two-sided confidence interval

The below image illustrates the difference between one-sided and two-sided confidence interval. You can use either the one-sided or two-sided interval when you are seeking only the lower bound, but it is essential to understand that they will give very different results for the same CI. They will give equivalent results if the CI is set appropriately (eg. 90% one-sided is the same as 80% two-sided). If you are unsure which to use, the more conservative approach is to use the two-sided interval. If you want the point estimate, use the one-sided interval with a CI=0.5.

One-sided confidence interval**Two-sided confidence interval**

In both cases, the confidence interval is 50% and the total area outside of the confidence interval is also 50%.

The lower bound of the two-sided confidence interval is more conservative (lower) than that of the one-sided confidence interval.



RELIABILITY TEST DURATION

This function is an extension of the `reliability_test_planner` which allows users to calculate the required duration for a reliability test to achieve the specified producers and consumers risks. This is done based on the specified MTBF (mean time between failure) required and MTBF design (the MTBF that the manufacturer believes the system has been designed to).

This type of determination must be made when organisations looking to test an item are uncertain of how much testing is required, but they know the amount of risk they are willing to accept as well as the MTBF required and the MTBF to which the item has been designed.

API Reference

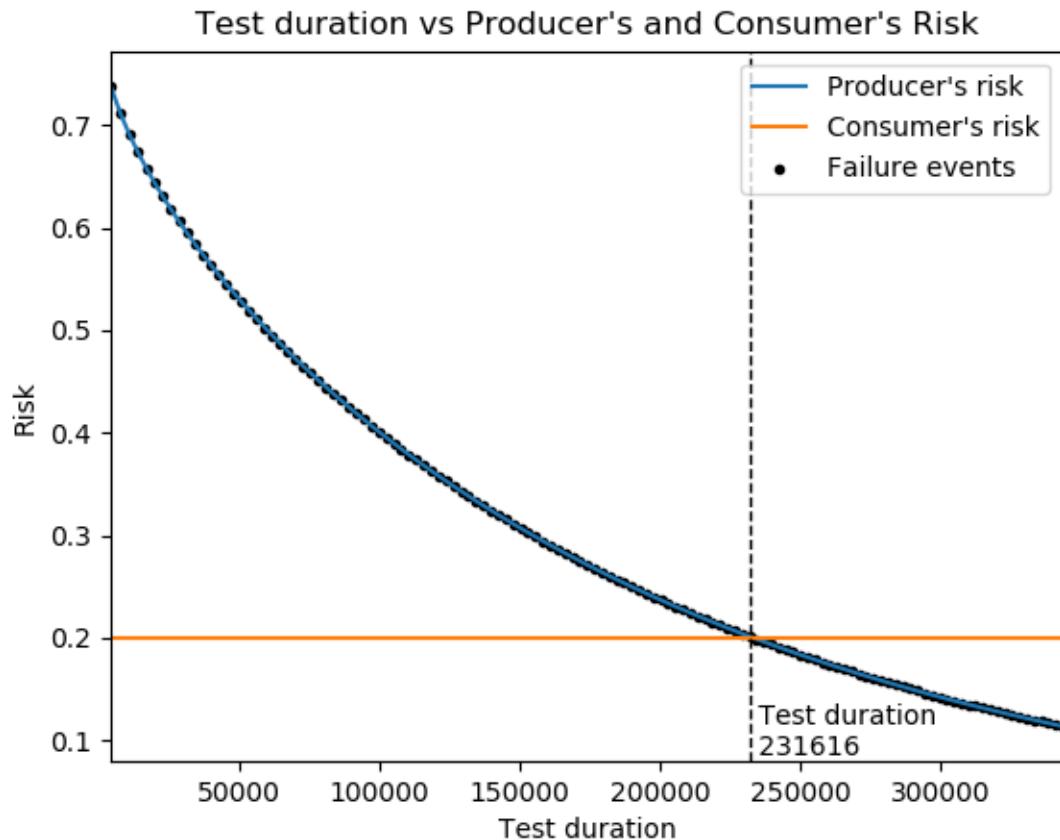
For inputs and outputs see the [API reference](#).

38.1 Example 1

In the example below the consumer requires a vehicle to achieve an MTBF of 2500km and is willing to accept 20% risk that they accept a bad item when they should have rejected it. The producer has designed the vehicle to have an MTBF of 3000km and they are willing to accept 20% risk that the consumer rejects a good item when they should have accepted it. How many kilometres should the reliability test be? Using the function we find the test needs to be 231616 km. Note that this duration is the total time on test and may be split across several vehicles. See the discussion points below on whether to split the test duration up among multiple vehicles.

```
from reliability.Reliability_testing import reliability_test_duration
import matplotlib.pyplot as plt
reliability_test_duration(MTBF_required=2500, MTBF_design=3000, consumer_risk=0.2,
                           producer_risk=0.2)
plt.show()

"""
Reliability Test Duration Solver for time-terminated test:
Required test duration: 231615.79491309822
Specified consumer's risk: 0.2
Specified producer's risk: 0.2
Specified MTBF required by the consumer: 2500
Specified MTBF designed to by the producer: 3000
""
```



Splitting the test up among several vehicles has both positives and negatives as follows:

Advantages of testing on only a few vehicles

- We can observe the failure behaviour later in life. If we tested 50 vehicles to 4632km each then we are unlikely to observe failures that typically occur after 50000km.
- It costs us less in vehicles since each vehicle has some of its life consumed during the testing.
- We may not have many items available for testing, particularly if it is a prototype that is yet to enter full production.
- We may not have many test facilities available so keeping the number of vehicles to a small number is often limited by the availability of the test facilities.

Advantages of splitting the testing up between many vehicles

- It is more representative of the population since all the testing on a single vehicle may not be accurate if that one vehicle is above or below average quality compared to the rest of the vehicles.
- Testing can be done faster which also means less cost on testing facilities. Reliability testing is often something that Project managers will put pressure on to cut if the project is falling behind schedule so using more vehicles may be a way to get the same amount of reliability testing done faster.

38.2 How does the algorithm work?

The underlying method is as follows:

Step 1) Begin with failures = 1. This will be iterated later.

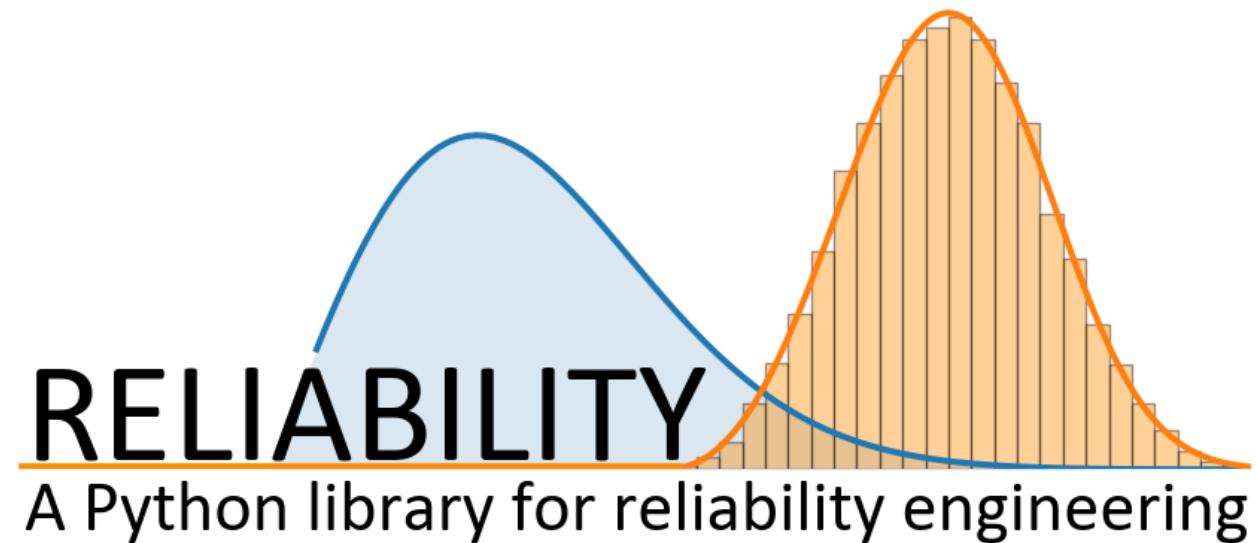
Step 2) Using the function `Reliability_testing.reliability_test_planner`, we set `CI = 1-consumer_risk`, `MTBF = MTBF_required` to solve for the `test_duration` that is achieved by this test. This is the test duration required if there was 1 failure which would give the specified MTBF required and specified consumer's risk.

Step 3) We again use the function `Reliability_testing.reliability_test_planner` but this time we set `MTBF = MTBF_design` and use the `test_duration` as the output from step 2. Still keeping failures = 1 we are solving for the `CI` achieved. This is effectively the producer's risk for the given `test_duration` and number of failures.

Step 4) The answer is higher than the specified producer's risk, so we now repeat steps 2 and 3 by increasing the number of failures by 1 each iteration. This is continued until the producer's risk is below what was specified. We then go back 1 failure since it is standard that the producer's risk can't be below what was specified (or the consumer may think the producer is cheating by lowering their risk).

We now have a value for `test_duration` that will give our required outputs in both equations. We also happen to arrive at the number of failures, though this is not particularly relevant since it is just part of the solution process and the actual number of failures will be determined based on the conduct of the reliability test.

The plot that is produced by `Reliability_testing.reliability_test_duration` displays a scatter plot at each failure. Since the number of failures must be an integer, we get results for reliability test durations that go in steps. The result returned corresponds to the `test_duration` at the last failure before the producer's risk dropped below what was specified. Also note that if the consumer's risk is different from the producer's risk, the solution for `test_duration` will not occur near the point on the graph where producer's risk and consumer's risk are equal.



CHI-SQUARED TEST

The Chi-squared test is a statistical test for goodness of fit to determine whether we can accept or reject the hypothesis that the data is from the specified distribution at the specified level of significance. This method is not a means of comparing distributions (which can be done with AICc, BIC, and AD), but instead allows us to accept or reject a hypothesis that data come from a distribution. Note that the result is sensitive to the bins. For this reason, it is recommended to leave bins as the default value.

The procedure for the test involves comparing the fitted CDF (from a hypothesised distribution) against the empirical CDF (from a cumulative histogram of the data). As with all histograms, the exact shape of the histogram depends on the bins. The difference between the fitted CDF and the empirical CDF is used to find the chi-squared statistic. The specified level of significance (analogous to confidence level), the number of parameters in the hypothesised distribution, and the number of data points is used to obtain the chi-squared critical value from the chi-squared distribution. By comparing the chi-squared statistic with the chi-squared critical value, we can determine whether the hypothesis (that the data are from the specified distribution) should be rejected or accepted. The acceptance criteria is when the the chi-squared statistic is below the critical value.

API Reference

For inputs and outputs see the [API reference](#).

In the example below we import a dataset called mileage which contains 100 values that appear to be normally distributed. Using the function chi2test we can determine whether we should accept the hypothesis that the data are from a Normal distribution with parameters $\mu=30011$ and $\sigma=10472$. This example is based on Example 2.31 (page 63) of Reliability Engineering and Risk Analysis (listed in [recommended resources](#)).

```
from reliability.Datasets import mileage
from reliability.Distributions import Normal_Distribution
from reliability.Reliability_testing import chi2test
import numpy as np
import matplotlib.pyplot as plt

data = mileage().failures
dist = Normal_Distribution(mu=30011, sigma=10472)
bins = [0, 13417, 18104, 22791, 27478, 32165, 36852, 41539, 46226, np.inf] #it is not
#necessary to specify the bins and leaving them unspecified is usually best
chi2test(distribution=dist, data=data, bins=bins)
plt.show()

"""

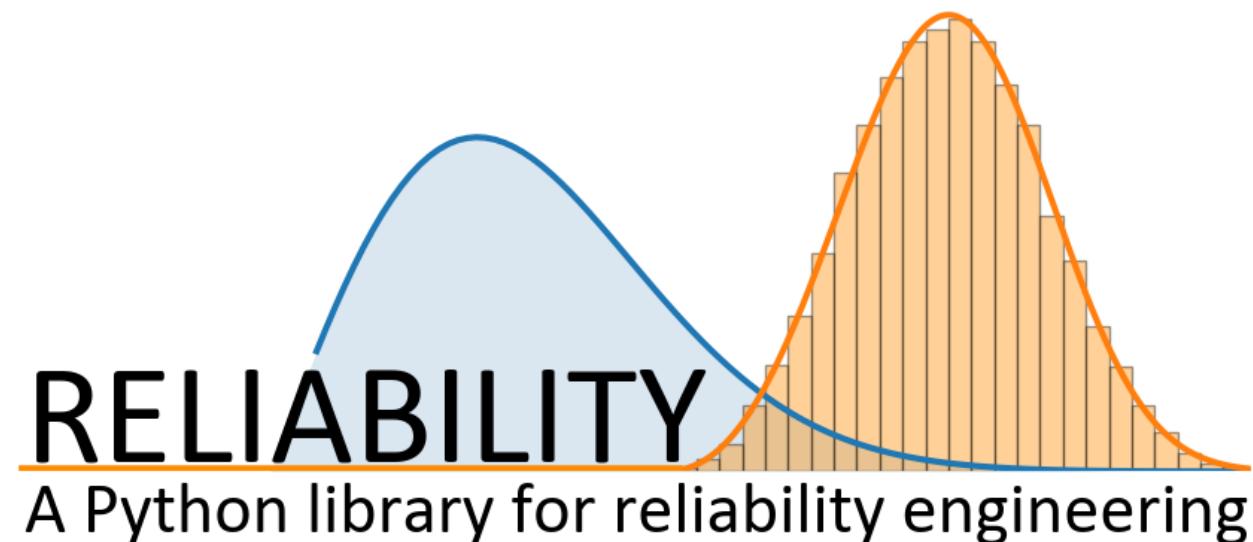
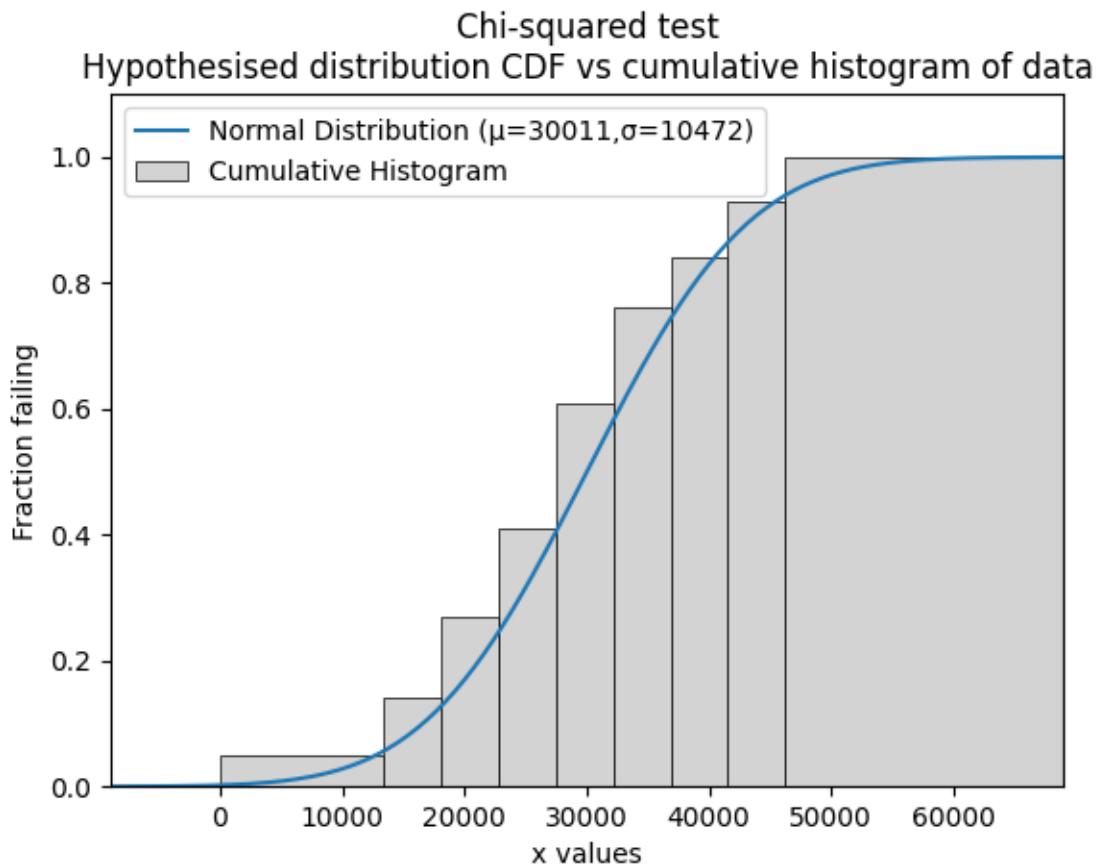
Results from Chi-squared test:
Chi-squared statistic: 3.1294947845652
```

(continues on next page)

(continued from previous page)

Chi-squared critical value: 12.591587243743977

At the 0.05 significance level, we can ACCEPT the hypothesis that the data comes from a \sim Normal Distribution ($=30011, =10472$)
 ""



KOLMOGOROV-SMIRNOV TEST

The Kolmogorov-Smirnov test is a statistical test for goodness of fit to determine whether we can accept or reject the hypothesis that the data is from the specified distribution at the specified level of significance. This method is not a means of comparing distributions (which can be done with AICc, BIC, and AD), but instead allows us to accept or reject a hypothesis that data come from a distribution. Unlike the [chi-squared test](#), the Kolmogorov-Smirnov test does not depend on the bins of a histogram, therefore making it a more consistent goodness of fit.

The procedure for the test involves comparing the fitted CDF (from a hypothesised distribution) against the empirical CDF (calculated using a rank order of the data of the form i/n). The difference between the fitted CDF and the empirical CDF is used to find the Kolmogorov-Smirnov statistic. The specified level of significance (analogous to confidence level) and the number of data points is used to obtain the Kolmogorov-Smirnov critical value from the Kolmogorov-Smirnov distribution. By comparing the Kolmogorov-Smirnov statistic with the Kolmogorov-Smirnov critical value, we can determine whether the hypothesis (that the data are from the specified distribution) should be rejected or accepted. The acceptance criteria is when the the Kolmogorov-Smirnov statistic is below the critical value.

API Reference

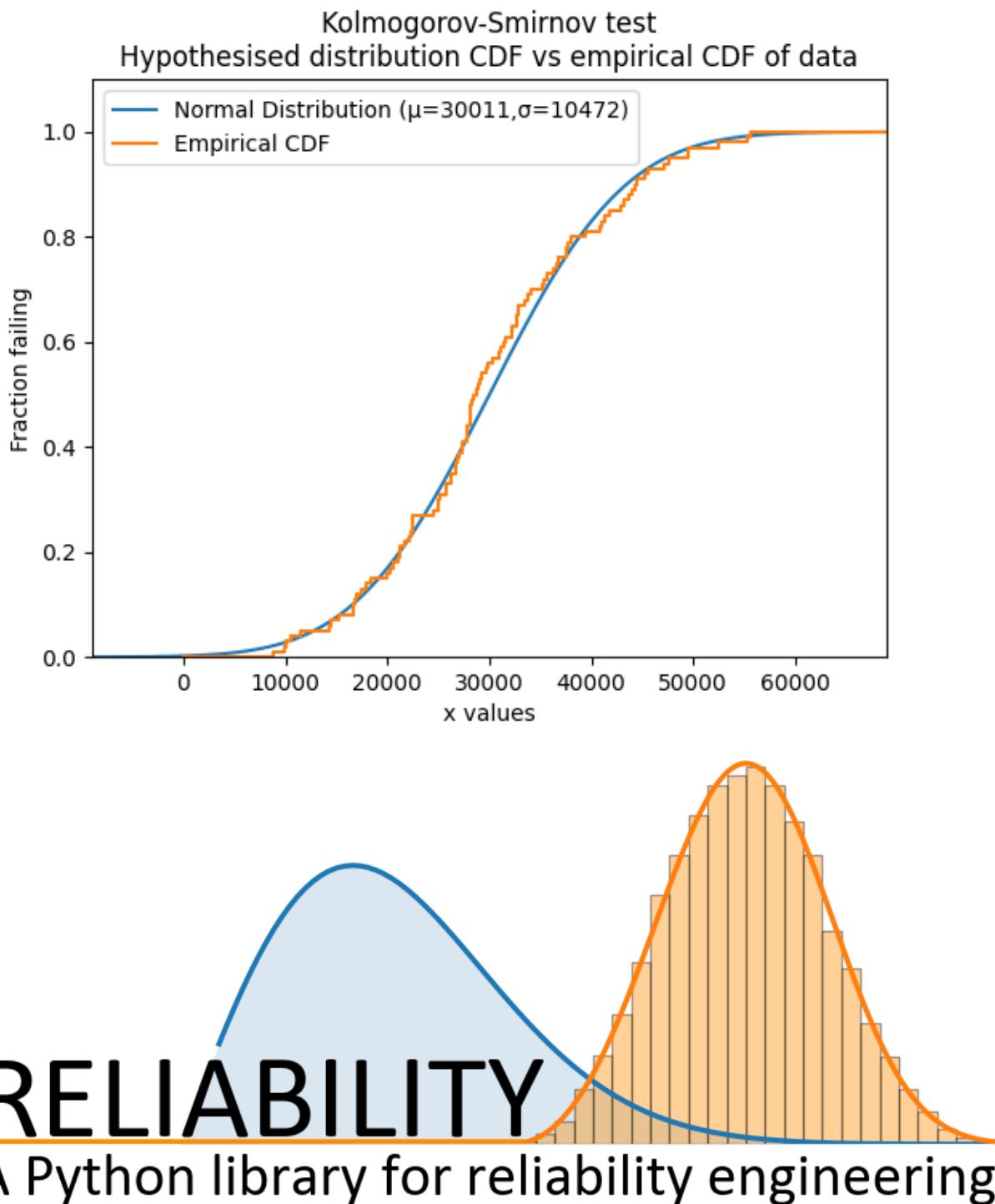
For inputs and outputs see the [API reference](#).

In the example below we import a dataset called mileage which contains 100 values that appear to be normally distributed. Using the function KStest we can determine whether we should accept the hypothesis that the data are from a Normal distribution with parameters $\mu=30011$ and $\sigma=10472$.

```
from reliability.Datasets import mileage
from reliability.Distributions import Normal_Distribution
from reliability.Reliability_testing import KStest
import matplotlib.pyplot as plt

data = mileage().failures
dist = Normal_Distribution(mu=30011, sigma=10472)
KStest(distribution=dist, data=data)
plt.show()

"""
Results from Kolmogorov-Smirnov test:
Kolmogorov-Smirnov statistic: 0.07162465859560846
Kolmogorov-Smirnov critical value: 0.13402791648569978
At the 0.05 significance level, we can ACCEPT the hypothesis that the data comes from a
Normal Distribution (=30011,=10472)
""
```



LIKELIHOOD PLOT

The likelihood plot provides a graphical representation of the confidence bounds on the parameters of a distribution at a particular confidence interval. When multiple distributions (of the same type) are plotted together, it is possible to determine whether there is a statistically significant difference between them by looking at whether their likelihood plots overlap.

Please note that it is not possible to overlay the likelihood plots of different types of distributions (e.g. Weibull and Normal) on the same plot because the axes must match.

The likelihood plot is only implemented for 2P distributions (Weibull, Normal, Lognormal, Gamma, Gumbel, Logistic, Beta) that are not location shifted. To compare two Exponential_1P distributions, simply look for overlap on the confidence bounds on the lambda parameter.

API Reference

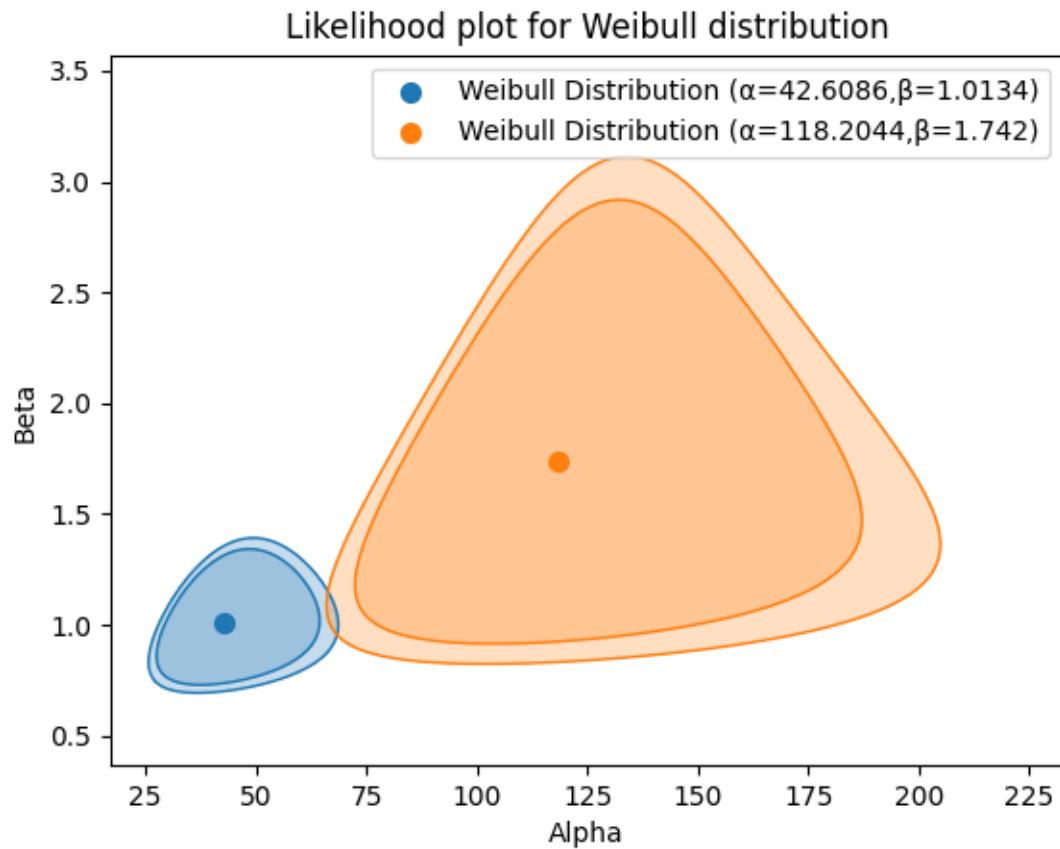
For inputs and outputs see the [API reference](#).

In the example below we have two datasets that are from two different design iterations of the same component. We want to know if the new design really is better than the old design. The parameters of the distribution certainly seem different, but is there a statistically significant difference between the distributions? To find out we need to plot the confidence bounds on the parameters. This is done at the 90% and 95% confidence levels.

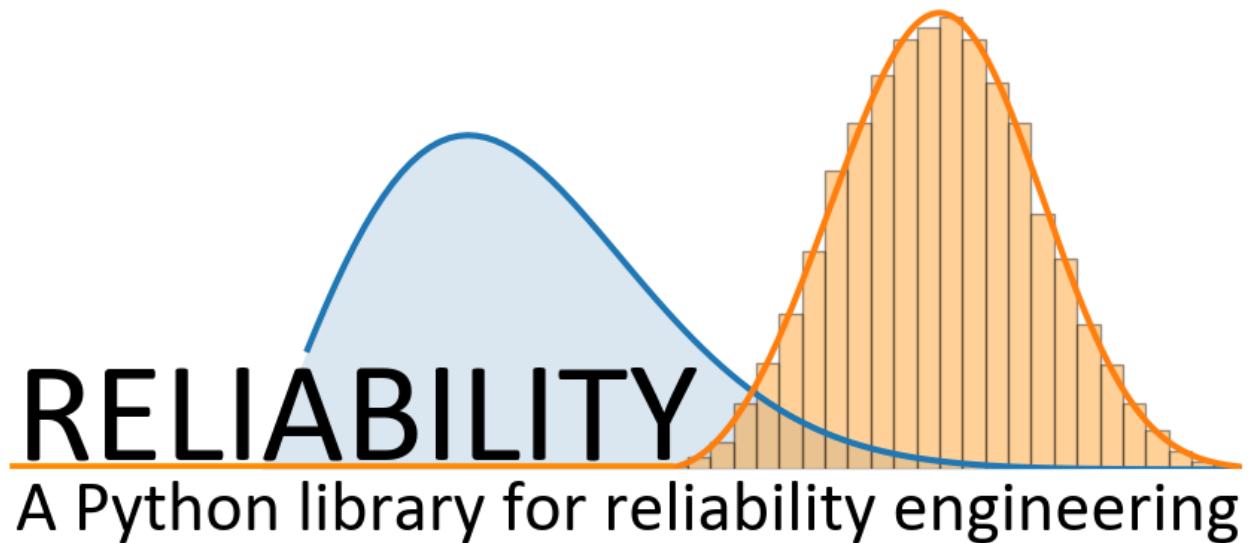
The resulting plot shows that there is no overlap at the 90% confidence level, but there is a small overlap at the 95% confidence level. From this plot we can say that we are 90% confident that the new design is better than the old one, but at the 95% confidence level, there is not a statistically significant difference between the two designs.

```
from reliability.Reliability_testing import likelihood_plot
import matplotlib.pyplot as plt

old_design = [2, 9, 23, 38, 67, 2, 11, 28, 40, 76, 3, 17, 33, 45, 90, 4, 17, 34, 55, 115,
             6, 19, 34, 56, 126, 9, 21, 37, 57, 197]
new_design = [15, 116, 32, 148, 61, 178, 67, 181, 75, 183]
likelihood_plot(distribution="Weibull", failures=old_design, CI=[0.9, 0.95])
likelihood_plot(distribution="Weibull", failures=new_design, CI=[0.9, 0.95])
plt.show()
```



This example uses the same data as the [example from reliawiki](#).



CHAPTER
FORTYTWO

SN DIAGRAM

This function will plot the stress vs number of cycles (S-N) diagram when supplied with data from a series of fatigue tests. An S-N diagram is a common procedure used to model the fatigue life of metals which are subject to known cyclic loads. Typically, the plot is done using a semilog scale where the number of cycles is scaled logarithmically. This has the effect of linearizing the plot and making the accuracy of the model much easier to visualize. For steels, titanium alloys, and some other metals, there exists an endurance limit. This limit is the minimum stress required to propagate fatigue cracks, and all stresses below this endurance limit do not contribute to fatigue growth. The plot can be adjusted to use an endurance limit using the optional inputs, however, there must be runout data (equivalent to right censored data) supplied in order for the program to determine where to set the endurance limit.

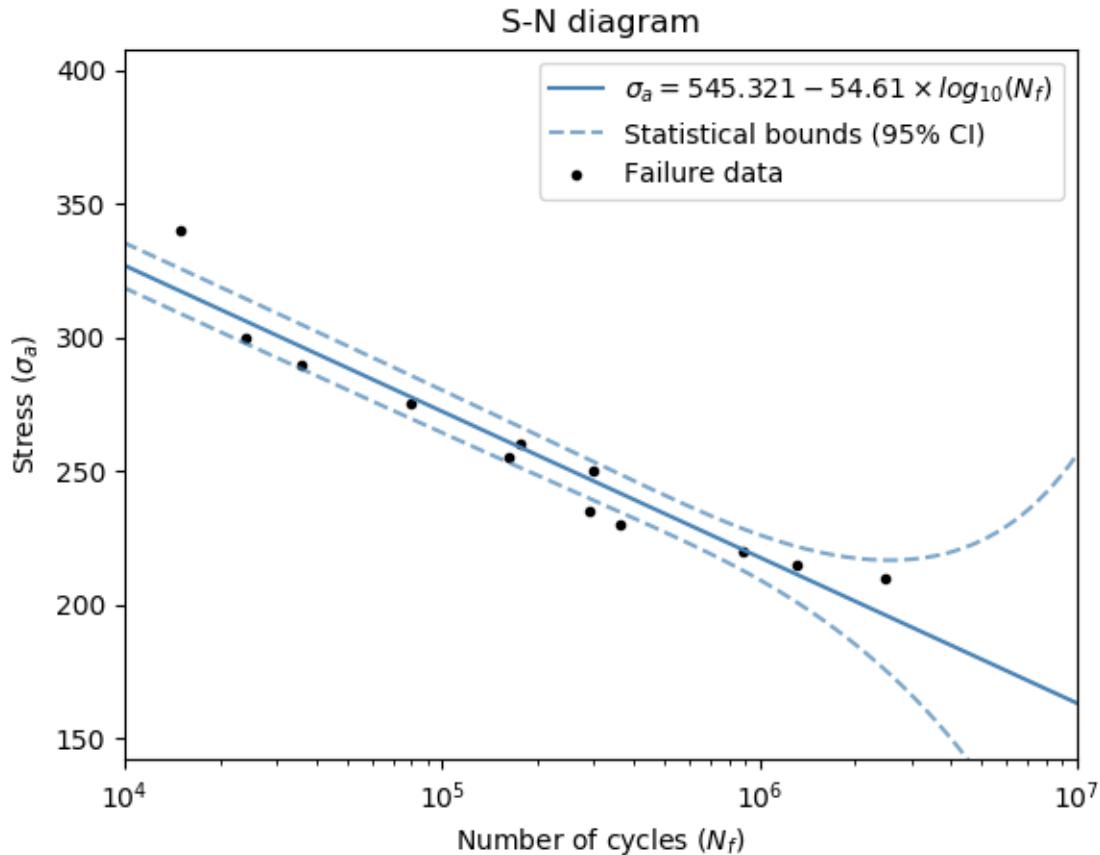
 **API Reference**

For inputs and outputs see the [API reference](#).

42.1 Example 1

In this first example, we use the data for stress and cycles to produce an S-N diagram. We will not provide any runout data here so the endurance limit will not be calculated.

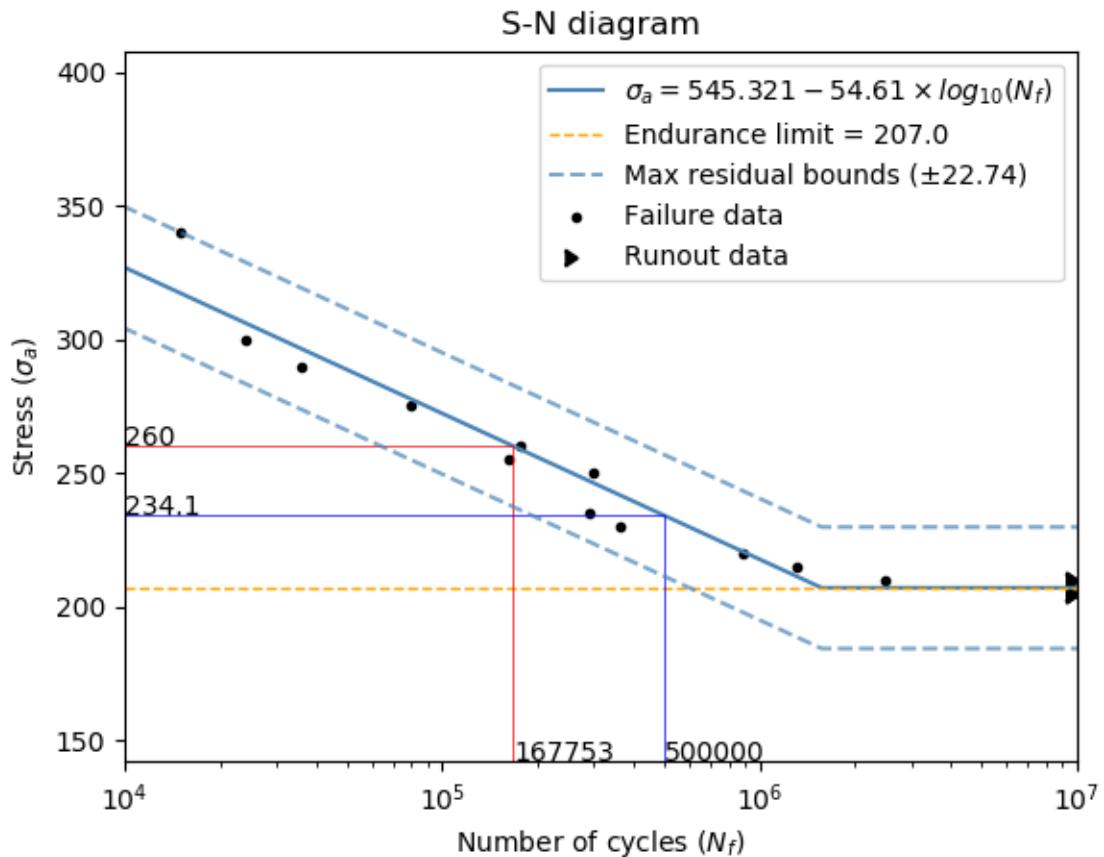
```
from reliability.PoF import SN_diagram
import matplotlib.pyplot as plt
stress = [340, 300, 290, 275, 260, 255, 250, 235, 230, 220, 215, 210]
cycles = [15000, 24000, 36000, 80000, 177000, 162000, 301000, 290000, 361000, 881000, ↴
          1300000, 2500000]
SN_diagram(stress=stress, cycles=cycles)
plt.show()
```



42.2 Example 2

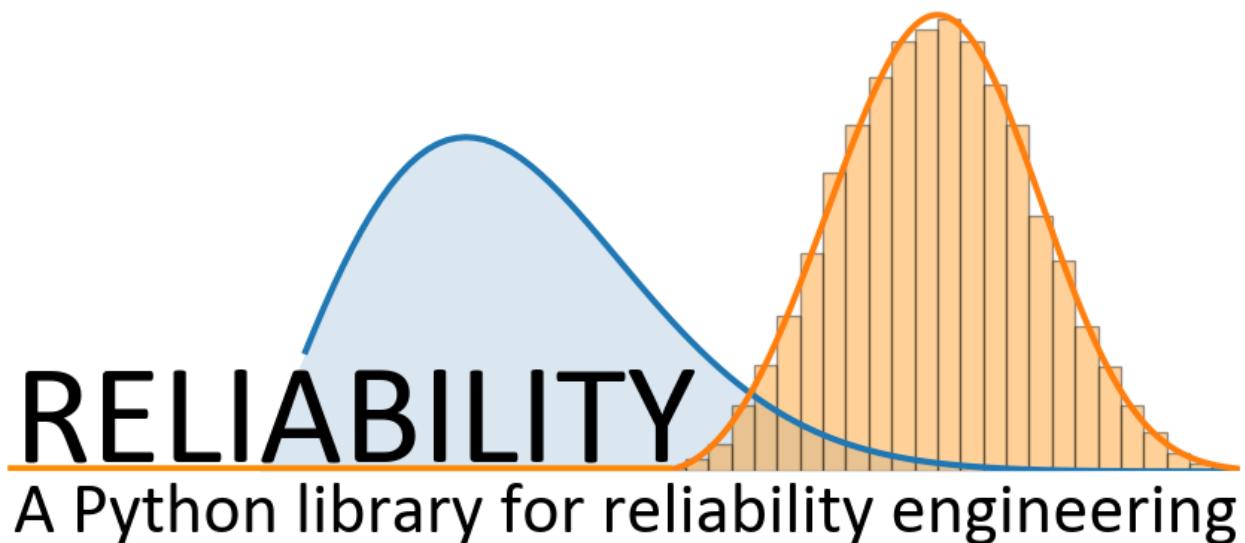
In this second example, we will use the same data as above, but also supply runout data so that the endurance limit will be calculated. We will also adjust the method_for_bounds to be 'residual'. We are also going to find the life (in cycles) at a stress of 260 by using stress_trace, and the stress required to achieve a life of 5×10^5 cycles using cycles_trace.

```
from reliability.PoF import SN_diagram
import matplotlib.pyplot as plt
stress = [340, 300, 290, 275, 260, 255, 250, 235, 230, 220, 215, 210]
cycles = [15000, 24000, 36000, 80000, 177000, 162000, 301000, 290000, 361000, 881000, 1300000, 2500000]
stress_runout = [210, 210, 205, 205, 205]
cycles_runout = [10 ** 7, 10 ** 7, 10 ** 7, 10 ** 7, 10 ** 7]
SN_diagram(stress=stress, cycles=cycles, stress_runout=stress_runout, cycles_
runout=cycles_runout, method_for_bounds='residual', cycles_trace=[5 * 10 ** 5], stress_
trace=[260])
plt.show()
```



References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 17-21.



CHAPTER
FORTYTHREE

STRESS-STRAIN AND STRAIN-LIFE

In the strain-life method of fatigue analysis, the elastic and plastic deformation of the material is used to determine how many cycles the material will last before failure. In this context, failure is defined as the formation of a small crack (typically 1mm) so the geometry of the material does not need to be considered provided the material properties have been accurately measured using a stress-strain test. This section of the documentation describes three functions which are useful in strain-life analysis. The first of these is useful to fit the stress-strain and strain-life models to available data, thereby providing the material properties. The second function is a diagram of the relationship between stress and strain during cyclic fatigue which shows the hysteresis loop and finds the min and max stress and strain. The third function produces the strain-life diagram and the equations for this diagram are used for calculating the number of cycles to failure. Further detail is available below for each of the respective functions.

The equations used for stress-strain and strain life are:

Ramberg-Osgood equation:

$$\varepsilon_{tot} = \underbrace{\frac{\sigma}{E}}_{\text{elastic}} + \underbrace{\left(\frac{\sigma}{K}\right)^{\frac{1}{n}}}_{\text{plastic}}$$

Hysteresis curve equation:

$$\Delta\varepsilon = \underbrace{\frac{\Delta\sigma}{E}}_{\text{elastic}} + 2 \underbrace{\left(\frac{\Delta\sigma}{2K}\right)^{\frac{1}{n}}}_{\text{plastic}}$$

Coffin-Manson equation:

$$\varepsilon_{tot} = \underbrace{\frac{\sigma_f}{E} (2N_f)^b}_{\text{elastic}} + \underbrace{\varepsilon_f (2N_f)^c}_{\text{plastic}}$$

Morrow Mean Stress Correction:

$$\varepsilon_{tot} = \underbrace{\frac{\sigma_f - \sigma_m}{E} (2N_f)^b}_{\text{elastic}} + \underbrace{\varepsilon_f (2N_f)^c}_{\text{plastic}}$$

Modified Morrow Mean Stress Correction:

$$\varepsilon_{tot} = \underbrace{\frac{\sigma_f - \sigma_m}{E} (2N_f)^b}_{\text{elastic}} + \underbrace{\varepsilon_f \left(\frac{\sigma_f - \sigma_m}{\sigma_f}\right)^{\frac{c}{b}} (2N_f)^c}_{\text{plastic}}$$

Smith-Watson-Topper Mean Stress Correction: $\varepsilon_{tot} = \underbrace{\frac{\sigma_f^2}{\sigma_{max} E} (2N_f)^{2b}}_{\text{elastic}} + \underbrace{\frac{\sigma_f \varepsilon_f}{\sigma_{max}} (2N_f)^{b+c}}_{\text{plastic}}$

43.1 Stress-Strain and Strain-Life parameter estimation

The function `stress_strain_life_parameters_from_data` will use stress and strain data to calculate the stress-strain parameters (K, n) from the Ramberg-Osgood relationship. If cycles is provided it will also produce the strain-life parameters (sigma_f, epsilon_f, b, c) from the Coffin-Manson equation. You cannot find the strain-life parameters without

stress as we must use stress to find elastic strain. Note that if you already have the parameters K, n, sigma_f, epsilon_f, b, c, then you can use the functions ‘stress_strain_diagram’ or ‘strain_life_diagram’ as described below.

ⓘ API Reference

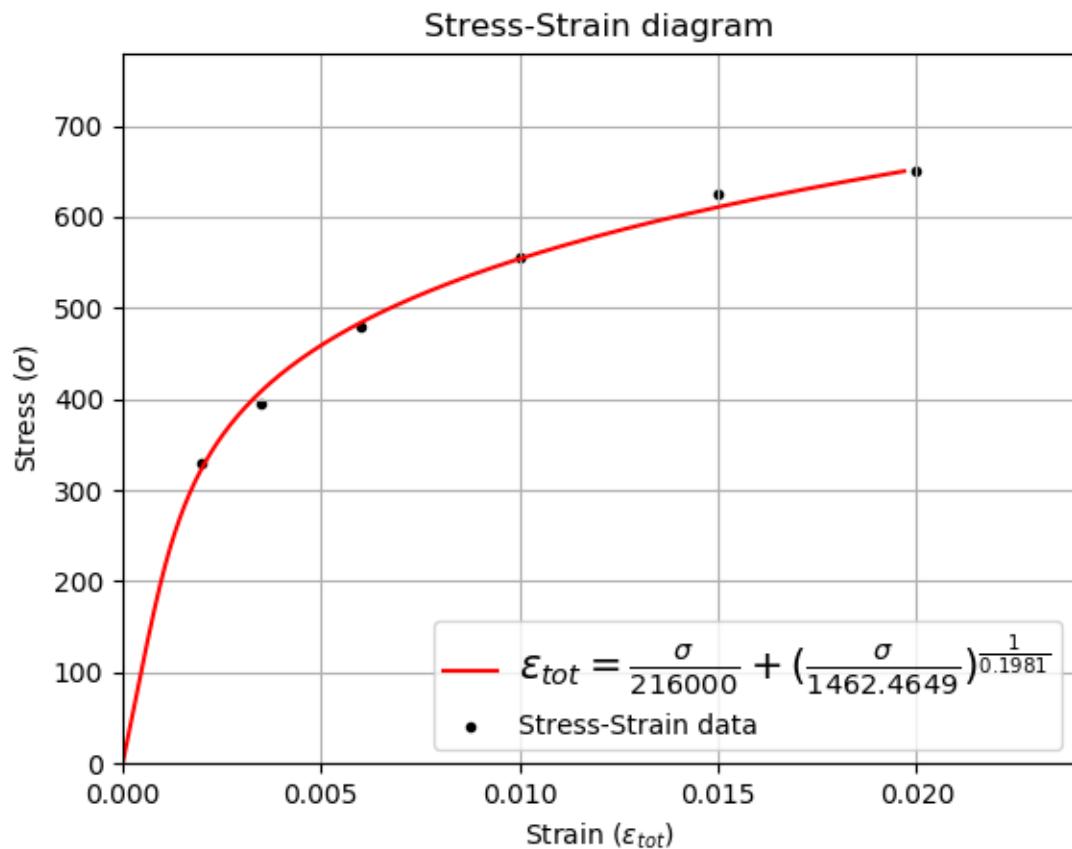
For inputs and outputs see the [API](#) reference.

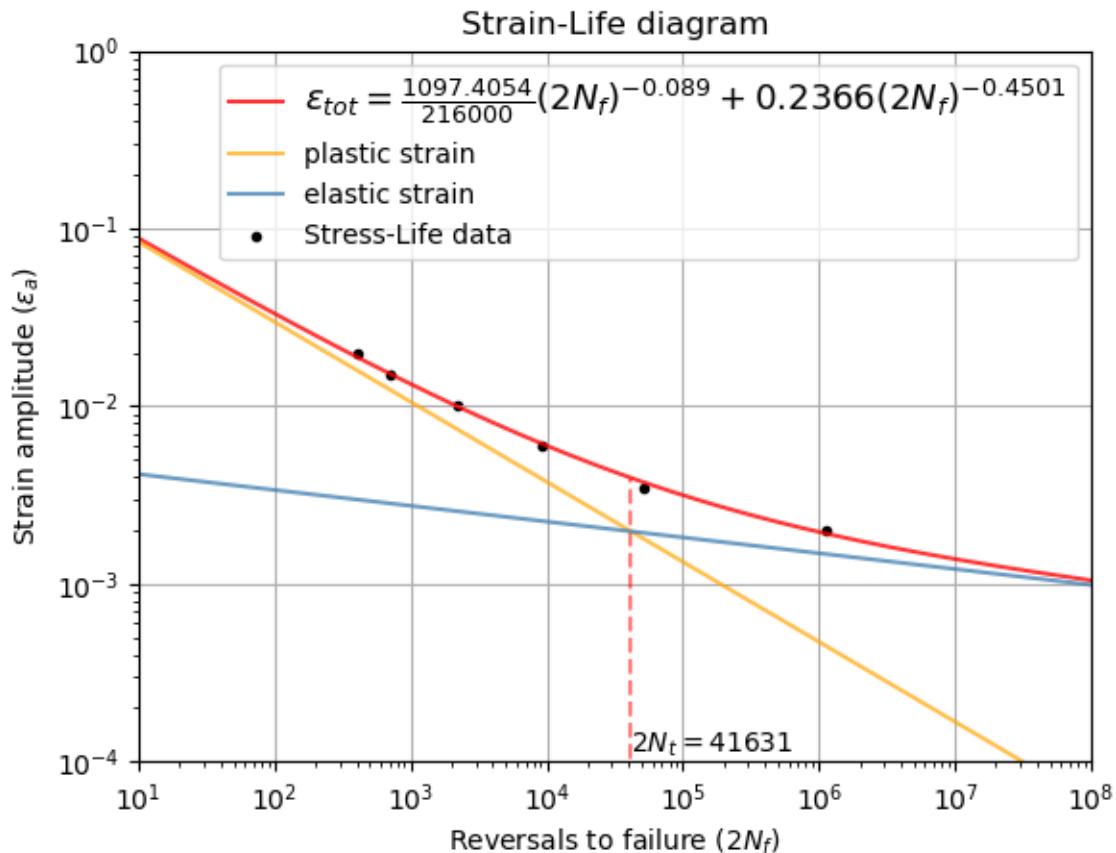
43.2 Example 1

In the example below, we provide data from a fatigue test including stress, strain, and cycles to failure. We must also provide the modulus of elasticity (E) for the material. All other options are left as default values. The plots shown below are provided and the results are printed to the console.

```
from reliability.PoF import stress_strain_life_parameters_from_data
import matplotlib.pyplot as plt
strain_data = [0.02, 0.015, 0.01, 0.006, 0.0035, 0.002]
stress_data = [650, 625, 555, 480, 395, 330]
cycles_data = [200, 350, 1100, 4600, 26000, 560000]
params = stress_strain_life_parameters_from_data(stress=stress_data, strain=strain_data,
                                                 cycles=cycles_data, E=216000)
plt.show()

"""
Results from stress_strain_life_parameters_from_data:
K (cyclic strength coefficient): 1462.4649152172044
n (strain hardening exponent): 0.19810419512368083
sigma_f (fatigue strength coefficient): 1097.405402055844
epsilon_f (fatigue strain coefficient): 0.23664541556833998
b (elastic strain exponent): -0.08898339316495743
c (plastic strain exponent): -0.4501077996416115
""
```





43.3 Stress-Strain diagram

The function `stress_strain_diagram` is used to visualize how the stress and strain vary with successive load cycles as described by the hysteresis curve equation. Due to residual tensile and compressive stresses, the stress and strain in the material does not unload in the same way that it loads. This results in a hysteresis loop being formed and this is the basis for crack propagation in the material leading to fatigue failure. The size of the hysteresis loop increases for higher strains. Fatigue tests are typically strain controlled; that is they are subjected to a specified amount of strain throughout the test, typically in a sinusoidal pattern. Fatigue tests may also be stress controlled, whereby the material is subjected to a specified amount of stress. This function accepts either input (max_stress or max_strain) and will find the corresponding stress or strain as required. If you do not specify min_stress or min_strain then it is assumed to be negative of the maximum value.

The cyclic loading sequence defaults to begin with tension, but can be changed using `initial_load_direction='compression'`. If your test begins with compression it is important to specify this as the residual stresses in the material from the initial loading will affect the results for the first reversal. This difference is caused by the Bauschinger effect. Only the initial loading and the first two reversals are plotted. For most materials the shape of the hysteresis loop will change over many hundreds of cycles as a result of fatigue hardening (also known as work-hardening) or fatigue-softening. More on this process is available in the [eFatigue training documents](#).

Note that if you do not have the parameters K , n , but you do have stress and strain data then you can use the function '`stress_strain_life_parameters_from_data`'. This will be shown in the first example below.

1 API Reference

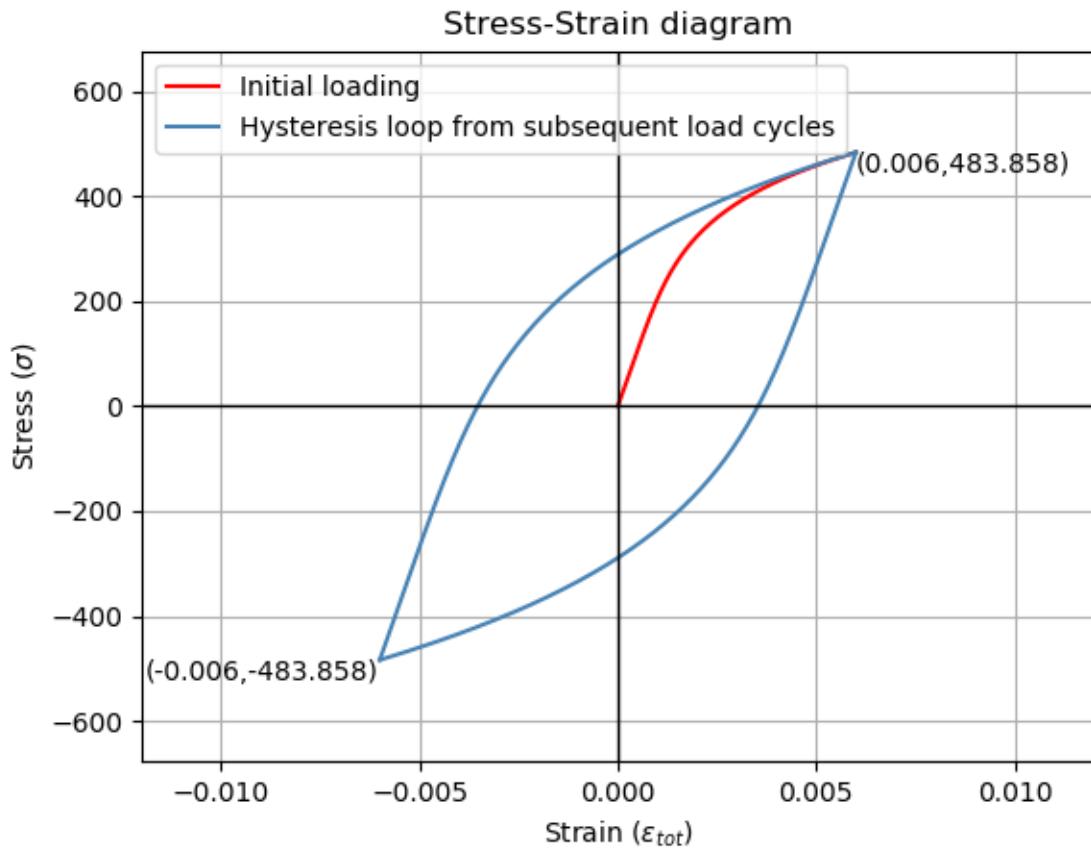
For inputs and outputs see the [API reference](#).

43.4 Example 2

In the example below, we are using the same data from the first example, but this time, we will store the calculated parameters in an object named ‘params’. Then we can specify the calculated parameters to the stress_strain_diagram function. The hysteresis loop generated is for a strain-controlled fatigue test where the strain goes from -0.006 to +0.006.

```
from reliability.PoF import stress_strain_life_parameters_from_data, stress_strain_
diagram
import matplotlib.pyplot as plt
strain_data = [0.02, 0.015, 0.01, 0.006, 0.0035, 0.002]
stress_data = [650, 625, 555, 480, 395, 330]
cycles_data = [200, 350, 1100, 4600, 26000, 560000]
params = stress_strain_life_parameters_from_data(stress=stress_data, strain=strain_data,_
cycles=cycles_data, E=216000, show_plot=False, print_results=False)
stress_strain_diagram(E = 216000, n = params.n, K = params.K, max_strain=0.006)
plt.show()

"""
Results from stress_strain_diagram:
Max stress: 483.8581623940648
Min stress: -483.8581623940648
Max strain: 0.006
Min strain: -0.006
""
```



43.5 Example 3

In this example, we will use the stress_strain_diagram to visualise the effects of residual stresses for a material subjected to non-zero mean stress. The material parameters (K and n) are already known so we do not need to obtain them from any data. We specify the max_stress is 378 MPa and the min_stress is -321 MPa. We will do this for two scenarios; initial tensile load, and initial compressive load. Upon inspection of the results we see for the initial tensile load, the min_stress in the material is actually -328.893 MPa which exceeds the min_stress we specified in our test. When we have an initial compressive load, the max_stress is 385.893 MPa which exceeds the max_stress we specified in our test. These results are not an error and are caused by the residual stresses in the material that were formed during the first loading cycle. In the case of an initial tensile load, when the material was pulled apart in tension by an external force, the material pulls back but due to plastic deformation, these internal forces in the material are not entirely removed, such that when the first compressive load peaks, the material's internal stresses add to the external compressive forces. This phenomenon is important in load sequence effects for variable amplitude fatigue.

```
from reliability.PoF import stress_strain_diagram
import matplotlib.pyplot as plt
plt.figure()
plt.subplot(121)
print('Tension first')
stress_strain_diagram(E=210000, K = 1200, n = 0.2, max_stress=378,min_stress=-321,
initial_load_direction='tension')
plt.title('Cyclic loading - tension first')
plt.subplot(122)
```

(continues on next page)

(continued from previous page)

```

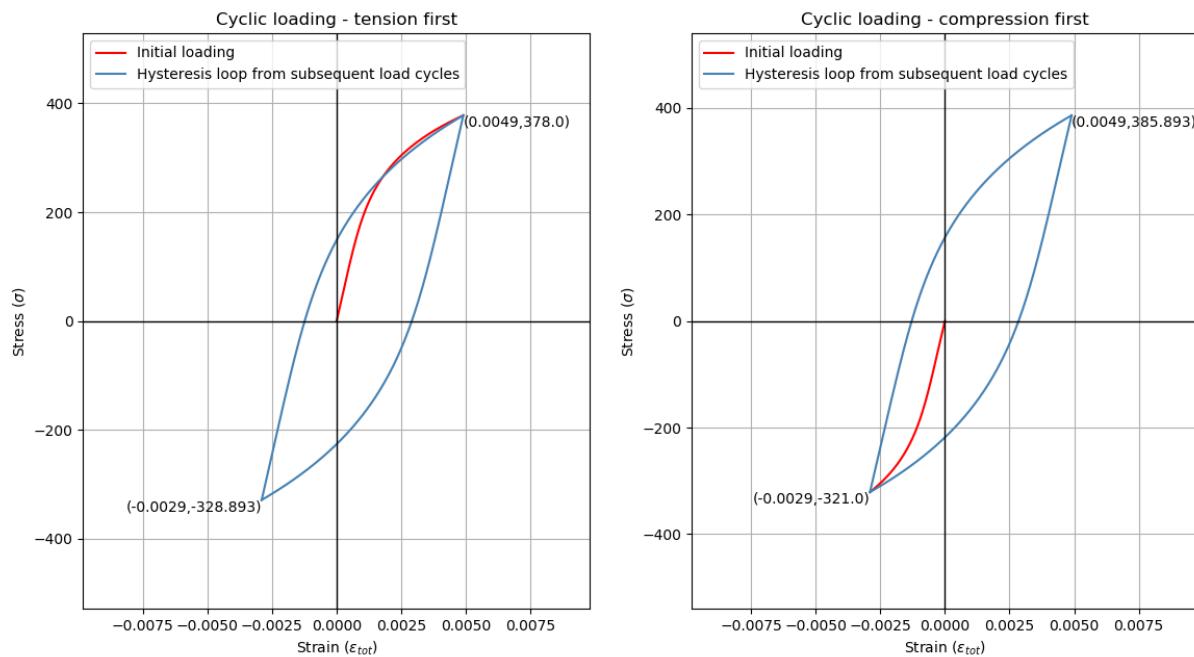
print('\nCompression first')
stress_strain_diagram(E=210000, K = 1200, n = 0.2, max_stress=378,min_stress=-321,
    initial_load_direction='compression')
plt.title('Cyclic loading - compression first')
plt.gcf().set_size_inches(12,7)
plt.show()

"""

Tension first
Results from stress_strain_diagram:
Max stress: 378.0
Min stress: -328.89311218003513
Max strain: 0.004901364196875
Min strain: -0.0028982508530831477

Compression first
Results from stress_strain_diagram:
Max stress: 385.89311218003513
Min stress: -321.0
Max strain: 0.004901364196875
Min strain: -0.0028982508530831477
"""

```



43.6 Strain-Life diagram

The function `strain_life_diagram` provides a visual representation of the Coffin-Manson relationship between strain and life. In this equation, strain is split into elastic strain and plastic strain which are shown on the plot as straight lines (on a log-log scale), and life is represented by reversals (with 2 reversals per cycle). The total strain amplitude is used to determine the fatigue life by solving the Coffin-Manson equation. When a `min_stress` or `min_strain` is specified

that results in a non-zero mean stress, there are several mean stress correction methods that are available. These are ‘morrow’, ‘modified_morrow’ (also known as Manson-Halford) , and ‘SWT’ (Smith-Watson-Topper). The default method is ‘SWT’ but can be changed using the options described below. The equation used is displayed in the legend of the plot. Also shown on the plot is the life of the material at the specified strain amplitude, and the transition life (2N_t) for which the material failure transitions from being dominated by plastic strain to elastic strain.

Note that if you do not have the parameters sigma_f, epsilon_f, b, c, but you do have stress, strain, and cycles data then you can use the function `stress_strain_life_parameters_from_data`.

The residual stress in a material subjected to non-zero mean stress (as shown in the previous example) are not considered in this analysis, and the specified max and min values for stress or strain are taken as the true values to which the material is subjected.

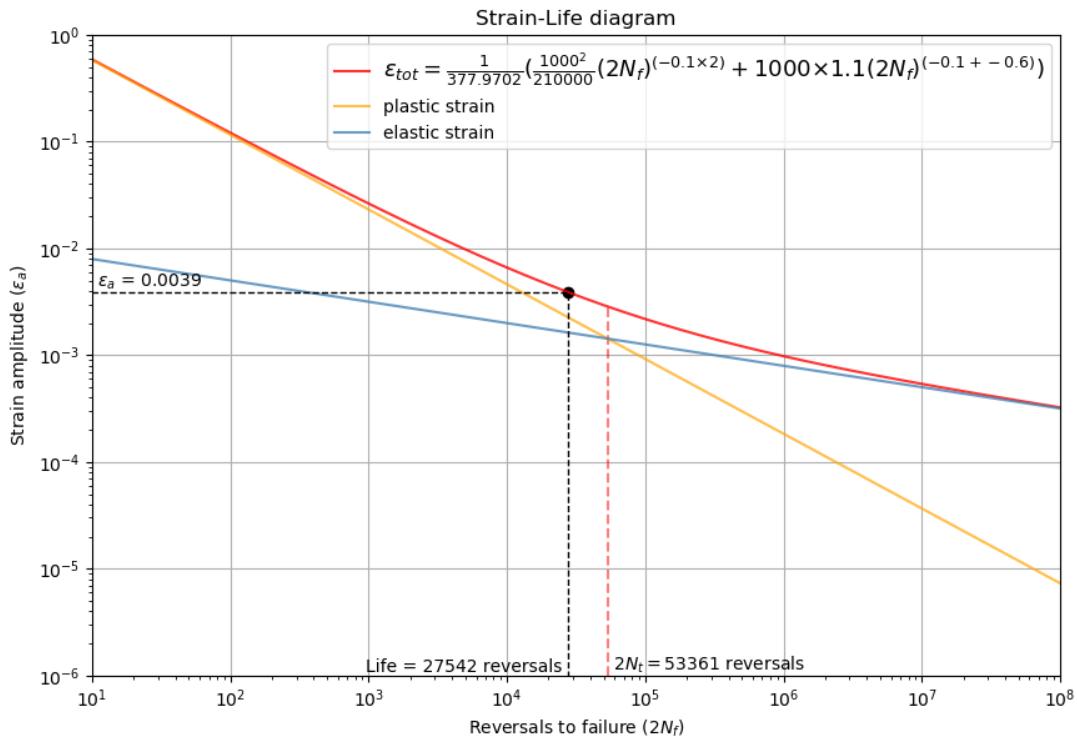
API Reference

For inputs and outputs see the [API reference](#).

43.7 Example 4

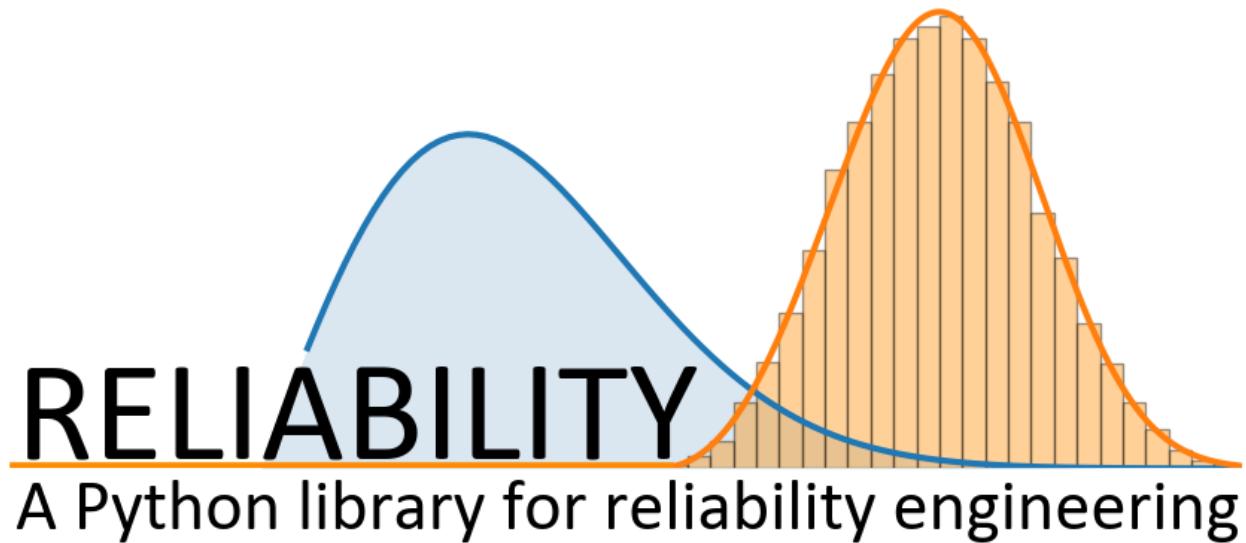
```
from reliability.PoF import strain_life_diagram
import matplotlib.pyplot as plt
strain_life_diagram(E=210000, sigma_f=1000, epsilon_f=1.1, b = -0.1, c=-0.6, K = 1200, n_u
                     = 0.2, max_strain=0.0049, min_strain=-0.0029)
plt.show()

"""
Results from strain_life_diagram:
Failure will occur in 13771.39 cycles (27542.78 reversals).
""
```



References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 23-33



CHAPTER
FORTYFOUR

FRACTURE MECHANICS

Fracture mechanics is an approach to fatigue analysis that involves calculating the number of cycles until failure of a component that is undergoing cyclic loading. We are generally interested in two values; the number of cycles needed to initiate a crack, and the number of cycles to grow the crack to a certain size (usually the size for brittle fracture to occur). The fracture mechanics functions described below are useful for solving typical fatigue problems given to students. Unfortunately, the limitation of these functions is that they are only applicable to thin plates with through-thickness cracks. As soon as you encounter a component that is not a thin plate, then the formulas required for analysis will be different from those used below. This is part of what makes fracture mechanics such a complex topic that is better handled by purpose-built fatigue analysis software such as [Altair Hyperlife](#). Even in the relatively simple thin flat plate geometry, there are many complications that make fracture mechanics a challenging subject. These include variable amplitude loading, surface roughness, frequency effect, environmental effects (temperature, corrosion) and other miscellaneous factors. Solving fracture mechanics problems for flat plates can still provide engineers with an appreciation for how fatigue operates and the factors affecting fatigue so that they can incorporate these lessons learned into their work.

Most textbooks (including Probabilistic Physics of Failure Approach to Reliability (2017) which was used to design the functions below) apply a few simplifications for solving crack growth problems. These simplifications involve an assumption that the stress in the component is constant, that the geometry factor is constant, and that the crack length to cause failure (which has the geometry factor in its formula) is constant. These simplifications are necessary for hand calculations, but in reality we know that they all must change as the crack length grows which necessitates an iterative calculation. Both the simplified and iterative methods are included in the crack growth function. Also included in both functions is the ability to solve these problems for notched components by providing the appropriate correction factors for the notched geometry.

44.1 Crack initiation

The function *fracture_mechanics_crack_initiation* uses the material properties, the local cross-sectional area, and force applied to the component to determine how many cycles until crack initiation (of a 1 mm crack). Units should always be in MPa (and mm² for area). This function may be used for an un-notched or notched component. If the component is un-notched, the parameters q and Kt may be left as their default values of 1.

While there are formulas to find the parameters q and Kt, these formulas have not been included here so that the function is reasonably generic to different materials and geometries. Resources for finding some of these parameters if they are not given to you:

$q = \frac{1}{1 + \frac{a}{r}}$ Where r is the notch radius of curvature (in mm), and a is $0.025 \times \frac{2070}{S_u}$. S_u is the ultimate strength in MPa. This only applies to high strength steels where $S_u > 550$ MPa.

Kt can be found from the [eFatigue website](#) which has an online calculator that will provide you with the appropriate Kt for your notched geometry.

ⓘ API Reference

For inputs and outputs see the [API reference](#).

44.2 Example 1

In the following example we will provide the function with the appropriate inputs for our problem (taken from Example 2.8 in Probabilistic Physics of Failure Approach to Reliability (2017)). The `mean_stress_correction_method` is changed to 'SWT' and the results will be printed to the console.

```
from reliability.PoF import fracture_mechanics_crack_initiation
fracture_mechanics_crack_initiation(P=0.15, A=5*80, Kt=2.41, q=0.9857, Sy=690, E=210000, ↵
K=1060, n=0.14, b=-0.081, c=-0.65, sigma_f=1160, epsilon_f=1.1, mean_stress_correction_ ↵
method='SWT')

"""
Results from fracture_mechanics_crack_initiation:
A crack of 1 mm will be formed after: 2919.91 cycles (5839.82 reversals).
Stresses in the component: Min = -506.7291 MPa , Max = 506.7291 MPa , Mean = -5. ↵
684341886080802e-14 MPa.
Strains in the component: Min = -0.0075 , Max = 0.0075 , Mean = 8.673617379884035e-19
Mean stress correction method used: SWT
""
```

44.3 Crack growth

The function `fracture_mechanics_crack_growth` uses the principles of fracture mechanics to find the number of cycles required to grow a crack from an initial length until a final length. The final length (`a_final`) may be specified, but if not specified then `a_final` will be set as the critical crack length (`a_crit`) which causes failure due to rapid fracture. This function performs the same calculation using two methods: simplified and iterative. The simplified method assumes that the geometry factor (`f(g)`), the stress (`S_net`), and the critical crack length (`a_crit`) are constant. This method is the way most textbooks show these problems solved as they can be done by hand in a few steps. The iterative method does not make those assumptions and as a result, the parameters `f(g)`, `S_net` and `a_crit` must be recalculated based on the current crack length at every cycle.

This function is applicable only to thin plates with a through thickness edge crack or a centre crack (which is to be specified using the parameter `crack_type`). You may also use this function for notched components (edge notches only, not centre holes) by specifying the parameters `Kt` and `D` which are based on the geometry of the notch. For any notched components, this method assumes the notched component has a "shallow notch" where the notch depth (`D`) is much less than the plate width (`W`). The value of `Kt` for notched components may be found on the [eFatigue website](#). In the case of notched components, the local stress concentration from the notch will often cause slower crack growth. In these cases, the crack length is calculated in two parts (stage 1 and stage 2) which can clearly be seen on the plot using the iterative method (as shown in the example below).

ⓘ API Reference

For inputs and outputs see the [API reference](#).

44.4 Example 2

In the following example, a crack of 1mm is grown to failure. The function determines that the notch (described by K_t and D) causes a local stress concentration which initially slows the propagation of the crack until the crack reaches the transition length. Once past the transition length, the crack grows much faster and results in brittle fracture of the material. This change in crack growth rate is evident on the plot from the iterative method. The reason for the different transition lengths between the simplified and iterative methods is that the simplified method uses 1.12 for the geometry factor whereas the iterative method finds the geometry factor using the local geometry (using W and D).

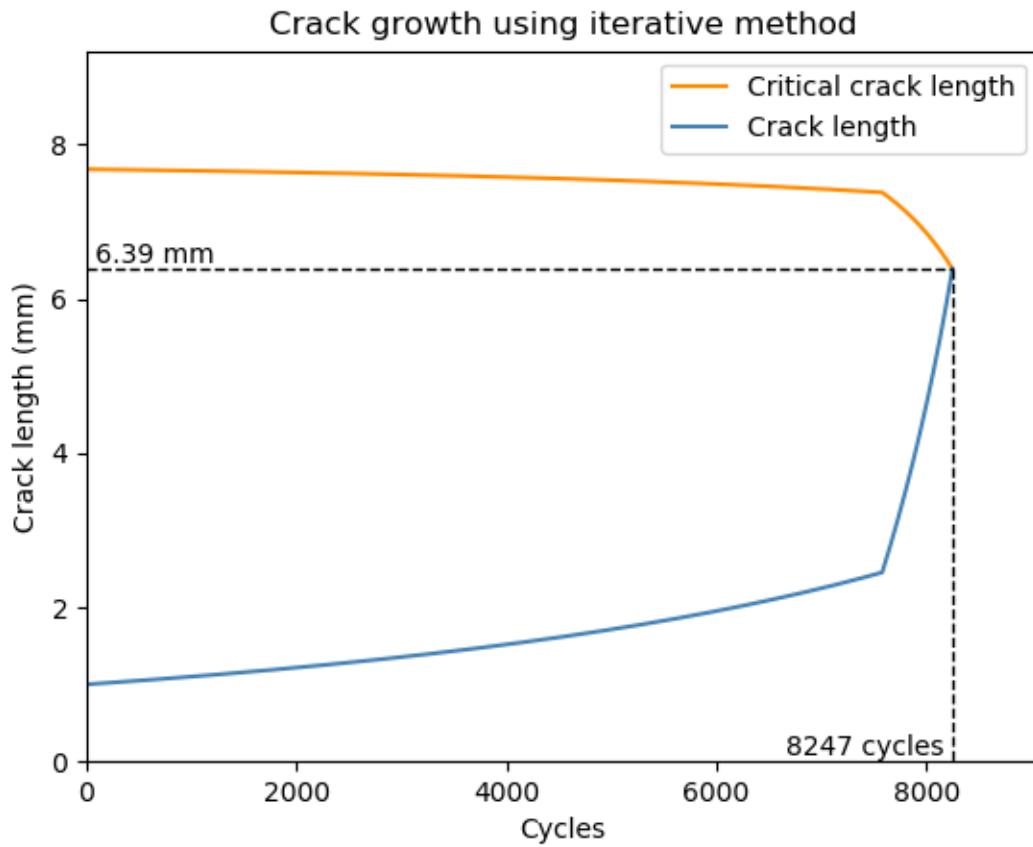
```
from reliability.PoF import fracture_mechanics_crack_growth
import matplotlib.pyplot as plt
fracture_mechanics_crack_growth(Kc=66,C=6.91*10**-12,m=3,P=0.15,W=100,t=5,Kt=2.41,D=10)
plt.show()

"""
Results from fracture_mechanics_crack_growth:
SIMPLIFIED METHOD (keeping  $f(g)$ ,  $S_{max}$ , and  $a_{crit}$  as constant):
Crack growth was found in two stages since the transition length ( 2.08 mm ) due to the
notch, was greater than the initial crack length ( 1.0 mm ).

Stage 1 (a_initial to transition length): 6802 cycles
Stage 2 (transition length to a_final): 1133 cycles
Total cycles to failure: 7935 cycles.
Critical crack length to cause failure was found to be: 7.86 mm.

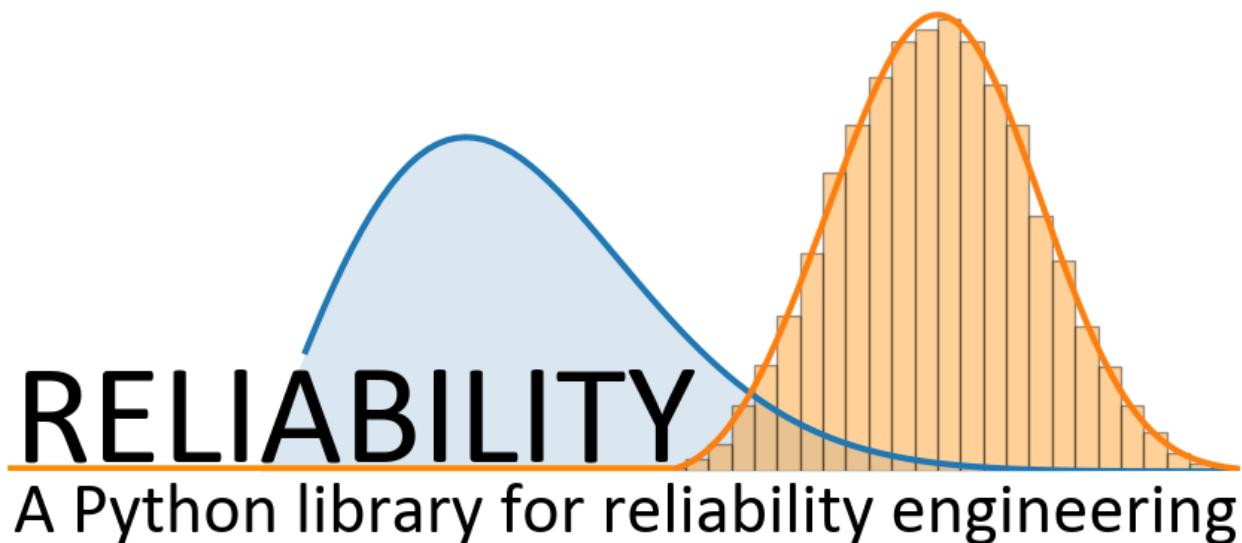
ITERATIVE METHOD (recalculating  $f(g)$ ,  $S_{max}$ , and  $a_{crit}$  for each cycle):
Crack growth was found in two stages since the transition length ( 2.45 mm ) due to the
notch, was greater than the initial crack length ( 1.0 mm ).

Stage 1 (a_initial to transition length): 7576 cycles
Stage 2 (transition length to a_final): 671 cycles
Total cycles to failure: 8247 cycles.
Critical crack length to cause failure was found to be: 6.39 mm.
""
```



References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 37-57



CHAPTER **FORTYFIVE**

CREEP

Creep is the progressive accumulation of plastic strain in a component under stress at an elevated temperature over a period of time. All creep modelling requires data that is unique to the material undergoing creep since all materials behave differently. This data may be stress, temperature, and time to failure data, or it may be material constants which are derived from the former. This section of reliability contains two functions to determine time to failure due to creep. These functions are *creep_rupture_curves* and *creep_failure_time*. Creep is generally modelled using the [Larson-Miller](#) relation or the Manson-Haferd relation.

The function `creep_rupture_curves` plots the creep rupture curves for a given set of creep data. The function also fits the lines of best fit to each temperature. The time to failure for a given temperature can be found by specifying `stress_trace` and `temp_trace`.

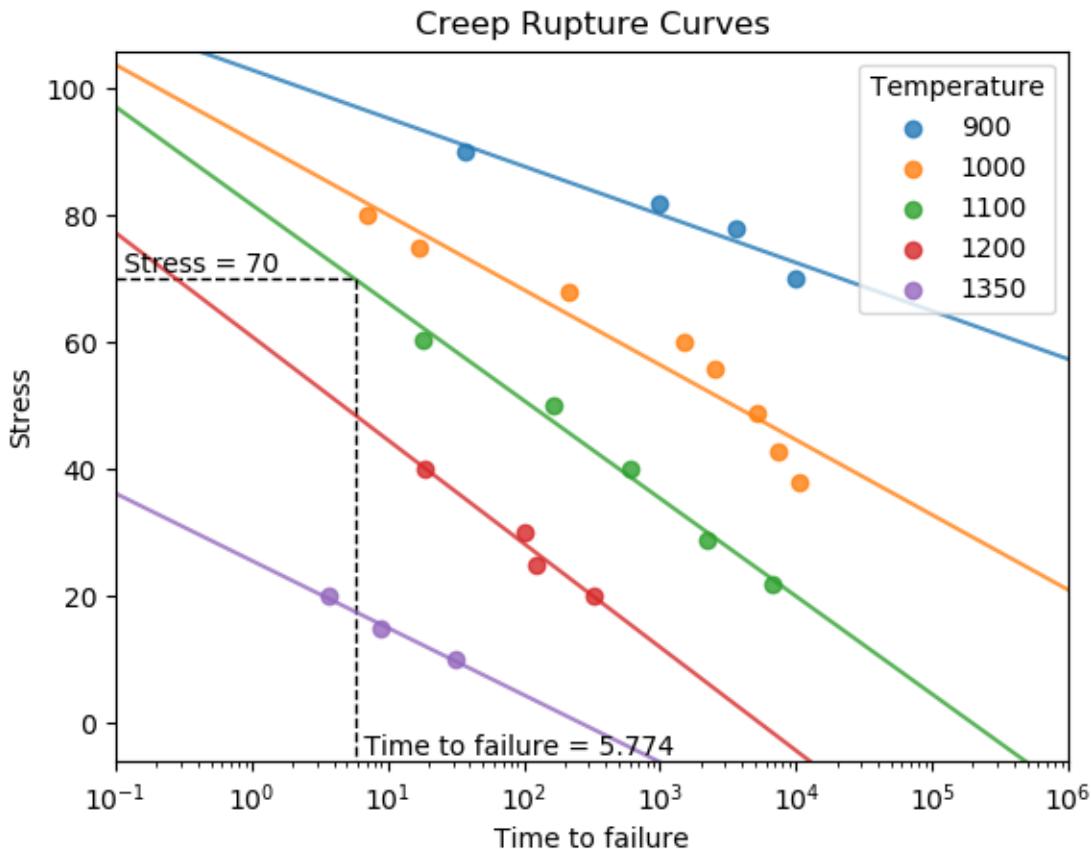
API Reference

For inputs and outputs see the [API reference](#).

45.1 Example 1

In the following example (taken from example 2.16 of Probabilistic Physics of Failure Approach to Reliability (2017)), we provide creep data in the form of temperatures, stresses, and times to failure in order to obtain the creep rupture curves. We also are interested in the time to failure of a component at a stress of 70 and a temperature of 1100.

```
from reliability.PoF import creep_rupture_curves
import matplotlib.pyplot as plt
TEMP = [900, 900, 900, 900, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1100, 1100, 1100, 1100, 1100, 1100, 1200, 1200, 1200, 1200, 1350, 1350, 1350]
STRESS = [90, 82, 78, 70, 80, 75, 68, 60, 56, 49, 43, 38, 60.5, 50, 40, 29, 22, 40, 30, 25, 20, 20, 15, 10]
TTF = [37, 975, 3581, 9878, 7, 17, 213, 1493, 2491, 5108, 7390, 10447, 18, 167, 615, 2220, 6637, 19, 102, 125, 331, 3.7, 8.9, 31.8]
creep_rupture_curves(temp_array=TEMP, stress_array=STRESS, TTF_array=TTF, stress_
    ↴trace=70, temp_trace=1100)
plt.show()
```



The function `creep_failure_time` uses the Larson-Miller relation to find the time to failure due to creep. The method uses a known failure time (`time_low`) at a lower failure temperature (`temp_low`) to find the unknown failure time at the higher temperature (`temp_high`). This relation requires the input temperatures in Fahrenheit. To convert Celsius to Fahrenheit use $F = C \times (9/5) + 32$. Also note that the conversion between Fahrenheit and Rankine used in this calculation is $R = F + 459.67$.

API Reference

For inputs and outputs see the [API reference](#).

45.2 Example 2

In the following example (which follows on from the previous example), we will use the Larson-Miller relation to find the time to failure due to creep at 1100°F for a component which we know fails at 9878 hours when subjected to the same stress at 900°F.

```
from reliability.PoF import creep_failure_time
creep_failure_time(temp_low=900,temp_high=1100,time_low=9878)

"""
Results from creep_failure_time:
The time to failure at a temperature of 1100 °F is 8.27520045913433
```

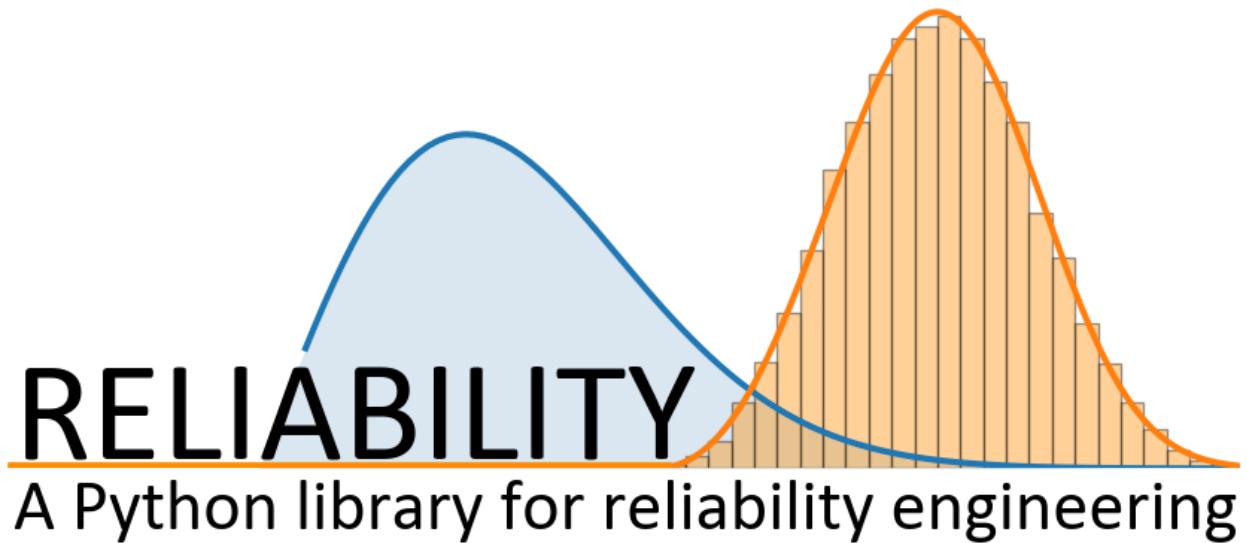
(continues on next page)

(continued from previous page)

The Larson-Miller parameter was found to be 32624.83162890552
"

References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 81-90



PALMGREN-MINER LINEAR DAMAGE MODEL

The function `palmgren_miner_linear_damage` uses the Palmgren-Miner linear damage hypothesis to find the outputs listed below.

API Reference

For inputs and outputs see the [API reference](#).

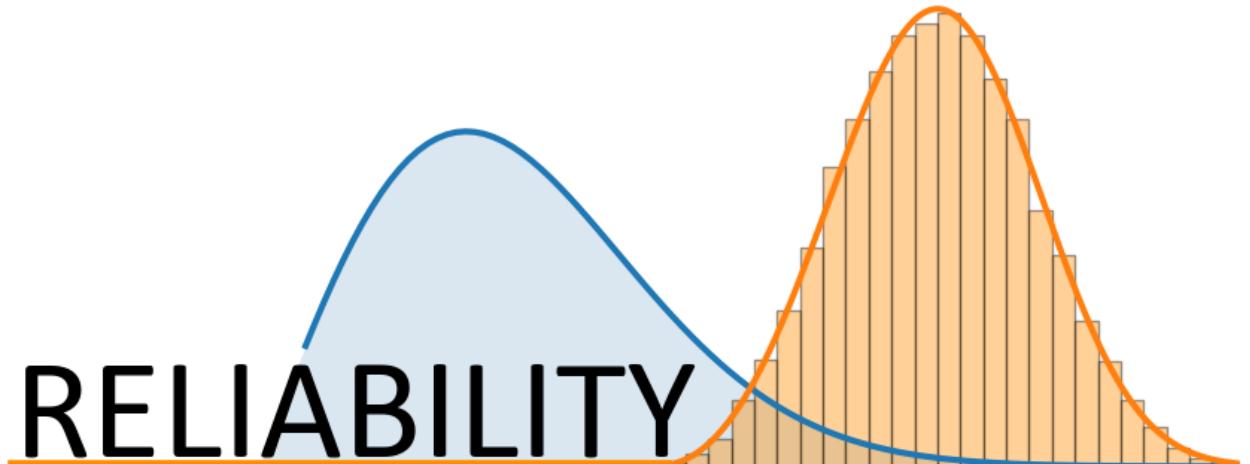
In the following example, we consider a scenario in which ball bearings fail after 50000 hrs, 6500 hrs, and 1000 hrs, after being subjected to a stress of 1kN, 2kN, and 4kN respectively. If each load cycle involves 40 mins at 1kN, 15 mins at 2kN, and 5 mins at 4kN, how long will the ball bearings last?

```
from reliability.PoF import palmgren_miner_linear_damage
palmgren_miner_linear_damage(rated_life=[50000,6500,1000], time_at_stress=[40/60, 15/60, 5/60], stress=[1, 2, 4])

"""
Palmgren-Miner Linear Damage Model results:
Each load cycle uses 0.01351 % of the components life.
The service life of the component is 7400.37951 load cycles.
The amount of damage caused at each stress level is:
Stress = 1 , Damage fraction = 9.86717 %.
Stress = 2 , Damage fraction = 28.463 %.
Stress = 4 , Damage fraction = 61.66983 %.
""
```

References:

- Probabilistic Physics of Failure Approach to Reliability (2017), by M. Modarres, M. Amiri, and C. Jackson. pp. 33-37



RELIABILITY
A Python library for reliability engineering

CHAPTER
FORTYSEVEN

ACCELERATION FACTOR

The Arrhenius model for Acceleration factor due to higher temperature is $AF = \exp \left[\frac{E_a}{K_B} \left(\frac{1}{T_{use}} - \frac{1}{T_{acc}} \right) \right]$ This function accepts `T_use` as a mandatory input and you may specify any two of the three other variables, and the third variable will be found.

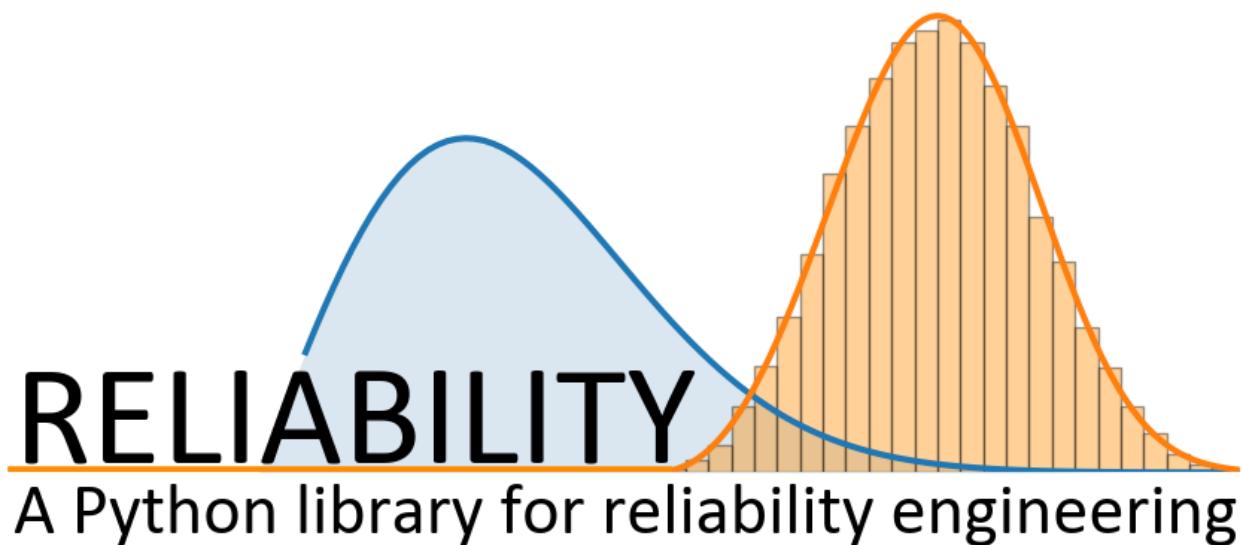
 **API Reference**

For inputs and outputs see the [API reference](#).

In the example below, the acceleration factor is found for an accelerated test at 100°C for a component that is normally run at 60°C and has an activation energy of 1.2 eV.

```
from reliability.PoF import acceleration_factor
acceleration_factor(T_use=60,T_acc=100,Ea=1.2)

"""
Results from acceleration_factor:
Acceleration Factor: 88.29574588463338
Use Temperature: 60 °C
Accelerated Temperature: 100 °C
Activation Energy: 1.2 eV
""
```



CHAPTER
FORTYEIGHT

SOLVING SIMULTANEOUS EQUATIONS WITH SYMPY

This document is a tutorial for how to use the Python module *sympy* to solve simultaneous equations. Since *sympy* does this so well, there is no need to implement it within *reliability*, but users may find this tutorial helpful as problems involving physics of failure will often require the solution of simultaneous equations. *sympy* is not installed by default when you install *reliability* so users following this tutorial will need to ensure *sympy* is installed on their machine. The following three examples should be sufficient to illustrate how to use *sympy* for solving simultaneous equations. Further examples are available in the [sympy documentation](#).

Example 1

Eqn 1: $x + y = 5$

Eqn 2: $x^2 + y^2 = 17$

Solving with *sympy*:

```
import sympy as sym
x,y = sym.symbols('x,y')
eq1 = sym.Eq(x+y,5)
eq2 = sym.Eq(x**2+y**2,17)
result = sym.solve([eq1,eq2],(x,y))
print(result)

"""
[(1, 4), (4, 1)] #these are the solutions for x,y. There are 2 solutions because the
→equations represent a line passing through a circle.
""
```

Example 2

Eqn 1: $a1000000^b = 119.54907$

Eqn 2: $a1000^b = 405$

Solving with *sympy*:

```
import sympy as sym
a,b = sym.symbols('a,b')
eq1 = sym.Eq(a*1000000**b,119.54907)
eq2 = sym.Eq(a*1000**b,405)
result = sym.solve([eq1,eq2],(a,b))
print(result)

""
```

(continues on next page)

(continued from previous page)

```
[ (1372.03074854535, -0.176636273742481) ] #these are the solutions for a,b
""
```

Example 3

Eqn 1: $2x^2 + y + z = 1$

Eqn 2: $x + 2y + z = c_1$

Eqn 3: $-2x + y = -z$

The actual solution to the above set of equations is:

$$\begin{aligned}x &= -\frac{1}{2} + \frac{\sqrt{3}}{2} \\y &= c_1 - \frac{3\sqrt{3}}{2} + \frac{3}{2} \\z &= -c_1 - \frac{5}{2} + \frac{5\sqrt{3}}{2}\end{aligned}$$

and a second solution:

$$\begin{aligned}x &= -\frac{1}{2} - \frac{\sqrt{3}}{2} \\y &= c_1 + \frac{3\sqrt{3}}{2} + \frac{3}{2} \\z &= -c_1 - \frac{5}{2} - \frac{5\sqrt{3}}{2}\end{aligned}$$

Solving with sympy:

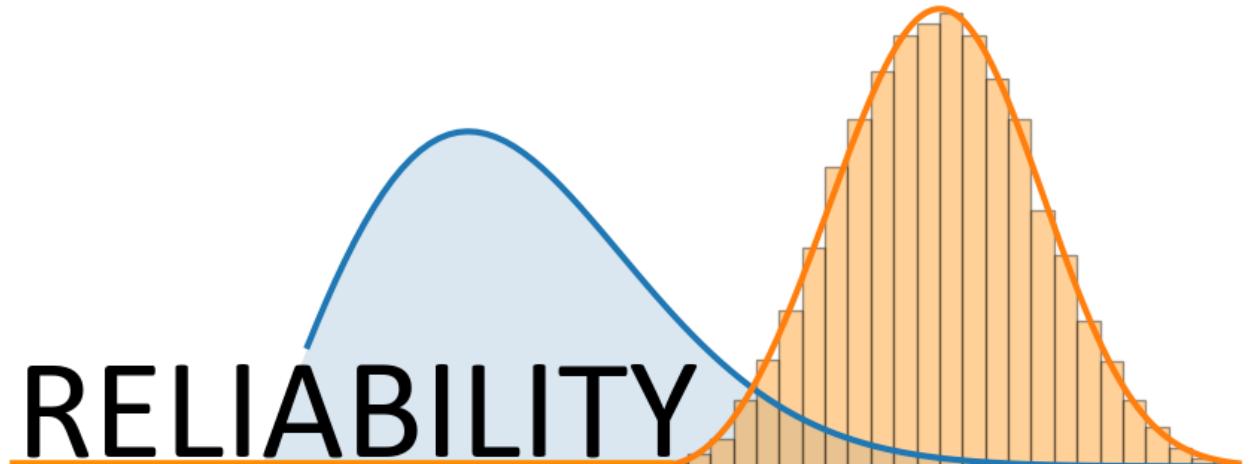
```
import sympy as sym
x,y,z = sym.symbols('x,y,z')
c1 = sym.Symbol('c1')
eq1 = sym.Eq(2*x**2+y+z,1)
eq2 = sym.Eq(x+2*y+z,c1)
eq3 = sym.Eq(-2*x+y,-z)
result = sym.solve([eq1,eq2,eq3],(x,y,z))
print(result)

"""
[(-1/2 + sqrt(3)/2, c1 - 3*sqrt(3)/2 + 3/2, -c1 - 5/2 + 5*sqrt(3)/2), (-sqrt(3)/2 - 1/2, c1 + 3/2 + 3*sqrt(3)/2, -c1 - 5*sqrt(3)/2 - 5/2)]
"""
```

Note

If you are using an iPython notebook, the display abilities are much better than the command line interface, so you can simply add `sym.init_printing()` after the import line and your equations should be displayed nicely.

A special thanks to Brigham Young University for offering [this tutorial](#).



RELIABILITY
A Python library for reliability engineering

STRESS-STRENGTH INTERFERENCE

Stress-Strength interference is a model to predict the probability of failure when the stress and strength probability distributions are known. Failure is defined as when stress > strength. If both the stress and strength distributions are Normal Distributions, then there exists a simple analytical solution which will give an exact result. To calculate stress-strength interference between distributions other than Normal Distributions requires the evaluation of an integral. These two cases are shown below using the functions `stress_strength_normal` (for two Normal Distributions) and `stress_strength` (for any two distributions).

49.1 Stress-Strength Interference for two Normal Distributions

The probability of failure for two Normal distributions is found using the equation:

$$\text{Probability of failure} = \Phi \left(\frac{\mu_{\text{strength}} - \mu_{\text{stress}}}{\sqrt{\sigma_{\text{strength}}^2 + \sigma_{\text{stress}}^2}} \right)$$

Where Φ is the standard Normal CDF with $\mu = 0$ and $\sigma = 1$

API Reference

For inputs and outputs see the [API reference](#).

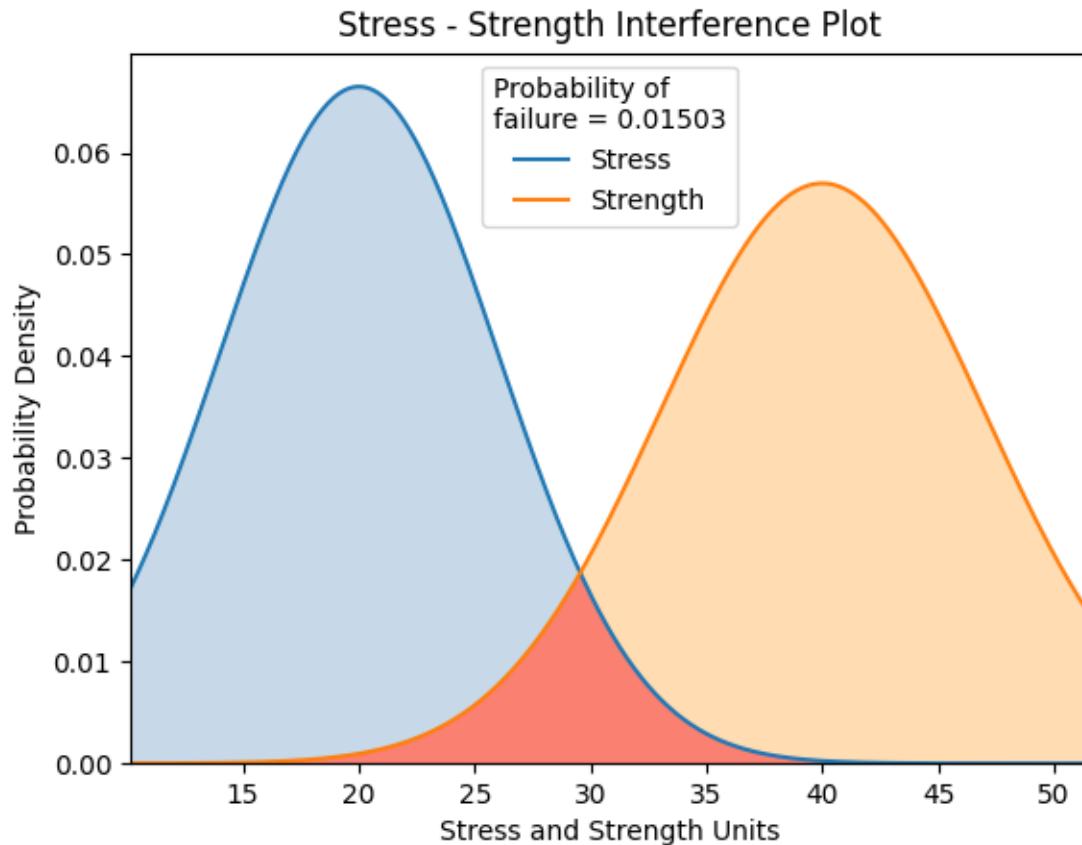
49.1.1 Example 1

In this example, we will create a stress and strength distribution (both of which are Normal distributions), and leaving everything else as default, we will see the results plotted and printed.

```
from reliability import Distributions
from reliability.Other_functions import stress_strength_normal
import matplotlib.pyplot as plt

stress = Distributions.Normal_Distribution(mu=20, sigma=6)
strength = Distributions.Normal_Distribution(mu=40, sigma=7)
stress_strength_normal(stress=stress, strength=strength)
plt.show()

"""
Stress - Strength Interference
Stress Distribution: Normal Distribution (=20,=6)
Strength Distribution: Normal Distribution (=40,=7)
Probability of failure (stress > strength): 1.50298 %
""
```



49.2 Stress-Strength Interference for any two Distributions

If either the stress or strength distributions are not Normal Distributions, the analytical method above can not be used and integration is required.

The equation to find the probability of failure any two distributions is:

$$\text{Probability of failure} = \int_0^{\infty} (f_{\text{strength}} \times R_{\text{stress}})$$

Where f is the PDF and R is the SF. The above integral can be evaluated using the `trapz` function in numpy:
`probability of failure = np.trapz(strength.PDF(x) * stress.SF(x), x)`

API Reference

For inputs and outputs see the [API reference](#).

49.2.1 Example 2

In this example, we will create a Weibull stress distribution and a Gamma strength distribution, and leaving everything else as default, we will see the results printed and the distribution plot.

```
from reliability import Distributions
from reliability.Other_functions import stress_strength
```

(continues on next page)

(continued from previous page)

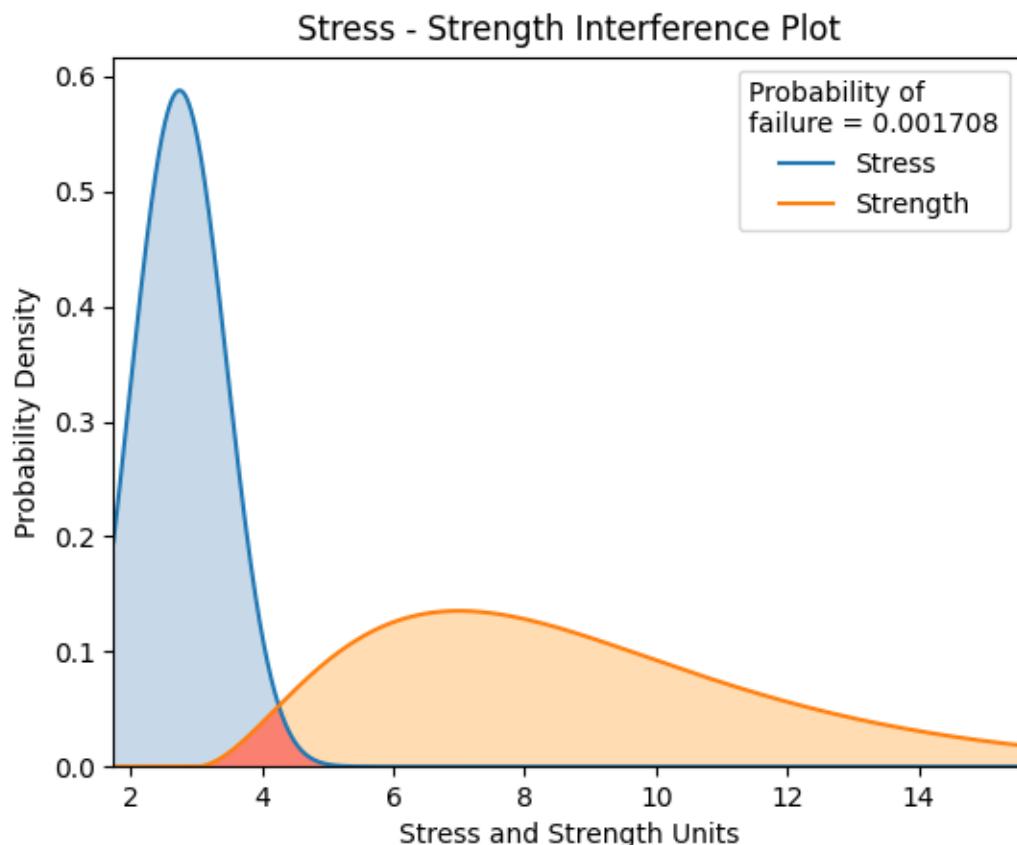
```

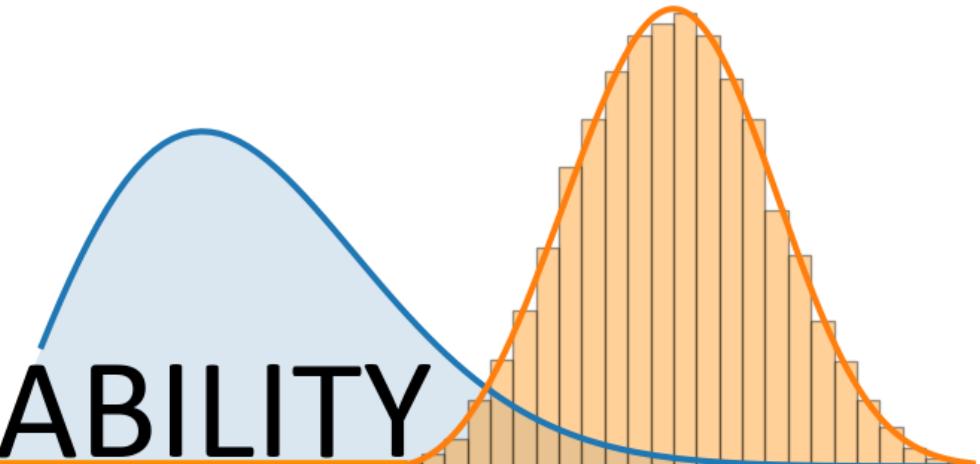
import matplotlib.pyplot as plt

stress = Distributions.Weibull_Distribution(alpha=2, beta=3, gamma=1)
strength = Distributions.Gamma_Distribution(alpha=2, beta=3, gamma=3)
stress_strength(stress=stress, strength=strength)
plt.show()

"""
Stress - Strength Interference
Stress Distribution: Weibull Distribution (=2,=3,=1.0)
Strength Distribution: Gamma Distribution (=2,=3,=3)
Probability of failure (stress > strength): 0.17078 %
"""

```





RELIABILITY
A Python library for reliability engineering

SIMILAR DISTRIBUTIONS

The function *similar_distributions* is a tool for finding the probability distributions that are most similar to an input distribution. It samples the CDF of an input distribution and then fits all other distributions to those samples to determine the best fitting and therefore most similar distributions.

API Reference

For inputs and outputs see the [API reference](#).

In the example below, we create a Weibull Distribution object using the `reliability.Distributions` module. We then provide the Weibull Distribution as input to *similar_distributions* and the output reveals the top 3 most similar distributions. The optional input of `include_location_shifted` has been set to False.

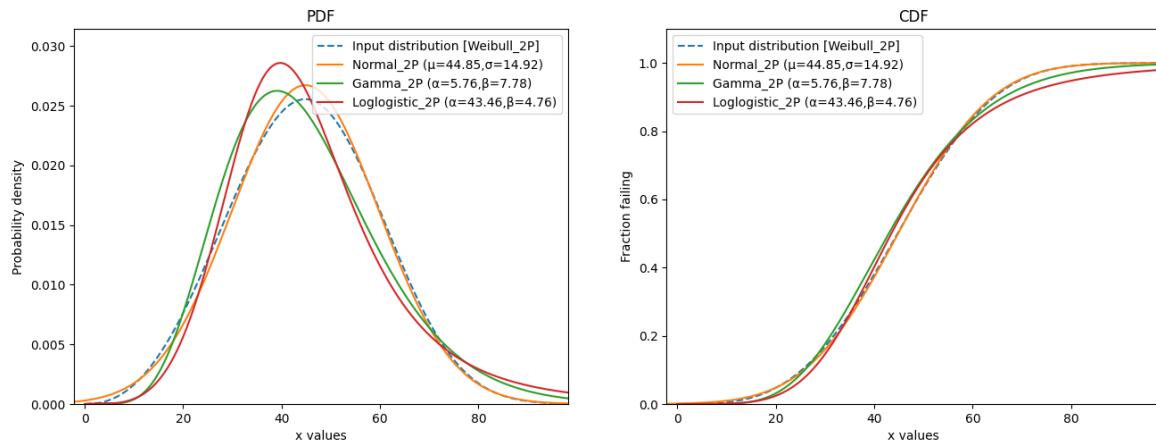
```
from reliability.Distributions import Weibull_Distribution
from reliability.Other_functions import similar_distributions
dist = Weibull_Distribution(alpha=50,beta=3.3)
similar_distributions(distribution=dist,include_location_shifted=False)

"""

Results from similar_distributions:
The input distribution was:
Weibull Distribution (=50,=3.3)

The top 3 most similar distributions are:
Normal Distribution (=44.8471,=14.9226)
Gamma Distribution (=5.7607,=7.785)
Loglogistic Distribution (=43.465,=4.7564)
""
```

Plot of similar distributions to Weibull Distribution ($\alpha=50, \beta=3.3$)



RELIABILITY
A Python library for reliability engineering

MAKE RIGHT CENSORED DATA

This function is a tool to convert complete data to complete and right censored data. Two methods are available which enable the production of either singly-censored or multiply-censored data. This function is often used in testing of the Fitters or Nonparametric functions when some right censored data is needed.

API Reference

For inputs and outputs see the [API reference](#).

51.1 Example 1

In this first example we will look at the production of **singly censored data**. That is data which is all censored at the same value (defined by threshold).

```
from reliability.Other_functions import make_right_censored_data
output = make_right_censored_data(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], threshold=6)
print('Failures:',output.failures)
print('Right Censored:',output.right_censored)

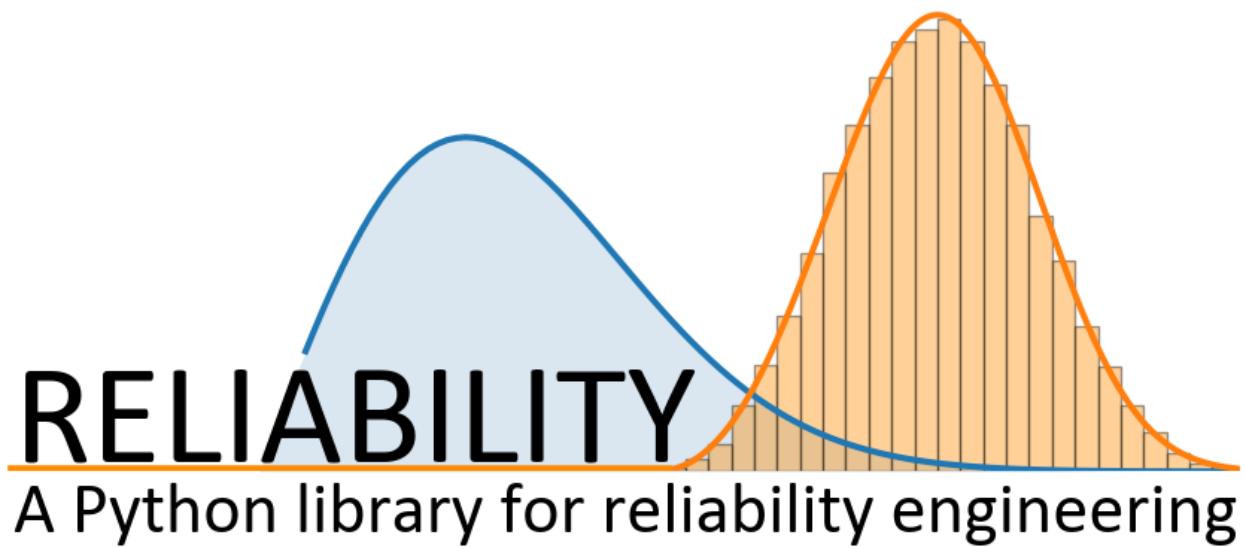
"""
Failures: [1 2 3 4 5 6]
Right Censored: [6 6 6 6] #the numbers 7 to 10 have been set equal to the threshold
""
```

51.2 Example 2

In this second example we will look at the production of **multiply censored data**. That is data which is censored at different values. The amount of data to be censored is governed by fraction_censored. If unspecified it will default to 0.5 resulting in 50% of the data being right censored. Note that there is randomness to the censoring. For repeatability set the seed. To apply the censoring, the data is first randomized then sliced into two portions; the items to be censored (based on the fraction_censored) and the items not to be censored. The items to be censored are each multiplied by a number between 0 and 1. These numbers are chosen randomly using `np.random.rand(len(items_to_censor))`. The `np.random.rand` algorithm samples from a uniform distribution bounded by 0 and 1. In theory, there are an infinite number of ways the censoring could be applied, since there are an infinite number of probability distributions bounded by 0 and 1 (think of the Beta distribution with different parameters). Custom censoring methods are not part of `make_right_censored` data, but it is important that as a user you understand the method used to multiply censor your data is as described above and this is not the only way to do it.

```
from reliability.Other_functions import make_right_censored_data
output = make_right_censored_data(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10], fraction_
    ↪censored=0.5, seed=1)
print('Failures:', output.failures)
print('Right Censored:', output.right_censored)

"""
Failures: [4 2 8 9 6] # half of the data has not been censored. It has been shuffled so
    ↪its order will be different from the order of the input data.
Right Censored: [1.16373222 6.69746037 6.5487735 4.23155458 0.31327352] # half of the
    ↪data has been censored at some value between 0 and the original value
""
```



MAKE ALT DATA

This function is used to generate accelerated life testing (ALT) data. It is primarily used for testing the functions within ALT_fitters. The function *Other_functions.make_ALT_data* accepts the life distribution (Weibull, Lognormal, Normal, Exponential) and the life-stress model (Exponential, Eyring, Power, Dual_Exponential, Dual_Power, Power_Exponential), along with the parameters of the model and will create an object with the data in the correct format for the ALT models contained within *reliability. ALT_fitters*. The function contains many more inputs than are required and these inputs will only be used if they are part of the model. Please see the [equations](#) of the ALT model you are using to determine what parameters are required. The function is designed to automatically censor a fraction of the data using the input `fraction_censored`.

API Reference

For inputs and outputs see the [API reference](#).

52.1 Example 1

In this first example we will create ALT data from a Weibull_Eyring model. To verify the accuracy of the fitter we can compare the fitted model's parameters to the parameters we used to generate the data. Note that we only need to specify a, c, and beta since these are the three parameters of the Weibull_Exponential model.

```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Weibull_Eyring

ALT_data = make_ALT_data(distribution='Weibull', life_stress_model='Eyring', a=1500, c=-10,
                           beta=2, stress_1=[500, 400, 350], number_of_samples=100, fraction_censored=0.2, seed=1)
Fit_Weibull_Eyring(failures=ALT_data.failures, failure_stress=ALT_data.failure_stresses,
                    right_censored=ALT_data.right_censored, right_censored_stress=ALT_data.right_censored_
                    stresses, use_level_stress=300, show_probability_plot=False, show_life_stress_
                    plot=False)

"""
Results from Fit_Weibull_Eyring (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 240/60 (20% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
      a        1458.14      92.7751   1276.31   1639.98
      c       -10.1231      0.230245  -10.5743   -9.6718
    beta       1.92551      0.0921087   1.75318   2.11477
```

(continues on next page)

(continued from previous page)

stress	original alpha	original beta	new alpha	common beta	beta change	acceleration
500	945.271	1.94553	920.348	1.92551	-1.03%	
400	2193.3	1.75376	2385.03	1.92551	+9.79%	
350	4822.96	2.11256	4588.29	1.92551	-8.85%	
300	2.33615					
<i>Goodness of fit</i>						
<i>Log-likelihood</i> -2000.5						
<i>AICc</i> 4007.09						
<i>BIC</i> 4018.12						
At the use level stress of 300, the mean life is 9507.77152						
""						

52.2 Example 2

In this second example we will create ALT data from a Lognormal_Dual_Power model. To verify the accuracy of the fitter we can compare the fitted model's parameters to the parameters we used to generate the data. Note that we only need to specify c, m, n, and sigma since these are the four parameters of the Lognormal_Dual_Power model.

```
from reliability.Other_functions import make_ALT_data
from reliability. ALT_fitters import Fit_Lognormal_Dual_Power

use_level_stress = [250, 7]
ALT_data = make_ALT_data(distribution='Lognormal', life_stress_model='Dual_Power',  

c=1e15, m=-4, n=-2, sigma=0.5, stress_1=[500, 400, 350, 420, 245], stress_2=[12, 8, 6,  

9, 10], number_of_samples=100, fraction_censored=0.5, seed=1, use_level_stress=use_level_stress)
Fit_Lognormal_Dual_Power(failures=ALT_data.failures, failure_stress_1=ALT_data.failure_stresses_1, failure_stress_2=ALT_data.failure_stresses_2, right_censored=ALT_data.right_censored, right_censored_stress_1=ALT_data.right_censored_stresses_1, right_censored_stress_2=ALT_data.right_censored_stresses_2, use_level_stress=use_level_stress, show_probability_plot=False, show_life_stress_plot=False)
print('The mean life from the true model is',ALT_data.mean_life_at_use_stress)

"""

Results from Fit_Lognormal_Dual_Power (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 250/250 (50% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
c 9.48288e+14 6.67128e+14 2.38844e+14 3.76502e+15
m -3.9731 0.11982 -4.20795 -3.73826
n -1.99518 0.123271 -2.23678 -1.75357
sigma 0.491039 0.0212097 0.45118 0.534419
```

(continues on next page)

(continued from previous page)

stress	original mu	original sigma	new mu	common sigma	sigma change	acceleration
<i>factor</i>						
500, 12	4.85616	0.496646	4.83656	0.491039	-1.13%	46.
↳ 0321						
420, 9	6.15963	0.525041	6.10326	0.491039	-6.48%	↳
↳ 12.97						
400, 8	6.39217	0.392671	6.53211	0.491039	+25.05%	8.
↳ 44684						
350, 6	7.69905	0.550747	7.63662	0.491039	-10.84%	2.
↳ 79905						
245, 10	8.02546	0.457947	8.03454	0.491039	+7.23%	1.
↳ 88017						
<i>Goodness of fit</i>						
<i>Log-likelihood</i> -1859.62						
AICc	3727.32					
BIC	3744.1					
<i>At the use level stress of 250, 7, the mean life is 6545.04098</i>						
<i>The mean life from the true model is 5920.122530308318</i>						
'''						

Recommended values

Some parameters are more suitable than others for these models. The following parameters are recommended for use as a starting point if you are having difficulty in determining the rough order of magnitude of the values you should use:

- Exponential: a=2000, b=10
- Eyring: a=1500, c=-10
- Power: a=5e15, n=-4
- Dual_Exponential: a=50, b=0.1, c=500
- Dual_Power: c=1e15, m=-4, n=-2
- Power_Exponential: a=200, c=400, n=-0.5



RELIABILITY
A Python library for reliability engineering

CROSSHAIRS

This function provides interactive crosshairs on matplotlib plots. The crosshairs will follow the users' mouse cursor when they are near lines or points and will snap to these lines and points. Upon a mouse click the crosshairs will add an annotation to the plot. This annotation can be dragged to a new position. To delete the annotation, right click on it. To temporarily hide all annotations, toggle 'h' on your keyboard.

Note that crosshairs should be called after everything is added to the plot (but before plt.show()) so that the objects in the plot are identified for the 'snap to' feature. If something is added to the plot after calling crosshairs then you will not be able to move the crosshairs onto it.

If your interactive development environment does not generate the plot in its own window then your plot is not interactive and this will not work. For iPython notebook users, the interactive window should be available by typing "%matplotlib qt" after importing matplotlib as described [here](#).

There are some customisable attributes of the crosshairs and annotations as described in the API reference.

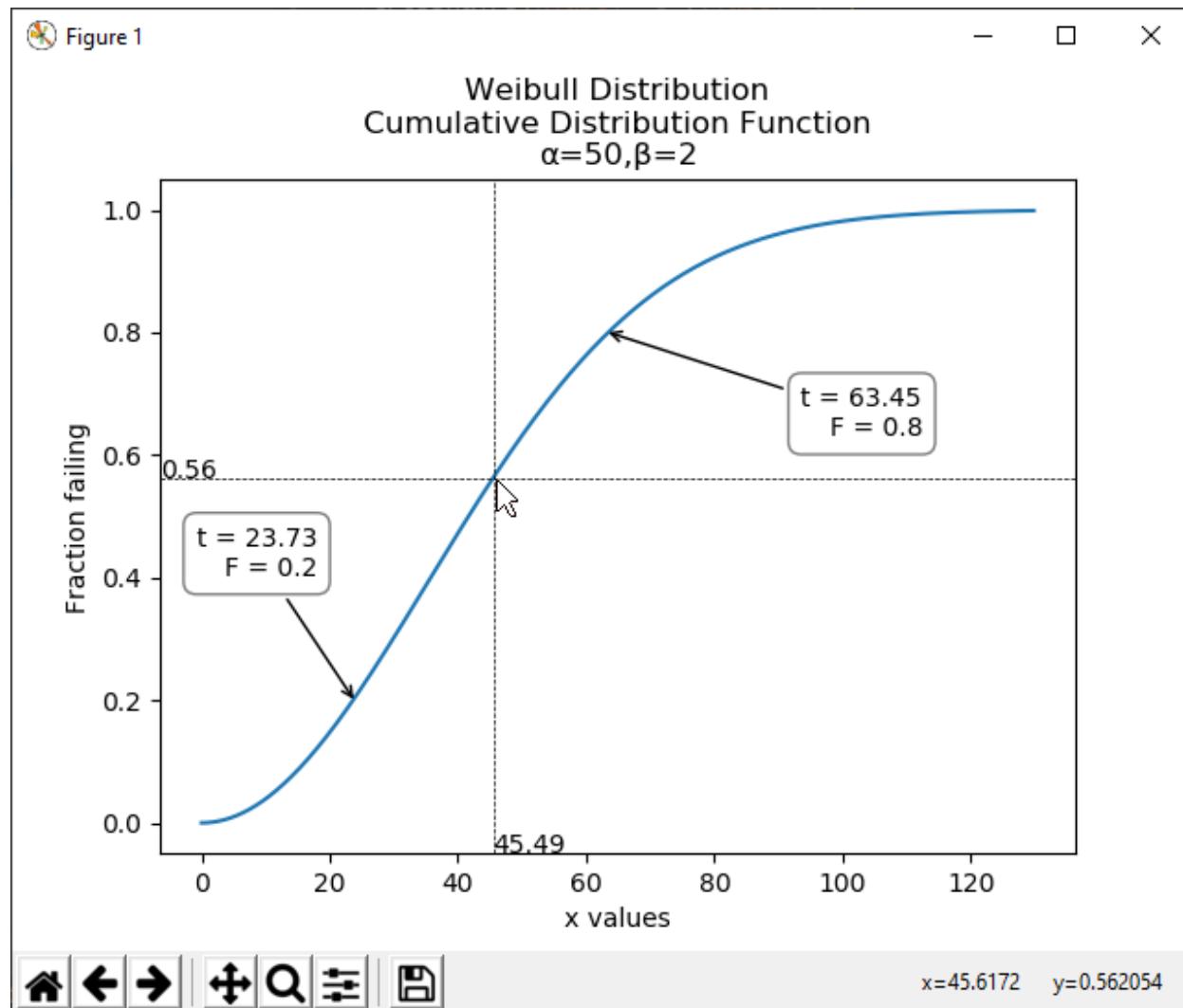
ⓘ API Reference

For inputs and outputs see the [API reference](#).

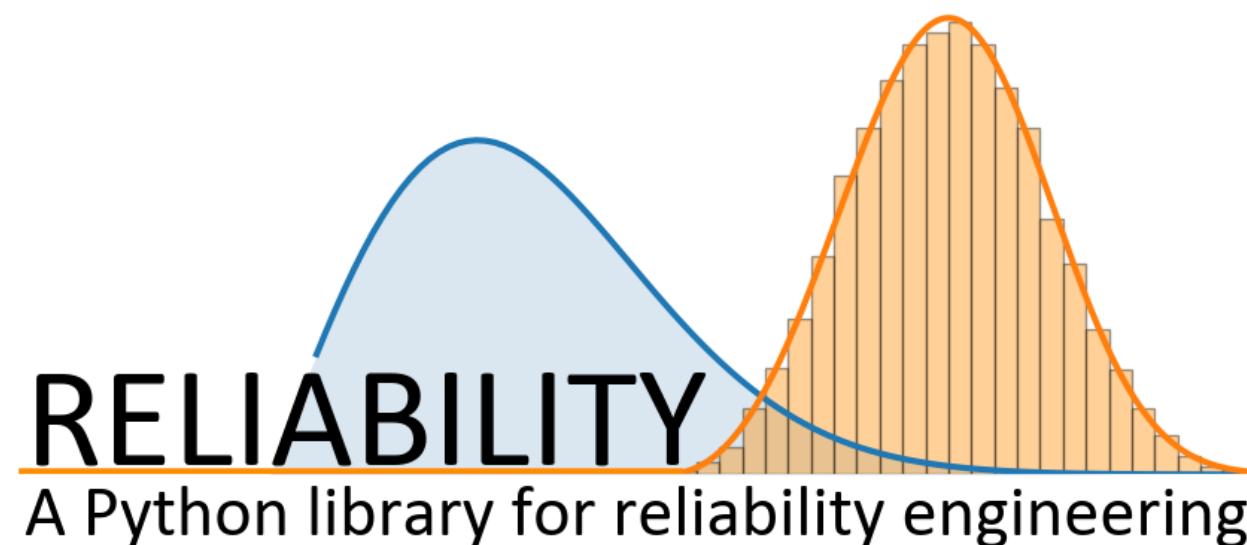
In the following example, we see the crosshairs being used to display the value of the Weibull CDF. The dynamic nature of this feature is shown in the video at the bottom of this page.

```
from reliability.Other_functions import crosshairs
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt

Weibull_Distribution(alpha=50,beta=2).CDF()
crosshairs(xlabel='t',ylabel='F') #it is important to call this last
plt.show()
```



A special thanks goes to Antony Lee, the author of [mplcursors](#). The crosshairs function works using [mplcursors](#) to enable the 'snap to' feature and the annotations. Antony was very helpful in getting this to work.



CHAPTER FIFTYFOUR

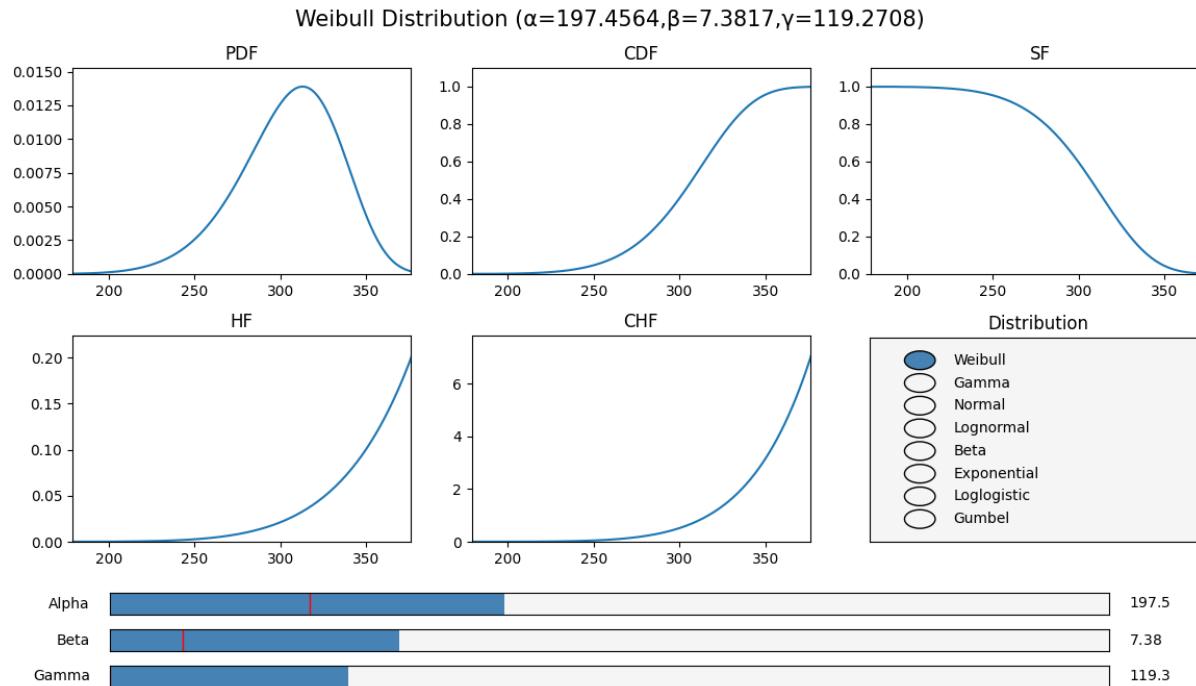
DISTRIBUTION EXPLORER

The distribution explorer is a simple way to explore the shape of each distribution based on its parameters. To achieve this, an interactive window is shown with the 5 characteristic functions (PDF, CDF, SF, HF, CHF) of each probability distribution. Parameters can be changed using slider widgets. Distributions can be changed using the radio button widget.

There are no inputs or outputs. Everything is done within the interactive matplotlib window. Please see the video for an example of the interactive features of the distribution explorer.

To open the distribution explorer, use the following code:

```
from reliability.Other_functions import distribution_explorer  
distribution_explorer()
```





RELIABILITY
A Python library for reliability engineering

HISTOGRAM

This function plots a histogram using the matplotlib histogram (plt.hist()), but adds some additional features. Default formatting is improved, the number of bins is optimized by default, and there is an option to shade the bins white above a chosen threshold. If you would like to specify the number of bins rather than having the optimal number calculated, then the bins argument allows this.

API Reference

For inputs and outputs see the [API reference](#).

The following example shows the difference between the appearance of the default histogram in matplotlib, and the histogram in reliability.

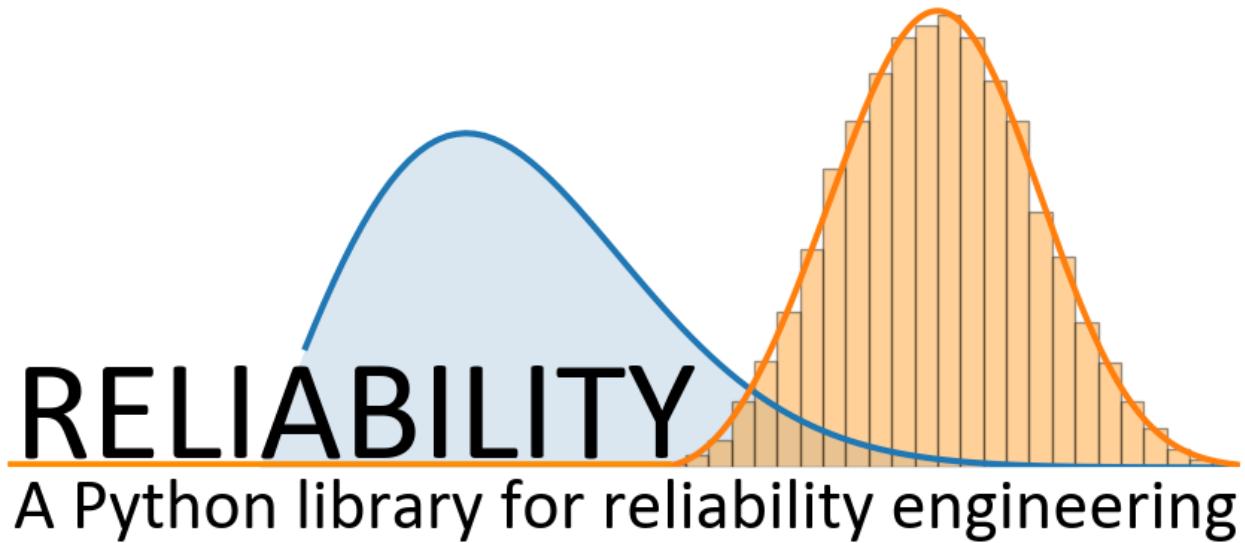
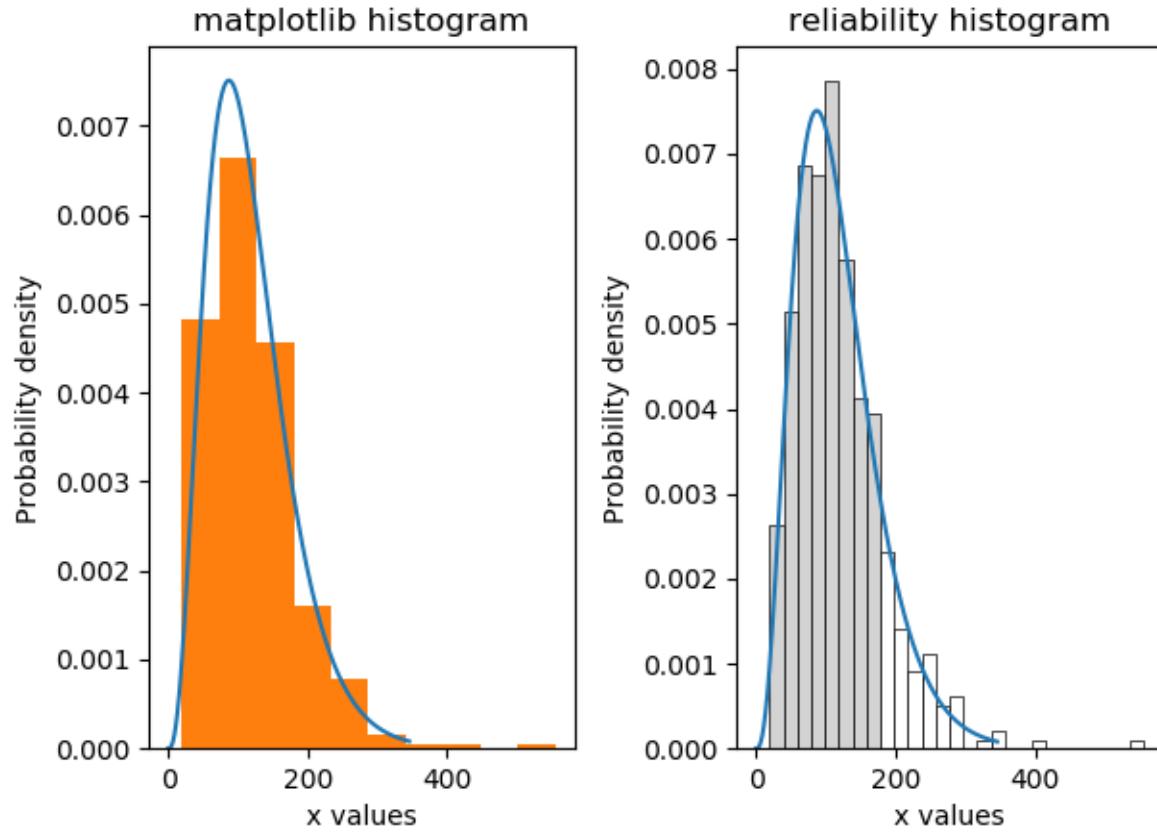
```
from reliability.Distributions import Gamma_Distribution
from reliability.Fitters import Fit_Gamma_2P
from reliability.Other_functions import make_right_censored_data, histogram
import matplotlib.pyplot as plt

a = 30
b = 4
threshold = 180 # this is used when right censoring the data
dist = Gamma_Distribution(alpha=30, beta=4)
raw_data = dist.random_samples(500, seed=2) # create some data. Seeded for repeatability
data = make_right_censored_data(raw_data, threshold=threshold) # right censor the data
gf = Fit_Gamma_2P(failures=data.failures,right_censored=data.right_censored,show_
↪probability_plot=False,print_results=False)

plt.subplot(121)
gf.distribution.PDF()
plt.hist(raw_data, density=True) # default histogram from matplotlib
plt.title('matplotlib histogram')

plt.subplot(122)
gf.distribution.PDF()
histogram(raw_data, white_above=threshold) # histogram from reliability - better_
↪formatting, optimal bin width by default, white_above option
plt.title('reliability histogram')

plt.subplots_adjust(right=0.95, wspace=0.38)
plt.show()
```



WHAT IS CENSORED DATA

Censored data is any data for which we do not know the exact event time. There are three types of censored data; right censored, left censored, and interval censored. Data for which the exact event time is known is referred to as complete data. In addition to the three types of censored data, there are also two ways in which censored data may be grouped; singly censored or multiply censored. Tests may be terminated after a certain time (time-terminated) or after a certain number of failures (failure-terminated). Each of these types of test termination lead to a different type of censoring (type I and type II censoring). An explanation of each of these terms is provided below.

In the context of reliability engineering we typically refer to events as “failures”. In other industries a range of terminology may be used to describe events. These often include “deaths” if studying living things such as in medical studies, or simply “events” if studying natural phenomena like flood events. Throughout *reliability* we will use “failures” to describe events.

It is common to refer to data as “times” but a variety of other units of measure may also be used such as cycles, rounds, landings, etc. This depends on the type of data being collected and the units in which the life is measured. Throughout the Python reliability library we will use “times” to describe the units of measure of when failures occur.

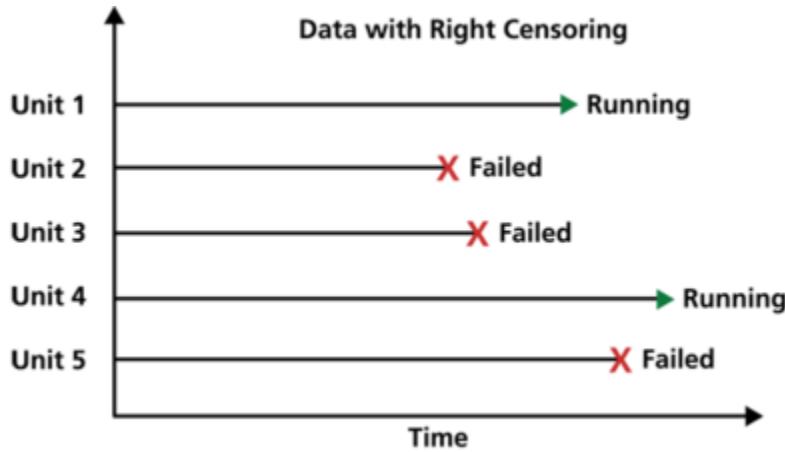
56.1 Complete data

Complete data is data for which we know the exact failure time. This is seen when all items under analysis have their exact failure times recorded. For example if we have 10 components under test, all of these components fail during the test, and the exact failure time is recorded then we have complete data. It is common to have a mixture of complete and censored data as most tests will have some failures and some survivors. It is not possible to perform an analysis with no failures and only right censored data. It is possible to perform analysis using all interval censored data as interval censoring (explained below) represents failures but the exact failure time is not known.



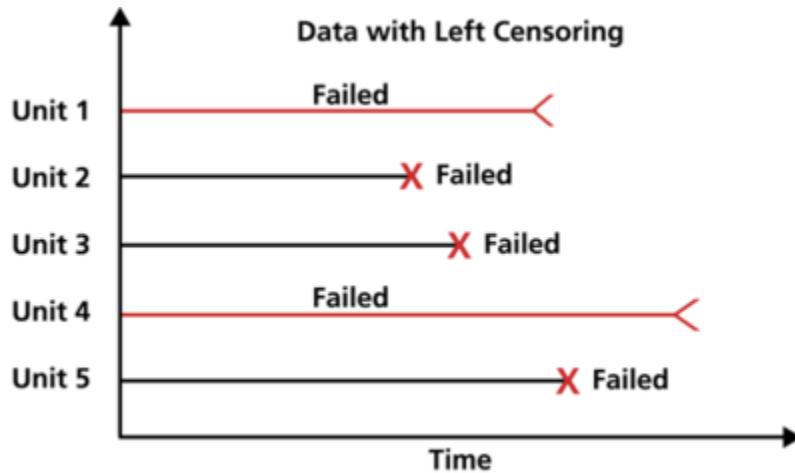
56.2 Right censored data

Right censored data is data for items that have not yet failed. They are considered “still alive” as their failure time has not yet occurred, though it is expected to occur at some point in the future. For example, consider a fatigue test with 10 components under test. The test is run for 100000 cycles and during this time 8 of the components fail. We record the 8 failure times and we also record the end of the test as the “right censored” time for the 2 unfailed components. Right censored data is also referred to as “suspensions” or “survivors”.



56.3 Left censored data

Left censored data is data for items that failed before the start of the test. Left censored data is the same as interval censored data, however the lower interval is 0. We rarely see this type of data in reliability engineering.



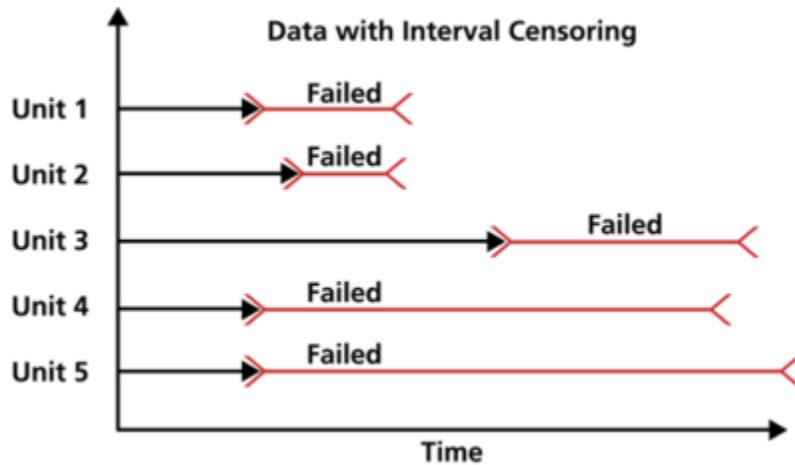
If you need to work with left censored data, please check out [lifelines](#) or [survival](#) as reliability only supports complete and right censored data.

56.4 Interval censored data

Interval censored data is when the exact failure time is not known but the lower and upper bounds of an interval surrounding the failure are known. For example, consider a failure that is discovered as part of an overhaul (deeper maintenance event). The exact failure time is not known but we do know that the item had not failed at the time of the last overhaul and it did fail before the current overhaul. We record the lower and upper times and treat this failure as interval censored data.

We could consider all censored data to be interval censored data, with the following bounds:

- right censored data: lower bound = end of observation time, upper bound = infinity
- left censored data: lower bound = 0, upper bound = start of observation time
- interval censored data: lower bound = last observation time before failure, upper bound = first observation time after failure



If you need to work with interval censored data, please check out [lifelines](#) or [surpyval](#) as reliability only supports complete and right censored data.

56.5 Singly censored data

This is not a type of censored data, but it used to describe how censored data is grouped. In singly censored data, all censoring times are the same. As an example we may have the following failures and right censored times (right censored represented by +): 50, 56, 78, 89, 100+, 100+, 100+. We can say this is “singly censored” as all the censoring occurs at a single point. This is often the case in a test where the end of the test is used as the same censoring time for all unfailed items.

56.6 Multiply censored data

This is not a type of censored data, but it used to describe how censored data is grouped. In multiply censored data, the censoring times occur at many different times. As an example we may have the following failures and right censored times (right censored represented by +): 50, 55+, 56, 72+, 74+, 78, 89, 98+. We can say this is “multiply censored” as the censoring times occur at multiple points. This is frequently seen when items have different start times or different amounts of usage so their times in service are not aligned. While the end of the observation period may be the same (in terms of the calendar date), the accumulated life will be different between items so their censoring times do not necessarily align.

56.7 Type I and Type II censoring

If a test is stopped after a certain amount of time then the test is “time-terminated”. This type of test will produce type I censoring. If a test is stopped after a certain number of failures then the test is “failure-terminated”. This type of test will produce type II censoring. The formulas for some calculation methods (such as in some [MTBF calculations](#)) differ between type I and type II censoring so it is important to know what type of censoring your test produces and whether the formulas being used need to be adjusted to reflect the type of censoring.

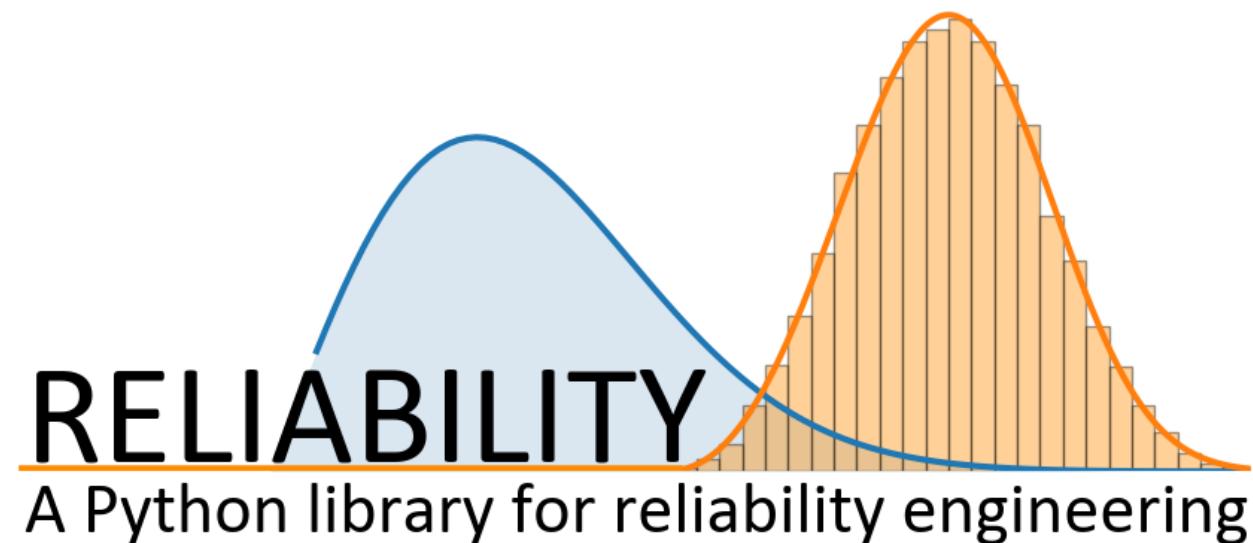
56.8 Considering failures as right censored

Items may fail by a variety of failure modes. If the failure modes are being analysed separately, we must consider any failures that did not occur via the failure mode under analysis, as right censored. Consider a fleet of vehicles with the following failures:

- Vehicle 1 - brake failure at 45000 km
- Vehicle 2 - brake failure at 37000 km
- Vehicle 3 - engine failure at 55000 km
- Vehicle 4 - battery failure at 28000 km
- Vehicle 5 - brake failure at 22000 km

If we are studying brake failures in our sample of 5 vehicles, we should use the following failure times: 22000, 28000+, 37000, 45000, 55000+. In this case the failures for vehicles 3 and 4 are treated as right censored data (shown with +) since the failure mode observed did not match the failure mode under analysis.

If you find any errors, think this needs to be explained better, or have any suggestions for improvements, please email me (alpha.reliability@gmail.com).



HOW ARE THE PLOTTING POSITIONS CALCULATED

When we want to fit a probability distribution to a dataset (such as failure times), there are a variety of methods we can use. The most popular of these methods are Least Squares estimation (LS) and Maximum Likelihood Estimation (MLE). As a prerequisite to Least Squares Estimation, we need an estimate of the CDF (y-values) for a given dataset (x-values). Once we have both the x-values and the y-values we can plot the points (x,y) on a graph. These are called the plotting positions.

There are a variety of different algorithms for obtaining the plotting positions, but the most popular is the rank adjustment method which will be described in detail below. To introduce the algorithm, we will start with complete data (ie. no censoring) and then we will see how the algorithm needs to be modified when we have censored data.

57.1 Rank adjustment for complete data

The plotting positions algorithm for complete data is as follows:

1. sort the data in ascending order
2. create a column (i) for the rank from 1 to n.
3. estimate the CDF using $y = \frac{i-a}{n+1-2a}$.

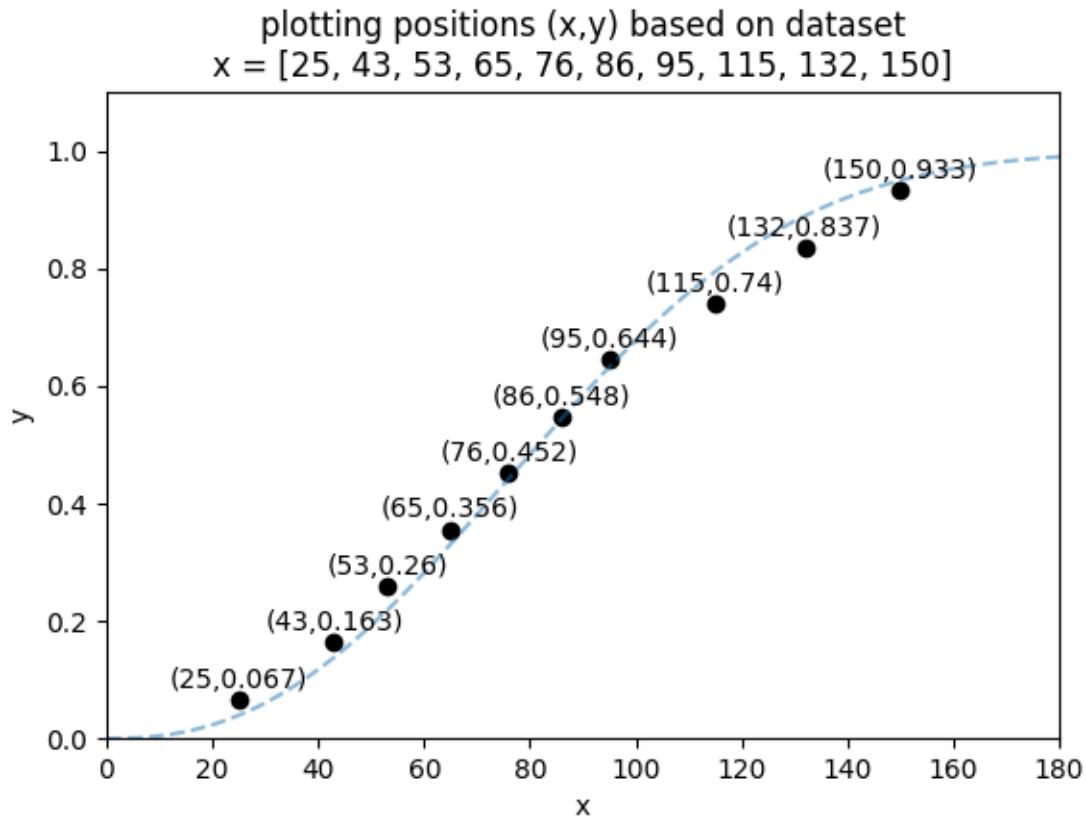
Where “n” is the number of items (len(x)) and “a” is the heuristic constant. For this example we will let $a = 0.3$ which will give Benard’s approximation of the median rank plotting positions (the default in most software). Other heuristics are discussed below. Something you may notice about the formula for y is that it is independent of x. You will always obtain the same y values for any array of x values of the same length.

Let’s do an example using the dataset $x = [25, 43, 53, 65, 76, 86, 95, 115, 132, 150]$

x	i	$y = (i-a)/(n+1-2a)$
25	1	0.067307692
43	2	0.163461538
53	3	0.259615385
65	4	0.355769231
76	5	0.451923077
86	6	0.548076923
95	7	0.644230769
115	8	0.740384615
132	9	0.836538462
150	10	0.932692308

a	0.3
n	10

We can now plot the x and y values to obtain the plotting positions as shown in the image below. The dashed blue line is a Weibull_2P distribution that has been fitted to the data. This is just for illustrative purposes to show that the empirical CDF (the calculated y-values) and the CDF of the fitted model should roughly align.



57.2 Rank adjustment for censored data

The algorithm above provides the rank (i) simply by using the item number (1 to n) when the x-values are sorted. When we have right censored data, the ranks need to be adjusted using a few modifications to the original algorithm. The rank adjustment algorithm for right censored data is as follows:

1. sort the data in ascending order
2. create a column (i) for the rank from 1 to n.
3. create a column (m) of the reverse rank from n to 1.
4. calculate the adjusted rank as $j_i = j_{i-1} + \frac{n+1-j_{i-1}}{1+m}$. If the first item is a failure, then the adjusted rank of the first failure is $j_1 = 1$. If the first item is not a failure, the the adjusted rank of the first failure is $j_1 = \frac{\text{number of leading censored values}}{n-1}$. Leave the rows with censored items blank.
5. estimate the CDF using $y = \frac{j-a}{n+1-2a}$.

Let's do an example using the dataset $x = [150, 340+, 560, 800, 1130+, 1720, 2470+, 4210+, 5230, 6890]$. In this dataset the values with + are right censored.

x	i	reverse rank (m)	adjusted rank (j)	$y = (j-a)/(n+1-2a)$
150	1	10	1	$=(1-0.3)/(11-2 \times 0.3)=0.067$
340+	2	9		
560	3	8	$=1+(11-1)/(1+8)=2.111$	$=(2.111-0.3)/(11-2 \times 0.3)=0.174$
800	4	7	$=2.111+(11-2.111)/(1+7)=3.222$	$=(3.222-0.3)/(11-2 \times 0.3)=0.281$
1130+	5	6		
1720	6	5	$=3.222+(11-3.222)/(1+5)=4.519$	$=(4.519-0.3)/(11-2 \times 0.3)=0.406$
2470+	7	4		
4210+	8	3		
5230	9	2	$=4.519+(11-4.519)/(1+2)=6.679$	$=(6.679-0.3)/(11-2 \times 0.3)=0.613$
6890	10	1	$=6.679+(11-6.679)/(1+1)=8.840$	$=(8.840-0.3)/(11-2 \times 0.3)=0.821$

a	0.3
n	10
n+1	11

You can check this using Python like this:

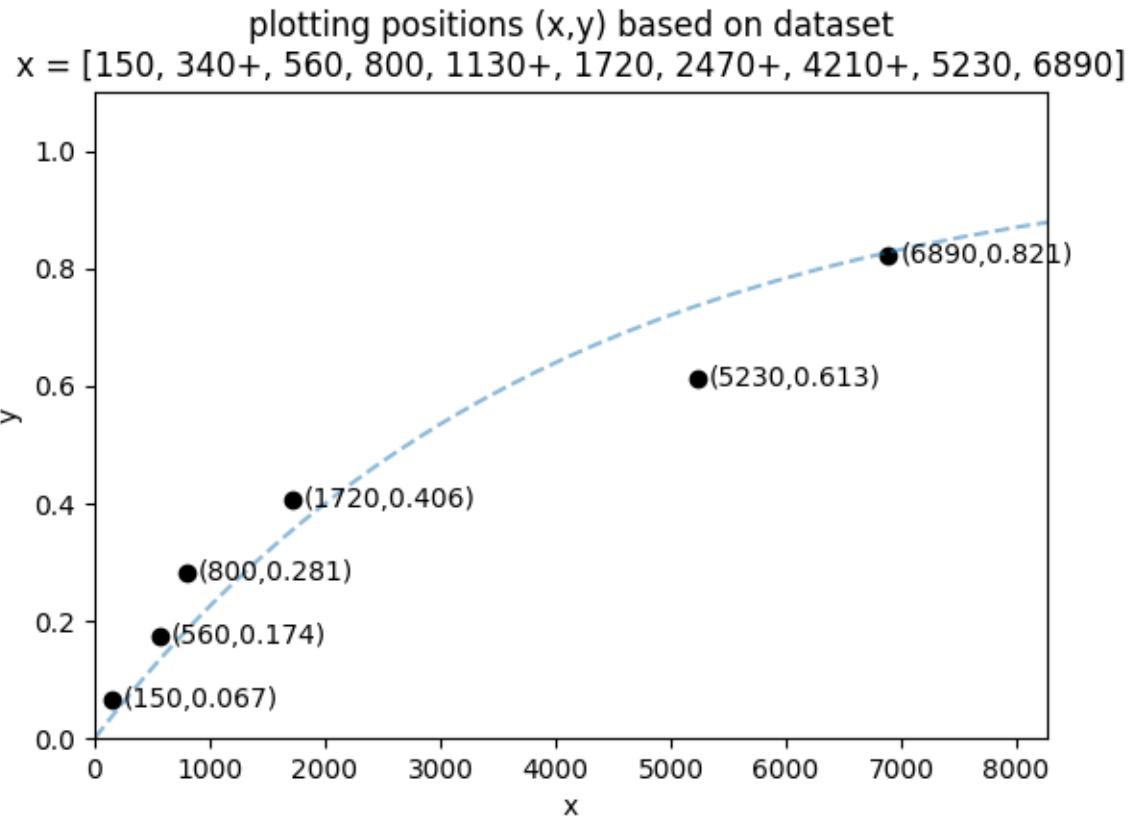
```
from reliability.Probability_plotting import plotting_positions
failures = [150, 560, 800, 1720, 5230, 6890]
right_censored = [340, 1130, 2470, 4210]
x,y=plotting_positions(failures=failures,right_censored=right_censored)

print('x =',x)
print('y =',y)

"""
x = [ 150  560  800 1720 5230 6890]
y = [ 0.06730769  0.1741453   0.28098291  0.40562678  0.61336657  0.82110636]
"""

```

We can now plot the x and y values to obtain the plotting positions as shown in the image below. The dashed blue line is an Exponential_1P distribution that has been fitted to the data. This is just for illustrative purposes to show that the empirical CDF (the calculated y-values) and the CDF of the fitted model should roughly align. Note that only the failures are plotted as the right censored data does not have an empirical estimate for the CDF.



57.3 Plotting heuristics

The plotting positions algorithm uses the formula $y = \frac{i-a}{n+1-2a}$. We can set the heuristic constant “a” to be any value from 0 to 1 and we will get different estimates. Some of these are better than others, but the most popular is $a = 0.3$ (Benard’s approximation of the median ranks (typically just called “median rank”)) as this is generally the most accurate.

Published literature has been produced on the following Heuristics:

Method	Heuristic (a)
Blom	0.375
Benard (Median)	0.3
Hazen (Modified Kaplan Meier)	0.5
Herd-Johnson (Mean)	0
Modal	1
Beard	0.31
Griengorten	0.44
Larsen	0.567
One-Third	1/3
Cunane	0.4

There is another modification to the $y = \frac{i-a}{n+1-2a}$ formula to make it $y = \frac{i-a}{n+b}$ which allows “b” to be independent of

“a”. The Kaplan Meier method uses this formula with $a=0$ and $b=0$ (making it $y = \frac{i}{n}$). The [Filliben estimate](#) also uses this method with further modifications to the first and last items of the CDF.

The formula of $y = \frac{i-a}{n+1-2a}$ is not the only way to obtain plotting positions. There are other methods involving [Beta](#) and [F distributions](#).

Within reliability, the heuristic constant “a” is accepted for all the probability plots as well as in the [Nonparametric.RankAdjustment](#) method. The median ranks method is generally the default for most software (including in Reliasoft and MINITAB).

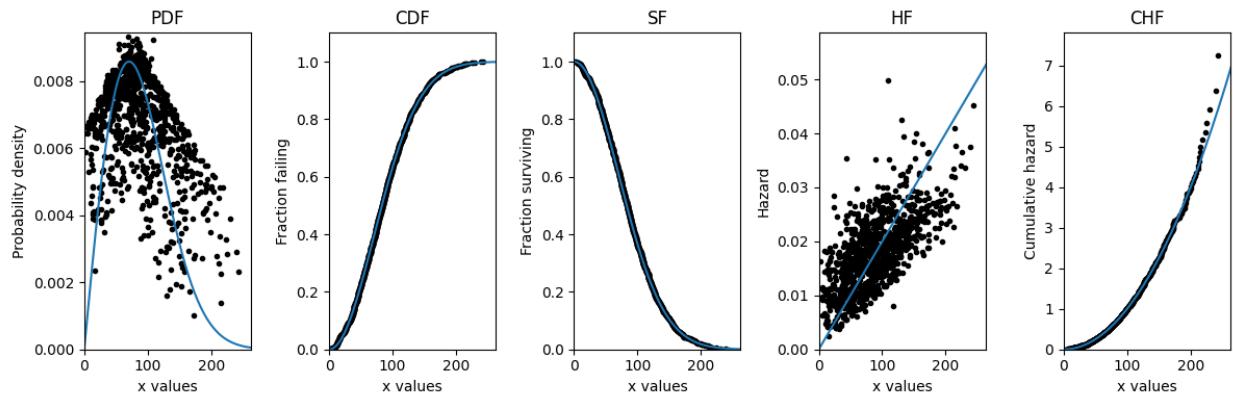
57.4 Transformations for PDF, SF, HF, CHF

The algorithms described above provide the empirical estimate of the CDF. With some simple [transformations](#) it is possible to obtain the empirical estimate of the SF and CHF. Less commonly (but still mathematically possible) we can obtain the empirical estimate of the PDF and HF. As you can see in the image below, the PDF and HF do not form smooth curves due to the need to take the derivative of a non-continuous function. The following example illustrates how `plot_points` can be used to generate a scatterplot of the plotting positions for any of the five functions. The Weibull distribution used to generate the data is also overlaid for comparison.

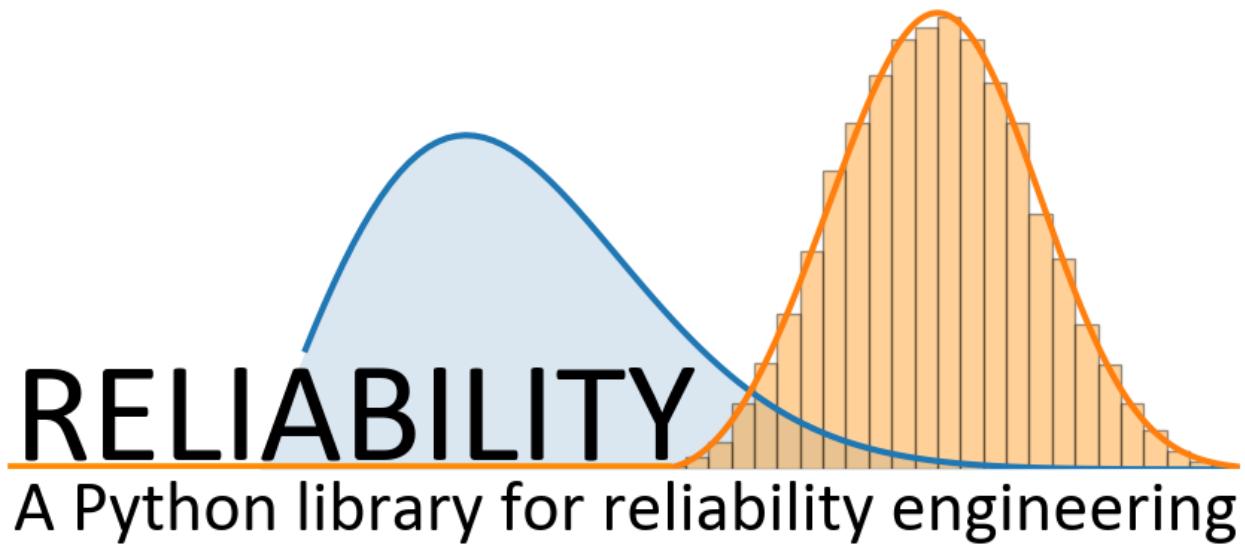
```
from reliability.Distributions import Weibull_Distribution
from reliability.Probability_plotting import plot_points
import matplotlib.pyplot as plt

dist = Weibull_Distribution(alpha=100,beta=2)
data = dist.random_samples(1000,seed=1)

functions = ['PDF', 'CDF', 'SF', 'HF', 'CHF']
i = 0
for function in functions:
    plt.subplot(151+i)
    if function == 'PDF':
        dist.PDF()
    elif function == 'CDF':
        dist.CDF()
    elif function == 'SF':
        dist.SF()
    elif function == 'HF':
        dist.HF()
    elif function == 'CHF':
        dist.CHF()
    plot_points(failures=data,func=function)
    plt.title(function)
    i+=1
plt.gcf().set_size_inches(12,4)
plt.tight_layout()
plt.show()
```



If you find any errors, think this needs to be explained better, or have any suggestions for improvements, please email me (alpha.reliability@gmail.com).



HOW DOES LEAST SQUARES ESTIMATION WORK

Least Squares Estimation is a method of fitting a probability distribution to a set of data. It works by transforming the CDF (by linearizing the equation) and then using least squares estimation to find the parameters of the line of best fit for the linearized data. We then perform the reverse transform (un-linearizing the linearized CDF) to obtain the parameters of the probability distribution which we were interested in fitting.

Least Squares Estimation (henceforth referred to as Least Squares) is also known as the method of probability plotting because we can either transform the data or transform the plot in order to get a straight line. Transforming the plot results in a probability plot, hence the name “method of probability plotting”. There is not really any plotting necessary as it can all be done with equations, but when plotted the equations provide an excellent visual illustration of the process.

In addition to Least Squares, there are several other methods to obtain the parameters of a probability distribution including Maximum Likelihood Estimation (MLE), Method of Moments, Mean Square Error, and Maximum Product of Spacings Estimation. The most popular methods are Least Squares and MLE which are both implemented in *reliability*. Users seeking to use the other methods listed will find them as part of the [Surpyval](#) library.

58.1 The least squares algorithm

We can either transform the data (using the same transform required to linearize the CDF) or we can transform the plot (into a probability plot). The probability plotting method involves special [probability paper](#), the line of best fit is typically done by hand, and the process of extracting the parameters is very rough. Many decades ago, when computers were not readily available, the probability plotting method was the most popular way to estimate distribution parameters. All probability plots that software presents to you are done by transforming the data and displaying the result on the probability plot. The explanations that follow all involve transformation of the data and examples with Excel and Python to find the line of best fit.

The least squares algorithm is as follows:

1. Obtain the plotting positions (y-values) for a given dataset (x-values).
2. Transform the x and y plotting positions based on the transformations necessary to linearize the CDF (described below).
3. Use simple linear regression to fit a line (see the section below on RRX and RRY) and extract the parameters of the line.
4. Convert the parameters of the fitted line to the parameters of the probability distribution using the inverse transform for the CDF (the reverse of step 2).

This algorithm is best explained with an example. For this example we will use least squares estimation to fit a Weibull Distribution to the following dataset $x = [25, 43, 53, 65, 76, 86, 95, 115, 132, 150]$. We firstly need the [plotting positions](#). In Python this is done as:

```
from reliability.Probability_plotting import plotting_positions
data = [25, 43, 53, 65, 76, 86, 95, 115, 132, 150]
```

(continues on next page)

(continued from previous page)

```
t,F = plotting_positions(failures=data)
print('t =',t)
print('F =',F)

"""
t = [ 25.  43.  53.  65.  76.  86.  95. 115. 132. 150.]
F = [0.06730769 0.16346154 0.25961538 0.35576923 0.45192308 0.54807692 0.64423077 0.
    ↪ 0.74038462 0.83653846 0.93269231]
""
```

We now need to find the transforms required to linearize the Weibull CDF (to get it in the form $y = mx + c$).

$$F = 1 - \exp\left(-\left(\frac{t}{\alpha}\right)^\beta\right)$$

$$-ln(1 - F) = \left(\frac{t}{\alpha}\right)^\beta$$

$$\underbrace{\ln(-\ln(1 - F))}_y = \underbrace{\beta}_{m} \cdot \underbrace{\ln(t)}_x - \underbrace{\beta \ln(\alpha)}_c$$

So the forward transforms for x and y are:

$$x = \ln(t)$$

$$y = \ln(-\ln(1 - F))$$

Once we fit the straight line to the transformed data, we will need the reverse transforms to obtain α and β which are:

$$\beta = m$$

$$c = -\beta \ln(\alpha) \text{ which becomes } \alpha = \exp\left(-\frac{c}{\beta}\right)$$

The table below shows the transformed data (from t and F into x and y) and a plot in Excel with the line of best fit. It also shows α and β which are obtained using the reverse transforms described above.

t	F	x = ln(t)	y = ln(-ln(1-F))
25	0.06730769	3.218875825	-2.663843121
43	0.16346154	3.761200116	-1.72326314
53	0.25961538	3.970291914	-1.202023136
65	0.35576923	4.17438727	-0.821666518
76	0.45192308	4.33073334	-0.508595384
86	0.54807692	4.454347296	-0.230365453
95	0.64423077	4.553876892	0.032924964
115	0.74038462	4.744932128	0.299032945
132	0.83653846	4.882801923	0.593977211
150	0.93269231	5.010635294	0.992688942

Excel formulas can get the parameters without the need to plot

m = SLOPE(y,x)	2.027390745
c = INTERCEPT(y,x)	-9.26158948

Reverse transforms to convert m and c into alpha and beta

alpha = exp(-c/beta)	96.37348522
beta = m	2.027390745

Here's how to do the same thing in Python, using `numpy.polyfit` for the line of best fit.

```
from reliability.Probability_plotting import plotting_positions
import numpy as np
```

(continues on next page)

(continued from previous page)

```

data = [25, 43, 53, 65, 76, 86, 95, 115, 132, 150]

# plotting positions
t,F = plotting_positions(failures=data)
print('t =',t)
print('F =',F)

# forward transform
x = np.log(t)
y = np.log(-np.log(1-F))
m, c = np.polyfit(x, y, 1)
print('m =',m)
print('c =',c)

# reverse transform
beta = m
alpha = np.exp(-c/beta)
print('alpha =',alpha)
print('beta =',beta)

"""

t = [ 25.  43.  53.  65.  76.  86.  95. 115. 132. 150.]
F = [0.06730769 0.16346154 0.25961538 0.35576923 0.45192308 0.54807692 0.64423077 0.
    ↪74038462 0.83653846 0.93269231]
m = 2.02739072618974
c = -9.261589398516517
alpha = 96.37348533880761
beta = 2.02739072618974
"""

```

Let's plot the Weibull Distribution that we have fitted alongside the data.

```

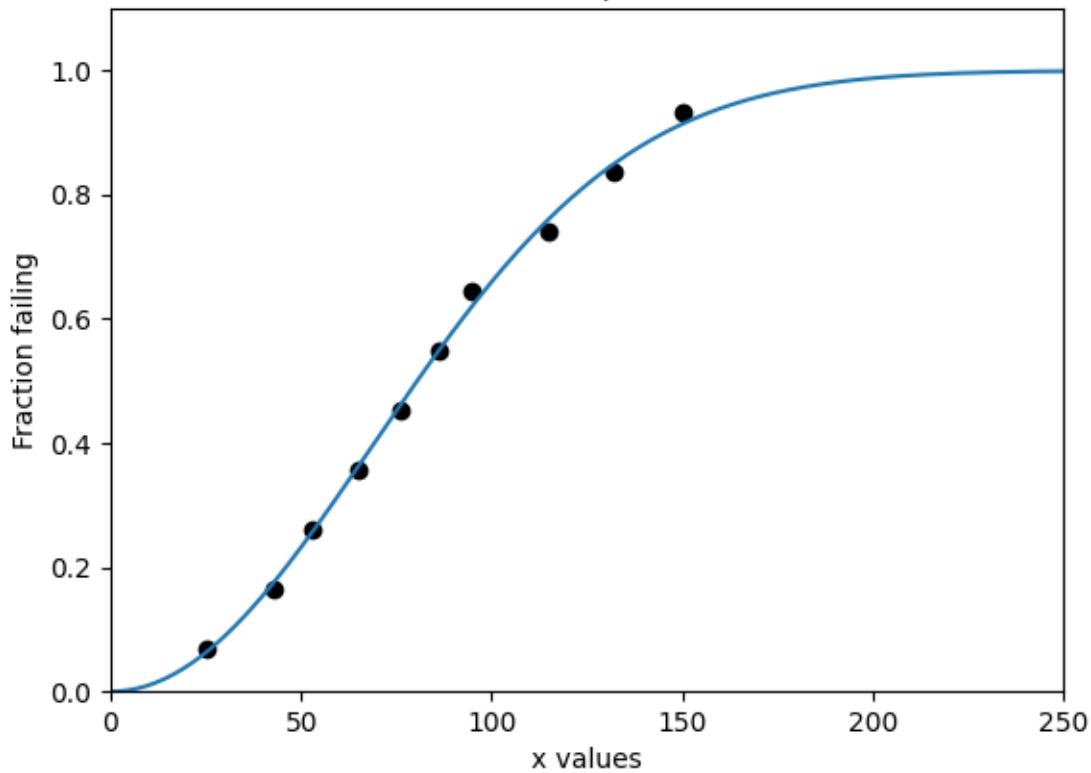
from reliability.Distributions import Weibull_Distribution
from reliability.Probability_plotting import plot_points
import matplotlib.pyplot as plt

data = [25, 43, 53, 65, 76, 86, 95, 115, 132, 150]
alpha = 96.37348533880761
beta = 2.02739072618974

plot_points(failures=data,marker='o')
Weibull_Distribution(alpha=alpha,beta=beta).CDF()
plt.show()

```

Weibull Distribution
Cumulative Distribution Function
 $\alpha=96.3735, \beta=2.0274$

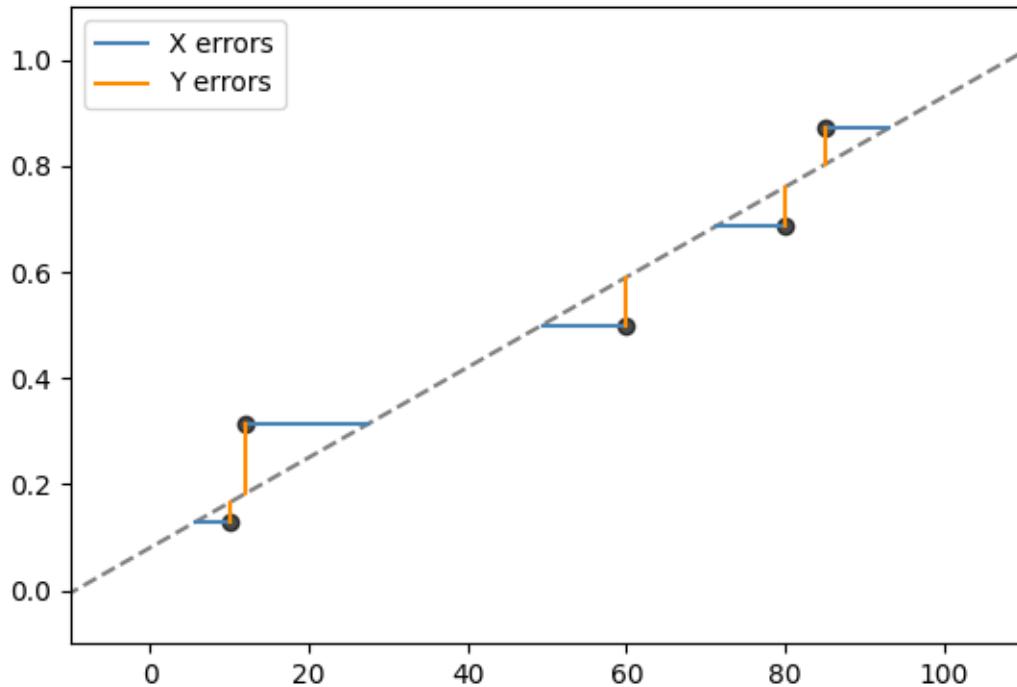


If you have right censored data, the only thing that will change will be the plotting positions. If you use different software to find the parameters of the line of best fit, the results may differ slightly. This is because there are several different algorithms to find the line of best fit, some of which use an optimizer and some of which do not. The RRX and RRY difference (discussed below) will also cause discrepancies in the results if each of the software packages you are using do not use the same approach.

58.2 RRX and RRY

Least squares is sometimes known as Rank Regression on X (RRX) or Rank Regression on Y (RRY). These two names are simply the two possible ways of fitting a line to the data. We can minimize the sum of the squared errors on X or we can minimize the sum of the squared errors on Y as shown below.

When finding the line of best fit, we need to minimize the errors between the line and the points. We can either minimize the X errors or minimize the Y errors. These two methods do not necessarily produce the same line.



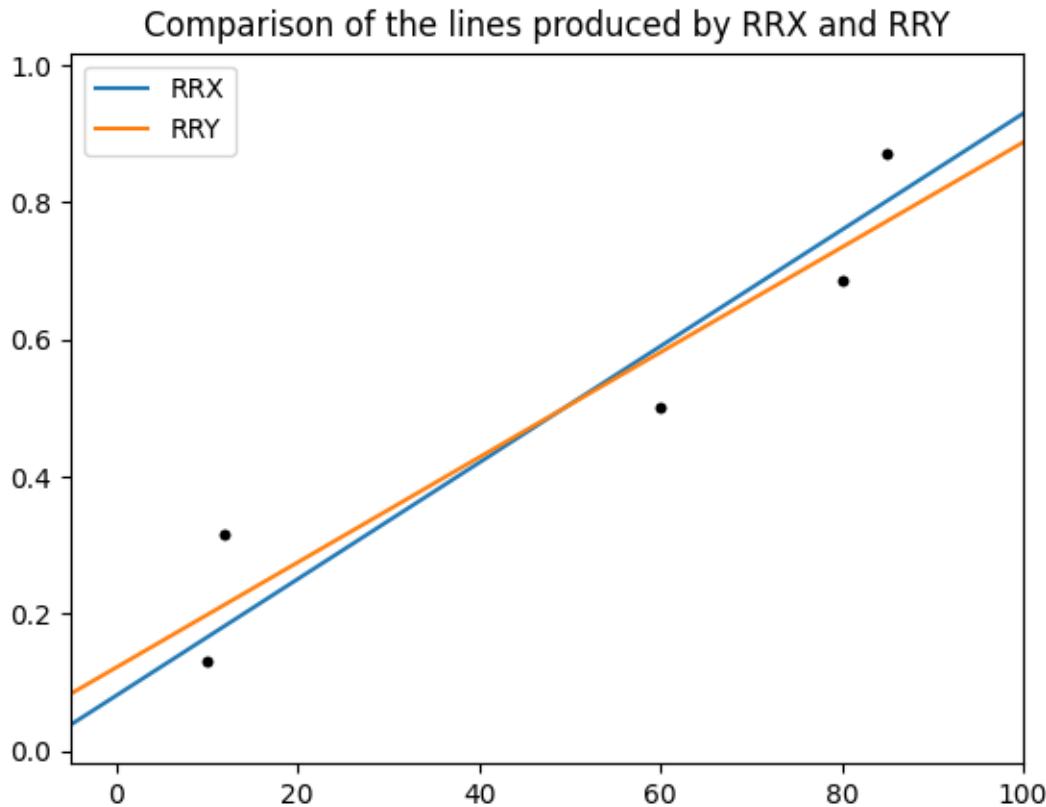
These two methods can give very different results, particularly if there is a small dataset. Most software (including MINITAB, Excel and numpy) use RRY. Reliasoft's Weibull++ gives the options for RRX or RRY, as does *reliability* in all of the fitters.

To illustrate the difference between RRX and RRY we can use one of the functions inside *reliability.Utils* which accepts RRX_or_RRY as an argument.

```
from reliability.Probability_plotting import plotting_positions
from reliability.Utils import linear_regression
import matplotlib.pyplot as plt

data = [10,12,60,80,85]
t,F = plotting_positions(failures=data)

linear_regression(x=t,y=F,RRX_or_RRY="RRX",show_plot=True,label='RRX')
linear_regression(x=t,y=F,RRX_or_RRY="RRY",show_plot=True,label='RRY')
plt.legend()
plt.title('Comparison of the lines produced by RRX and RRY')
plt.show()
```



58.3 Non-linear least squares

In the first example above, the CDF of the Weibull Distribution was able to be linearized without too much trouble into the form $y = m.x + c$. Some distributions cannot be linearized. These include 3 parameter distributions (such as Weibull_3P) and distributions involving special functions (such as the Gamma and Beta Distributions). I encourage you to try this yourself using the equations for the CDF available [here](#). The Normal (and Lognormal) distributions can be linearized quite easily because there is an algorithm to compute the Normal CDF (Φ) as well as its inverse (Φ^{-1}).

When the equation of the CDF cannot be linearized, we can use non-linear least squares (NLLS). The NLLS algorithm still seeks to minimize the sum of the square errors (usually the errors on Y), but it does not use the linear regression formula and can therefore work on any function. You can see this in action inside Excel when you chose a higher order polynomial for the line of best fit. To achieve this complicated fitting process, *reliability* calls `scipy.optimize.curve_fit` to find the parameters of the distribution directly. There is no forward and reverse transform required, just the appropriate setup of `scipy`'s `curve_fit`. The hardest part (and one possible source of failure) is obtaining a reasonable initial guess for the optimizer to begin. There are several different ways in which *reliability* obtains an initial guess, depending on the function being fitted.

58.4 Is MLE better than Least Squares

Sometimes yes, but sometimes no. It really depends on the distribution, the amount of data, and the amount of censoring. Least squares is computationally easier so it was invented first and remains popular today as it is easier for students to learn and can be faster for computers if doing a lot of calculations. MLE is the default method for most reliability engineering software including Weibull++, MINITAB, *reliability*, and many others. For most cases, MLE is generally

regarded as more accurate.

The best way to check whether MLE or Least squares is more accurate is through a Monte-Carlo simulation. In the following code, we will draw some random parameters (alpha and beta) to create a Weibull Distribution. In this simulation alpha is between 1 and 1000, while beta is between 0.5 and 10. We will then draw some random data from the Weibull distribution. This is done 3 times (10 samples, 100 samples, 1000 samples). We will right censor a fraction of the data (from 0 (no censoring) to 0.9 (90% censored)). Then we will fit a distribution to the random data using MLE and LS. The percentage error in the parameters (alpha and beta) is calculated and plotted. The following code performs this simulation 1000 times for each fraction censored. The code took about 45 minutes to run as it is fitting around 60K distributions (1000 trials x 10 fraction censored increments x 2 methods (MLE and LS) x 3 groups of samples).

```
import numpy as np
from reliability.Distributions import Weibull_Distribution
from reliability.Fitters import Fit_Weibull_2P
from tqdm import tqdm
from reliability.Other_functions import make_right_censored_data
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter

def MLE_or_LS(trials,number_of_samples):

    fraction_censored = [0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

    MLE_alpha_error_mean_array = []
    MLE_beta_error_mean_array = []
    LS_alpha_error_mean_array = []
    LS_beta_error_mean_array = []

    for frac in tqdm(fraction_censored):

        MLE_alpha_error_array = []
        MLE_beta_error_array = []
        LS_alpha_error_array = []
        LS_beta_error_array = []
        for trial in range(trials):
            alpha = (np.random.randint(1,1000,1)+np.random.rand())[0] # alpha between 1 and 1000
            beta = (np.random.randint(50,900,1)/100+np.random.rand())[0] # beta between 0.5 and 10
            true_dist = Weibull_Distribution(alpha=alpha,beta=beta)
            raw_samples = true_dist.random_samples(number_of_samples=number_of_samples)
            samples = make_right_censored_data(data=raw_samples,fraction_censored=frac)

            if len(np.unique(samples.failures))>1:
                MLE = Fit_Weibull_2P(failures=samples.failures,right_censored=samples.right_censored,show_probability_plot=False,print_results=False,method='MLE')
                MLE_alpha = MLE.distribution.alpha
                MLE_beta = MLE.distribution.beta
                MLE_alpha_error_array.append(abs(alpha-MLE_alpha)/alpha)
                MLE_beta_error_array.append(abs(beta-MLE_beta)/beta)

                LS = Fit_Weibull_2P(failures=samples.failures,right_censored=samples.right_censored,show_probability_plot=False,print_results=False,method='LS')
```

(continues on next page)

(continued from previous page)

```

LS_alpha = LS.distribution.alpha
LS_beta = LS.distribution.beta
LS_alpha_error_array.append(abs(alpha-LS_alpha)/alpha)
LS_beta_error_array.append(abs(beta-LS_beta)/beta)

MLE_alpha_error_mean_array.append(np.average(MLE_alpha_error_array))
MLE_beta_error_mean_array.append(np.average(MLE_beta_error_array))
LS_alpha_error_mean_array.append(np.average(LS_alpha_error_array))
LS_beta_error_mean_array.append(np.average(LS_beta_error_array))

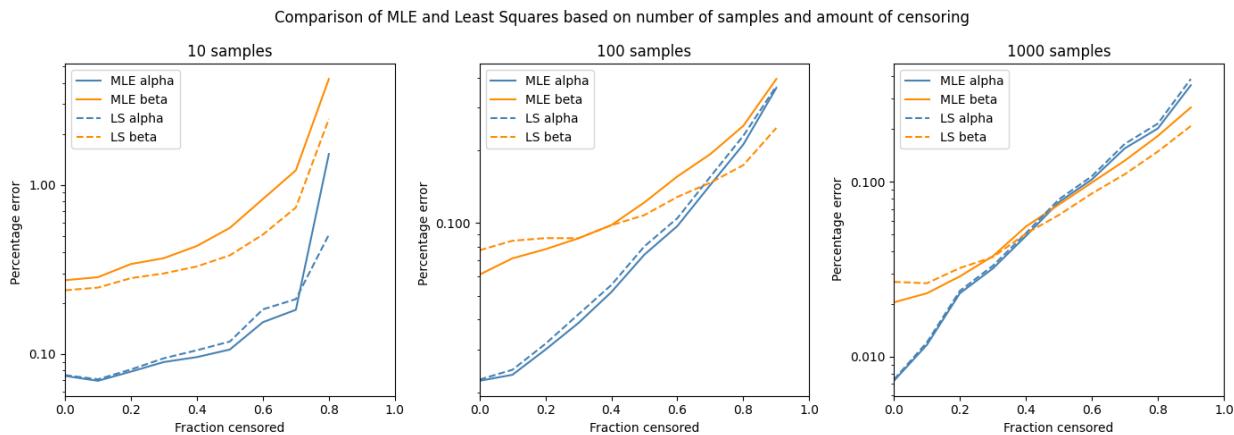
plt.plot(fraction_censored,MLE_alpha_error_mean_array,label='MLE alpha',color=
'steelblue')
plt.plot(fraction_censored,MLE_beta_error_mean_array,label='MLE beta',color=
'darkorange')
plt.plot(fraction_censored,LS_alpha_error_mean_array,label='LS alpha',color=
'steelblue',linestyle='--')
plt.plot(fraction_censored,LS_beta_error_mean_array,label='LS beta',color='darkorange',
',linestyle='--')
plt.yscale('log')
plt.xlim(0,1)
plt.gca().yaxis.set_major_formatter(ScalarFormatter())
plt.legend()
plt.xlabel('Fraction censored')
plt.ylabel('Percentage error')

trials = 10
plt.figure(figsize=(14,5))
plt.subplot(131)
MLE_or_LS(trials=trials, number_of_samples=10)
plt.title('10 samples')

plt.subplot(132)
MLE_or_LS(trials=trials, number_of_samples=100)
plt.title('100 samples')

plt.subplot(133)
MLE_or_LS(trials=trials, number_of_samples=1000)
plt.title('1000 samples')
plt.suptitle('Comparison of MLE and Least Squares based on number of samples and amount
of censoring')
plt.tight_layout()
plt.show()

```

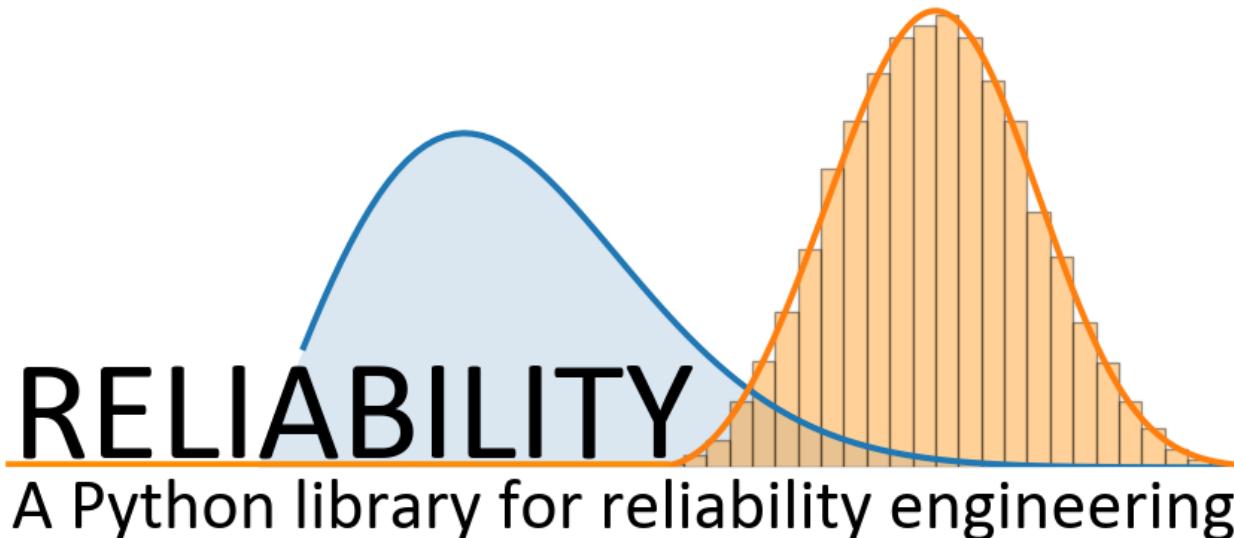


The y-axis is a log plot of the percentage error, so where you see 1 that means it is 100% in error (eg. correct value of 2, predicted value of 4). The fraction censored ranges from 0 to 0.9, except for the 10 sample case as a minimum of 2 samples are needed to fit the distribution making 0.8 the maximum possible fraction censored. From the above plots we can see a few things:

- The percentage error in beta is much higher than the percentage error in alpha for smaller sample sizes, but about the same for large sample sizes.
- Both MLE and LS perform very similarly in terms of their percentage error.
- Least squares is generally better than MLE for small sample sizes, while MLE is generally better than Least squares for large sample sizes.
- MLE tends to have more error in the beta parameter than Least squares, and less error in the alpha parameter than least squares. A correction method exists for this, though it is not currently implemented in *reliability*.

The trends we see in the above plot may differ if we chose another distribution, different ranges for the parameters, or different numbers of samples.

If you find any errors, think this needs to be explained better, or have any suggestions for improvements, please email me (alpha.reliability@gmail.com).



HOW DOES MAXIMUM LIKELIHOOD ESTIMATION WORK

Maximum Likelihood Estimation (MLE) is a method of estimating the parameters of a model using a set of data. While MLE can be applied to many different types of models, this article will explain how MLE is used to fit the parameters of a probability distribution for a given set of failure and right censored data.

MLE works by calculating the probability of occurrence for each data point (we call this the likelihood) for a model with a given set of parameters. These probabilities are summed for all the data points. We then use an optimizer to change the parameters of the model in order to maximise the sum of the probabilities. This is easiest to understand with an example which is provided below.

There are two major challenges with MLE. These are the need to use an optimizer (making hand calculations almost impossible for distributions with more than one parameter), and the need for a relatively accurate initial guess for the optimizer. The initial guess for MLE is typically provided using [Least Squares Estimation](#). A variety of [optimizers](#) are suitable for MLE, though some may perform better than others so trying a few is sometimes the best approach.

There are several advantages of MLE which make it the standard method for fitting probability distributions in most software. These are that MLE does not need the equation to be linearizable (which is needed in Least Squares Estimation) so any equation can be modeled. The other advantage of MLE is that unlike Least Squares Estimation which uses the plotting positions and does not directly use the right censored data, MLE uses the failure data and right censored data directly, making it more suitable for heavily censored datasets.

59.1 The MLE algorithm

The MLE algorithm is as follows:

1. Obtain an initial guess for the model parameters (typically done using least squares estimation).
2. Calculate the probability of occurrence of each data point ($f(t)$ for failures, $R(t)$ for right censored, $F(t)$ for left censored).
3. Multiply the probabilities (or sum their logarithms which is much more computationally efficient).
4. Use an optimizer to change the model parameters and repeat steps 2 and 3 until the total probability is maximized.

As mentioned in step 2, different types of data need to be handled differently:

Type of observation	Likelihood function
Failure data	$L_i(\theta t_i) = f(t_i \theta)$
Right censored data	$L_i(\theta t_i) = R(t_i \theta)$
Left censored data	$L_i(\theta t_i) = F(t_i \theta)$
Interval censored data	$L_i(\theta t_i) = F(t_i^{RI} \theta) - F(t_i^{LI} \theta)$

In words, the first equation above means “the likelihood of the parameters (θ) given the data (t_i) is equal to the probability of failure ($f(t)$) evaluated at each time t_i with that given set of parameters (θ)”. The equations for the PDF ($f(t)$),

CDF ($F(t)$), and SF ($R(t)$) for each distribution is provided [here](#).

Once we have the likelihood (L_i) for each data point, we need to combine them. This is done by multiplying them together (think of this as an AND condition). If we just had failures and right censored data then the equation would be:

$$L(\theta|D) = \prod_{i=1}^n f_i(t_i^{\text{failures}}|\theta) \times R_i(t_i^{\text{right censored}}|\theta)$$

In words this means that “the likelihood of the parameters of the model (θ) given the data (D) is equal to the product of the values of the PDF ($f(t)$) with the given set of parameters (θ) evaluated at each failure (t_i^{failures}), multiplied by the product of the values of the SF ($R(t)$) with the given set of parameters (θ) evaluated at each right censored value ($t_i^{\text{right censored}}$ ”).

Since probabilities are between 0 and 1, multiplying many of these results in a very small number. A loss of precision occurs because computers can only store so many decimals. Multiplication is also slower than addition for computers. To overcome this problem, we can use a logarithm rule to add the log-likelihoods rather than multiply the likelihoods. We just need to take the log of the likelihood function (the PDF for failure data and the SF for right censored data), evaluate the probability, and sum the values. The parameters that will maximize the log-likelihood function are the same parameters that will maximize the likelihood function.

59.2 An example using the Exponential Distribution

Let's say we have some failure times: $t = [27, 64, 3, 18, 8]$

We need an initial estimate for time model parameter (λ) which we would typically get using Least Squares Estimation. For this example, let's start with 0.1 as our first guess for λ .

For each of these values, we need to calculate the value of the PDF (with the given value of λ).

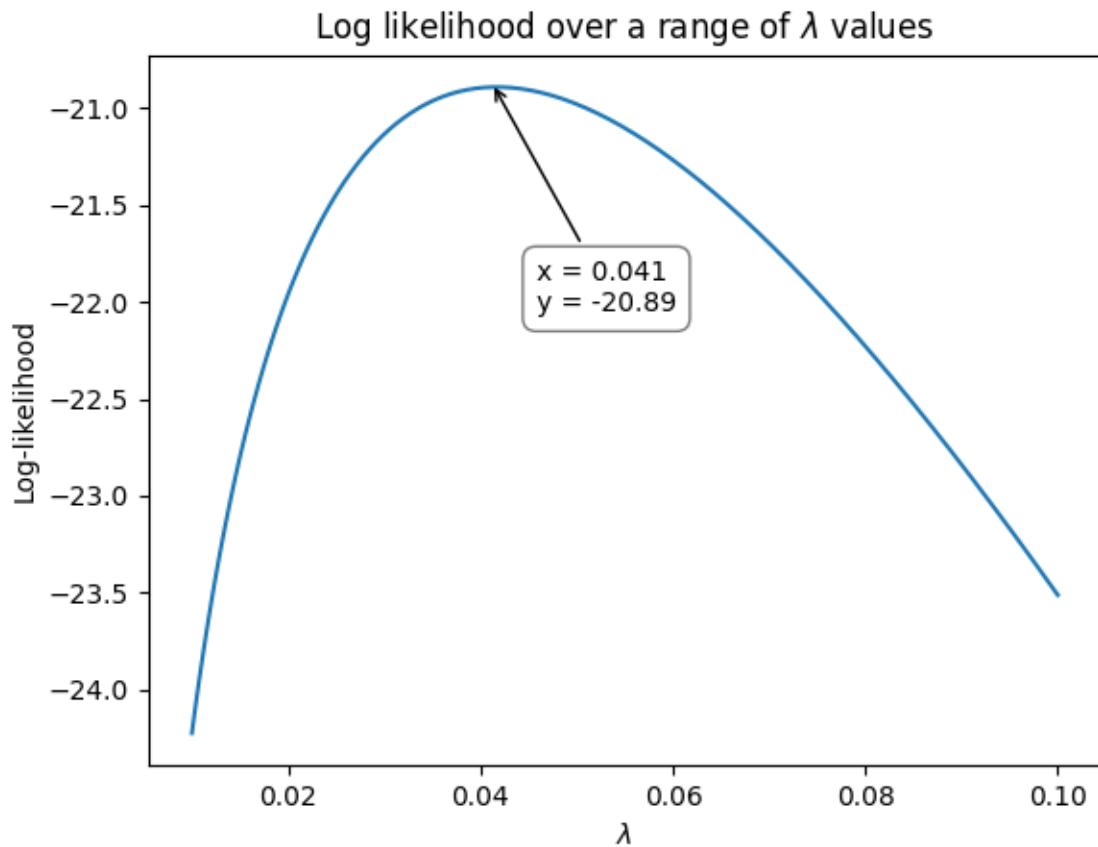
Exponential PDF: $f(t) = \lambda e^{-\lambda t}$

Exponential Log-PDF: $\ln(f(t)) = \ln(\lambda) - \lambda t$

Now we substitute in $\lambda = 0.1$ and $t = [27, 64, 3, 18, 8]$

$$\begin{aligned} L(\lambda = 0.1|t = [27, 64, 3, 18, 8]) &= \\ & (ln(0.1) - 0.1 \times 27) + (ln(0.1) - 0.1 \times 64) + (ln(0.1) - 0.1 \times 3) \\ & + (ln(0.1) - 0.1 \times 18) + (ln(0.1) - 0.1 \times 8) \\ & = -23.512925 \end{aligned} \tag{59.1}$$

Here's where the optimization part comes in. We need to vary λ until we maximize the log-likelihood. The following graph shows how the log-likelihood varies as λ varies.



This was produced using the following Python code:

```
import matplotlib.pyplot as plt
import numpy as np

data = np.array([27, 64, 3, 18, 8])

lambda_array = np.geomspace(0.01, 0.1, 100)
LL = []
for L in lambda_array:
    loglik = np.log(L) - L * data
    LL.append(loglik.sum())

plt.plot(lambda_array, LL)
plt.xlabel('$\lambda$')
plt.ylabel('Log-likelihood')
plt.title('Log likelihood over a range of $\lambda$ values')
plt.show()
```

The optimization process can be done in Python (using `scipy.optimize.minimize`) or in Excel (using `Solver`), or a variety of other software packages. Optimization becomes a bit more complicated when there are two or more parameters that need to be optimized simultaneously (such as in a Weibull Distribution).

So, using the above method, we see that the maximum for the log-likelihood occurred when λ was around 0.041 at a log-likelihood of -20.89. We can check the value using *reliability* as shown below which achieves an answer of

$\lambda = 0.0416667$ at a log-likelihood of -20.8903:

```
from reliability.Fitters import Fit_Exponential_1P

data = [27, 64, 3, 18, 8]
Fit_Exponential_1P(failures=data, show_probability_plot=False)

"""

Results from Fit_Exponential_1P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 5/0 (0% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
Lambda      0.0416667  0.0186339  0.0173428  0.100105
1/Lambda      24        10.7331    9.98947    57.6607

Goodness of fit      Value
Log-likelihood -20.8903
AICc      45.1139
BIC       43.39
AD        2.43793
"""


```

59.3 Another example using the Exponential Distribution with censored data

Lets use a new dataset that includes both failures and right censored values.

failures = [17, 5, 12] and right_censored = [20, 25]

Once again, we need an initial estimate for the model parameters, and for that we would typically use Least Squares Estimation. For the purposes of this example, we will again use an initial guess of $\lambda = 0.1$.

For each of the failures, we need to calculate the value of the PDF, and for each of the right censored values, we need to calculate the value of the SF (with the given value of λ).

Exponential PDF: $f(t) = \lambda e^{-\lambda t}$

Exponential Log-PDF: $\ln(f(t)) = \ln(\lambda) - \lambda t$

Exponential SF: $R(t) = e^{-\lambda t}$

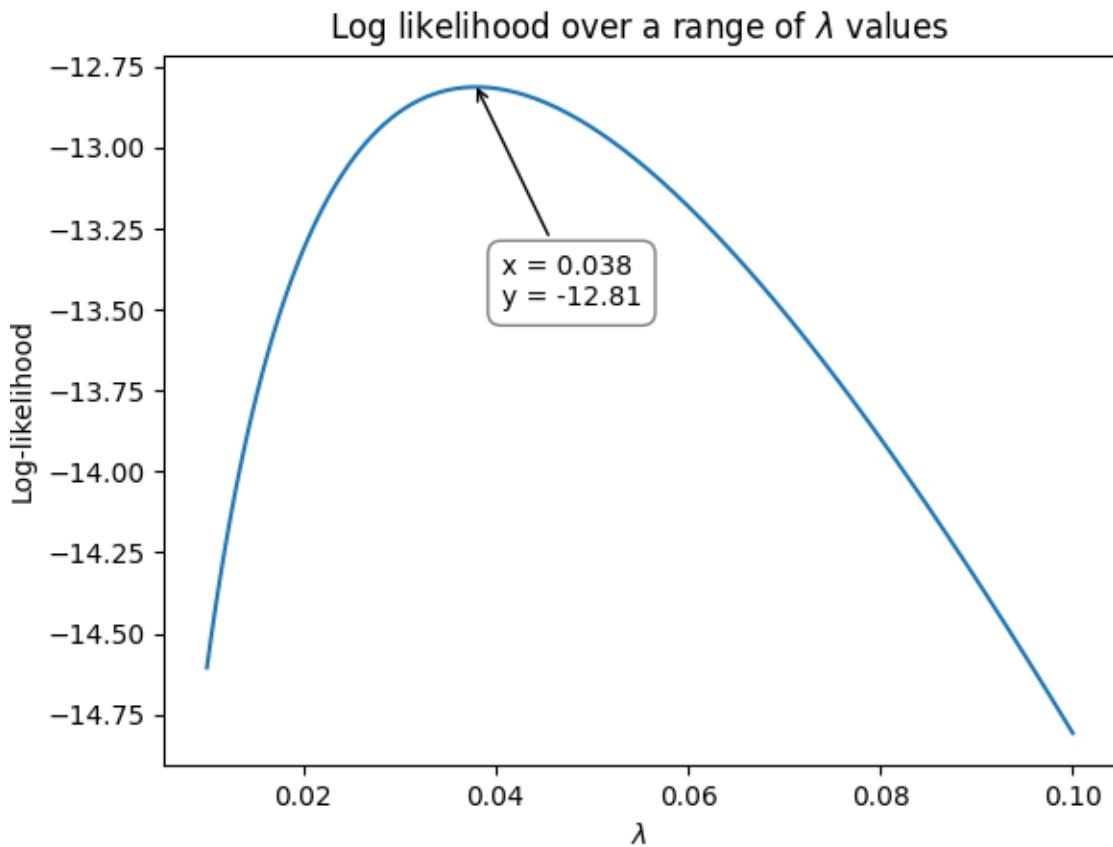
Exponential Log-SF: $\ln(R(t)) = -\lambda t$

Now we substitute in $\lambda = 0.1$, $t_{\text{failures}} = [17, 5, 12]$, and $t_{\text{right censored}} = [20, 25]$.

$$\begin{aligned}
 L(\lambda = 0.1 | t_{\text{failures}} = [17, 5, 12] \text{ and } t_{\text{right censored}} = [20, 25]) &= \\
 & (\ln(0.1) - 0.1 \times 17) + (\ln(0.1) - 0.1 \times 5) + (\ln(0.1) - 0.1 \times 20) \\
 & + (-0.1 \times 25) \\
 & = -14.8075928
 \end{aligned} \tag{59.5}$$

Note that the last two terms are the right censored values. Their contribution to the log-likelihood is added in the same way that the contribution from each of the failures is added, except that right censored values use the the log-SF.

As with the previous example, we again need to use optimization to vary λ until we maximize the log-likelihood. The following graph shows how the log-likelihood varies as λ varies.



This was produced using the following Python code:

```
import matplotlib.pyplot as plt
import numpy as np

failures = np.array([17, 5, 12])
right_censored = np.array([20, 25])

lambda_array = np.geomspace(0.01, 0.1, 100)
LL = []
for L in lambda_array:
    loglik_failures = np.log(L) - L * failures
    loglik_right_censored = -L * right_censored
    LL.append(loglik_failures.sum() + loglik_right_censored.sum())

plt.plot(lambda_array, LL)
plt.xlabel('$\lambda$')
plt.ylabel('Log-likelihood')
plt.title('Log likelihood over a range of $\lambda$ values')
plt.show()
```

So, using the above method, we see that the maximum for the log-likelihood occurred when λ was around 0.038 at a log-likelihood of -12.81. We can check the value using *reliability* as shown below which achieves an answer of $\lambda = 0.0379747$ at a log-likelihood of -12.8125:

```

from reliability.Fitters import Fit_Exponential_1P

failures = [17,5,12]
right_censored = [20, 25]
Fit_Exponential_1P(failures=failures, right_censored=right_censored, show_probability_
plot=False)

"""

Results from Fit_Exponential_1P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 3/2 (40% right censored)

Parameter  Point Estimate  Standard Error  Lower CI  Upper CI
Lambda      0.0379747    0.0219247  0.0122476  0.117743
1/Lambda    26.3333     15.2036    8.49306   81.6483

Goodness of fit      Value
Log-likelihood -12.8125
AICc      28.9583
BIC       27.2345
AD        19.3533
"""

```

59.4 An example using the Weibull Distribution

Because it requires optimization, MLE is only practical using software if there is more than one parameter in the distribution. The rest of the process is the same, but instead of the likelihood plot (the curves shown above) being a line, for 2 parameters it would be a surface, as shown in the example below.

We'll use the same dataset as in the previous example with failures = [17,5,12] and right_censored = [20, 25].

We also need an estimate for the parameters of the Weibull Distribution. We will use $\alpha = 15$ and $\beta = 2$

For each of the failures, we need to calculate the value of the PDF, and for each of the right censored values, we need to calculate the value of the SF (with the given value of λ).

$$\text{Weibull PDF: } f(t) = \frac{\beta}{\alpha} \left(\frac{t}{\alpha}\right)^{\beta-1} e^{-(\frac{t}{\alpha})^\beta}$$

$$\text{Weibull Log-PDF: } \ln(f(t)) = \ln\left(\frac{\beta}{\alpha}\right) + (\beta - 1).\ln\left(\frac{t}{\alpha}\right) - \left(\frac{t}{\alpha}\right)^\beta$$

$$\text{Weibull SF: } R(t) = e^{-(\frac{t}{\alpha})^\beta}$$

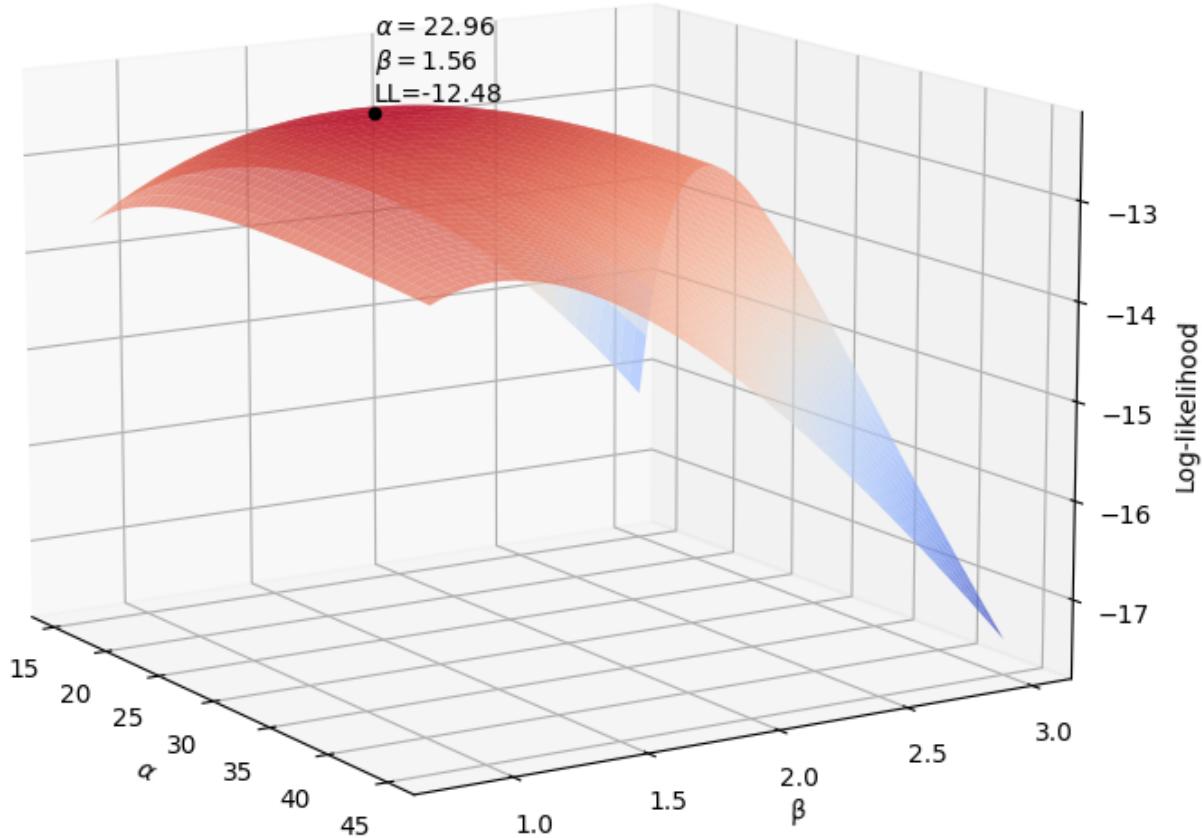
$$\text{Weibull Log-SF: } \ln(R(t)) = -\left(\frac{t}{\alpha}\right)^\beta$$

Now we substitute in $\alpha = 15$, $\beta = 2$, $t_{\text{failures}} = [17, 5, 12]$, and $t_{\text{right censored}} = [20, 25]$ to the log-PDF and log-SF.

$$\begin{aligned}
 L(\alpha = 15, \beta = 2 | t_{\text{failures}} = [17, 5, 12] \text{ and } t_{\text{right censored}} = [20, 25]) = & \quad (59.9) \\
 \ln\left(\frac{2}{15}\right) + (2-1).\ln\left(\frac{17}{15}\right) - \left(\frac{17}{15}\right)^2 \\
 + \ln\left(\frac{2}{15}\right) + (2-1).\ln\left(\frac{5}{15}\right) - \left(\frac{5}{15}\right)^2 \\
 + \ln\left(\frac{2}{15}\right) + (2-1).\ln\left(\frac{12}{15}\right) - \left(\frac{12}{15}\right)^2 \\
 + \left(-\left(\frac{20}{15}\right)^2\right) \\
 + \left(-\left(\frac{25}{15}\right)^2\right) \\
 = -16.58354
 \end{aligned}$$

As with the previous example, we again need to use optimization to vary α and β until we maximize the log-likelihood. The following 3D surface plot shows how the log-likelihood varies as α and β are varied. The maximum log-likelihood is shown as a scatter point on the plot.

Log-likelihood over a range of α and β values



This was produced using the following Python code:

```

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

failures = np.array([17, 5, 12])
right_censored = np.array([20, 25])

points = 50
alpha_array = np.linspace(15, 45, points)
beta_array = np.linspace(0.8, 3, points)

A, B = np.meshgrid(alpha_array, beta_array)

LL = np.empty((len(alpha_array), len(beta_array)))
for i, alpha in enumerate(alpha_array):
    for j, beta in enumerate(beta_array):
        loglik_failures = np.log(beta/alpha)+(beta-1)*np.log(failures/alpha)-(failures/
        alpha)**beta
        loglik_right_censored = -(right_censored/alpha)**beta
        LL[i][j] = loglik_failures.sum()+loglik_right_censored.sum()

LL_max = LL.max()
idx = np.where(LL==LL_max)
alpha_fit = alpha_array[idx[0]][0]
beta_fit = beta_array[idx[1]][0]

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(A, B, LL.transpose(), cmap="coolwarm", linewidth=1, antialiased=True, alpha=0.
    ↵7, zorder=0)
ax.set_xlabel(r'$\alpha$')
ax.set_ylabel(r'$\beta$')
ax.set_zlabel('Log-likelihood')
ax.scatter([alpha_fit], [beta_fit], [LL_max], color='k', zorder=1)
text_string = str(r'$\alpha=')+str(round(alpha_fit, 2))+r'\n'+r'$\beta='+str(round(beta_
    ↵fit, 2))+r'\nLL='+str(round(LL_max, 2)))
ax.text(x=alpha_fit, y=beta_fit, z=LL_max+0.1, s=text_string)
ax.computed_zorder = False
plt.title(r'Log-likelihood over a range of $\alpha$ and $\beta$ values')
plt.show()

```

So, using the above method, we see that the maximum for the log-likelihood (shown by the scatter point) occurred when α was around 22.96 and β was around 1.56 at a log-likelihood of -12.48. Once again, we can check the value using *reliability* as shown below which achieves an answer of $\alpha = 23.0653$ and $\beta = 1.57474$ at a log-likelihood of -12.4823. The trickiest part about MLE is the optimization step, which is discussed briefly in the next section.

```

from reliability.Fitters import Fit_Weibull_2P
failures = [17, 5, 12]
right_censored = [20, 25]
Fit_Weibull_2P(failures=failures, right_censored=right_censored, show_probability_
    ↵plot=False)

```

(continues on next page)

(continued from previous page)

```

"""
Results from Fit_Weibull_2P (95% CI):
Analysis method: Maximum Likelihood Estimation (MLE)
Optimizer: TNC
Failures / Right censored: 3/2 (40% right censored)

Parameter Point Estimate Standard Error Lower CI Upper CI
Alpha      23.0653      8.76119  10.9556  48.5604
Beta       1.57474     0.805575  0.577786  4.2919

Goodness of fit      Value
Log-likelihood -12.4823
AICc      34.9647
BIC       28.1836
AD        19.2756
"""

```

59.5 How does the optimization routine work in Python

Optimization is a complex field of study, but thankfully there are several software packages where optimizers are built into (relatively) easy-to-use tools. Excel's `Solver` is a perfect example of an easy to use optimizer, and it is capable of changing multiple cells so it can be used to fit the Weibull Distribution. In this section, we will look at how optimization can be done in Python using `scipy.optimize.minimize`.

Scipy requires four primary inputs:

- An objective function that should be minimized
- An initial guess
- The optimization routine to use
- Bounds on the solution

The objective function is the log-likelihood function as we used in the previous examples. For the initial guess, we use `least squares estimation`. The optimization routine is a string that tells scipy which optimizer to use. There are several options, but `reliability` only uses four bounded `optimizers`, which are 'TNC', 'L-BFGS-B', 'powell', and 'nelder-mead'. The bounds on the solution are things like specifying $\alpha > 0$ and $\beta > 0$. Bounds are not essential, but they do help a lot with stability (preventing the optimizer from failing).

Another important point to highlight is that when using an optimizer for the log-likelihood function in Python, it is more computationally efficient to find the point of minimum slope (which is the same as the peak of the log-likelihood function). The slope is evaluated using the gradient (∇), which is done using the Python library `autograd` using the `value_and_grad` function. This process is quite mathematically complicated, but `reliability` does all of this internally and as a user you do not need to worry too much about how the optimizer works. The most important thing to understand is that as part of MLE, an optimizer is trying to maximize the log-likelihood by varying the parameters of the model. Sometimes the optimizer will be unsuccessful or will give a poor solution. In such cases, you should try another optimizer, or ask `reliability` to try all optimizers by specifying `optimizer='best'`.



RELIABILITY
A Python library for reliability engineering

HOW ARE THE CONFIDENCE INTERVALS CALCULATED

There are confidence intervals on the model parameters, and confidence intervals on the plots. This document explains how both are calculated. The confidence intervals on the plots use the confidence intervals on the model parameters in the procedure that is explained in the second section of this document. Before reading this document, you should have a reasonable understanding of how [Maximum Likelihood Estimation \(MLE\) works](#) and also understand [what partial derivatives are](#) and [how to calculate them](#).

60.1 Confidence intervals on the parameters

Once the model parameters have been found (whether by Least Squares Estimation or Maximum Likelihood Estimation), we can use these parameters along with the data and the log-likelihood function, to calculate the hessian matrix. We can use `autograd.differential_operators.hessian` to calculate the hessian matrix, which is a square matrix of partial derivatives. The inverse of the hessian matrix gives us the covariance matrix, which contains the numbers we need for finding the confidence intervals. The covariance matrix is a matrix containing the variance of the parameters along the diagonal, and the covariance of the parameters outside of the diagonal. The size of the covariance matrix will match the number of parameters. For the Weibull Distribution (with parameters α and β), the covariance matrix looks like this:

$$\begin{bmatrix} \text{Var}(\alpha) & \text{Cov}(\alpha, \beta) \\ \text{Cov}(\alpha, \beta) & \text{Var}(\beta) \end{bmatrix}$$

To find the confidence intervals on the parameters, we need the parameters, the standard error ($\sqrt{\text{variance}}$), and the desired confidence interval. For this example, we will use the default of 95% confidence (two sided). The formulas for the confidence intervals on the parameters have two forms, depending on whether they are strictly positive (like α and β are for the Weibull Distribution) or unbounded (like μ is for the Normal Distribution).

For strictly positive parameters, the formula is:

$$X_{lower} = \hat{X} \cdot e^{-Z \cdot \frac{\hat{X}_{SE}}{\hat{X}}}$$

$$X_{upper} = \hat{X} \cdot e^{+Z \cdot \frac{\hat{X}_{SE}}{\hat{X}}}$$

For unbounded parameters (which can be positive or negative), the formula is:

$$X_{lower} = \hat{X} - Z \cdot \hat{X}_{SE}$$

$$X_{upper} = \hat{X} + Z \cdot \hat{X}_{SE}$$

Where:

- Z = Standard Normal Distribution Quantile of $\frac{1-CI}{2}$. This can be calculated using `-scipy.stats.norm.ppf((1 - CI) / 2)`. For $CI = 0.95$, $Z = 1.95996$. If finding a 1 sided interval, don't divide by 2 in the formula.

- \hat{X}_{SE} = The standard error $\left(\sqrt{\text{Var}(\hat{X})} \right)$.

- X_{lower} = The lower confidence bound on the parameter based on the specified confidence interval (CI).
 - X_{upper} = The upper confidence bound on the parameter based on the specified confidence interval (CI).

If you would like to calculate these values yourself, you can check your result with *reliability* like this:

There are a few exceptions to the above formulas for confidence intervals on the parameters. For three parameter distributions (such as Weibull_3P), the mathematics somewhat breaks down, requiring a minor modification. The bounds for α and β are calculated using the Weibull_2P log-likelihood function and the bounds on γ are calculated using the Weibull_3P log-likelihood function. This method is used by *reliability* and *reliasoft*'s Weibull++ software.

In the case of the Exponential Distribution, the covariance matrix is a 1×1 matrix containing just $Var(\lambda)$. The upper and lower bounds on λ are found using the formula above for strictly positive parameters.

Some matrices are non-invertable due to their values. While rare, if it occurs for the hessian matrix, it means that the inverse of the hessian matrix cannot be calculated so the covariance matrix is not able to be obtained. In such cases, a warning will be printed by *reliability*, the standard errors will be set to 0 and the upper and lower bounds of the parameters will match the parameters.

60.2 Confidence intervals on the plots

The confidence intervals on the plots (usually called confidence bounds) are available for the CDF, SF, and CHF. It is not possible to calculate confidence intervals for the PDF or HF. There are two types of confidence bounds, these are bounds on time (Type I) and bounds on reliability (Type II). Depending on the amount of data, these bounds may be almost the same (for large sample sizes) or quite different (for small sample sizes). The following example shows the differences in these bounds for the CDF, SF, and CHF.

```
from reliability.Fitters import Fit_Weibull_2P
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```

data = [43, 81, 41, 44, 52, 99, 64, 25, 41, 7]
fit = Fit_Weibull_2P(failures=data, show_probability_plot=False, print_results=False)

plt.figure(figsize=(10,4))

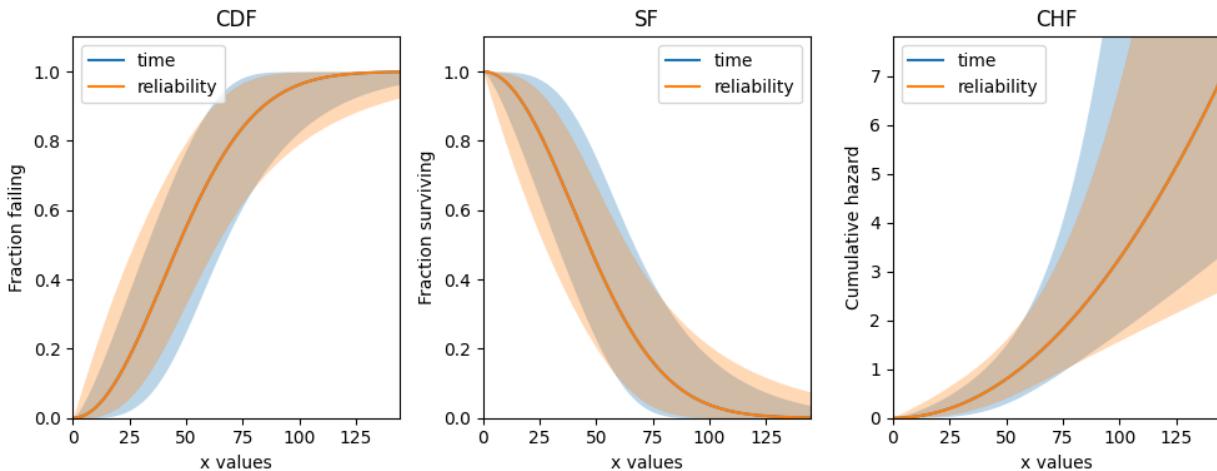
plt.subplot(131)
fit.distribution.CDF(CI_type='time',label='time')
fit.distribution.CDF(CI_type='reliability',label='reliability')
plt.title('CDF')
plt.legend()

plt.subplot(132)
fit.distribution.SF(CI_type='time',label='time')
fit.distribution.SF(CI_type='reliability',label='reliability')
plt.title('SF')
plt.legend()

plt.subplot(133)
fit.distribution.CHF(CI_type='time',label='time')
fit.distribution.CHF(CI_type='reliability',label='reliability')
plt.title('CHF')
plt.legend()

plt.tight_layout()
plt.show()

```



For larger values of CI (the default is 0.95), the distance between the solid line and the confidence bounds will increase.

Due to the relationship between the CDF, SF, and CHF, we only need to calculate the confidence bounds on the SF and we can use a few simple [transformations](#) to obtain the bounds for the CDF and CHF.

60.2.1 Bounds on time

The formulas for the confidence bounds on time (T) for the Weibull Distribution can be obtained as follows:

Begin with the equation for the SF: $R = e^{-(\frac{T}{\alpha})^\beta}$

Linearize the equation: $\ln(-\ln(R)) = \beta \cdot (\ln(T) - \ln(\alpha))$

Rearrange to make T the subject: $\ln(T) = \frac{1}{\beta} \ln(-\ln(R)) + \ln(\alpha)$

Substitute $u = \ln(T)$: $u = \frac{1}{\beta} \ln(-\ln(R)) + \ln(\alpha)$

The upper and lower bounds on u are:

$$u_U = \hat{u} + Z \cdot \sqrt{\text{Var}(\hat{u})}.$$

$$u_L = \hat{u} - Z \cdot \sqrt{\text{Var}(\hat{u})}.$$

You'll notice that this is the same formula for the bounds on the parameters (when unbounded) provided in the previous section. The formula for Z is also listed in the previous section.

Here's the tricky part. We need to find $\text{Var}(\hat{u})$. The formula for this comes from something called the [Delta Method](#) which states that:

$$\text{Var}(h_r) = \sum_i \left(\frac{\partial h_r}{\partial B_i} \right)^2 \text{Var}(B_i) + \sum_i \sum_{j \neq i} \left(\frac{\partial h_r}{\partial B_i} \right) \left(\frac{\partial h_r}{\partial B_j} \right) \text{Cov}(B_i, B_j)$$

Applying this to $u = \frac{1}{\beta} \ln(-\ln(R)) + \ln(\alpha)$ gives:

$$\begin{aligned} \text{Var}(u) &= \left(\frac{\partial u}{\partial \beta} \right)^2 \text{Var}(\beta) \\ &\quad + \left(\frac{\partial u}{\partial \alpha} \right)^2 \text{Var}(\alpha) \\ &\quad + 2 \left(\frac{\partial u}{\partial \beta} \right) \left(\frac{\partial u}{\partial \alpha} \right) \text{Cov}(\beta, \alpha) \\ &= \left(-\frac{1}{\beta^2} \ln(-\ln(R)) \right)^2 \text{Var}(\beta) \\ &\quad + \left(\frac{1}{\alpha^2} \right)^2 \text{Var}(\alpha) \\ &\quad + 2 \left(-\frac{1}{\beta^2} \ln(-\ln(R)) \right) \left(\frac{1}{\alpha} \right) \text{Cov}(\beta, \alpha) \\ &= \frac{1}{\beta^4} (\ln(-\ln(R)))^2 \text{Var}(\beta) \\ &\quad + \frac{1}{\alpha^2} \text{Var}(\alpha) \\ &\quad + 2 \left(-\frac{1}{\beta^2} \right) \left(\frac{\ln(-\ln(R))}{\alpha} \right) \text{Cov}(\beta, \alpha) \end{aligned} \tag{60.1}$$

Since we made the substitution $u = \ln(T)$, we can obtain the upper and lower bounds on T using the reverse of this substitution:

$$T_U = \exp(u_U)$$

$$T_L = \exp(u_L)$$

The result we have produced will accept a value from the SF (a reliability between 0 and 1) and output the corresponding upper and lower times. It tells us that we can be 95% certain that the system reliability (R) will be reached somewhere between T_L and T_U (if CI=0.95).

60.2.2 Bounds on reliability

Beginning with the linearized equation for the SF: $\ln(-\ln(R)) = \beta \cdot (\ln(T) - \ln(\alpha))$

We make R the subject, which it already is (yay!) so no rearranging needed.

Now substitute $u = \ln(-\ln(R))$: $u = \beta \cdot (\ln(T) - \ln(\alpha))$

As with the bounds on time, the bounds on reliability in terms of u are:

$$u_U = \hat{u} + Z \cdot \sqrt{\text{Var}(\hat{u})}.$$

$$u_L = \hat{u} - Z \cdot \sqrt{\text{Var}(\hat{u})}.$$

This time we have a different formula for $\text{Var}(u)$. Using the delta method on $u = \beta \cdot (\ln(T) - \ln(\alpha))$ we can derive the following expression:

$$\begin{aligned} \text{Var}(u) &= \left(\frac{\partial u}{\partial \beta} \right)^2 \text{Var}(\beta) \\ &+ \left(\frac{\partial u}{\partial \alpha} \right)^2 \text{Var}(\ln(T)) \\ &+ 2 \left(\frac{\partial u}{\partial \beta} \right) \left(\frac{\partial u}{\partial \alpha} \right) \text{Cov}(\ln(T), \ln(\alpha)) \\ &= (\ln(T) - \ln(\alpha))^2 \text{Var}(\ln(T)) \\ &+ \left(-\frac{\beta}{\alpha} \right)^2 \text{Var}(\ln(\alpha)) \\ &+ 2(\ln(T) - \ln(\alpha)) \left(-\frac{\beta}{\alpha} \right) \text{Cov}(\ln(T), \ln(\alpha)) \end{aligned} \quad (60.10)$$

Once we have the full expression for u we need to make the reverse substitution:

$$R_U = \exp(-\exp(u_U))$$

$$R_L = \exp(-\exp(u_L))$$

The result we have produced will accept any time value (T) and will output the bounds on reliability (R) between 0 and 1 at that corresponding time. It tells us that we can be 95% certain that the reliability of the system lies between R_L and R_U (if CI=0.95) at the specified time.

60.2.3 How are the confidence bounds calculated using Python

The above derivations are tedious and become extremely difficult for more complicated equations (such as the Gamma Distribution). Within `reliability` the linearized forms of the SF (in terms of time and reliability) are specified, and then the partial derivatives are calculated using `autograd.jacobian`. In code (for bounds on time) it looks like this:

```
# weibull SF rearranged for t with v = ln(t)
def v(R, alpha, beta):
    return (1 / beta) * anp.log(-anp.log(R)) + anp.log(alpha)

dv_da = jacobian(v, 1) # derivative w.r.t. alpha
dv_db = jacobian(v, 2) # derivative w.r.t. beta

def var_v(self, u): # u is reliability
    return (
        dv_da(u, self.alpha, self.beta) ** 2 * self.alpha_SE ** 2
        + dv_db(u, self.alpha, self.beta) ** 2 * self.beta_SE ** 2
        + 2 * dv_da(u, self.alpha, self.beta) * dv_db(u, self.alpha, self.beta) * self.
        Cov_alpha_beta
    )

# v is ln(t) and Y is reliability
v_lower = v(Y, self.alpha, self.beta) - Z * (var_v(self, Y) ** 0.5)
v_upper = v(Y, self.alpha, self.beta) + Z * (var_v(self, Y) ** 0.5)
```

(continues on next page)

(continued from previous page)

```
# transform back from v = ln(t)
t_lower = np.exp(v_lower)
t_upper = np.exp(v_upper)
```

There are several other challenges to getting Python to do this correctly, such as where to incorporate γ for location shifted distributions, how to distribute the points so they look smooth, how to correct for things (like reversals in the bounds) that are mathematically correct but practically (in the world of reliability engineering) incorrect, and how to correctly transform the bounds on the SF to get the bounds on the CDF or CHF. Some distributions (such as the Gamma Distribution) are particularly difficult and require a slightly different method to that which is explained above.

60.2.4 How can I extract the confidence bounds from the plot

For bounds on time, this can be done using the quantiles option in the fitter as shown below in option 1. Alternatively, once we have the fitted distribution object, we can extract values from bounds on time or reliability directly from the CDF, SF, or CHF. This is shown below in option 2. Multiple examples of the second method are provided in the document on [working with fitted distributions](#).

```
from reliability.Fitters import Fit_Weibull_2P
import matplotlib.pyplot as plt

data = [43, 81, 41, 44, 52, 99, 64, 25, 41, 7]
q = [0.2, 0.3, 0.4]

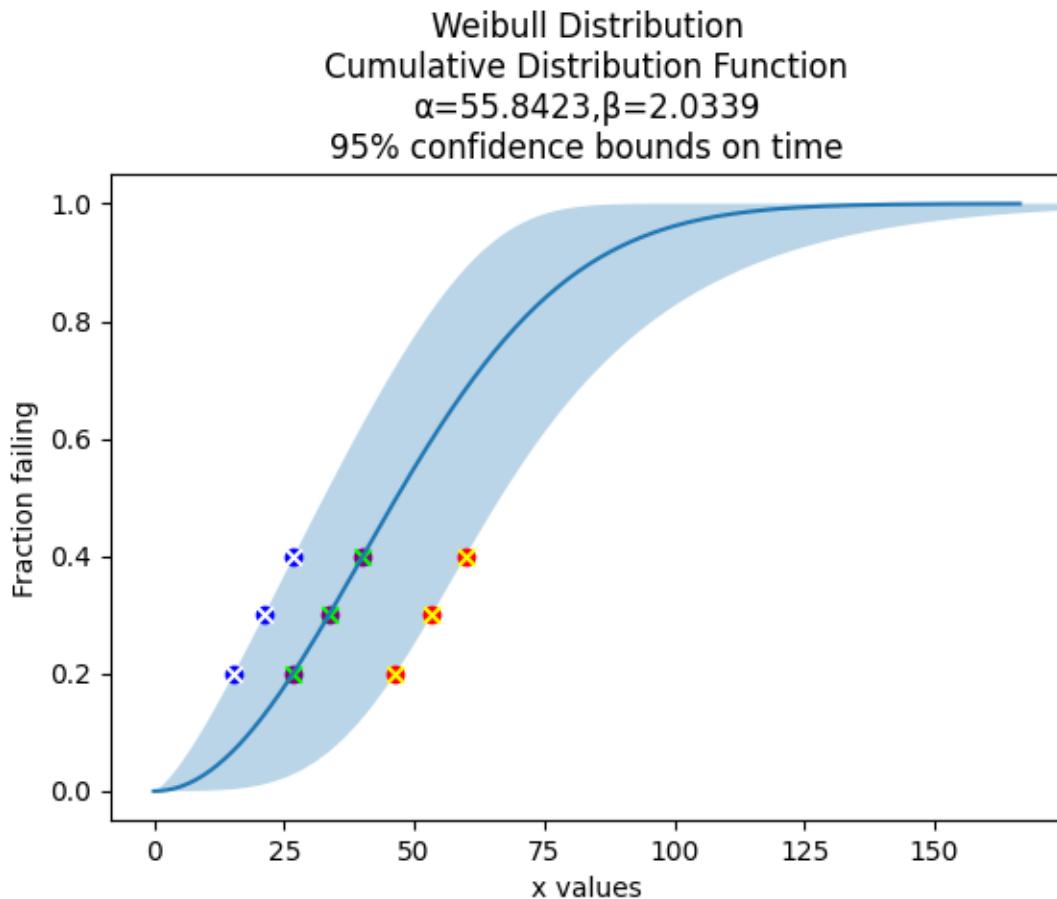
# option 1 using quantiles argument
fit = Fit_Weibull_2P(failures=data, show_probability_plot=False, print_results=False, CI=0.95, quantiles=q)
lower = fit.quantiles['Lower Estimate']
point = fit.quantiles['Point Estimate']
upper = fit.quantiles['Upper Estimate']
fit.distribution.CDF()

# option 2 extracting values directly from the CDF
lower2, point2, upper2 = fit.distribution.CDF(CI_y=q, show_plot=False)

plt.scatter(lower, q, color='blue')
plt.scatter(point, q, color='purple')
plt.scatter(upper, q, color='red')

plt.scatter(lower2, q, color='white', marker='x')
plt.scatter(point2, q, color='lime', marker='x')
plt.scatter(upper2, q, color='yellow', marker='x')

plt.show()
```

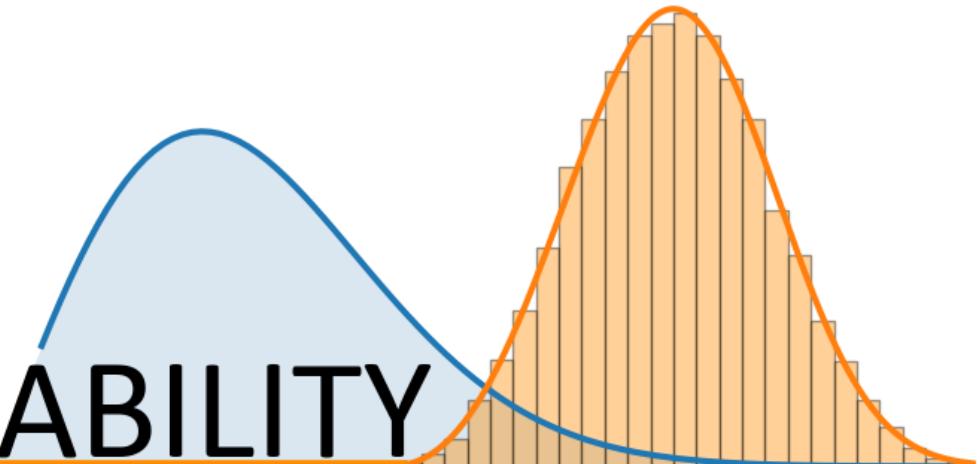


Lastly, for bounds on either time or reliability, the `Other_functions.crosshairs` function provides an interactive set of crosshairs which can be used to find the values using the mouse. A demo of how this works is shown [here](#).

60.2.5 Further reading

More information and formulas are available in the following references from weibull.com, reliawiki and reliasoft:

- Differences Between Type I (Time) and Type II (Reliability) Confidence Bounds
- Confidence Bounds
- The Weibull Distribution



RELIABILITY
A Python library for reliability engineering

CHANGELOG

61.1 Version: 0.8.16 — Released: 27 Dec 2023

Summary of changes

This is a minor release to fix two bugs. The first bug was in plot legend for PoF.SN_diagram which had the confidence intervals hardcoded to 95%. The legend now reflects the specified CI. The second bug was from [this issue](#) where the Lognormal Distribution CDF didn't return a tuple. This is now fixed and aligns with the remainder of the distributions. A few broken hyperlinks in the docs have also been fixed.

61.2 Version: 0.8.15 — Released: 31 Oct 2023

Summary of changes

This is a minor release to add the ability to use Mixture_Model and Competing_Risks_Model as distributions in Other_functions.stress_strength. There is also a minor bugfix to allow Loglogistic distributions with beta<1 to be used. The bug occurred because a text string was created for the mean of a Loglogistic distribution with beta<1 (since there is no mean for such distributions) and the mean was used for a calculation causing an error. The calculation works equally well using the median which does exist for these distributions. Thanks to Cole Tenold for raising this [issue](#).

61.3 Version: 0.8.14 — Released: 07 Oct 2023

Summary of changes

This is a minor release to fix a bug in Other_functions.stress_strength and Other_functions.stress_strength_normal. The bug occurred when stress was greater than strength causing the integration to fail and return incorrect values for the probability of failure. This has been fixed and the plots are also improved. Thanks to github user ctenold for identifying [this issue](#) and providing a working solution. There is also a speed improvement for Fitters.Fit_Weibull_2P_grouped which takes advantage of the np.repeat function to optimize the process of generating the arrays for large datasets of repeating values. Thanks to Tomas Santos for identifying this improvement.

61.4 Version: 0.8.13 — Released: 02 Sep 2023

Summary of changes

This is a minor release to fix a bug in the plot of Reliability_testing.chi2test identified in [this issue](#). The normed parameter in numpy.histogram is deprecated and had been replaced by the density parameter.

61.5 Version: 0.8.12 — Released: 02 Aug 2023

Summary of changes

This is a minor release to fix a bug in the plot of Nonparametric.RankAdjustment identified in [this issue](#). The values weren't being sorted. This has been fixed by adding an optional 'sort' argument to Probability_plotting.plotting_positions.

61.6 Version: 0.8.11 — Released: 07 Jul 2023

Summary of changes

This is a minor release to remove the docutils requirement. There was a [bugfix](#) in docutils and an update to sphinx so we no longer need to specify docutils<0.18.

61.7 Version: 0.8.10 — Released: 15 May 2023

Summary of changes

This is a bugfix release.

Bug Fixes

- Requirements needed to be updated. Specifically some changes in pandas 2.0.0 needed to be promulgated to reliability so it is no longer compatible with earlier versions of pandas.

61.8 Version: 0.8.9 — Released: 23 Apr 2023

Summary of changes

This is a bugfix release.

Bug Fixes

- There was a bug in Competing Risks and Mixture Models which occurred when the PDF or HF had an asymptote, resulting in an error from matplotlib. Values that are almost inf have been capped to 1e100 to prevent overflow in matplotlib.
- A change to the python core across multiple versions resulted in key value pairs from iterators returning the key as a tuple rather than a float. This only affected ALT Fitters and caused all of them to fail. Within Utils the key float is now extracted from the key tuple to resolve this bug.

Other

- Added PDF and EPUB downloads in readthedocs.

61.9 Version: 0.8.8 — Released: 21 Feb 2023

Summary of changes

This is a bugfix release.

Bug Fixes

- API changes in matplotlib which deprecated some keywords and axes classes resulted in two serious bugs being introduced into reliability. This has been fixed and the requirements for matplotlib have been updated to >=3.7.0.

61.10 Version: 0.8.7 — Released: 28 Jan 2023

Summary of changes

This is primarily a bugfix release, with some minor API changes and new features.

New features

- `Reliability_testing.likelihood_plot` is a new function to generate a likelihood plot. See the [documentation](#) for more detail.

API Changes

- `Utils.round_to_decimals` has been replaced by `Utils.round_and_string`. This function always returns a string and the rounding rules applied are better than before as they apply scientific notation for very large and very small numbers. This was an issue in plot titles and plot text where very large numbers weren't being displayed in scientific notation.
- Within Fitters and Distributions that have been created by Fitters, the confidence intervals can be turned off using `CI_type='none'`. Previously this was `CI_type=None` which caused some issues with confidence interval inheritance between Fitters and Distributions and an inability to turn them off when called from Distributions.
- `Utils.life_stress_plot` will swap the x and y axes if `ax='swap'`. This is useful for making a stress-life plot which is similar to an SN_diagram.
- All of the ALT_Fitters will now accept `show_life_stress_plot='swap'` to return a stress-life plot by swapping the axes of the standard life-stress plot.

Bug Fixes

- The axes limits for the histogram plot in `Fit_Everything` crashed when a left asymptote occurred.
- `KStest` and `chi2test` now accept Mixture Model, Competing Risks Model, and DSZI model for the distribution.
- The line of best fit in life stress plots started at 1. This was a problem for data less than 1. The line of best fit now starts at 0.
- When using `force_beta` with `RRX` in `Fit_Weibull_2P`, the beta that is returned is the inverse of `force_beta`. This only occurred for `RRX` for `Fit_Weibull_2P`. Thanks to Github user `ctenold` for identifying this bug and providing the solution in [this issue](#).

Other

- Added a new textbook to Recommended resources.
- The CDF, SF, and CHF methods in Distributions did not include all the parameters in the documentation.

61.11 Version: 0.8.6 — Released: 06 Aug 2022

Summary of changes

This is primarily a bugfix release to deal with a few minor bugs and a few minor features have been added to make it easier to save figures programatically for later use.

New features

- `Fit_Everything` and `ALT_Fit_Everything` now return figure handles for the figures. This behaviour is similar to each of the individual functions in Fitters and ALT_Fitters which return axes handles. This was done to allow the figures to be saved programatically for use later.
- A new `Utils` function `reshow_figure` will reshown a figure that has been closed. The plot interactivity is lost due to an unresolved [issue](#) with matplotlib, but the function still allows a figure to be reshown from an axes or figure handle.

API Changes

- The returned figure handles from Fit_Everything and ALT_Fit_Everything are available from the results. For example “results.probability_plot” will return the figure handle for the probability plot. The the API reference of [Fit_Everything_ALT](#) and [Fit_Everything](#) for details.
- ALT_Fit_Everything now accepts 2 kwargs of failure_stress and right_censored_stress as alternatives to the arguments of failure_stress_1 and right_censored_stress_1. This is used to provide consistency with the other functions in ALT_Fitters which also accept failure_stress and right_censored_stress.

Bug Fixes

- There was a floating point precision error in Distributions.Mixture_Model when the check for sum(proportions) was done. See [this issue](#) for details.
- Mixture_Model and Competing_Risks_Model encountered an error when provided with a single value. This was due to two bugs: a type mismatch in the combiner sub function and an inability to iterate a single element array when trying to find the plotting limits if there is only a single value provided (solved by disabling plotting for this case). Thanks to Smita for identifying this bug.

Other

- Minor optimisations to the probability plotting in ALT_Fit_Everything to merge a function that is only used once and did not need to be its own function.
- Minor corrections to the docs and fixing a broken link.

61.12 Version: 0.8.5 — Released: 25 May 2022

Summary of changes

This is bugfix release to deal with a minor bug.

Bug Fixes

- The equation for the Loglogistic Hazard function was wrong. It mistakenly used the equation for the Weibull Hazard Function. Fitted results are unaffected. This bug only affected the plot and numerical outputs of the Loglogistic Hazard Function. Thanks to Kaiya Raby for finding and reporting this bug.

61.13 Version: 0.8.4 — Released: 05 May 2022

Summary of changes

This is bugfix release to deal with a minor bug.

Bug Fixes

- Weibull Mixture models and Weibull Competing Risks models obtain their initial guesses by splitting the data into two groups and fitting a Weibull distribution to each group. If there are not two distinct groups, the estimate for the dividing line between the two groups may result in one of the groups having fewer than 2 failures allocated to the group. This causes an error when fitting the Weibull distribution to that group. It has been fixed by checking the number of elements in the group and moving the dividing line if required. The dividing line is only an estimate used to obtain the initial guess for the parameters and the dividing line is not used in the final MLE or LS estimates in the fitter.

61.14 Version: 0.8.3 — Released: 17 Apr 2022

Summary of changes

This is bugfix release to deal with a minor bug.

Bug Fixes

- When a distribution was fitted and the defaults for CI and CI_type were overridden, these new values should have been inherited by the distribution object. They were not causing the probability plot to always show confidence intervals with CI=0.95 and CI_type='time'. This has been resolved so that the probability plot will use the values for CI and CI_type that the user specifies and the distribution object returned with the results will also inherit these values as defaults. The distribution object can still be used with any CI and CI_type but the defaults are now those which have been inherited from the values given to the fitter. This bug affected the plots generated by all fitters which display confidence intervals.

Other

- Fixed a deprecation in pandas associated with df.append being replaced with pd.concat.

61.15 Version: 0.8.2 — Released: 23 Mar 2022

Summary of changes

This is bugfix release to deal with a minor bug.

Bug Fixes

- When only 1 failure time was entered and the shape parameter was specified, the Anderson Darling calculation would crash due to receiving a float not an array. Thanks to Francois Besnard for reporting this. This is now resolved.

61.16 Version: 0.8.1 — Released: 18 Jan 2022

Summary of changes

This is bugfix release to deal with a minor bug.

Bug Fixes

- The histogram plot in Fit_Everything had a bug which caused plotting to fail, as identified by [this issue](#). This is now resolved.

61.17 Version: 0.8.0 — Released: 09 Jan 2022

Summary of changes

The major changes in this release include enabling confidence bounds to be extracted programmatically from the distribution object, and a complete rewrite of the reliability_growth function. There are also several minor changes, mainly to the documentation and a minor bugfix.

New features

- Extracting confidence bounds on CDF, SF, and CHF for bounds on time or bounds on reliability can now be done directly from the distribution object that is created by the fitter, as shown [here](#). This required a large number of functions to be modified and resulted in several API changes (see below).

- Repairable_systems.reliability growth has been completely rewritten. This function now includes both the Duane and Crow-AMSAA reliability growth models. The parametrisation of the Duane model has been modified to match what reliasoft uses.
- New dataset called system_growth has been added to the Datasets module.

API Changes

- Repairable_systems.reliability growth has been completely rewritten, so the inputs and outputs are completely different. The old version of the Duane model has been replaced without deprecation. Users needing to use the old version should use v0.7.1 of reliability.
- All references to percentiles have now been replaced by quantiles. Note that the previous percentiles argument in the Fitters mandated values between 0 and 100. The quantiles argument mandates values between 0 and 1. They are otherwise the same, just a factor of 100 different. This has been changed without deprecation, so it may cause your code to break if you are using the percentiles argument. This change was made for simplicity, since the plots show quantiles so there was no real need to have a new argument for the same thing multiplied by 100.
- The subfunctions (.CDF(), .SF(), .CHF()) for each Distribution (that has confidence intervals) now have all relevant arguments visible as args rather than kwargs. This refers to plot_CI, CI_type, CI, CI_y, CI_x. Previously plot_CI, CI_type, and CI were kwargs so your IDE would not show you these. They have been converted to args for ease of use. The arguments CI_x and CI_y are new, and are used to extract the confidence bounds from the plot of a fitted distribution object.

Bug Fixes

- The ALT life stress plots for dual stress models now plot the scatter plot above the surface plot. This is enabled from matplotlib 3.5.0 onwards using the new parameter computed_zorder which respects the zorder specified rather than always plotting surfaces above scatter points.

Other

- Improvements to the API documentation for Convert_data, Datasets, PoF, and Utils modules. This has been a long term body of work to reformat the documentation, and it is finally complete.
- The required version of matplotlib has been upgraded to 3.5.0 to enable the above bugfix for the computed_zorder in ALT life stress plots.
- Theory documents are finished for [censored data](#), [plotting positions](#), [Least Squares Estimation](#), [Maximum Likelihood Estimation](#), and [Confidence Intervals](#).
- Updates pytests for new reliability_growth function.
- New document on [working with fitted distributions](#).
- Added several new utils functions including, distributions_input_checking, extract_CIs, unpack_single_arrays
- Within the Distributions module, the returns from each PDF, CDF, SF, HF, CHF, quantile, inverse_SF function will automatically unpack arrays of length 1. This means that if given an array of length 1 as input, you will now get a float instead of an array as output. This makes it easier for users to avoid the need to manually unpack single values for later use.
- Updated API documentation for Distributions to reflect the numerous changes to the inputs and outputs.

61.18 Version: 0.7.1 — Released: 26 Oct 2021

Summary of changes

This is primarily a bugfix release to deal with some minor bugs.

Bug Fixes

- Other_functions.crosshairs returned the labels as a float which resulted in numbers like 10.0 rather than 10 when decimals=0. The labels are now converted to int when decimals=0 so they will indeed return zero decimals when told to.
- PP_plot_parametric, PP_plot_semiparametric, QQ_plot_parametric, and QQ_plot_semiparametric all had a bug that would cause complete failure. This bug was caused by incorporating an argument that didn't exist. Further details in [this issue](#).

Other

- Added a documentation section (Reliability Theory) which explains how some important algorithms work.
- Made Fit_Everything more tolerant of different names to exclude distributions. For example to exclude the Weibull_CR model, users may type “Weibull_CR”, “CR”, “Weibull_Competing_Risks”, “Competing Risks” and many more variations.

61.19 Version: 0.7.0 — Released: 8 Oct 2021

Summary of changes

Version 0.7.0 has a few really useful enhancements. The first of these is the addition of three of the special models (mixture, competing risks, defective subpopulation) to the Fit_Everything function. The second major enhancement is faster plotting for large datasets using downsampling. There are also numerous bug fixes that resolve several longstanding minor issues as well as some minor changes that make some of the algorithms more reliable.

New features

- Added Weibull_Mixture, Weibull_CR, and Weibull_DS to Fit Everything. This also required changes to the plot window sizes.
- Changed the histogram plot from Fit_Everything. Legend is now on the left as it was getting too long with 15 items. New method used to scale the CDF histogram when there is censored data which is more accurate than the previous scaling method.
- Downsampling for the scatter plot on probability plots. This makes plotting much faster for large datasets (>1000 failures), particularly when using Fit_Everything which has many subplots.

API Changes

- The keyword “downsample_scatterplot” now appears in all fitters and probability plots. It controls whether the scatterplot will apply downsampling. This only affects the plot (when there are over 1000 data points) not the calculations.

Bug Fixes

- Resolved Deprecation Warning from numpy in PoF.strain_life_diagram.
- Fit_Everything had a bug when the best distribution was Weibull_Mixture, Weibull_CR, Weibull_DS due to these distributions not having param_title_long.
- Probability plots with very large datasets (>10000 items) sometimes resulted in the upper ylim being 1. This is equivalent to infinity on a probability plot and caused an error. It is now manually corrected for.
- The least squares method for Fit_Gamma_3P produced a very poor estimate due to a bug. This carried across into the MLE result since LS is used for the MLE initial guess.
- The confidence intervals would sometimes not be displayed on probability plots with very small datasets. This was due to the CI arrays still containing 1's at the extremities. This is now corrected using a more robust filter before plotting.

- Numerous bug fixes for ALT_Fitters, including the ability to exclude distributions (which was previously being ignored due to an error) and greater stability in the initial guess (which previously could crash with some datasets due to non-invertable matrices).
- Fixed a rare bug in all Fitters and all ALT_Fitters when the hessian matrix was non-invertable. This would cause a LinAlgError. Now it will be excepted and print a warning that the confidence intervals can't be obtained.

Other

- Changed the method used by curve_fit within least_squares. Previously was ‘dogleg’ which was very slow. Changed to ‘trf’. This significantly speeds up the location shifted distributions (Weibull_3P, etc.)
- Changed the group splitting algorithm used in Fit_Weibull_Mixture and Fit_Weibull_CR. The new method is more robust and provides a better initial guess of the parameters for MLE.
- Completed the reformatting of the API docs for all the ALT_Fitters. Still need to do this for the Convert_data, Datasets, PoF, Utils modules. Reformatted API docs for these remaining modules will be part of a future release.

61.20 Version: 0.6.0 — Released: 23 July 2021

Summary of changes

Version 0.6.0 has two main improvements. Firstly the behaviour of the optimizers has been changed to be more efficient, and to allow users to try multiple optimizers easily by specifying optimizer='best'. Secondly, the addition of the Defective Subpopulation (DS) and Zero Inflated (ZI) Model now provides a model for which the CDF can range from above 0 to below 1. There are several new Fitters added to take advantage of this as detailed below.

New features

- Ability to specify “best” optimizer will result in multiple optimizers being tried and the best result being used. Optimizers tried are “L-BFGS-B”, “TNC”, “powell” and “nelder-mead”. For more detail see the documentation on [Optimizers](#).
- DSZI_Model has been added to the Distributions module. This model allows for the CDF to start above 0 and finish below 1.
- Fitters for DSZI models, including Fit_Weibull_DS, Fit_Weibull_ZI, Fit_Weibull_DSZI

API Changes

- The optimizer “nelder-mead” will now be accepted as a bounded optimization method. This requires scipy 1.7.0 or higher.

Bug Fixes

- Due to a new Utils function implemented in 0.5.7, a runtime error would occur when the confidence intervals could not be plotted due to too many NaNs in the arrays. This error has now been bypassed.

Other

- The default optimizer has been changed. Previously it was ‘L-BFGS-B’ for < 97% censored data and ‘TNC’ above 97% censored data. Now it is ‘TNC’. For more detail and a flowchart description of the default behaviour, see the documentation on [Optimizers](#).
- The optimizer used is now reported in the printed results for all of the Fitters and ALT_Fitters.
- Removed support for Python 3.6 due to scipy 1.7.0 dropping support for this Python version.
- Change to the algorithm used in Other_functions.make_right_censored_data when making multiply censored data. The algorithm used is explained [here](#).
- Significant speed improvement to Other_functions.make_right_censored_data when making multiply censored data.

- Change to the versioning system. The new system is major.minor.bugfix whereas the previous system was re-served.major.minor. This should allow more frequent bugfix releases.
- Fixed all the tests for ALT_Fitters since this relied upon Other_functions.make_right_censored_data which had a change of algorithm
- Speed improvement to Probability_plotting.plotting_positions to make it 7% faster.

61.21 Version: 0.5.7 — Released: 25 June 2021

Summary of changes

Version 0.5.7 of *reliability* completes a part of this project that has taken almost one year by providing confidence intervals for all standard distributions (except Beta_2P). This release now incorporates confidence intervals for the Gamma_2P and Gamma_3P distributions which were the last remaining to be implemented and proved quite a mathematical challenge. In addition to these enhancements, version 0.5.7 contains numerous minor bug fixes and API changes.

New features

- Fit_Gamma_2P and Fit_Gamma_3P now have confidence intervals implemented. This involved changes to Distributions, Utils, Fitters, and Probability_plotting modules.

API Changes

- Added “dateformat” argument to Other_functions.crosshairs. This provides datetime formatting capability for x axis crosshair labels and annotations. Useful if the plot contains datetime data on the x axis.
- Fully deprecated Other_functions.convert_dataframe_to_grouped_lists
- Fully deprecated the ALT_probability_plotting module as this was made redundant by the improvements to ALT_Fitters in v0.5.6
- Fit_Weibull_Mixture and Fit_Weibull_CR didn’t accept kwargs. All kwargs are now passed directly to matplotlib making it possible to change color, label, linestyle, etc on the probability plot of these distributions.
- In stress_strength and stress_strength_normal the argument show_distribution_plot has been changed to show_plot. This is done for simplicity and standardisation.
- The outputs from all nonparametric functions (.KM, .RA, .NA) are now arrays. Previously these were lists.
- Repairable_systems.optimal_replacement_time argument “show_plot” has been changed to “show_time_plot”. There is another argument “show_ratio_plot” which has been added. While normally expecting True/False, these arguments will also accept axes subclasses if you want them to plot on a specific axes.
- All of the ALT_Fitters (except Fit_Everything_ALT) will now accept an axes object into their show_probability_plot and show_life_stress_plot arguments. If an axes object is passed, the plot will be added to the axes specified. This enables the plots to be placed in subplots rather than always being in their own figures.

Bug Fixes

- Reliability_testing.reliability_test_planner had an error when solving for number of failures. It gave a number 1 more than it should. The number of failures should ensure the MTBF is always above the minimum requirement.
- Incorrect formula for stress strength interference was used. This created negligible difference at small probabilities of failure but when stress.mean > strength.mean the difference was significant. Thanks to Jake Sadie for discovering this.
- All fitters that extracted the covariance (eg. Cov_alpha_beta) took the abs value. This was incorrect as covariance can be negative. This may have led to minor errors in some of the confidence intervals on the plots as covariance is used for these confidence intervals.

- Other_functions.distribution_explorer had a bug due to a change that matplotlib made to the type of error raised. This caused axes to be removed and not redrawn when the radio buttons were toggled. This has been fixed by hiding the axes rather than removing them.
- CI_type of None was not being passed from Fitters resulting in an inability to hide the confidence intervals on the plot as the presence of None resulted in the default of ‘time’ being used. CI_type=None as a kwarg from fitters will now suppress the confidence intervals in the probability plot.
- Exponential_probability_plot and Exponential_probability_plot_Weibull_Scale now allow fitting with 1 failure. Previously required 2 failures. This change was made because Fit_Exponential_1P only requires 1 failure so the limitation was rule based not a mathematical limitation.
- Minor fixes to how the confidence intervals are prepared to ensure the arrays are cleaned of illegal values caused by precision errors.

Other

- Improvements to API documentation. This has been a long term work in progress, but is nearly finished.
- Speed enhancement (x10) to Repairable_systems.optimal_replacement_time and the addition of a new plot (cost ratio vs replacement interval). Thanks to Ed Burrows for contributing the speed enhancement.
- chi2test and KTest will no longer produce their own figure and show the plot automatically. This now enables the plot to be added to an existing figure as a subplot. If not part of a subplot the behaviour is unchanged except that you now need to use plt.show() to show the plot.

61.22 Version: 0.5.6 — Released: 7 March 2021

Summary of changes

Version 0.5.6 of *reliability* is focused on enhancing the accelerated life testing (ALT) section of the library. This release includes a complete rewrite of ALT fitters and supporting Utils, comprising around 13000 lines of code (about 28% of the total codebase). This is the biggest update in terms of lines of code for this library. The rewrite also includes new ALT models (bringing the total from 20 to 24) and tremendous speed enhancements. In addition to the rewrites done to ALT_fitters, there are numerous other small enhancements and bug fixes detailed below.

New features

- Fitters.Fit_Everything now includes an option to show_best_distribution_probability_plot. Default is True.
- Each of the functions within ALT fitters now has a goodness of fit dataframe printed with results.
- Other_functions.make_ALT_data is a new function that enables ALT data to be created. This is useful for testing the functions within ALT_Fitters.
- ALT_fitters was sensitive to the initial guess as it used curve_fit. The initial guess has been changed to use least squares to obtain the initial guess since the stress-life equations are all linearizable.
- ALT_fitters.Fit_Everything_ALT is a new function that enables users to fit all the ALT models.
- ALT_fitters now has Dual_Power models, bringing the total available models to 24.

API Changes

- The ALT_probability_plotting module has been deprecated. Functions will still run with a Deprecation Warning. This was done because all the functionality has been included in the new ALT_fitters module.
- ALT_fitters functions have several changes to the inputs and outputs. Please see the documentation for detail of the new input and output arguments.
- All the probability plots now have a new argument “show_scatter_points” which allows the scatter plot to be hidden if set to False. This was implemented based on [this issue](#).

Bug Fixes

- Failure to fit any of the ALT_fitters will now report the failure and run with the initial guess, rather than crashing.
- make_right_censored_data used a seed but this seed was ineffective due to the use of both the random module and numpy.random. Changed to use only numpy.random so now the seed achieves repeatability.
- ALT_fitters had incorrect confidence intervals for b in Exponential, a in Power, and c in Dual-Exponential
- ALT_fitters Eyring models would crash if not given right_censored data.
- Some ALT models didn't accept data with < 2 failures at each stress level. The new requirement is to have at least as many failures as there are parameters in the model. It is possible to have a single failure at each stress level and still fit the model.
- The percentiles dataframe in Fit_Weibull_3P had the first column set as the index. This has been corrected to retain the original index. Identified in [this issue](#).
- The function plotting_positions sorted the failure data and returned sorted lists. This made it difficult if users wanted to specify different colors for each of the points. plotting_positions now returns the results in the same order the input was given, as per [this issue](#).
- Some datasets with some optimisers could cause a crash due to a non-invertable hessian matrix. This error is now caught and a warning is issued about the confidence intervals without causing a crash.

Other

- Minor improvement to scaling and text positions in stress_strain_diagram
- CodeCov was broken when the continuous integration was changed from Travis_CI to GitHub Actions. CodeCov reporting is now fixed and the coverage will be improved upon progressively.
- All the Fitters now return the axes handles in the probability_plot output object.
- Started work on API documentation. This is already available using the help function in Python, but adding it to `readthedocs` makes it much easier to read.
- Fit_Expon_1P and Fit_Expon_2P are now fully deprecated and have been removed. These were replaced by Fit_Exponential_1P and Fit_Exponential_2P in version 0.5.4 (released Nov 2020).
- The Stress_strength module is now fully deprecated and has been removed. The functions from within this module were renamed and moved to the Other_functions module in version 0.5.5 (released Jan 2021).

61.23 Version: 0.5.5 — Released: 6 January 2021

Summary of changes

Version 0.5.5 of *reliability* has significant improvements to the initial guess methods for the Fitters functions. This makes all the fitters much faster and more accurate. There are also many new enhancements including functions to help with importing data from Excel and converting data between different formats. There are many bug fixes in this release. The other major change is in code formatting using Black.

New features

- All of the standard fitters have been significantly improved with the following features:
 - Least Squares estimation is now available. Previously the fit was solely achieved using MLE. MLE remains the default.
 - For the least squares estimation, users may select RRX, RRY, LS. RRX and RRY are rank regression on X and rank regression on Y respectively. LS will perform both RRX and RRY and use the one with the best log-likelihood.

- There are 3 optimisers to choose from for all of the standard fitters. These are L-BFGS-B, TNC, powell. Previously there was only an option for some of the fitters and the optimiser was not standardized. L-BFGS-B is default if there is less than 97% censored data, otherwise TNC is the default optimizer above 97% censored data.
- Removal of scipy as the method to obtain the initial guess for MLE. With the inclusion of least squares estimation, the MLE method is much faster since it is not reliant on scipy to provide an initial guess (which failed to account for right censored data and often gave a poor guess).
- Addition of a new module for converting data between different formats. The module `reliability.Convert_data` allows for conversion between FR (failures, right censored), FNRR (failures, number of failures, right censored, number of right censored), and XCN (event time, censoring code, number of events). It also provides a streamlined process for importing data from xlsx files, for exporting data to xlsx files, and for printing the dataset in a dataframe for easy visualisation.

API Changes

- All of the standard fitters now include method and optimizer arguments.
- The non-standard fitters (`Fit_Everything`, `Fit_Weibull_Mixture` and `Fit_Weibull_CR`) now include optimizer argument.
- `Fitters.Fit_Weibull_2P`, `Fitters.Fit_Weibull_3P`, `Fitters.Fit_Weibull_2P_grouped` have had some changes to their input arguments so that they all include method and optimizer. The `initial_guess_method` option is gone as it has been replaced by least squares estimation.
- The function `Other_functions.Convert_dataframe_to_grouped` lists is now deprecated. The functionality is captured within the new `Convert_data` module.
- The entire `Stress_strength` module has been deprecated. This is because there were (and likely only ever would be) two functions in this module which is not enough to justify a separate module. The two function have been moved into `Other_functions` and renamed. Full deprecation will occur in March 2021 (in version 0.5.6), and until then a `DeprecationWarning` will be printed and the old functions will still work. The renaming is as follows:
 - `reliability.Stress_strength.Probability_of_failure` ⇒ `reliability.Other_functions.stress_strength`
 - `reliability.Stress_strength.Probability_of_failure_normdist` ⇒ `reliability.Other_functions.stress_strength_normal`

Bug Fixes

- fixed a bug in `Reliability_testing.reliability_test_duration` in which certain inputs resulted in 1 failure and the plot limits caused a crash when `left=right` limit.
- fixed a bug in `ALT_Fitters` where the `CI` string in the results title would be rounded to an integer. This would cause 0.975 to appear as 97% rather than 97.5%.
- fixed a bug in `Fit_Weibull_Mixture` and `Fit_Weibull_CR`. When given input as a list of integers, it failed to convert these to floats and then crashed due to an error with type conversion error between `int32` and `float64`
- `probability_plot_xylims` had a bug when there is only 1 datapoint as `xlower=xupper` and `ylower=yupper`. Cases with only 1 datapoint are now handled appropriately.
- `Fitters` had a bug where `force_beta` or `force_sigma` needed to be a float. It would crash if an int was supplied.
- Fixed a bug in all the ALT fitters where a crash would occur when use level stress was not provided. This was due to the use life being referenced in all cases rather than just in cases where the use level stress was specified.
- ROCOF had a bug that was only evident when the ROCOF was found to be constant. This was caused by a formula using `n` instead of `n+1` for the sample size.

Other

- `Utils` has 2 new functions (`linear_regression` and `least_squares`). These are now used by `Fitters` to obtain the least squares estimates.

- The format of all the printed fitters outputs has been improved. More detail is provided, goodness of fit parameters are provided and the formatting is better.
- Dataframes everywhere are formatted better to retain the index but not display it.
- Text output for sample_size_no_failures.
- Text output for one_sample_proportion.
- Text output for two_proportion_test.
- one_sample_proportion will now return 0 or 1 for the lower and upper reliability estimates instead of NaN in cases when there are all failures or all successes.
- ALT_Fitters has 2 new results: alpha_at_use_stress (mu for Lognormal and Normal, Lambda for Exponential) and distribution_at_use_stress. These are provided for convenience and were able to be calculated from the previous results.
- Title added to all nonparametric results printed.
- Bold and underline enhancements to results titles in all ALT_fitters and in MCF_parametric and MCF_nonparametric.
- Changed Build and Test from Travis CI to GitHub Actions.
- Reformatted all code using [Black](#). This resulted in a significant increase in the lines of code (LOC) count but in actual fact there was not that many new lines added.
- Added another standard dataset called “mixture” and an ALT dataset called “ALT_temperature4”.
- In all the ALT fitters, the initial guess process is now bypassed if an initial guess is specified by the user. Previously the initial guess was always obtained by curve_fit but not used if a user specified initial guess was given. This change enhances speed and enables a failure of curve_fit to be bypassed through specifying an accurate initial guess.
- Documentation updates to reflect version 0.5.5 API changes and results printed.
- Updated the Logo for *reliability* and provided the [code](#) for generating the new logo.
- Changed the structure of the README to put the link to the documentation up higher.

61.24 Version: 0.5.4 — Released: 7 November 2020

Summary of changes

Version 0.5.4 of *reliability* brings in confidence intervals for many more distributions, as well as the inclusion of the Gumbel distribution. Due to the time it took to get the confidence intervals working, there have been many other minor changes to formatting of plots and printed results that are included in this release.

New features

- Confidence intervals added for Normal, Lognormal, Loglogistic, and Gumbel Distributions. *Confidence intervals for the Gamma and Beta Distributions will be part of 0.5.6 in Feb/Mar 2021*
- Added Gumbel_Distribution to Distributions
- Added Gumbel_Distribution to Other_functions.distribution_explorer
- Added Fit_Gumbel_2P to Fitters
- Added Gumbel_probability_plot to Probability_plotting
- Added Gumbel Distribution to Fitters.Fit_Everything
- Added Gumbel Distribution to Other_functions.similar_distributions

- Added Gumbel Distribution to Stress_strength.Probability_of_failure
- Added Gumbel Distribution to Reliability_testing.chi2test and Reliability_testing.KStest
- Added Loglogistic and Gumbel Distributions to PP_plot_parametric, QQ_plot_parametric, PP_plot_sempiparametric, and QQ_plot_sempiparametric. Loglogistic should have been added in version 0.5.3 but it was missed.
- Added Loglogistic and Gumbel Distributions to Mixture Model and Competing Risks Model. Loglogistic should have been added in version 0.5.3 but it was missed.
- Fit_Everything now plots everything in order of best fit for all 3 of the plots generated.
- Both the Competing Risks Model and Mixture Model now work for negative xvals when the mixture contains one or more Normal and/or Gumbel Distributions. Previously these were be truncated at 0 which could lead to inaccuracies if the model contained Normal Distributions (or Gumbel Distributions, though Gumbel was not available previously).

API Changes

- Confidence intervals were previously available for the Hazard functions of the Weibull and Exponential distributions. This capability has been removed as it was not useful (just as confidence intervals on the PDF are not useful). Any attempt to use confidence interval related keywords (such as CI and CI_type) on the HF of any distribution will generate an error.
- Fit_Everything now includes an option to exclude distributions.
- Fit_Expon_1P and Fit_Expon_2P are deprecated. These have been replaced by Fit_Exponential_1P and Fit_Exponential_2P. Using the old functions will still work and will issue a DeprecationWarning printed to the console. Full deprecation/removal will occur in March 2021 (in version 0.5.6). The reason for the change is to minimize the use of abbreviated terms. It was originally abbreviated because the word Exponential_Distribution seemed too long, but this is no longer valid with Loglogistic_Distribution being added. Also, scipy's function for Exponential is "expon" so Fit_Expon_1P initially seemed like an appropriate abbreviation.
- percentiles have been added to all fitters (except Gamma and Beta). This will print a table of percentiles (with bounds on time) to the console. This is similar to the output that Minitab gives when fitting a distribution.

Bug Fixes

- Other_functions.distribution_explorer had a bug caused by a recent update to matplotlib. When a non-existent axis was deleted, the error matplotlib generated was a ValueError and that is now changed to AttributeError which was not being appropriately handled by distribution_explorer.
- All of the standard distributions expected a list or array for their 5 functions (PDF, CDF, SF, HF, CHF). A command like this "dist.SF(1)" would cause an error and should have been entered as dist.SF([1]). This is now fixed such that if the input is not in a list or array then it will no longer produce an error and the output type will be np.float64.
- Within Fit_Everything if only 3 points were entered some of the AIC values would be 'Insufficient Data'. If the user also specified sort_by='AIC' then an error would be raised by pandas trying to sort by strings and numbers. In this case the sort_by method will automatically be changed to BIC.
- The Exponential confidence intervals were invisible if there were only 2 failures for the fit. This was cause by the upper CI reaching 1 which is effectively infinity on a probability plot. 1's are now filtered out so the CI will always appear.

Other

- Removed margins in the stress_strength plots so that the xaxis coincides with the plot window.
- Changed layout of Fitters.Fit_Everything probability plot and PP plot to be 4x3 without Beta fitted and 5x3 with Beta fitted. This was necessary to include the Gumbel Distribution in the space that Beta previously used.

- Formatting changes to Fitters.Fit_Everything PP plot so the red line extends to the edges of the plot.
- The histogram plot in Fitters.Fit_Everything now has its legend in the order of the the results, such that the best fitting distribution will appear first in the legend.
- Within Other_functions.similar_distributions there were cases when a 3P distribution was fitted and the optimal gamma was 0 (making it the same as its 2P distribution). A filter has been added so the 3P distribution will only be shown if the gamma parameter is non-zero.
- Improved plots for Stress_strength so the distribution xvals extend beyond the plot xlims. This is only noticeable if the plot is moved.
- Adjusted scaling and line colors for all QQ and PP plots to improve the way they are displayed.
- PP_plot_parametric now has labels for quantile lines which are linked to the axes coords, so if the plot is moves / zoomed the labels will follow the plotting window.
- Improved the Mixture Model PDF and HF using the actual formula rather than taking the numerical derivatives of CDF and CHF respectively.
- Fit_Everything can now accept a minimum of 2 failures (previously the minimum was 3) and it will automatically exclude the 3P distributions
- All warnings throughout reliability are now printed in red.
- New Utils function colorprint. This provides a simple API for printing in color, bold, underline and italic.
- Improved input checking for all the fitters. This has been standardised in a Utils function so nothing is missed for each of the fitters.
- Probability_plotting.plot_points previously has a minimum of 2 failures required to plot the points. The minimum is now 1 failure required.

61.25 Version: 0.5.3 — Released: 29 September 2020

Summary of changes

Version 0.5.3 of *reliability* is a major release, adding in the Loglogistic distribution, the RankAdjustment nonparametric method, a new goodness of fit measure (anderson darling) and many other new functions.

New features

- Added Loglogistic_Distribution to Distributions
- Added Fit_Loglogistic_2P and Fit_Loglogistic_3P to Fitters
- Added Loglogistic_probability_plot to Probability_plotting
- Added Fit_Loglogistic_2P and Fit_Loglogistic_3P to Fitters.Fit_Everything
- Added Loglogistic distribution to Other_functions.similar_distributions
- Added Loglogistic distribution to Stress_strength.probability_of_failure
- Added the function Reliability_testing.reliability_test_duration
- Added the function Other_functions.distribution_explorer
- Added Utils.probability_plot_xylims and Utils.probability_plot_xyticks which provide better axes limits and tick labels. These are now incorporated into all probability plots, ALT probability plots and ALT Fitters.
- Added Chi-squared and Kolmogorov-Smirnov goodness of fit tests to Reliability_testing

- Added Anderson-Darling goodness of fit test statistic into all Fitters (It is not appropriate to use for ALT_fitters for the entire model). This now allows users to compare distributions goodness of fit using Log-likelihood, AICc, BIC, or AD. Note that the Anderson-Darling test statistic is the default goodness of fit test statistic in Minitab.
- Added Utils.anderson_darling to simplify the process of calculating the AD statistic. It's a lot of formulas that are best packaged into a function that is called by each of the Fitters.
- Added Datasets.mileage which is a simple dataset with no right censored data.
- Added Nonparametric.RankAdjustment. This method is similar in results to Kaplan-Meier and Nelson-Aalen but very different in the method used.
- Other_functions.make_right_censored_data can now create either singly-censored or multiply-censored data. Previously it only created singly-censored data.

API Changes

- Reliability_testing.reliability_test_planner has an optional argument of two_sided which was set to True as default. This has been changed to one_sided=True, making the default calculation use the one-sided confidence interval and changing the argument name. The reason for this change was to align the function with the approach more commonly used in industry.
- All probability plots had h1 and h2 options for the plotting heuristics. These have been replaced by the argument “a” which is the same as what h1 was. h2 can be calculated from h1 and the length of the dataset so it was redundant. “a” was chosen to align with [wikipedia](#).
- Thanks to the addition of the Nonparametric.RankAdjustment, the functions Probability_plotting.QQ_plot_semiparametric and Probability_plotting.PP_plot_semiparametric now allow for ‘RA’ as the option in their method. Previously the methods were limited to ‘KM’, and ‘NA’ for Kaplan-Meier and Nelson-Aalen.
- Other_functions.make_right_censored_data now has an additional argument of fraction_censored which controls the amount of data to right censor when producing multiply-censored data. There is also a random seed argument added for repeatability.
- All the ALT_fitters were missing loglik as an output. They had loglik2 which is the same as loglik*-2 but this is added for completeness and to match the outputs from Fitters.

Bug Fixes

- Fixed autoscale for cases where the HF is constant so it no longer lies along the yaxis upper limit
- Fit_Everything had a bug in the default xvals for the Beta_Distribution’s histogram which caused an error in some special cases.
- All the quantile functions in each distribution didn’t accept np.float64 and raised an error. They now accept this data type.
- The AICc and BIC in all the ALT_fitters was slightly wrong due to a small coding error.

Other

- Fixed the HF and CHF equations for Exponential_Distribution to be actual equations. This is preferred than using the HF = PDF/SF and CHF=-ln(SF) relationships which breakdown when SF=0 at high xvals. This has also been implemented for the logistic distribution. Can’t do it for Normal, Lognormal, Gamma, and Beta distributions as these do not have closed form solutions for HF and CHF which don’t involve the SF.
- Changed the Gamma_Distribution and Weibull_Distribution mode to be self.gamma when beta < 1. Previously it was “No mode exists when beta < 1” which is true from a formula perspective but it is clear that the mode is equal to gamma as that’s where the asymptote occurs. The only distribution with “no mode exists...” is the Beta distribution as it can have 2 modes for certain values of alpha and beta.

- Updated `Utils.generate_X_array` to use 200 points (rather than 100) and allocated more points to the right hand side of the plot (beyond b99). This was because plots were not displaying smoothly enough for distributions with high skewness.
- Changed default plotting upper limit to b9999. Previously it was slightly more and was not a round quantile. Done for simplicity and minimal change will be noticed.
- Changed the layout of the Probability plots and PP plots in `Fit_Everything` from a 5x2 grid to a 4x3 grid. This made more sense due to the addition of the Loglogistic Distribution which would have made the layout 6x2 which is too long.
- Plotting enhancements to increase the detail in plots using less points (by generating more points where the plots curve and less where the plots are flat). Using 200 instead of 1000 points will make the plots much faster, particularly when multiple distributions are layered. In version 0.5.2 this was just done for the Weibull Distribution but it has now been implemented for all 7 of the standard probability distributions.
- Plotting enhancements to the x and y scale such that the limits are based on the quantiles. This will ensure more relevant detail is shown, particularly for location shifted distributions. In version 0.5.2 this was just done for the Weibull Distribution but it has now been implemented for all 7 of the standard probability distributions.
- Within `Stress_strength.Probability_of_failure`, the integration method has been changed from `quad` to `trapz` based on this issue.
- Within `Stress_strength` the legend text for both plots no longer formats the probability of failure as a percentage and the format is changed to use scientific notation which is much more appropriate for very small failure probabilities.
- Within `Stress_strength` both functions will issue a warning if `stress.mean > strength.mean` to indicate that the user may have assigned the distributions in the wrong order.
- The version requirements for all dependancies have been updated to their most recent versions. This is most important for `scipy` which recently had an update that affects the covariance matrix results.
- Added `__version__` to the `__init__.py` file so that the version number is recorded in the same way as other packages record it.
- `Other_functions.histogram` has an argument for `bins`. Previously this accepted the exact bins to be used and if left blank calculated them using the [Freedman-Diaconis rule](#). In addition to accepting the exact bins to use, the `bins` argument now accepts strings just like `matplotlib` and `numpy`, and the default is now 'auto'. See [numpy](#) for more detail on the strings available.
- `KaplanMeier` and `NelsonAalen` now consider previous `xlim` when plotting. This prevents plot limits from being overridden by the most recent plot.

61.26 Version: 0.5.2 — Released: 14 August 2020

Summary of changes

Version 0.5.2 of *reliability* includes two special distributions, the mixture distribution and the competing risks distribution, along with their respective fitters. Autoscaling is also a great improvement to ensure that plots appear mostly the same, just with their axes scaled appropriately.

New features

- New distributions
 - `Mixture_Distribution`
 - `Competing_Risks_Distribution`
- A new fitter for the Weibull competing risks model (`Fit_Weibull_CR`)

- The output of the Fit_Weibull_Mixture now includes a probability plot instead of a histogram of the PDF and CDF
- The output of the Fit_Weibull_Mixture now prints the confidence interval estimates of the parameters
- Added some datasets for use with the mean cumulative function (MCF_1 and MCF_2).

API Changes

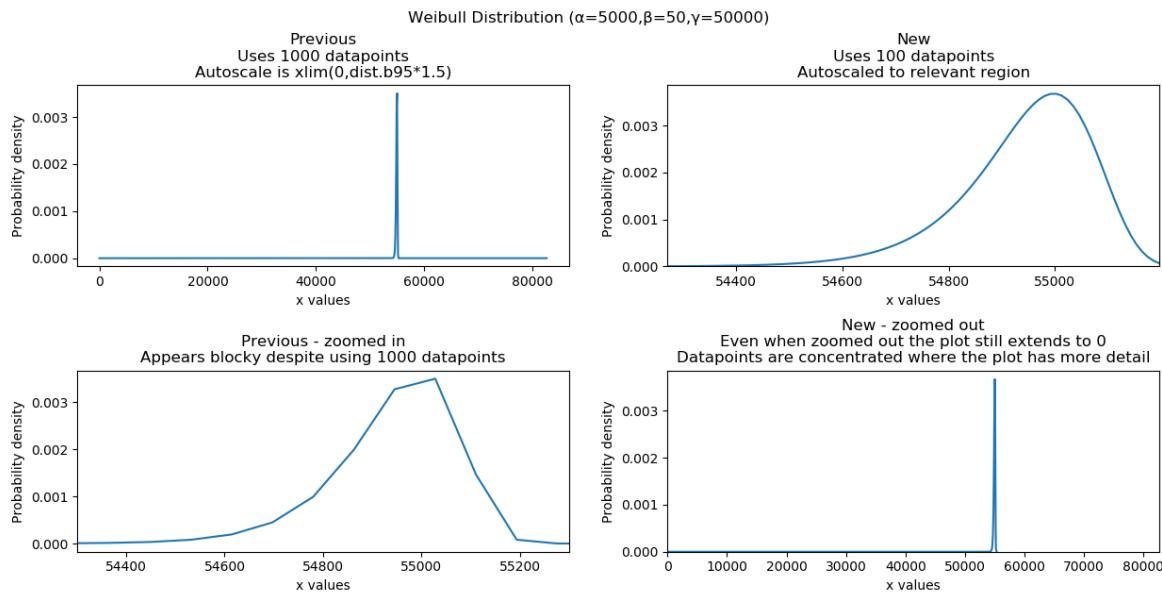
- Within Fitters.Fit_Weibull_mixture the option show_plot has been changed to show_probability_plot to align with all the other fitters.

Bug Fixes

- Fixed the autoscale in Weibull and Exponential distributions that locked autoscaling when confidence intervals were plotted sequentially.
- Automatic removal of zeros for all fitters (except Normal_2P). Previously the zeros were left in the data and resulted in NaNs and crashes. Also added a dedicated error to report input with times below zero.
- Fixed the confidence interval bounds for Kaplan-Meier and Nelson-Aalen CHF plots. Some of the bounds were inf since the $CHF = -\ln(SF)$ which will be inf when $SF=0$.
- MCF_Nonparametric and MCF_Parametric had a bug which caused crashes when the dataset included a system with only one censored time. This has now been fixed.

Other

- Minor clean up of code. Removed unnecessary imports, removed unused variables, etc. Hopefully this will have no noticeable effects.
- Within Fitters.Fit_Everything the histogram output has been improved with better formatting and it now uses the Freedman-Diaconis rule for obtaining optimal bin width.
- Fixed Weibull HF and CHF equations to use actual equations and not PDF/SF or $-\ln(SF)$ as these result in NaN when $SF=0$ (an issue at high xvals). These changes are currently only implemented for Weibull_Distribution.
- Improved creation of xvals for PDF,CDF,SF,HF,CHF within the Weibull Distribution. The changes now generate datapoints where there is more detail (between the 0.1% and 99.9% quantiles) such that only 100 datapoints are needed to show more detail than was previously achieved with 1000 datapoints. This is most noticeable with Weibull distributions that have high beta values and are significantly location shifted. An example of this is shown in the plot below. These changes are only implemented for Weibull_Distribution but will be extended to all distributions in the very near future.
- Improved autoscaling for the Weibull Distribution plots. For location shifted distributions, this zooms in on the 0.1% to 99.9% quantiles allowing users to see more detail. The HF and CHF ylimits are also limited based on the quantiles so that they do not obscure the detail if there is an asymptote to large values or infinity. An example of this is shown in the plot below. These changes are only implemented for Weibull_Distribution but will be extended to all distributions in the very near future.



61.27 Version: 0.5.1 — Released: 08 July 2020

Summary of changes

Version 0.5.1 of *reliability* is a fairly minor release.

New features

- More efficient method used within `Other_functions.similar_distributions`. Results are always consistent and more accurate now.
- `Other_functions.histogram`. This plots a histogram with optimal bin width, better default formatting, and an option to shade bins white above a threshold.

API Changes

- Some of the functions in `reliability.Other_functions` have been moved into `reliability_Utils` and `reliability_Reliability_testing`. The new layout is:
 - `Utils` ⇒ `round_to_decimals`, `transform_spaced`, `axes_transforms`
 - `Other_functions` ⇒ `similar_distributions`, `convert_dataframe_to_grouped_lists`, `crosshairs`, `make_right_censored_data`
 - `Reliability_testing` ⇒ `one_sample_proportion`, `two_proportion_test`, `sample_size_no_failures`, `sequential_sampling_chart`, `reliability_test_planner`
- Within `Other_functions.similar_distributions` the option ‘`monte_carlo_trials`’ has been removed as the distribution sampling method is no longer random.

Bug Fixes

- Fixed confidence interval color inheritance for `Nonparametric.Kaplan_Meier` and `Nonparametric.Nelson_Aalen`. Previously the color was only inherited if specified rather than left as default.
- The default axes labels for both `Stress_strength.Probability_of_failure` and `Stress_strength.Probability_of_failure_normdist` were reversed. The have now been switched to the correct labels.

Other

- Documentation updates to reflect the API changes in Version 0.5.1

61.28 Version: 0.5.0 — Released: 04 July 2020

Summary of changes

Version 0.5.0 of *reliability* is a major release that includes the first introduction of confidence intervals, and many other new features. Significant structural changes have also been made including the use of a `Utils` function and the introduction of automated testing.

New features

- Confidence intervals on fitted distributions ==> this has only been implemented for Weibull and Exponential. It is quite difficult and takes considerable time and testing. I will do Normal and Lognormal distributions next, then Gamma and Beta distributions. I hope to finish them all by September 2020.
- Confidence intervals have been disabled in `ALT_probability_plotting` and `ALT_fitters` to avoid cluttering on the plot.
- The probability plot in `Fit_Everything` now uses the `Exponential_probability_plot_Weibull_Scale` instead of `Exponential_probability_plot`. It is much clearer to see the effectiveness of the fit using the Weibull scale.
- Added an option to seed the `random_samples` functions within the `Distributions` module. This allows for repeatable results.
- Improvements to rounding of all titles, labels, and stats in `Distributions` and `Probability_plotting` using a new function, `round_to_decimals`.
- Added `Other_functions.round_to_decimals` which keeps the specified number of decimals after leading zeros. This is useful as `round` would make very small values appear as 0.
- Minor improvements to color inheritance for `probability_plotting`.
- Minor improvements to confidence interval color inheritance for `Nonparametric.Kaplan_Meier` and `Nonparametric.Nelson_Aalen`.
- Within `Stress_strength`, the method of obtaining the solution has been changed from monte carlo to integration. Thanks to Thomas Enzinger for providing the formula for this method in response to an [Issue](#) that was raised. Using the integration method, accuracy is much higher (1e-11 error now vs 1e-3 error previously) and always consistent, and the speed is significantly improved over the monte carlo method. As noted below in API changes, there is no need to specify the number of `monte_carlo_samples` and no option to obtain the convergence plot.
- Within `Stress_strength`, the colors used for shading have been changed to improve the style.
- `Probability_plotting.plot_points` now includes the option to plot the points for the PDF and HF. These are not very useful as they appear messy due to the discontinuous nature of the function, but they are added for completeness.
- Added `Other_functions.transform_spaced`. This is similar to `np.linspace` and `np.logspace` but it creates an array that is ‘weibull spaced’, ‘normal spaced’, ‘exponential spaced’, ‘beta spaced’, or ‘gamma spaced’. It is used to get data points for the confidence intervals so they are as evenly spaced as possible, particularly on probability paper. This function is likely to be moved into `utils`.
- `Other_functions.make_right_censored_data` has been added. This function accepts uncensored data and a threshold, and returns failures and `right_censored` arrays.
- Added `mplcursors` to requirements in `setup.py` as it is needed for the crosshairs function.
- Added `crosshairs` function to `Other_functions`. This is a very useful feature that provides interactive crosshairs to the plot using snap-to feature and also adds annotations on click events. Thanks to Antony Lee (the author of `mplcursors`) for help with getting this to work using his library.

Bug fixes

- Within Stress_strength, there are improvements to the fill_between method as it had errors in some special cases.
- Fixed an [Issue](#) in Lognormal_Probability_Plot that occurred for very large numbers (above 1e20)

API Changes

- Within Stress_strength, the output format has changed from an object to a returned value of the probability of failure. This makes it much more simple to access the answer since the object had only one value.
- Within Stress_strength, the method of obtaining the solution has been changed from monte carlo to integration. As a result, there is now no need to specify the number of monte_carlo_samples and no option to obtain the convergence plot.
- Added the options initial_guess_method and optimizer to Fit_Weibull_2P and Fit_Weibull_3P. They were previously only in Fit_Weibull_2P_grouped. It is planned to add these options to all fitters.
- There is now the option CI_type for the Weibull and Exponential fitters. This allows users to chose between confidence bounds on reliability and time. This option will be added to all fitters as the confidence intervals for the other distributions are completed.

Other

- Added tests folder. This is planned to include automated tests.
- Created utils module. I plan to move some utilities into here that are currently inside other modules where users can access them, but users should never need to access them so they just create clutter in the dropdown lists of your IDE.
- Added Reliability_testing module. I plan to move everything related to reliability testing out of Other_functions as there is now enough functions to justify a new module dedicated to reliability testing.
- Documentation updates to reflect the changes in Version 0.5.0

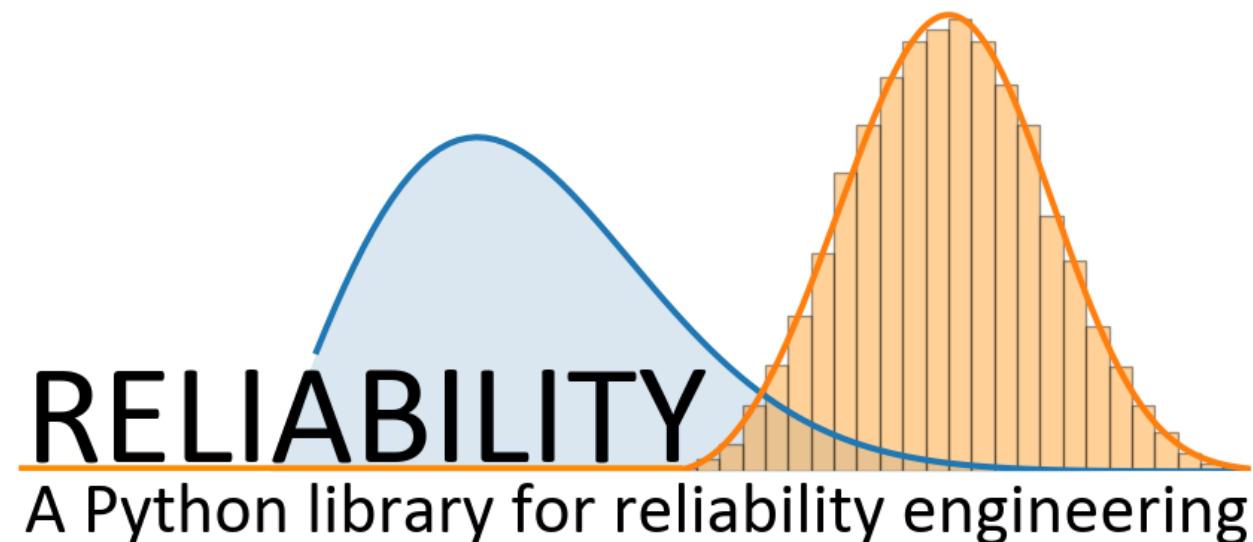
61.29 Version: 0.4.9 — Released: 27 April 2020

New features

- Updates to reliability_test_planner to include option for failure terminated test

Other

- Addition of this Changelog to the documentation



CHAPTER
SIXTYTWO

DEVELOPMENT ROADMAP

As of January 2022, development of *reliability* has transitioned to primarily codebase maintenance (bug fixes). I am turning my attention to learning Web Development in order to bring the power of this library to an interactive Web Application. I hope this approach will help a wider audience who may not be familiar enough with Python to use this library in its current form. If you're an experienced web developer and willing to help get this project off the ground, please get in touch via email (alpha.reliability@gmail.com).

The following items are residual “TO DO” tasks that I hope to some day return to. Listed in priority order they are:

- Within all fitters, use the FNRM format to give speed improvements in the same way as Fit_Weibull_2P_grouped works internally. This will subsequently result in the deprecation of Fit_Weibull_2P_grouped once its advantage is integrated in Fit_Weibull_2P. Need to confirm this method does not introduce too much cumulative error due to floating point precision limitations.
- Add confidence intervals for Weibull_Mixture, Weibull_CR, Weibull_DS, Weibull_ZI, and Weibull_DSZI
- Proportional Hazards Models - This is available in [Lifelines](#).
- Warranty Module. A new module of tools for warranty calculations.
- Add [step-stress models](#) to ALT models.
- Add the [Kijima G-renewal process](#) to repairable systems.
- Improvement to the online documentation for how these methods work, including the addition of more formulas, algorithms, and better referencing.
- Make tests for everything that doesn't have a test yet.
- Add plotting to all things that can plot in order to increase test coverage.

I welcome the addition of new suggestions, both large and small, as well as help with writing the code if you feel that you have the ability. If you have something to add:

- new features - please send me an email (alpha.reliability@gmail.com) or fill out the [feedback form](#).
- bugs - please send me an email (alpha.reliability@gmail.com) or raise an [Issue](#) on Github.



RELIABILITY
A Python library for reliability engineering

COPYRIGHT INFORMATION

reliability is licensed under the [LGPLv3](#). Some FAQs on the GPL and LGPL are answered [here](#).

Under the GPLv3 license, the use of *reliability* is FREE and unrestricted for both individuals and commercial users.

Individuals and commercial users are FREE to incorporate *reliability* as a part of their own software, subject to the following conditions:

- If linked dynamically (see below):
 - Your software license is unaffected
 - No credit / attribution is required
 - No changes are permitted (or possible since it is dynamically linked)
 - You must not in any way suggest that the licensor endorses you or your use.
 - You must not sell your software.
- If linked statically (see below):
 - Your software must also be released under the GPLv3 license and you must provide a link to the license.
 - You must give appropriate credit / attribution to the author of *reliability* and the version you have copied.
 - You must indicate any changes that were made.
 - You must not in any way suggest that the licensor endorses you or your use.
 - You must not sell your software.

The major limitation imposed by the GPLv3 license is that individuals and commercial users must not incorporate *reliability* into their own software (either in part or as a whole, either statically or dynamically), and then sell their software. If you or your company is looking to use *reliability* as part of your software, and you intend to sell your software, then this use falls under a Commercial License and will attract a small fee to permit redistribution. Please contact alpha.reliability@gmail.com if you believe your usage falls under the Commercial License.

63.1 Static vs Dynamic linking

Static linking is where you include part or all of the source code for *reliability* in your package. i.e. you are redistributing source code obtained from *reliability*. Static linking is strongly discouraged as it does not benefit from improvements and bug fixes made with each release.

Dynamic linking is where your software (typically another Python library or repository) uses *reliability* as a dependency. In dynamic linking you are not redistributing any of the source code from *reliability* and are simply using it as an external tool. Dynamic linking is encouraged as it enables users to use the most recent version of *reliability* which benefits from improvements and bug fixes.

63.2 The License



GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <<https://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document,
but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and
conditions of version 3 of the GNU General Public License, supplemented by the
additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public
License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an
Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but
which is not otherwise based on the Library. Defining a subclass of a class defined by
the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the
Library. The particular version of the Library with which the Combined Work was made
is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source
for the Combined Work, excluding any source code for portions of the Combined Work
that, considered in isolation, are based on the Application, and not on the Linked
Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or
source code for the Application, including any data and utility programs needed for
reproducing the Combined Work from the Application, but excluding the System Libraries
of the Combined Work.

(continues on next page)

(continued from previous page)

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
 - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
 - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.
 - e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If

(continues on next page)

(continued from previous page)

→ you use option 4d1, you must provide the Installation Information in the manner
→ specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a
→ single library together with other library facilities that are not Applications and
→ are not covered by this License, and convey such a combined library under terms of
→ your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library,
→ uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on
→ the Library, and explaining where to find the accompanying uncombined form of the same
→ work.

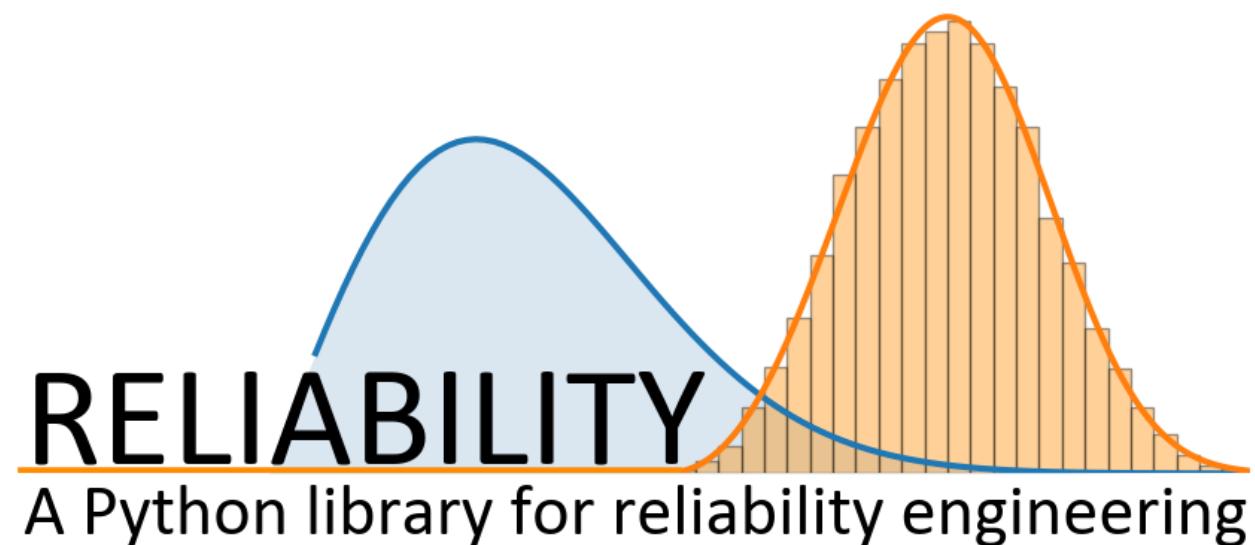
6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser
→ General Public License from time to time. Such new versions will be similar in spirit
→ to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it
→ specifies that a certain numbered version of the GNU Lesser General Public License "or
→ any later version" applies to it, you have the option of following the terms and
→ conditions either of that published version or of any later version published by the
→ Free Software Foundation. If the Library as you received it does not specify a version
→ number of the GNU Lesser General Public License, you may choose any version of the GNU
→ Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future
→ versions of the GNU Lesser General Public License shall apply, that proxy's public
→ statement of acceptance of any version is permanent authorization for you to choose
→ that version for the Library.

Reference: <https://www.gnu.org/licenses/lgpl-3.0.html>



CHAPTER
SIXTYFOUR

CITING RELIABILITY IN YOUR WORK

If *reliability* contributes to a project that leads to a scientific publication, please acknowledge this contribution by citing the DOI [10.5281/zenodo.3938000](https://doi.org/10.5281/zenodo.3938000).

The following reference is using APA:

Reid, M. (2022). Reliability – a Python library for reliability engineering (Version 0.8.2) [Computer software]. Zenodo. <https://doi.org/10.5281/ZENODO.3938000>

If you would like to use another referencing style, the details you may need are:

- **Author:** Matthew Reid
- **Year published:** 2022
- **Title:** Reliability – a Python library for reliability engineering
- **Version:** 0.8.2
- **Platform:** Python
- **Available from:** <https://pypi.org/project/reliability/>
- **DOI:** 10.5281/zenodo.3938000

Note that the version number is constantly changing so please check [PyPI](#) for the current version.

If you have used `reliability` in any published academic work, I would love to hear about it (alpha.reliability@gmail.com). Depending on the usage, I may provide a link to your work below.

Links to articles and papers that have used the Python reliability library:

- Reliability Engineering Using Python - by Matthew Reid.
- SurPyval: Survival Analysis with Python - by Derryn Knife.
- International trade distributions and their relation with random fragmentation processes - by Ricardo Bustos Guajardo. Note that this paper is about Econophysics, not reliability engineering, but it does use the Python reliability library to fit Weibull Distributions to trade data.
- Probabilistic characterization of random variables - Phase II - by Javier Alfonso Ochoa Moreno. Note that this article is in Spanish.
- A tutorial for reliability engineers: going from scratch to building Weibull Analyses using Python - by Dr Sarah Lukens.
- Predictive Modeling of a repairable system using Data Analytics Tool - by Chiranjit Pathak.



RELIABILITY
A Python library for reliability engineering

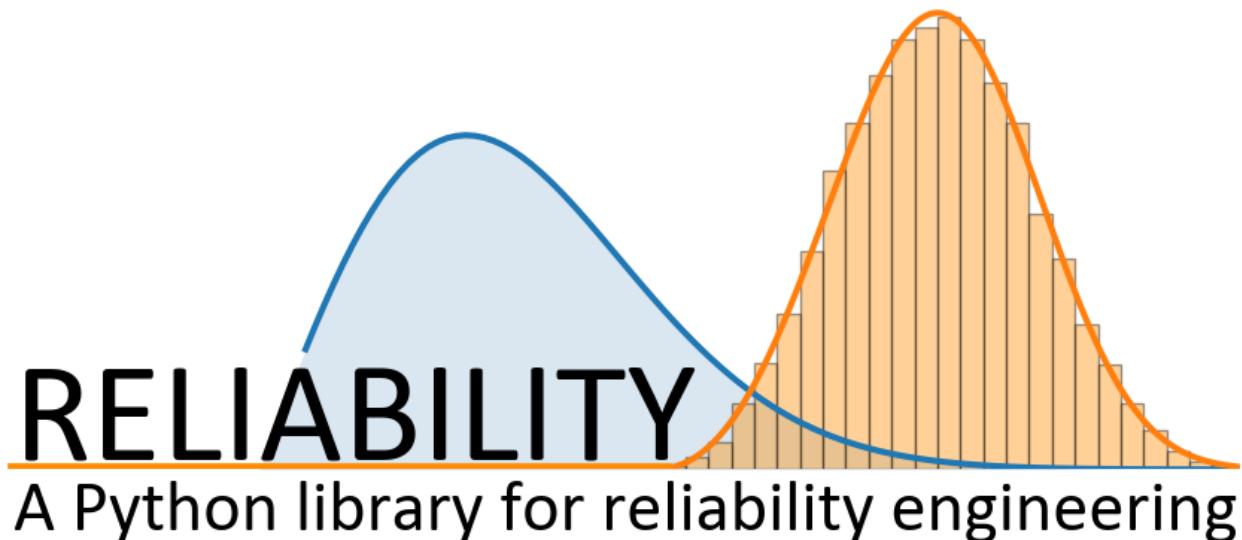
CONTRIBUTING

If you would like to see something added or an existing feature changed to make it better, please send me an email (alpha.reliability@gmail.com) and from there we can put together a plan on how to proceed. I greatly appreciate all help, even if it is just pointing out an error in the code or documentation. There are a large number of features currently identified for inclusion into this library so if you would like to contribute something but don't know what to help with, please email me and we can discuss the functions that are currently planned for development.

If you are proposing something new, it is helpful to include as much detail as possible, such as:

- a detailed explanation of what the new feature is and why you think it would be useful
- example Python code
- any references such as academic papers or textbooks which describe the necessary steps
- screenshots from other programs (such as Minitab, JMP Pro, Reliasoft) which show the feature in use

Remember to upgrade **reliability** to the newest version and double check that the requested feature is not in there, as the codebase is constantly being improved.



HOW TO GET HELP

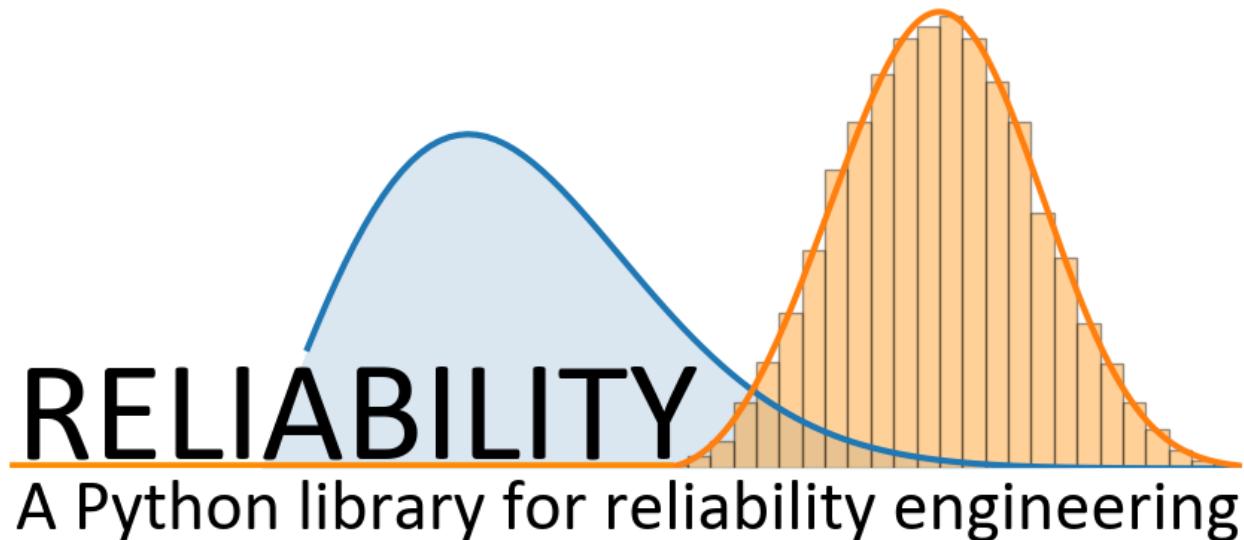
Questions on statistics or mathematics

If you have a question about a statistical or mathematical process that `reliability` performs, please consult Google and Wikipedia to see if you can find an existing explanation. If you still need to ask someone then I recommend asking your question on [Stack Exchange](#). If you still can't get an answer on there, you're welcome to email me directly (alpha.reliability@gmail.com) and I will try my best to help you.

Questions on using the Python `reliability` library

If you have a question about how to do something using `reliability` or how to use one of the features within `reliability` then you should firstly consult the [documentation](#) and the help files within Python. An example of how to access one of the help files is provided below. If the documentation and help files still do not answer your question then you're welcome to email me directly (alpha.reliability@gmail.com) and I will work to improve the documentation if it is unclear.

```
from reliability import Fitters
print(help(Fitters))
```



CHAPTER
SIXTYSEVEN

HOW TO DONATE TO THE PROJECT

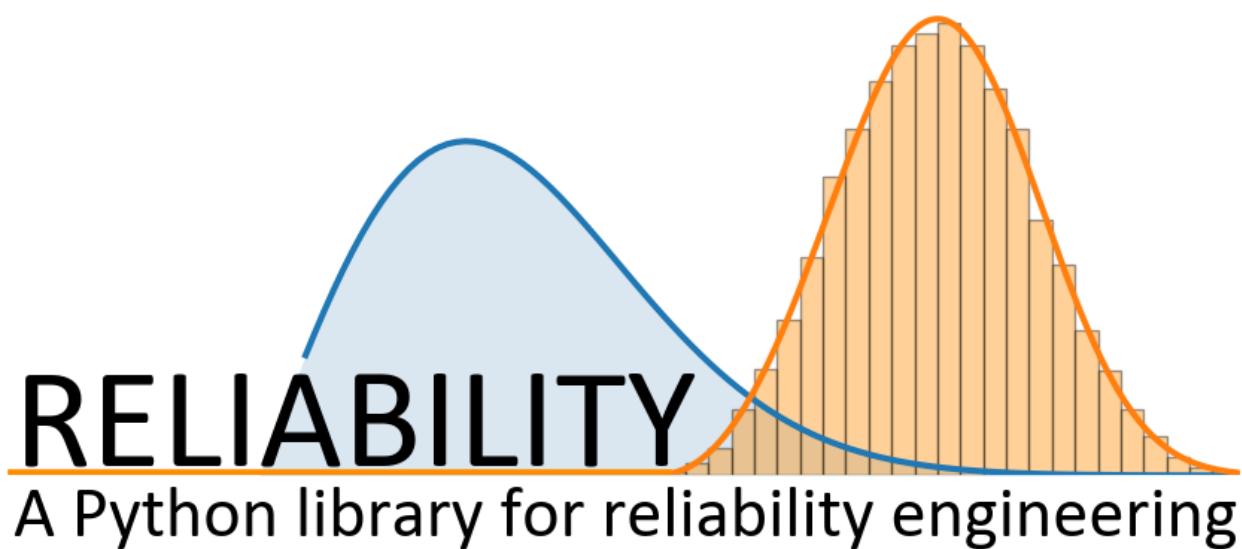
The Python reliability library is free and open source, and it always will be. It aims to provide students and professionals alike with a set of powerful tools to make reliability engineering more efficient. Many of these tools are otherwise only found in commercial software which is prohibitively expensive, especially for individuals. I began developing this library while I was a student because I found there was nothing like it available for free.

Developing and maintaining this library is all done in my spare time and is a very time consuming process. If you would like to donate as a way of showing your appreciation, you can send a one off donation using [Paypal](#) or a monthly donation by becoming a [GitHub Sponsor](#).

If you're wondering why you should pay for something that is already free, GitHub has a [great explanation](#) on the importance of investing in the software that powers your world.

Thank you to the following *reliability* donors for their generous support of open source development :)

- elriaral (GitHub user)
- Gabriel Felipe
- Juergen Leib
- Felix Nakovic
- Pedro Sa
- Augusto Mura



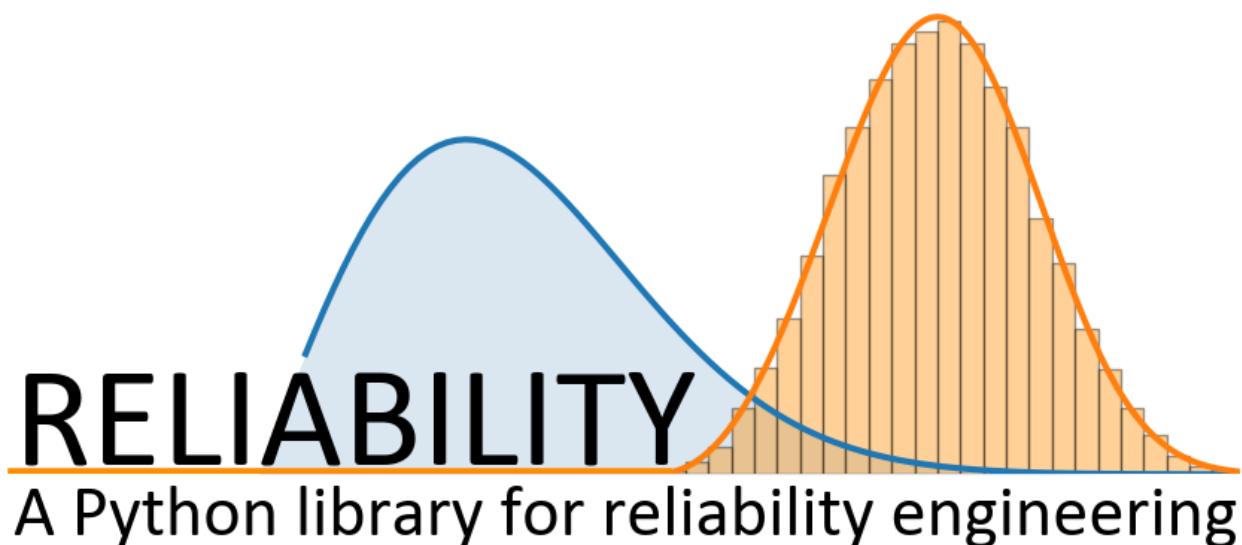
CHAPTER
SIXTYEIGHT

ABOUT THE AUTHOR

The Python reliability library was written by Matthew Reid. Matthew holds a Masters of Reliability Engineering from the University of Maryland, a Masters of Project Management from the University of New South Wales, and a Bachelor of Aeronautical Engineering from the University of New South Wales. Matthew lives in Canberra, Australia and currently works as an Aeronautical Engineer. If you would like to contact Matthew, you can send a message via [LinkedIn](#).

The Python reliability library was written because there were no dedicated reliability engineering libraries for Python, and Matthew found himself needing to use `scipy.stats`, `lifelines`, `Minitab`, `MATLAB`, `JMP Pro`, and his own Python scripts, for a variety of common reliability engineering tasks that were not available in one place. This library is intended to make reliability engineering more accessible to the world, particularly to those individuals and small businesses who find the high cost of proprietary software to be a barrier to entry into conducting reliability engineering analysis.

This is Matthew's first Python library on the Python Package Index and is currently in active development. In accordance with the [LGPLv3 license](#), every effort has been made to ensure the software is free of errors, however, no guarantees or warranties are provided in any form. Feedback on the Python reliability library is most welcome. If you find an error, have a suggestion, would like to request something to be included, or would like to contribute something, please send it through by email (alpha.reliability@gmail.com).



CHAPTER
SIXTYNINE

CREDITS TO THOSE WHO HELPED

During the process of writing *reliability* there have been many problems that I was unable to solve alone. I would like to thank the following people who provided help and feedback on problems with the code and with the reliability concepts:

- Cameron Davidson-Pilon for help with getting autograd to work to fit censored data and for writing `autograd-gamma` which makes it possible to fit the gamma and beta distributions. Also for providing help with obtaining the Fisher Information Matrix so that the confidence intervals for parameters could be estimated.
- Dr. Vasiliy Krivtsov for providing feedback on PP and QQ plots, for further explaining optimal replacement time equations, and for guidance in developing the Competing risk model. Dr. Krivtsov teaches “Collection and analysis of Reliability Data (ENRE640)” at the University of Maryland.
- Dr. Mohammad Modarres for help with PoF, ALT_fitters, and ALT_probability_plotting. Dr. Modarres teaches several reliability engineering subjects at the University of Maryland and has authored several of the textbooks listed under [recommended resources](#).
- The Stack Overflow user ImportanceOfBeingErnest for [this answer](#) that was necessary to get the probability plotting functions working correctly for Gamma and Beta distributions.
- Antony Lee for help in adapting parts of his `mplcursors` library into the crosshairs function in `reliability.Other_functions.crosshairs`
- Thomas Enzinger for help in improving the method of finding the area in stress-strength interference between any two distributions. Previously this was done using a monte-carlo method, but Thomas’ method is much more accurate and always consistent. This is incorporated in Version 0.5.0.
- Karthick Mani for help implementing the Loglogistic and Gumbel Distributions including implementation of these distributions in Fitters and Probability_plotting.
- Jake Sadie for identifying an error in the formula used for stress-strength interference of any two distributions. This error has been corrected in version 0.5.7.
- Ed Burrows for x10 speed improvement to `optimal_replacement_time` by using `numpy.vectorize`.
- Adam Gary for help incorporating the Crow-AMSAA model into the `reliability_growth` function.



RELIABILITY
A Python library for reliability engineering

CHAPTER
SEVENTY

LOGO

The logo for *reliability* can be created using the code below. The logo was generated using matplotlib version 3.3.3 and reliability version 0.5.5. The image produced requires subsequent cropping to remove surrounding white space.

```
from reliability.Distributions import Weibull_Distribution
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(10, 4))

# blue distribution
x_blue_fill = np.linspace(0, 19, 1000)
blue_dist = Weibull_Distribution(alpha=5.5, beta=2, gamma=0.63)
y_blue_fill = blue_dist.PDF(linewidth=3, xvals=x_blue_fill, show_plot=False)
plt.fill_between(
    x=x_blue_fill,
    y1=np.zeros_like(y_blue_fill),
    y2=y_blue_fill,
    color="steelblue",
    linewidth=0,
    alpha=0.2,
)
blue_dist.PDF(linewidth=3, xvals=np.linspace(1.5, 19, 100))

# orange distribution
orange_dist = Weibull_Distribution(alpha=6, beta=3.3, gamma=8)
x_orange = np.linspace(0, 19, 1000)
orange_dist.PDF(linewidth=3, xvals=x_orange)
plt.plot([-4, orange_dist.gamma + 0.27], [0, 0], linewidth=5.5, color="darkorange")

# orange histogram
samples = orange_dist.random_samples(20000, seed=3)
plt.hist(
    x=samples[samples < max(x_orange)],
    density=True,
    alpha=0.4,
    color="darkorange",
    bins=25,
    edgecolor="k",
)
```

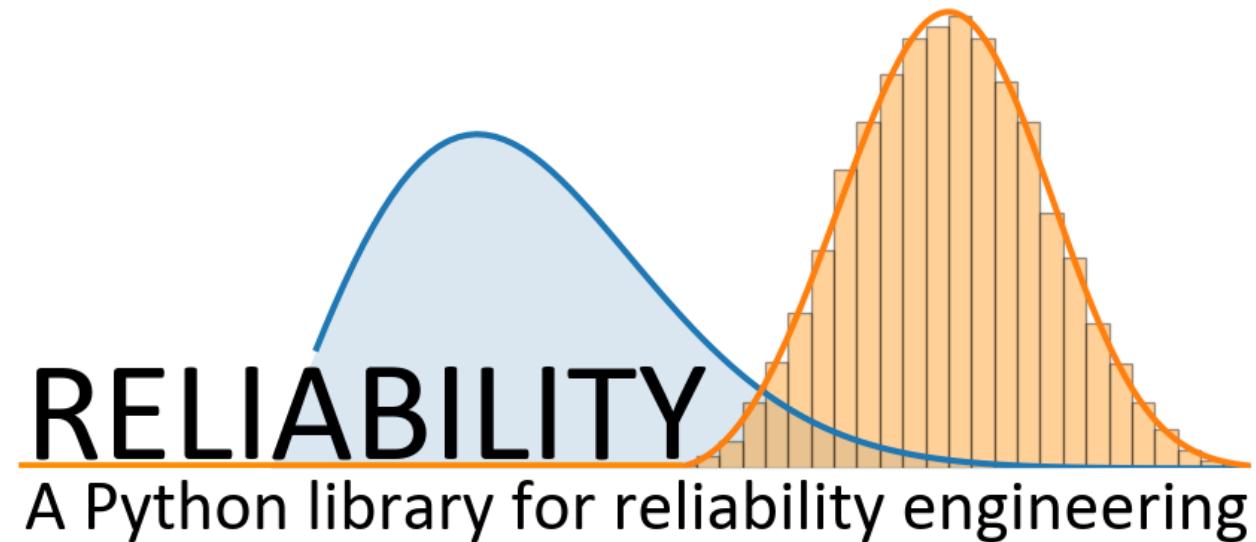
(continues on next page)

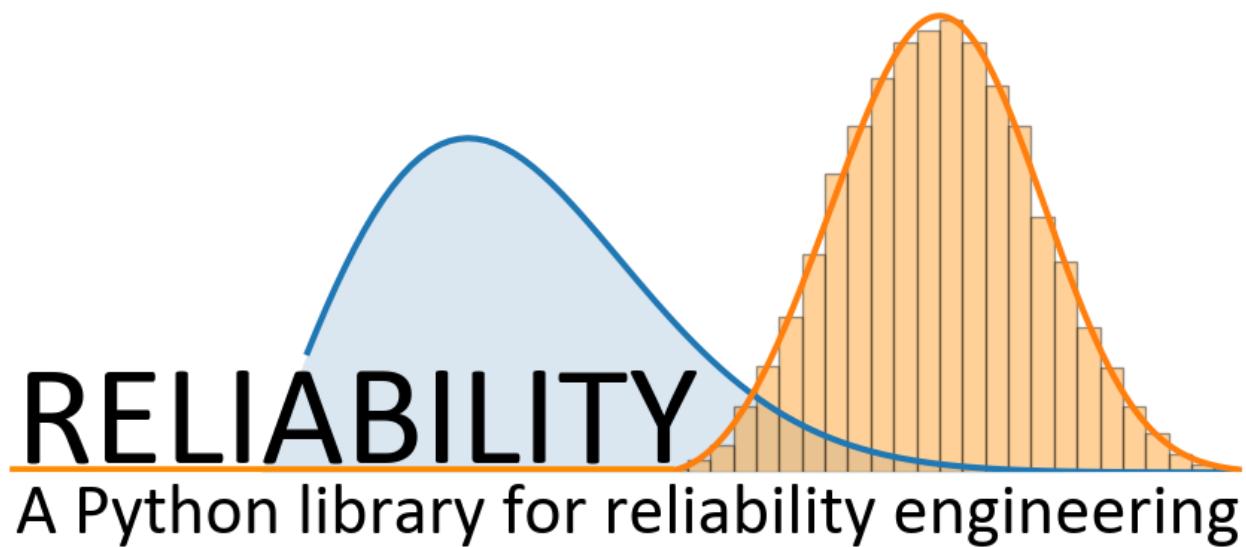
(continued from previous page)

```
# text objects
plt.text(x=-4, y=0.005, s="RELIABILITY", size=70, fontname="Calibri")
plt.text(
    x=-4,
    y=-0.005,
    va="top",
    s="A Python library for reliability engineering",
    size=34.85,
    fontname="Calibri",
)

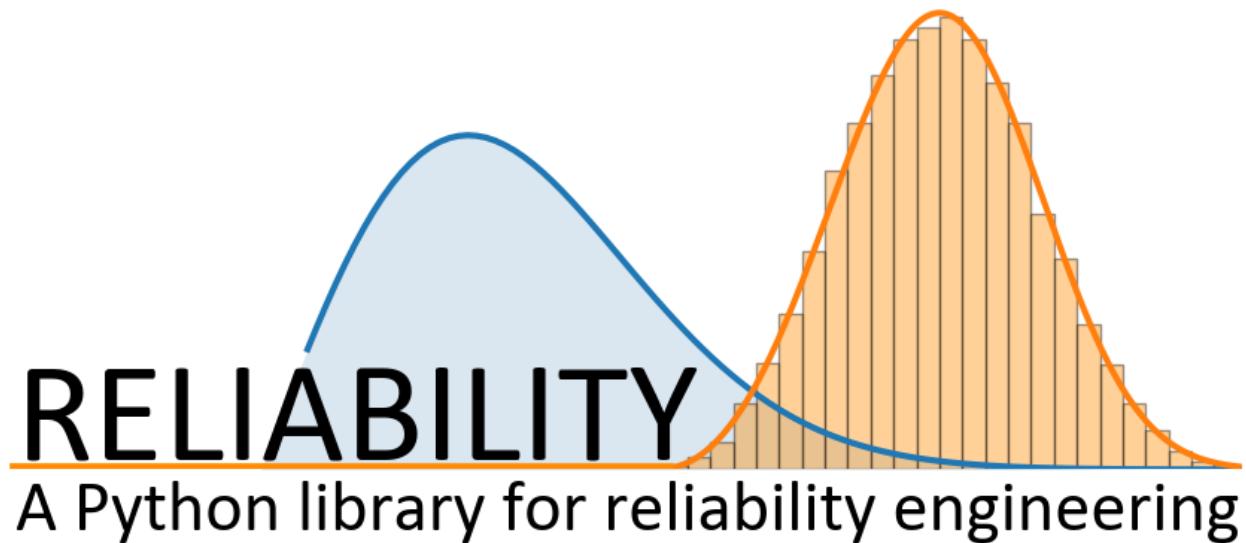
plt.xlim(-5, 20)
plt.title("")
plt.axis("off")
plt.tight_layout()
plt.show()
```

If you have any suggestions for future versions of this logo, please send them through by email to alpha.reliability@gmail.com





71.1 ALT_fitters



71.1.1 Fit_Everything_ALT

```
class reliability. ALT_fitters.Fit_Everything_ALT(failures, failure_stress_1=None,  
                                                failure_stress_2=None, right_censored=None,  
                                                right_censored_stress_1=None,  
                                                right_censored_stress_2=None,  
                                                use_level_stress=None, CI=0.95, optimizer=None,  
                                                show_probability_plot=True,  
                                                show_best_distribution_probability_plot=True,  
                                                print_results=True, exclude=None, sort_by='BIC',  
                                                **kwargs)
```

This function will fit all available ALT models for the data you enter, which may include right censored data.

ALT models are either single stress (Exponential, Eyring, Power) or dual stress (Dual_Exponential, Power_Exponential, Dual_Power).

Depending on the data you enter (ie. whether failure_stress_2 is provided), the applicable set of ALT models will be fitted.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list, optional*) – The corresponding stresses (such as temperature or voltage) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress. Alternative keyword of failure_stress is accepted in place of failure_stress_1.
- **failure_stress_2** (*array, list, optional*) – The corresponding stresses (such as temperature or voltage) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress. Optional input. Providing this will trigger the use of dual stress models. Leaving this empty will trigger the use of single stress models.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stresses (such as temperature or voltage) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided. Alternative keyword of right_censored_stress is accepted in place of right_censored_stress_1.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stresses (such as temperature or voltage) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if failure_stress_2 is provided.
- **use_level_stress** (*int, float, list, array, optional*) – The use level stress at which you want to know the mean life. Optional input. This must be a list or array [stress_1, stress_2] if failure_stress_2 is provided and you want to know the mean life.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, optional*) – True/False. Default is True. Provides a probability plot of each of the fitted ALT model.
- **show_best_distribution_probability_plot** (*bool, optional*) – True/False. Defaults to True. Provides a probability plot in a new figure of the best ALT model.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.

- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **sort_by** (*str, optional*) – Goodness of fit test to sort results by. Must be ‘BIC’, ‘AICc’, or ‘Log-likelihood’. Default is ‘BIC’.
- **exclude** (*list, array, optional*) – A list or array of strings specifying which distributions to exclude. Default is None. Options are: Weibull_Exponential, Weibull_Eyring, Weibull_Power, Weibull_Dual_Exponential, Weibull_Power_Exponential, Weibull_Dual_Power, Lognormal_Exponential, Lognormal_Eyring, Lognormal_Power, Lognormal_Dual_Exponential, Lognormal_Power_Exponential, Lognormal_Dual_Power, Normal_Exponential, Normal_Eyring, Normal_Power, Normal_Dual_Exponential, Normal_Power_Exponential, Normal_Dual_Power, Exponential_Exponential, Exponential_Eyring, Exponential_Power, Exponential_Dual_Exponential, Exponential_Power_Exponential, Exponential_Dual_Power
- **kwargs** – Accepts failure_stress and right_censored_stress as alternative keywords to failure_stress_1 and right_censored_stress_1. This is used to provide consistency with the other functions in ALT_Fitters which also accept failure_stress and right_censored_stress.

Returns

- **results** (*dataframe*) – The dataframe of results. Fitted parameters in this dataframe may be accessed by name. See below example.
- **best_model_name** (*str*) – The name of the best fitting ALT model. E.g. ‘Weibull_Exponential’. See above list for exclude.
- **best_model_at_use_stress** (*object*) – A distribution object created based on the parameters of the best fitting ALT model at the use stress. This is only provided if the use_level_stress is provided. This is because use_level_stress is required to find the scale parameter.
- **parameters and goodness of fit results** (*float*) – This is provided for each fitted model. For example, the Weibull_Exponential model values are Weibull_Exponential_a, Weibull_Exponential_b, Weibull_Exponential_beta, Weibull_Exponential_BIC, Weibull_Exponential_AICc, Weibull_Exponential_loglik
- **excluded_models** (*list*) – A list of the models which were excluded. This will always include at least half the models since only single stress OR dual stress can be fitted depending on the data.
- **probability_plot** (*object*) – The figure handle from the probability plot (only provided if show_probability_plot is True).
- **best_distribution_probability_plot** (*object*) – The figure handle from the best distribution probability plot (only provided if show_best_distribution_probability_plot is True).

Notes

From the results, the models are sorted based on their goodness of fit test results, where the smaller the goodness of fit value, the better the fit of the model to the data.

Example Usage:

```
failures = [619, 417, 173, 161, 1016, 512, 999, 1131, 1883, 2413, 3105, 2492]
failure_stresses = [500, 500, 500, 500, 400, 400, 400, 350, 350, 350, 350]
(continues on next page)
```

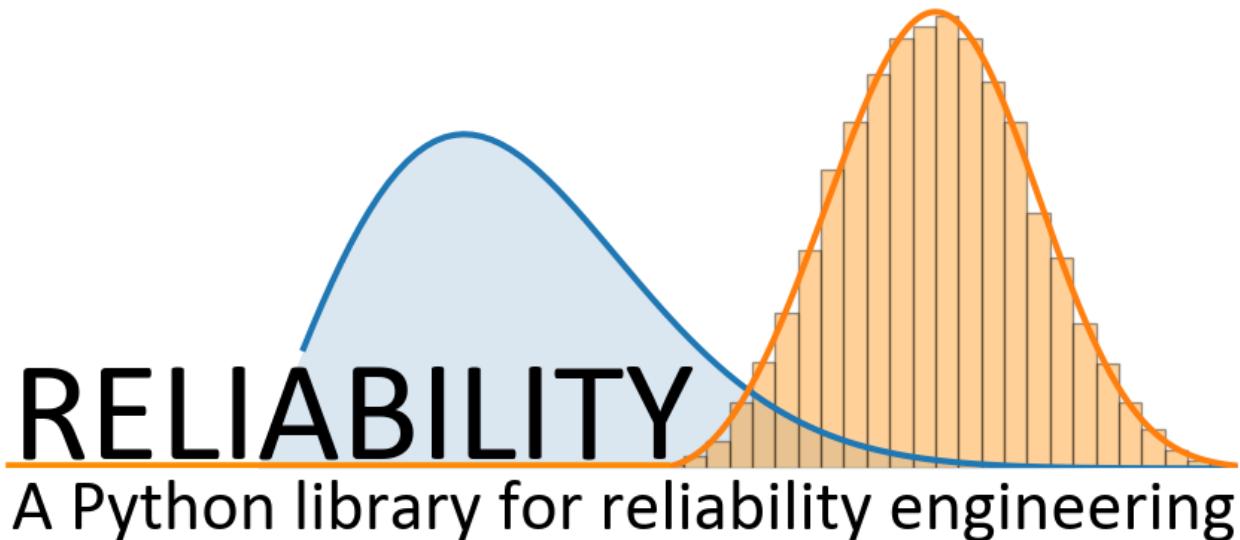
(continued from previous page)

```

right_censored = [29, 180, 1341]
right_censored_stresses = [500, 400, 350]
use_level_stress = 300
output = Fit_Everything_ALT(failures=failures,failure_stress_1=failure_stresses,
                           right_censored=right_censored, right_censored_stress_1=right_censored_stresses,
                           use_level_stress=use_level_stress)

# To extract the parameters of the Weibull_Exponential model from the results
# dataframe, you may access the parameters by name:
print('Weibull Exponential beta =',output.Weibull_Exponential_beta)
>>> Weibull Exponential beta = 3.0807072337386123

```



71.1.2 Fit_Exponential_Dual_Exponential

```

class reliability. ALT_fitters.Fit_Exponential_Dual_Exponential(failures, failure_stress_1,
                                                               failure_stress_2,
                                                               right_censored=None,
                                                               right_censored_stress_1=None,
                                                               right_censored_stress_2=None,
                                                               use_level_stress=None, CI=0.95,
                                                               optimizer=None,
                                                               show_probability_plot=True,
                                                               show_life_stress_plot=True,
                                                               print_results=True)

```

This function will Fit the Exponential_Dual_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two thermal stresses (such as temperature-humidity). It is recommended that you ensure your temperature data are in Kelvin and humidity data range from 0 to 1.

Parameters

- **failures** (*array, list*) – The failure data.

- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as humidity) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as humidity) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

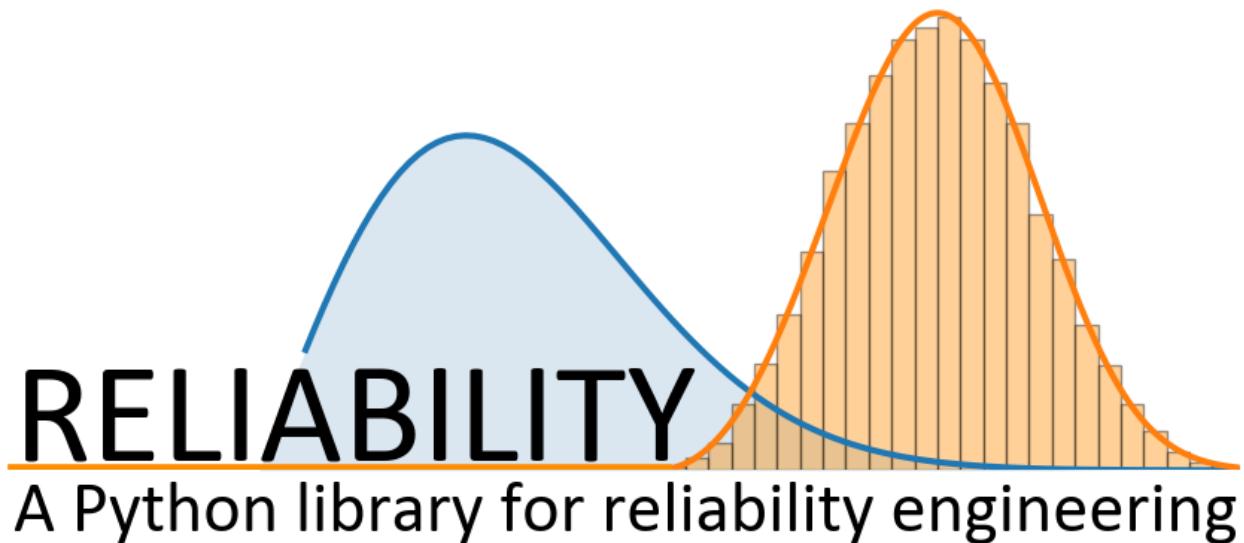
- **a** (*float*) – The fitted parameter from the Dual_Exponential model
- **b** (*float*) – The fitted parameter from the Dual_Exponential model
- **c** (*float*) – The fitted parameter from the Dual_Exponential model
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter

- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (equivalent Weibull alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **Lambda_at_use_stress** (*float*) – The equivalent Exponential Lambda parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Exponential distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
```

```
static logR(t, S1, S2, a, b, c)
```

```
static logf(t, S1, S2, a, b, c)
```



71.1.3 Fit_Exponential_Dual_Power

```
class reliability. ALT_fitters.Fit_Exponential_Dual_Power(failures, failure_stress_1, failure_stress_2,
                                                       right_censored=None,
                                                       right_censored_stress_1=None,
                                                       right_censored_stress_2=None,
                                                       use_level_stress=None, CI=0.95,
                                                       optimizer=None,
                                                       show_probability_plot=True,
                                                       show_life_stress_plot=True,
                                                       print_results=True)
```

This function will Fit the Exponential_Dual_Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two non-thermal stresses such as voltage and load.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as voltage) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as load) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as voltage) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as load) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

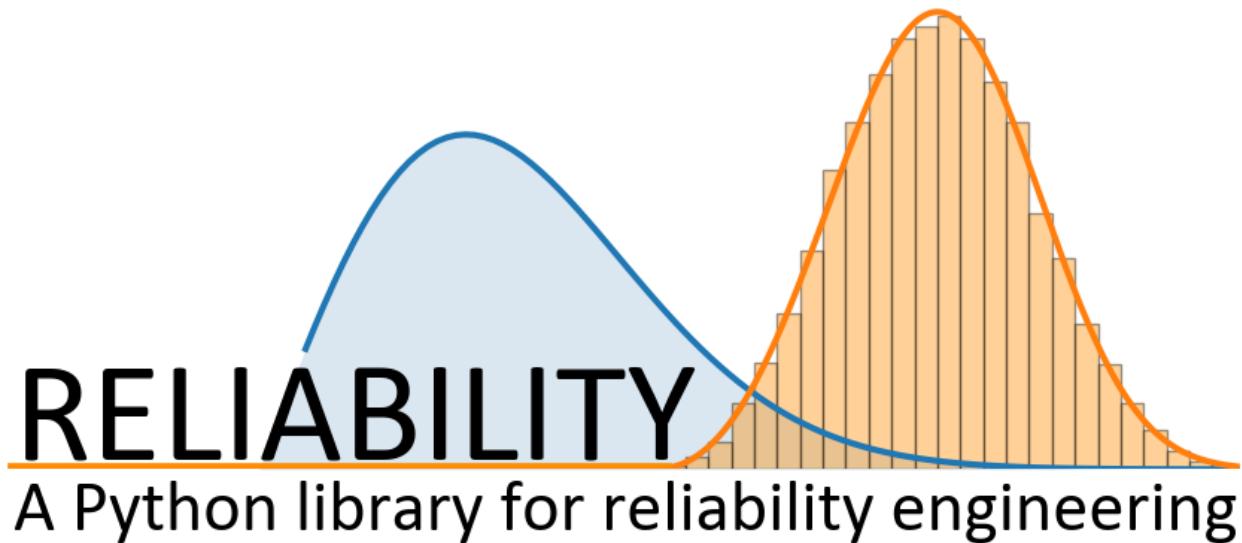
Returns

- **c** (*float*) – The fitted parameter from the Dual_Power model
- **n** (*float*) – The fitted parameter from the Dual_Power model
- **m** (*float*) – The fitted parameter from the Dual_Power model
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **m_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **m_upper** (*float*) – The upper CI estimate of the parameter
- **m_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (equivalent Weibull alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **Lambda_at_use_stress** (*float*) – The equivalent Exponential Lambda parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Exponential distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

static **LL**(*params*, *t_f*, *t_rc*, *S1_f*, *S2_f*, *S1_rc*, *S2_rc*)

static **logR**(*t*, *S1*, *S2*, *c*, *m*, *n*)

static **logf**(*t*, *S1*, *S2*, *c*, *m*, *n*)



71.1.4 Fit_Exponential_Exponential

```
class reliability. ALT_fitters.Fit_Exponential_Exponential(failures, failure_stress,
                                                               right_censored=None,
                                                               right_censored_stress=None,
                                                               use_level_stress=None, CI=0.95,
                                                               optimizer=None,
                                                               show_probability_plot=True,
                                                               show_life_stress_plot=True,
                                                               print_results=True)
```

This function will fit the Exponential-Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

If you are using this model for the Arrhenius equation, $a = Ea/K_B$. When results are printed Ea will be provided in eV.

Parameters

- **failures** (array, list) – The failure data.
- **failure_stress** (array, list) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (array, list, optional) – The right censored failure times. Optional input.
- **right_censored_stress** (array, list, optional) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (int, float, optional) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (bool, optional) – True/False. Default is True. Prints the results to the console.

- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

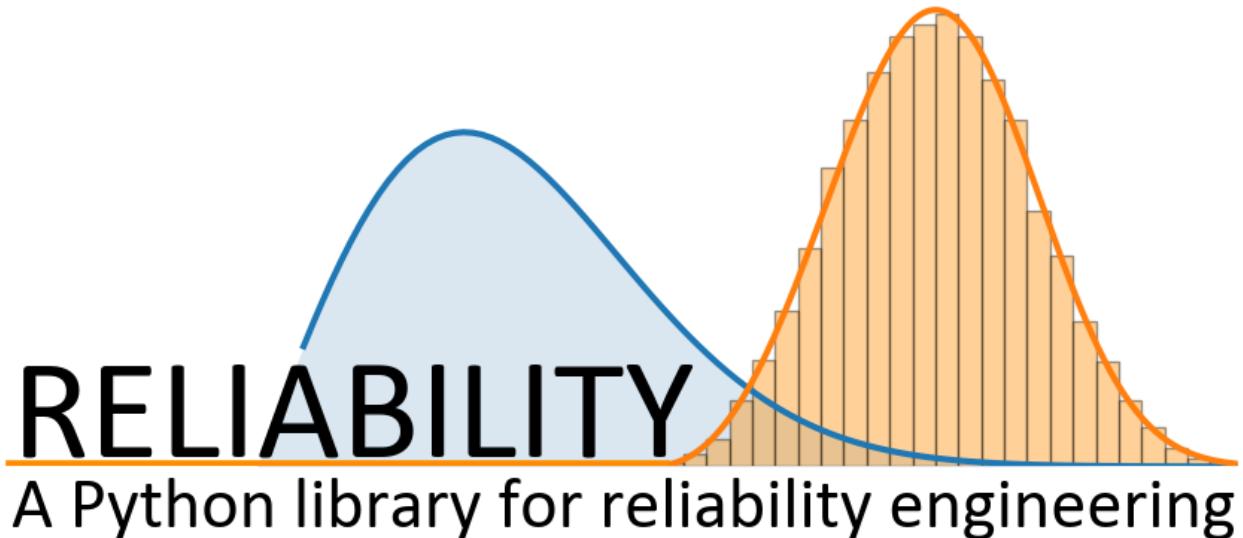
Returns

- **a** (*float*) – The fitted parameter from the Exponential model
- **b** (*float*) – The fitted parameter from the Exponential model
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (equivalent Weibull alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **Lambda_at_use_stress** (*float*) – The equivalent Exponential Lambda parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Exponential distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```

static LL(params, t_f, t_rc, T_f, T_rc)
static logR(t, T, a, b)
static logf(t, T, a, b)

```



71.1.5 Fit_Exponential_Eyring

```

class reliability. ALT_fitters.Fit_Exponential_Eyring(failures, failure_stress, right_censored=None,
                                                       right_censored_stress=None,
                                                       use_level_stress=None, CI=0.95,
                                                       optimizer=None, show_probability_plot=True,
                                                       show_life_stress_plot=True,
                                                       print_results=True)

```

This function will fit the Exponential-Eyring life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.

- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

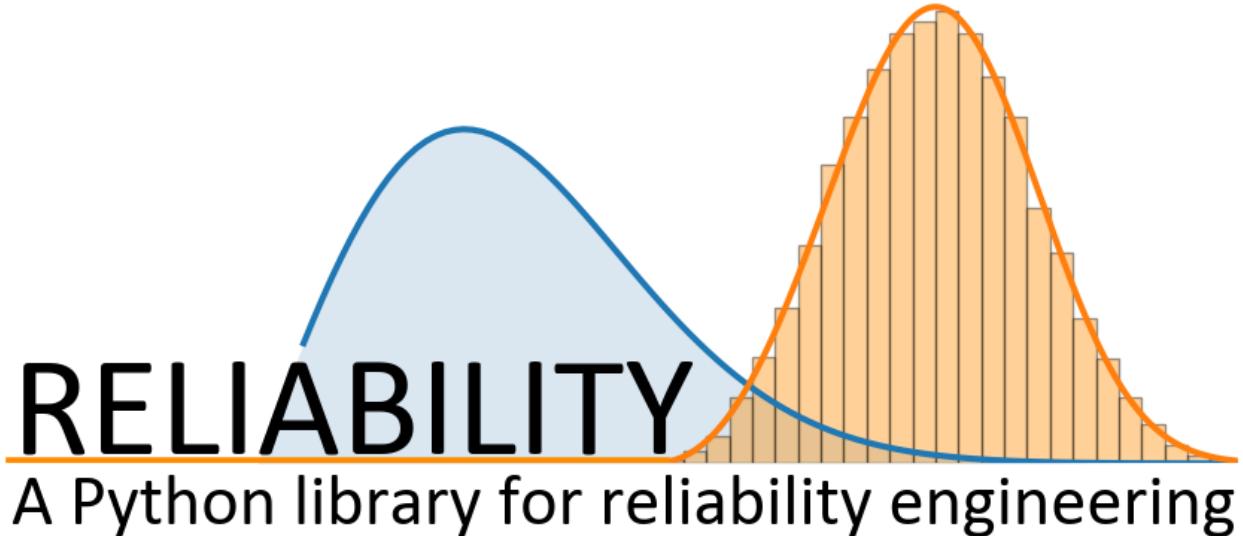
Returns

- **a** (*float*) – The fitted parameter from the Exponential model
- **c** (*float*) – The fitted parameter from the Exponential model
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (equivalent Weibull alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **Lambda_at_use_stress** (*float*) – The equivalent Exponential Lambda parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Exponential distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```

static LL(params, t_f, t_rc, T_f, T_rc)
static logR(t, T, a, c)
static logf(t, T, a, c)

```



71.1.6 Fit_Exponential_Power

```

class reliability.ALT_fitters.Fit_Exponential_Power(failures, failure_stress, right_censored=None,
right_censored_stress=None,
use_level_stress=None, CI=0.95,
optimizer=None, show_probability_plot=True,
show_life_stress_plot=True, print_results=True)

```

This function will Fit the Exponential-Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with non-thermal stresses (typically in fatigue applications).

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as load) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as load) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.

- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

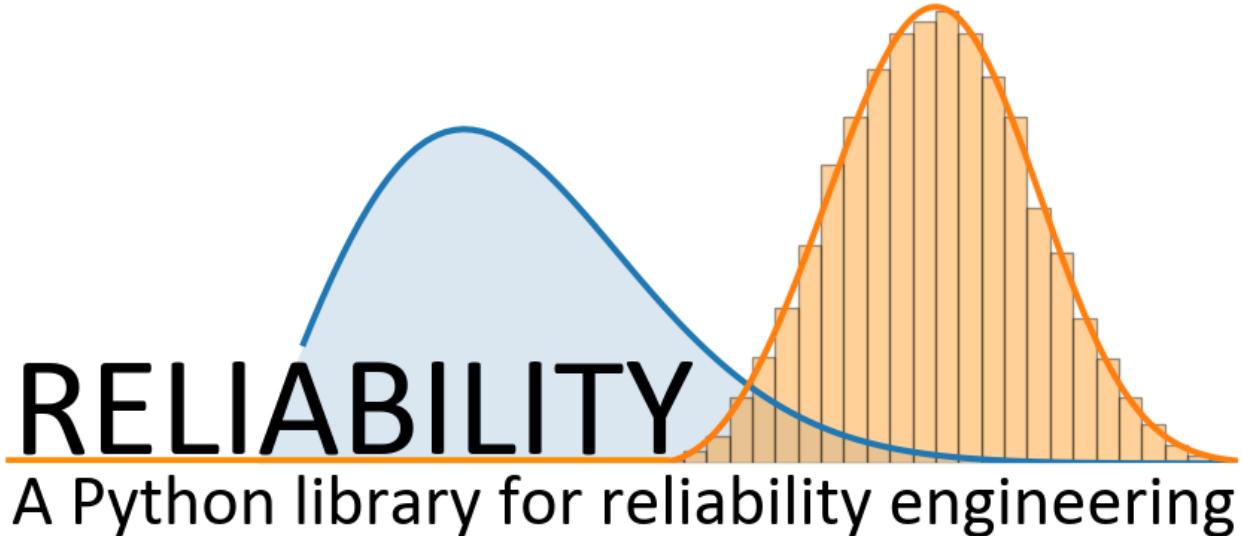
Returns

- **a** (*float*) – The fitted parameter from the Power model
- **n** (*float*) – The fitted parameter from the Power model
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (equivalent Weibull alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **Lambda_at_use_stress** (*float*) – The equivalent Exponential Lambda parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Exponential distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```

static LL(params, t_f, t_rc, T_f, T_rc)
static logR(t, T, a, n)
static logf(t, T, a, n)

```



71.1.7 Fit_Exponential_Power_Exponential

```

class reliability. ALT_fitters. Fit_Exponential_Power_Exponential(failures, failure_stress_1,
                                                               failure_stress_2,
                                                               right_censored=None,
                                                               right_censored_stress_1=None,
                                                               right_censored_stress_2=None,
                                                               use_level_stress=None, CI=0.95,
                                                               optimizer=None,
                                                               show_probability_plot=True,
                                                               show_life_stress_plot=True,
                                                               print_results=True)

```

This function will Fit the Exponential_Power_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with thermal and non-thermal stresses. It is essential that you ensure your thermal stress is stress_1 (as it will be modeled by the Exponential) and your non-thermal stress is stress_2 (as it will be modeled by the Power). Also ensure that your temperature data are in Kelvin.

Parameters

- **failures** (*array*, *list*) – The failure data.
- **failure_stress_1** (*array*, *list*) – The corresponding stress 1 (thermal stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array*, *list*) – The corresponding stress 2 (non-thermal stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.

- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (non-thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

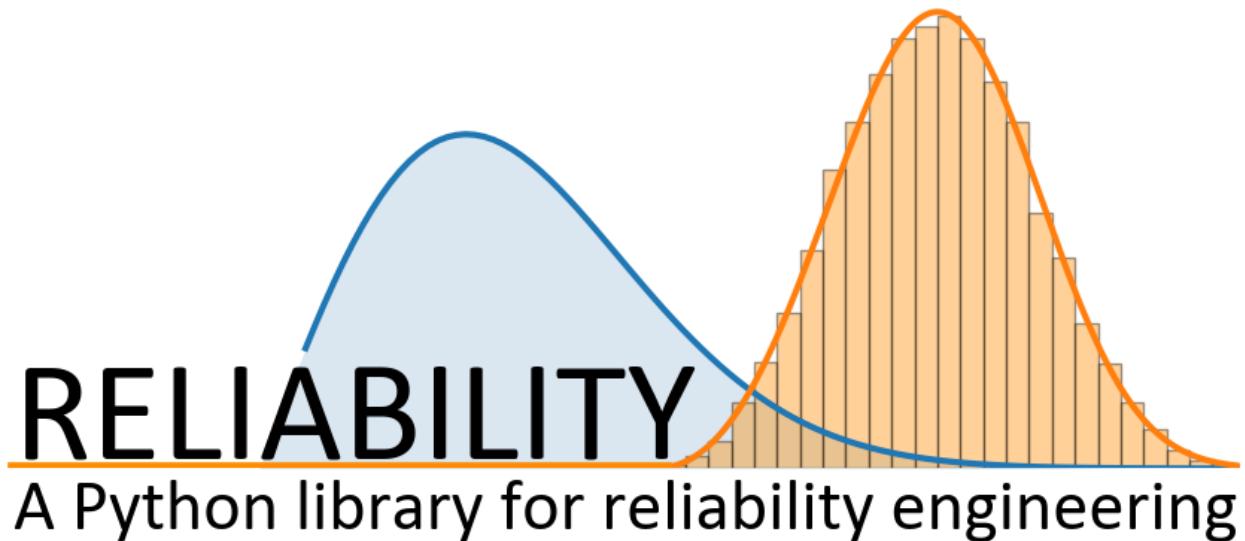
- **a** (*float*) – The fitted parameter from the Power_Exponential model
- **c** (*float*) – The fitted parameter from the Power_Exponential model
- **n** (*float*) – The fitted parameter from the Power_Exponential model
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter

- **n_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (equivalent Weibull alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **Lambda_at_use_stress** (*float*) – The equivalent Exponential Lambda parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Exponential distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)

static logR(t, S1, S2, a, c, n)

static logf(t, S1, S2, a, c, n)
```



71.1.8 Fit_Lognormal_Dual_Exponential

```
class reliability.ALT_fitters.Fit_Lognormal_Dual_Exponential(failures, failure_stress_1,
                                                               failure_stress_2,
                                                               right_censored=None,
                                                               right_censored_stress_1=None,
                                                               right_censored_stress_2=None,
                                                               use_level_stress=None, CI=0.95,
                                                               optimizer=None,
                                                               show_probability_plot=True,
                                                               show_life_stress_plot=True,
                                                               print_results=True)
```

This function will Fit the Lognormal_Dual_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two thermal stresses (such as temperature-humidity). It is recommended that you ensure your temperature data are in Kelvin and humidity data range from 0 to 1.

Parameters

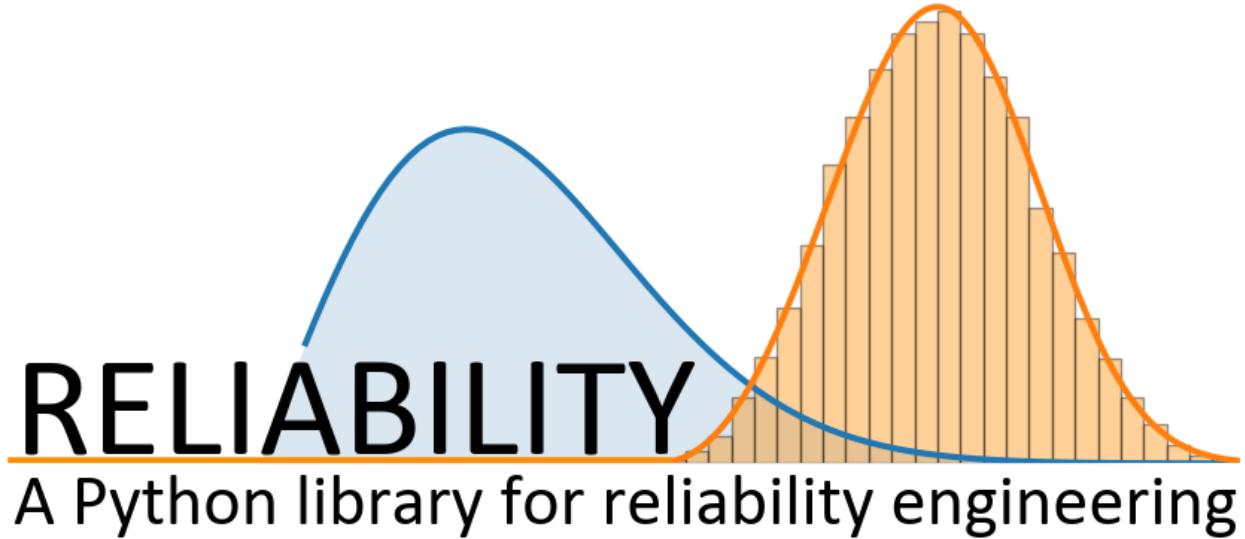
- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as humidity) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as humidity) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’,

‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

- **a** (*float*) – The fitted parameter from the Dual_Exponential model
- **b** (*float*) – The fitted parameter from the Dual_Exponential model
- **c** (*float*) – The fitted parameter from the Dual_Exponential model
- **sigma** (*float*) – The fitted Lognormal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Lognormal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Lognormal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
static logR(t, S1, S2, a, b, c, sigma)
static logf(t, S1, S2, a, b, c, sigma)
```



71.1.9 Fit_Lognormal_Dual_Power

```
class reliability. ALT_fitters.Fit_Lognormal_Dual_Power(failures, failure_stress_1, failure_stress_2,
                                                       right_censored=None,
                                                       right_censored_stress_1=None,
                                                       right_censored_stress_2=None,
                                                       use_level_stress=None, CI=0.95,
                                                       optimizer=None,
                                                       show_probability_plot=True,
                                                       show_life_stress_plot=True,
                                                       print_results=True)
```

This function will Fit the Lognormal_Dual_Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two non-thermal stresses such as voltage and load.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as voltage) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as load) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as voltage) at which each right_censored data point was obtained. This must match the length of

right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.

- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as load) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

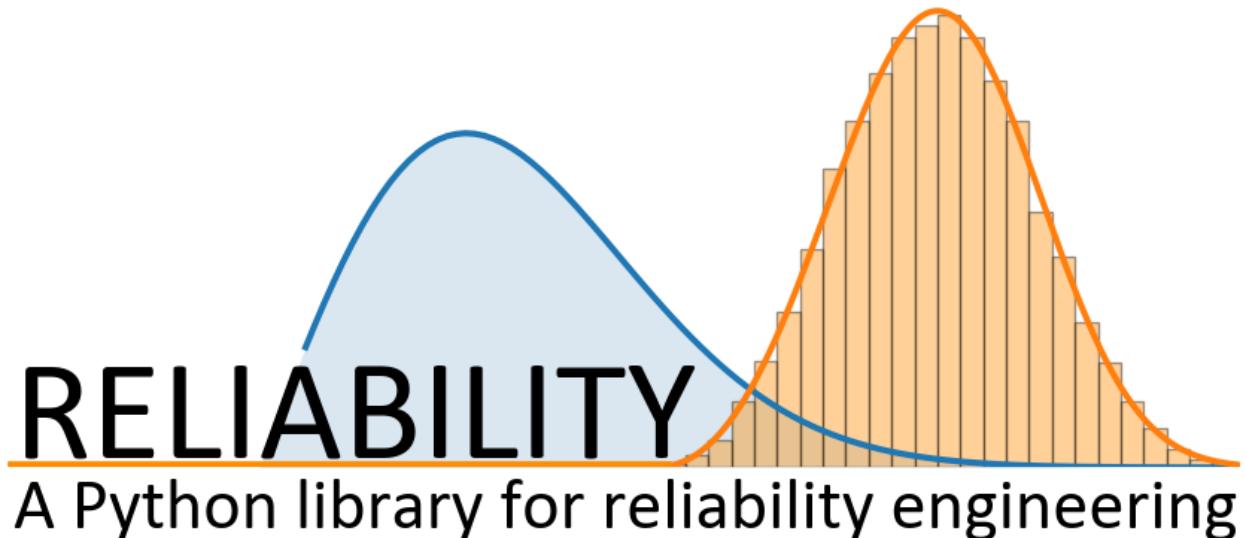
- **c** (*float*) – The fitted parameter from the Dual_Power model
- **m** (*float*) – The fitted parameter from the Dual_Power model
- **n** (*float*) – The fitted parameter from the Dual_Power model
- **sigma** (*float*) – The fitted Lognormal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **m_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **m_upper** (*float*) – The upper CI estimate of the parameter
- **m_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter

- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Lognormal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Lognormal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
```

```
static logR(t, S1, S2, c, m, n, sigma)
```

```
static logf(t, S1, S2, c, m, n, sigma)
```



71.1.10 Fit_Lognormal_Exponential

```
class reliability. ALT_fitters.Fit_Lognormal_Exponential(failures, failure_stress,
                                                       right_censored=None,
                                                       right_censored_stress=None,
                                                       use_level_stress=None, CI=0.95,
                                                       optimizer=None,
                                                       show_probability_plot=True,
                                                       show_life_stress_plot=True,
                                                       print_results=True)
```

This function will Fit the Lognormal-Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

If you are using this model for the Arrhenius equation, $a = Ea/K_B$. When results are printed Ea will be provided in eV.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

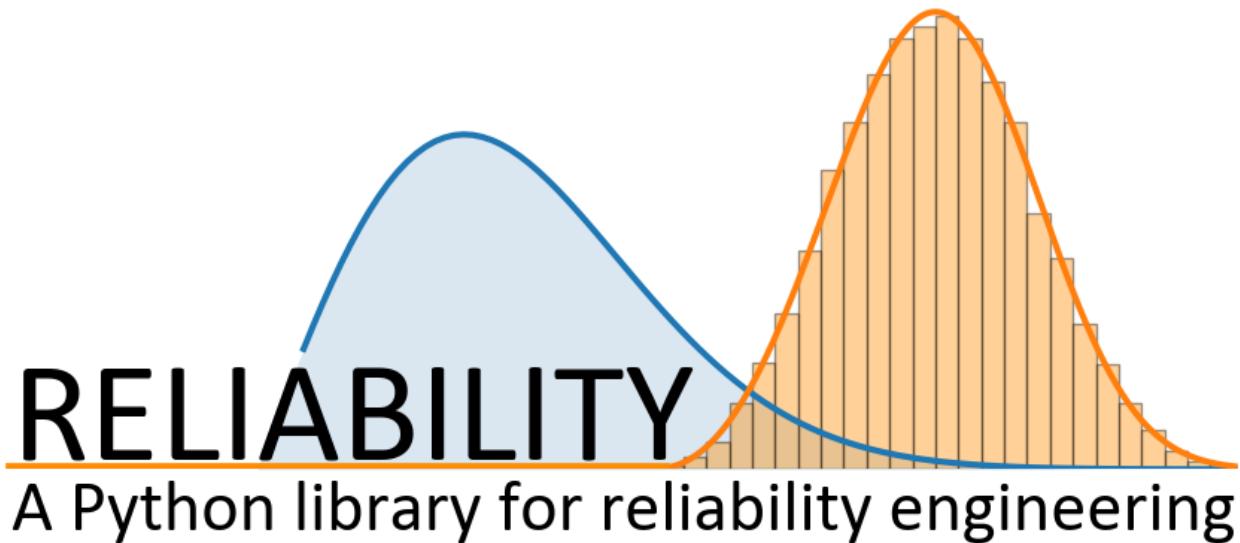
- **a** (*float*) – The fitted parameter from the Exponential model
- **b** (*float*) – The fitted parameter from the Exponential model
- **sigma** (*float*) – The fitted Lognormal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)

- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Lognormal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Lognormal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)

static logR(t, T, a, b, sigma)

static logf(t, T, a, b, sigma)
```



71.1.11 Fit_Lognormal_Eyring

```
class reliability. ALT_fitters.Fit_Lognormal_Eyring(failures, failure_stress, right_censored=None, right_censored_stress=None, use_level_stress=None, CI=0.95, optimizer=None, show_probability_plot=True, show_life_stress_plot=True, print_results=True)
```

This function will Fit the Lognormal-Eyring life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.

- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

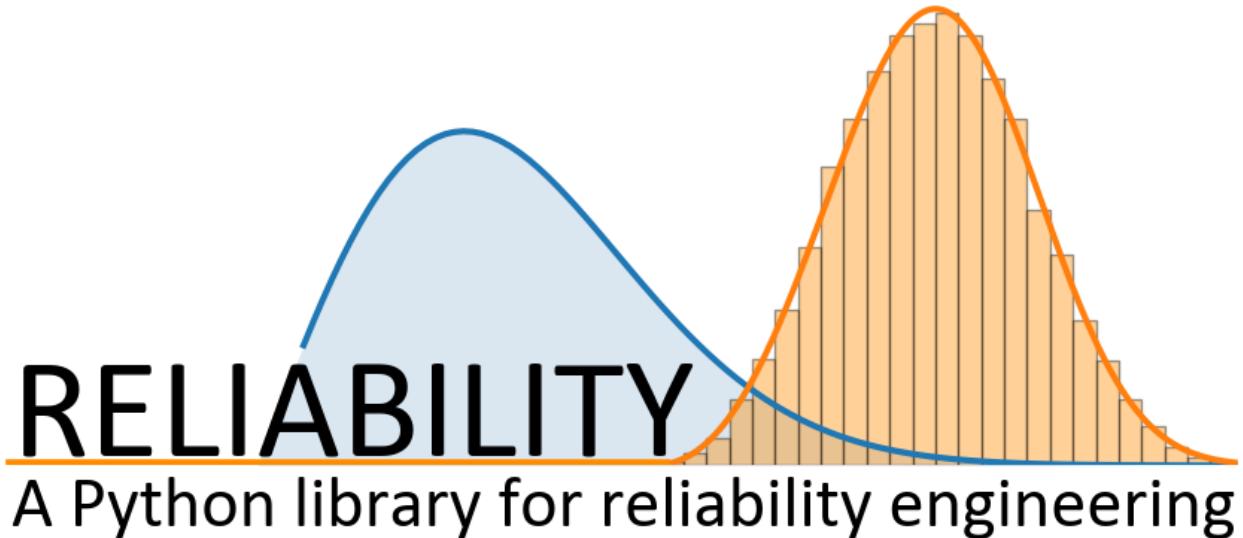
Returns

- **a** (*float*) – The fitted parameter from the Eyring model
- **c** (*float*) – The fitted parameter from the Eyring model
- **sigma** (*float*) – The fitted Lognormal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Lognormal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Lognormal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```

static LL(params, t_f, t_rc, T_f, T_rc)
static logR(t, T, a, c, sigma)
static logf(t, T, a, c, sigma)

```



71.1.12 Fit_Lognormal_Power

```

class reliability.ALT_fitters.Fit_Lognormal_Power(failures, failure_stress, right_censored=None,
                                                 right_censored_stress=None,
                                                 use_level_stress=None, CI=0.95, optimizer=None,
                                                 show_probability_plot=True,
                                                 show_life_stress_plot=True, print_results=True)

```

This function will Fit the Lognormal-Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with non-thermal stresses (typically in fatigue applications).

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as load) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as load) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.

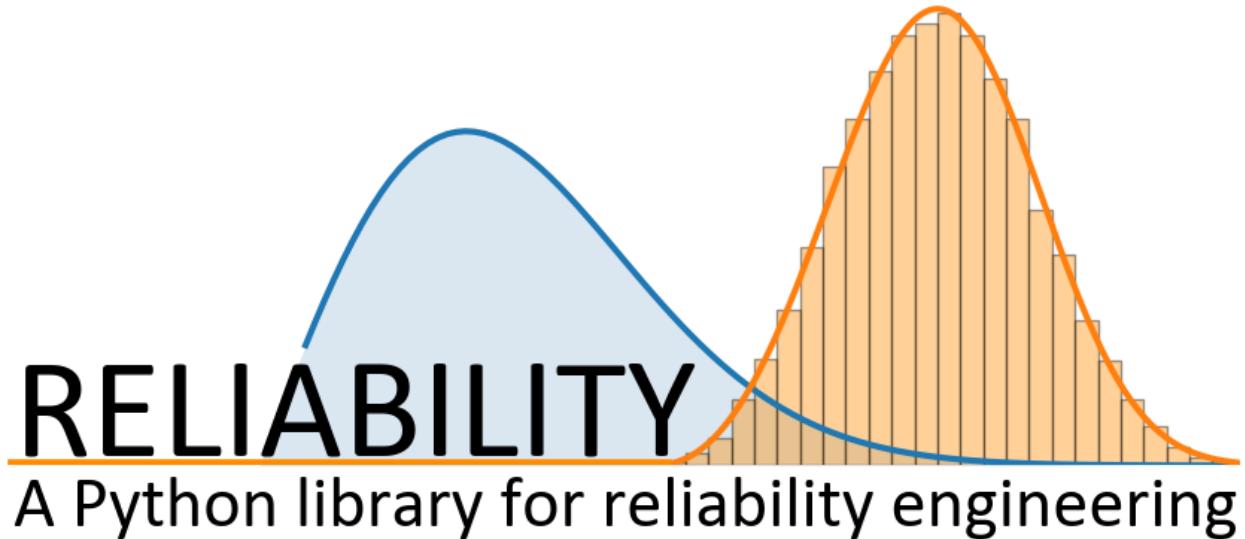
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

- **a** (*float*) – The fitted parameter from the Power model
- **n** (*float*) – The fitted parameter from the Power model
- **sigma** (*float*) – The fitted Lognormal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Lognormal mu parameter at the use level stress (only provided if use_level_stress is provided).

- **distribution_at_use_stress** (*object*) – The Lognormal distribution at the use level stress (only provided if `use_level_stress` is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if `show_probability_plot` is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if `show_life_stress_plot` is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)
static logR(t, T, a, n, sigma)
static logf(t, T, a, n, sigma)
```



71.1.13 Fit_Lognormal_Power_Exponential

```
class reliability. ALT_fitters.Fit_Lognormal_Power_Exponential(failures, failure_stress_1,
                                                               failure_stress_2,
                                                               right_censored=None,
                                                               right_censored_stress_1=None,
                                                               right_censored_stress_2=None,
                                                               use_level_stress=None, CI=0.95,
                                                               optimizer=None,
                                                               show_probability_plot=True,
                                                               show_life_stress_plot=True,
                                                               print_results=True)
```

This function will Fit the Lognormal_Power_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with thermal and non-thermal stresses. It is essential that you ensure your thermal stress is `stress_1` (as this will be modeled by the Exponential) and your non-thermal stress is `stress_2` (as this will be modeled by the Power). Also ensure that your temperature data are in Kelvin.

Parameters

- **failures** (*array, list*) – The failure data.

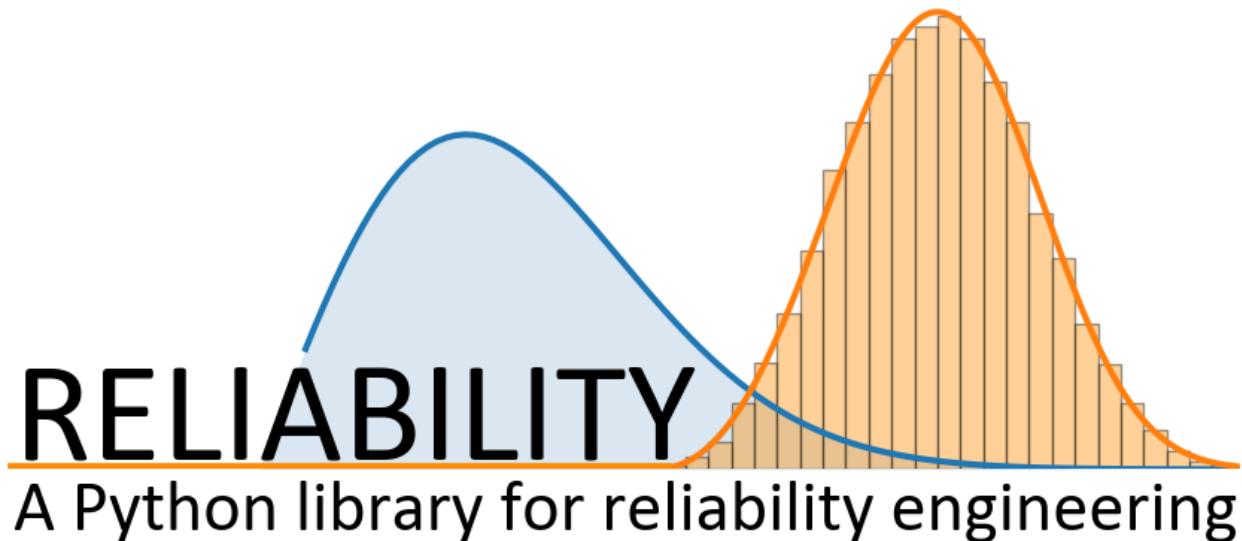
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (thermal-stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (non-thermal stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (non-thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

- **a** (*float*) – The fitted parameter from the Power_Exponential model
- **c** (*float*) – The fitted parameter from the Power_Exponential model
- **n** (*float*) – The fitted parameter from the Power_Exponential model
- **sigma** (*float*) – The fitted Lognormal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter

- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Lognormal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Lognormal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
static logR(t, S1, S2, a, c, n, sigma)
static logf(t, S1, S2, a, c, n, sigma)
```



71.1.14 Fit_Normal_Dual_Exponential

```
class reliability. ALT_fitters.Fit_Normal_Dual_Exponential(failures, failure_stress_1,  
                                                       failure_stress_2, right_censored=None,  
                                                       right_censored_stress_1=None,  
                                                       right_censored_stress_2=None,  
                                                       use_level_stress=None, CI=0.95,  
                                                       optimizer=None,  
                                                       show_probability_plot=True,  
                                                       show_life_stress_plot=True,  
                                                       print_results=True)
```

This function will Fit the Normal_Dual_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two thermal stresses (such as temperature-humidity). It is recommended that you ensure your temperature data are in Kelvin and humidity data range from 0 to 1.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as humidity) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.

- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as humidity) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

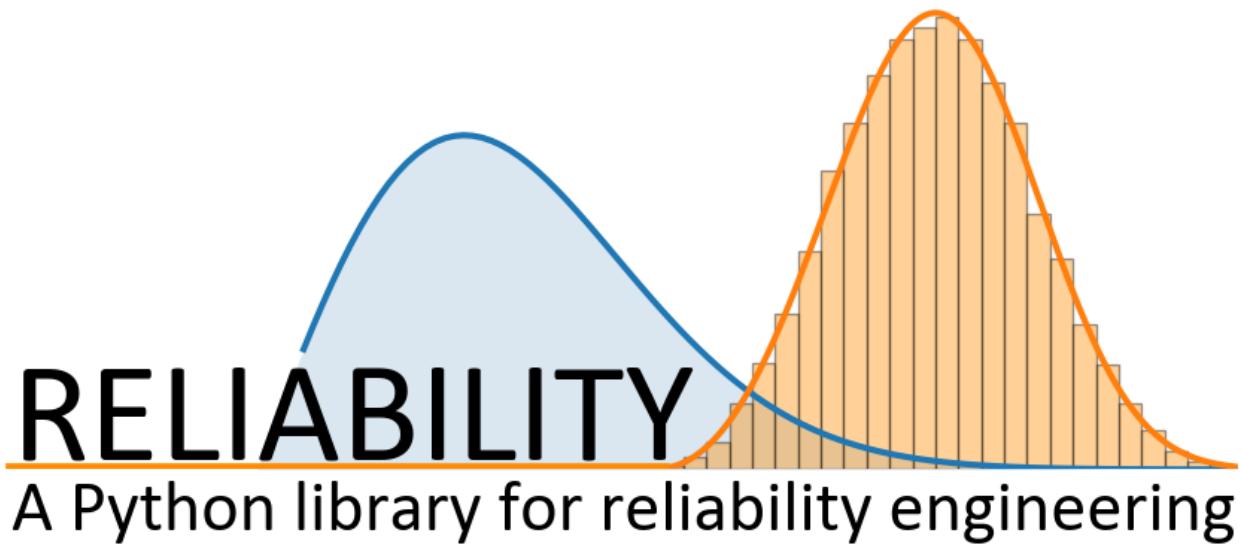
- **a** (*float*) – The fitted parameter from the Dual_Exponential model
- **b** (*float*) – The fitted parameter from the Dual_Exponential model
- **c** (*float*) – The fitted parameter from the Dual_Exponential model
- **sigma** (*float*) – The fitted Normal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter

- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Normal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Normal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
```

```
static logR(t, S1, S2, a, b, c, sigma)
```

```
static logf(t, S1, S2, a, b, c, sigma)
```



71.1.15 Fit_Normal_Dual_Power

```
class reliability. ALT_fitters.Fit_Normal_Dual_Power(failures, failure_stress_1, failure_stress_2,  
right_censored=None,  
right_censored_stress_1=None,  
right_censored_stress_2=None,  
use_level_stress=None, CI=0.95,  
optimizer=None, show_probability_plot=True,  
show_life_stress_plot=True, print_results=True)
```

This function will Fit the Normal_Dual_Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two non-thermal stresses such as voltage and load.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as voltage) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as load) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as voltage) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as load) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

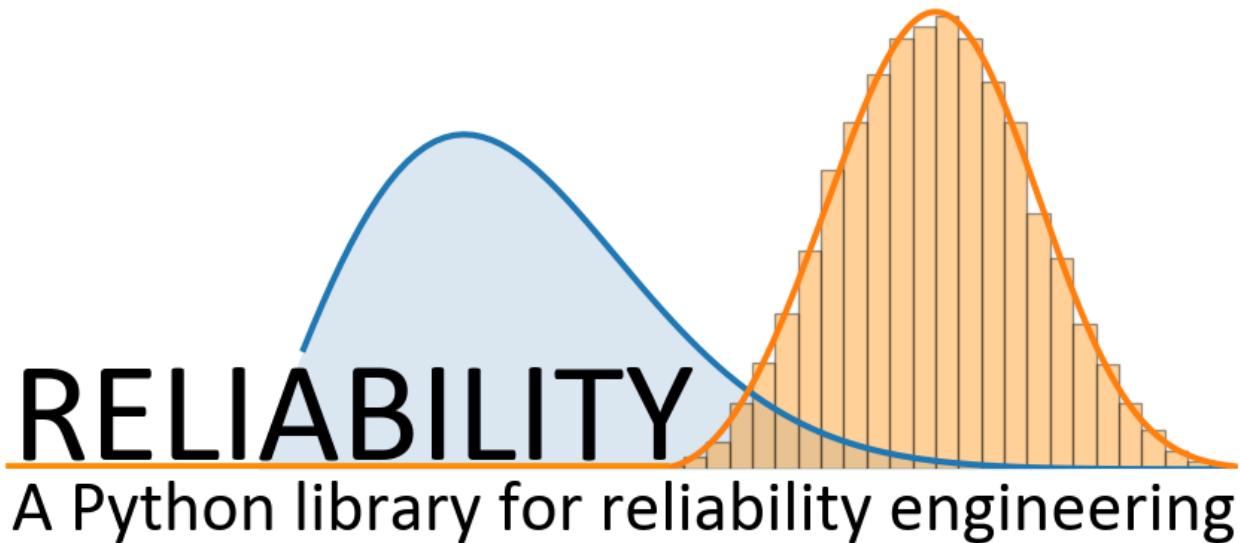
- **c** (*float*) – The fitted parameter from the Dual_Power model
- **m** (*float*) – The fitted parameter from the Dual_Power model
- **n** (*float*) – The fitted parameter from the Dual_Power model
- **sigma** (*float*) – The fitted Normal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)

- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **m_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **m_upper** (*float*) – The upper CI estimate of the parameter
- **m_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Normal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Normal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
```

```
static logR(t, S1, S2, c, m, n, sigma)
```

```
static logf(t, S1, S2, c, m, n, sigma)
```



71.1.16 Fit_Normal_Exponential

```
class reliability. ALT_fitters.Fit_Normal_Exponential(failures, failure_stress, right_censored=None,
right_censored_stress=None,
use_level_stress=None, CI=0.95,
optimizer=None, show_probability_plot=True,
show_life_stress_plot=True,
print_results=True)
```

This function will Fit the Normal-Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

If you are using this model for the Arrhenius equation, $a = Ea/K_B$. When results are printed Ea will be provided in eV.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.

- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

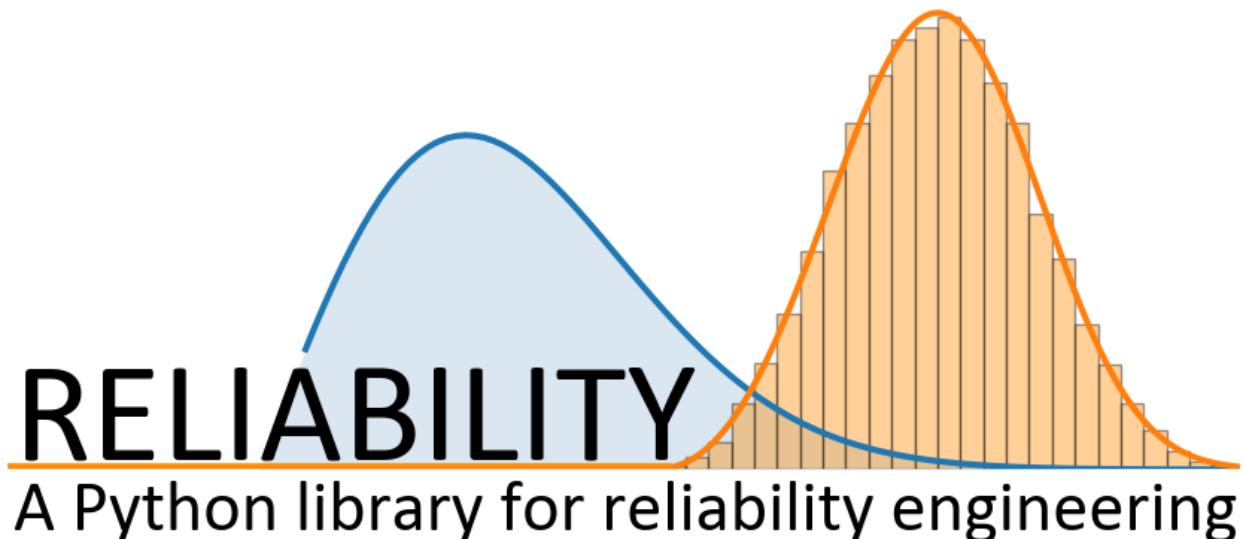
- **a** (*float*) – The fitted parameter from the Exponential model
- **b** (*float*) – The fitted parameter from the Exponential model
- **sigma** (*float*) – The fitted Normal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Normal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Normal distribution at the use level stress (only provided if use_level_stress is provided).

- **probability_plot (object)** – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot (object)** – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)
```

```
static logR(t, T, a, b, sigma)
```

```
static logf(t, T, a, b, sigma)
```



71.1.17 Fit_Normal_Eyring

```
class reliability. ALT_fitters.Fit_Normal_Eyring(failures, failure_stress, right_censored=None,
                                                right_censored_stress=None, use_level_stress=None,
                                                CI=0.95, optimizer=None,
                                                show_probability_plot=True,
                                                show_life_stress_plot=True, print_results=True)
```

This function will Fit the Normal-Eyring life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

Parameters

- **failures (array, list)** – The failure data.
- **failure_stress (array, list)** – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored (array, list, optional)** – The right censored failure times. Optional input.
- **right_censored_stress (array, list, optional)** – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.

- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

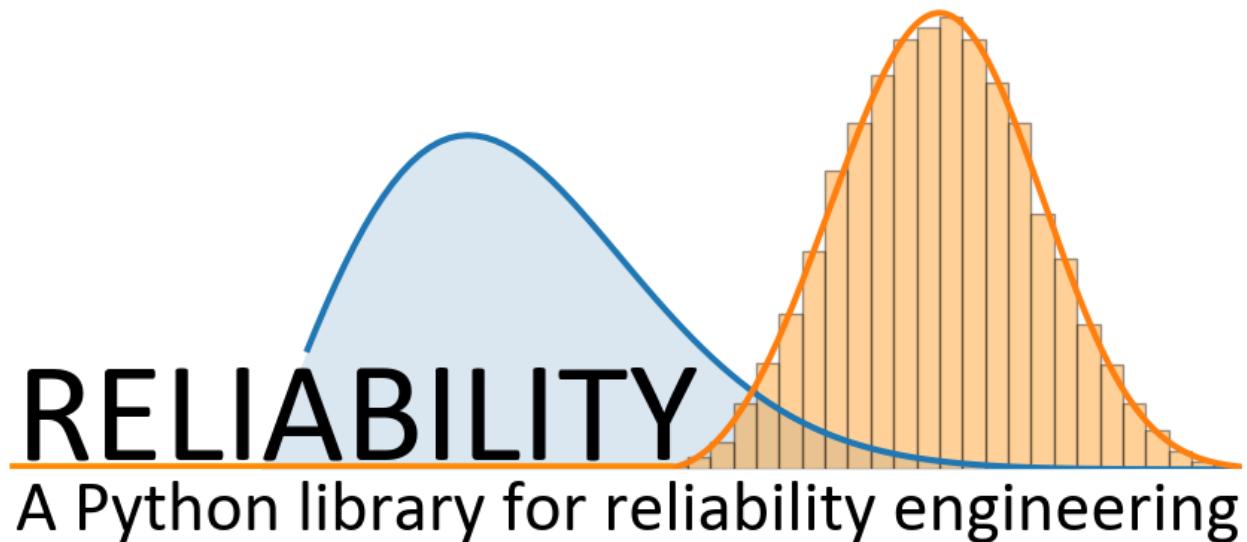
- **a** (*float*) – The fitted parameter from the Eyring model
- **c** (*float*) – The fitted parameter from the Eyring model
- **sigma** (*float*) – The fitted Normal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).

- **mu_at_use_stress** (*float*) – The equivalent Normal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Normal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)
```

```
static logR(t, T, a, c, sigma)
```

```
static logf(t, T, a, c, sigma)
```



71.1.18 Fit_Normal_Power

```
class reliability. ALT_fitters.Fit_Normal_Power(failures, failure_stress, right_censored=None,
                                                right_censored_stress=None, use_level_stress=None,
                                                CI=0.95, optimizer=None,
                                                show_probability_plot=True,
                                                show_life_stress_plot=True, print_results=True)
```

This function will Fit the Normal-Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with non-thermal stresses (typically in fatigue applications).

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as load) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.

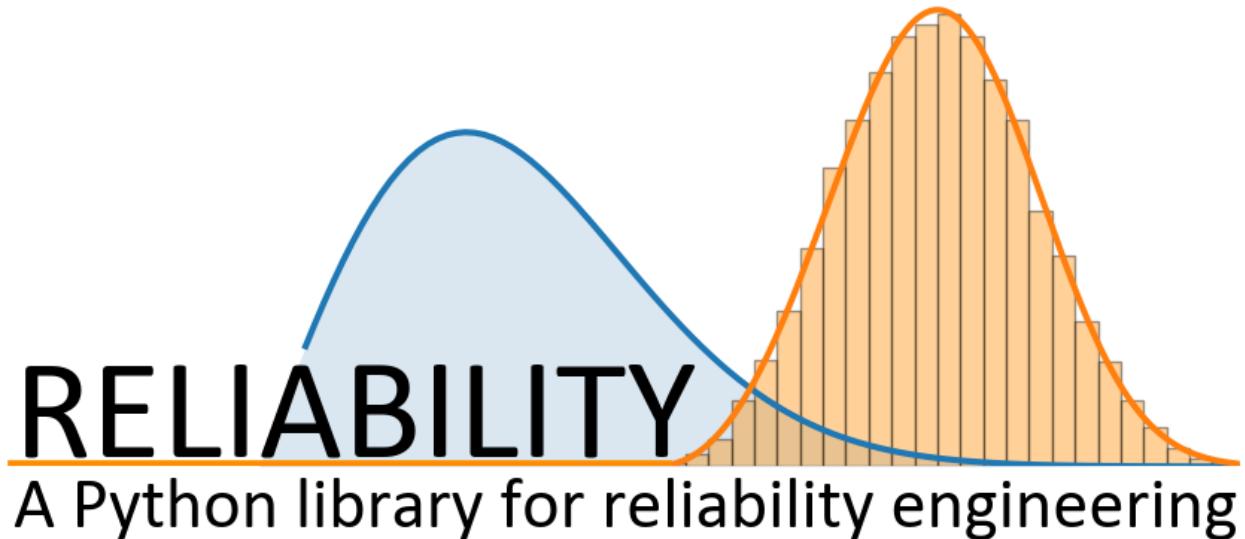
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as load) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

- **a** (*float*) – The fitted parameter from the Power model
- **n** (*float*) – The fitted parameter from the Power model
- **sigma** (*float*) – The fitted Normal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)

- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (*mu* and *sigma*) at each stress level.
- **mean_life** (*float*) – The mean life at the *use_level_stress* (only provided if *use_level_stress* is provided).
- **mu_at_use_stress** (*float*) – The equivalent Normal *mu* parameter at the use level stress (only provided if *use_level_stress* is provided).
- **distribution_at_use_stress** (*object*) – The Normal distribution at the use level stress (only provided if *use_level_stress* is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if *show_probability_plot* is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if *show_life_stress_plot* is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)
static logR(t, T, a, n, sigma)
static logf(t, T, a, n, sigma)
```



71.1.19 Fit_Normal_Power_Exponential

```
class reliability. ALT_fitters.Fit_Normal_Power_Exponential(failures, failure_stress_1,
failure_stress_2, right_censored=None,
right_censored_stress_1=None,
right_censored_stress_2=None,
use_level_stress=None, CI=0.95,
optimizer=None,
show_probability_plot=True,
show_life_stress_plot=True,
print_results=True)
```

This function will Fit the Normal_Power_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with thermal and non-thermal stresses. It is essential that you ensure your thermal stress is stress_1 (as this will be modeled by the Exponential) and your non-thermal stress is stress_2 (as this will be modeled by the Power). Also ensure that your temperature data are in Kelvin.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (thermal-stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (non-thermal stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (non-thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

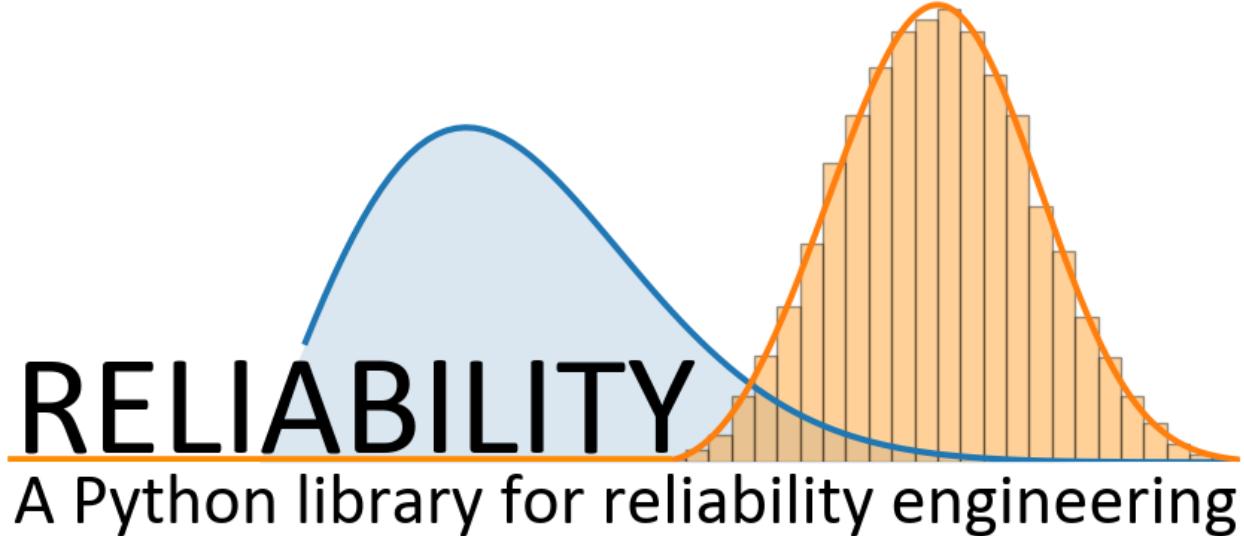
- **a** (*float*) – The fitted parameter from the Power_Exponential model
- **c** (*float*) – The fitted parameter from the Power_Exponential model
- **n** (*float*) – The fitted parameter from the Power_Exponential model
- **sigma** (*float*) – The fitted Normal_2P sigma parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)

- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **sigma_upper** (*float*) – The upper CI estimate of the parameter
- **sigma_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (mu and sigma) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **mu_at_use_stress** (*float*) – The equivalent Normal mu parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Normal distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
```

```
static logR(t, S1, S2, a, c, n, sigma)
```

```
static logf(t, S1, S2, a, c, n, sigma)
```



71.1.20 Fit_Weibull_Dual_Exponential

```
class reliability. ALT_fitters.Fit_Weibull_Dual_Exponential(failures, failure_stress_1, failure_stress_2, right_censored=None, right_censored_stress_1=None, right_censored_stress_2=None, use_level_stress=None, CI=0.95, optimizer=None, show_probability_plot=True, show_life_stress_plot=True, print_results=True)
```

This function will Fit the Weibull_Dual_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two thermal stresses (such as temperature-humidity). It is recommended that you ensure your temperature data are in Kelvin and humidity data range from 0 to 1.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as humidity) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.

- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as humidity) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

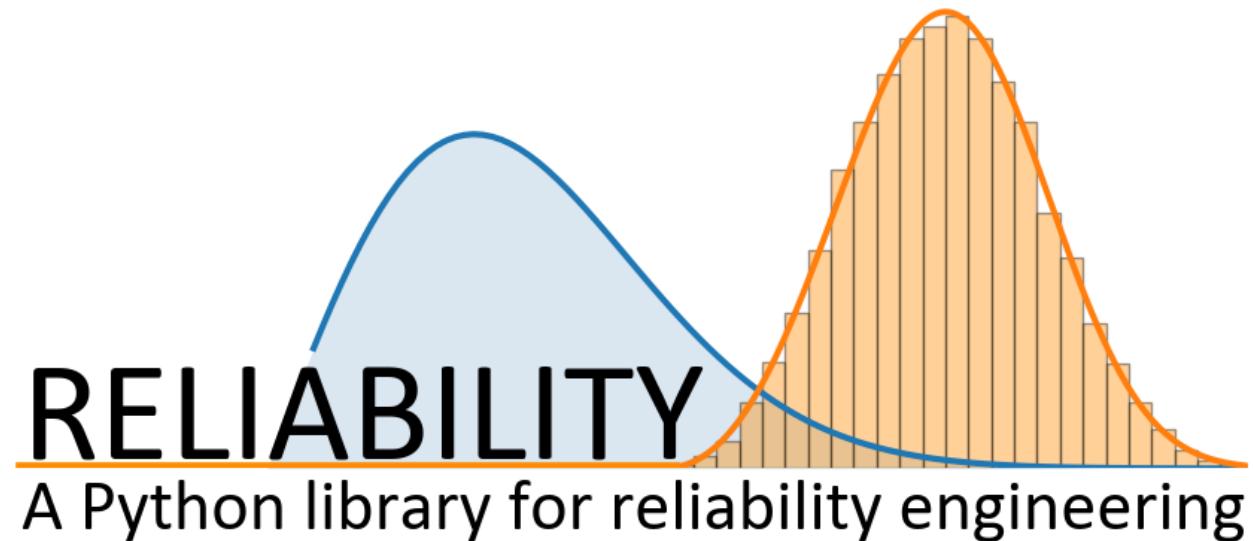
- **a** (*float*) – The fitted parameter from the Dual_Exponential model
- **b** (*float*) – The fitted parameter from the Dual_Exponential model
- **c** (*float*) – The fitted parameter from the Dual_Exponential model
- **beta** (*float*) – The fitted Weibull_2P beta parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **beta_upper** (*float*) – The upper CI estimate of the parameter

- **beta_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **alpha_at_use_stress** (*float*) – The equivalent Weibull alpha parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Weibull distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
```

```
static logR(t, S1, S2, a, b, c, beta)
```

```
static logf(t, S1, S2, a, b, c, beta)
```



71.1.21 Fit_Weibull_Dual_Power

```
class reliability. ALT_fitters.Fit_Weibull_Dual_Power(failures, failure_stress_1, failure_stress_2,
                                                       right_censored=None,
                                                       right_censored_stress_1=None,
                                                       right_censored_stress_2=None,
                                                       use_level_stress=None, CI=0.95,
                                                       optimizer=None, show_probability_plot=True,
                                                       show_life_stress_plot=True,
                                                       print_results=True)
```

This function will Fit the Weibull_Dual_Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with two non-thermal stresses such as voltage and load.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (such as voltage) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (such as load) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (such as voltage) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (such as load) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

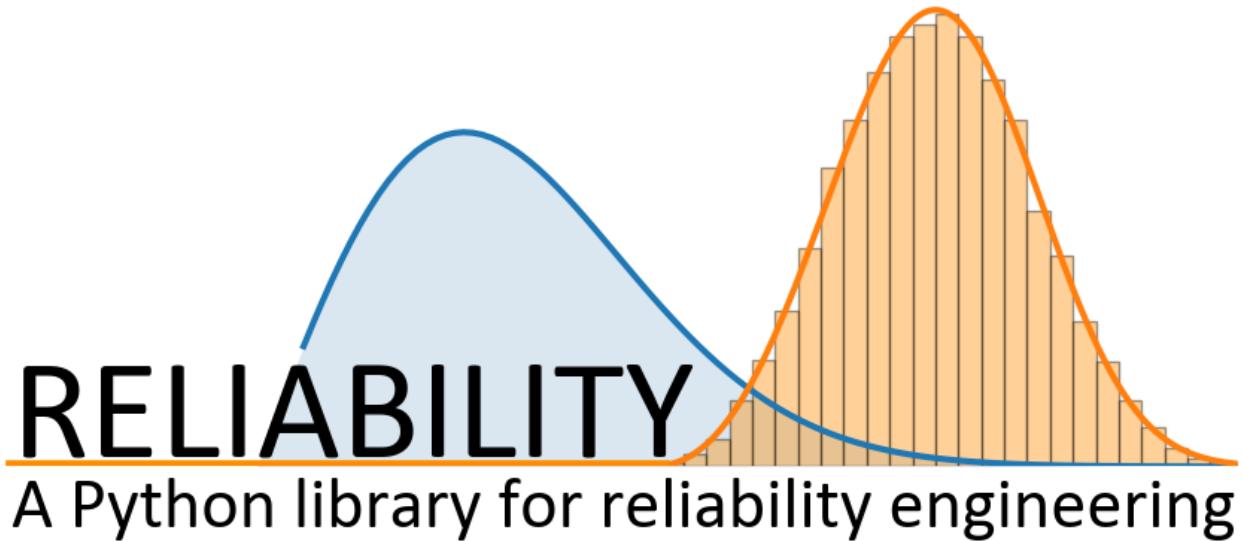
- **c** (*float*) – The fitted parameter from the Dual_Power model

- **n** (*float*) – The fitted parameter from the Dual_Power model
- **m** (*float*) – The fitted parameter from the Dual_Power model
- **beta** (*float*) – The fitted Weibull_2P beta parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **m_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **m_upper** (*float*) – The upper CI estimate of the parameter
- **m_lower** (*float*) – The lower CI estimate of the parameter
- **beta_upper** (*float*) – The upper CI estimate of the parameter
- **beta_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **alpha_at_use_stress** (*float*) – The equivalent Weibull alpha parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Weibull distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

static **LL**(*params*, *t_f*, *t_rc*, *S1_f*, *S2_f*, *S1_rc*, *S2_rc*)

static **logR**(*t*, *S1*, *S2*, *c*, *m*, *n*, *beta*)

```
static logf(t, S1, S2, c, m, n, beta)
```



71.1.22 Fit_Weibull_Exponential

```
class reliability. ALT_fitters.Fit_Weibull_Exponential(failures, failure_stress, right_censored=None,
                                                       right_censored_stress=None,
                                                       use_level_stress=None, CI=0.95,
                                                       optimizer=None,
                                                       show_probability_plot=True,
                                                       show_life_stress_plot=True,
                                                       print_results=True)
```

This function will fit the Weibull-Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

If you are using this model for the Arrhenius equation, $a = Ea/K_B$. When results are printed Ea will be provided in eV.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.

- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

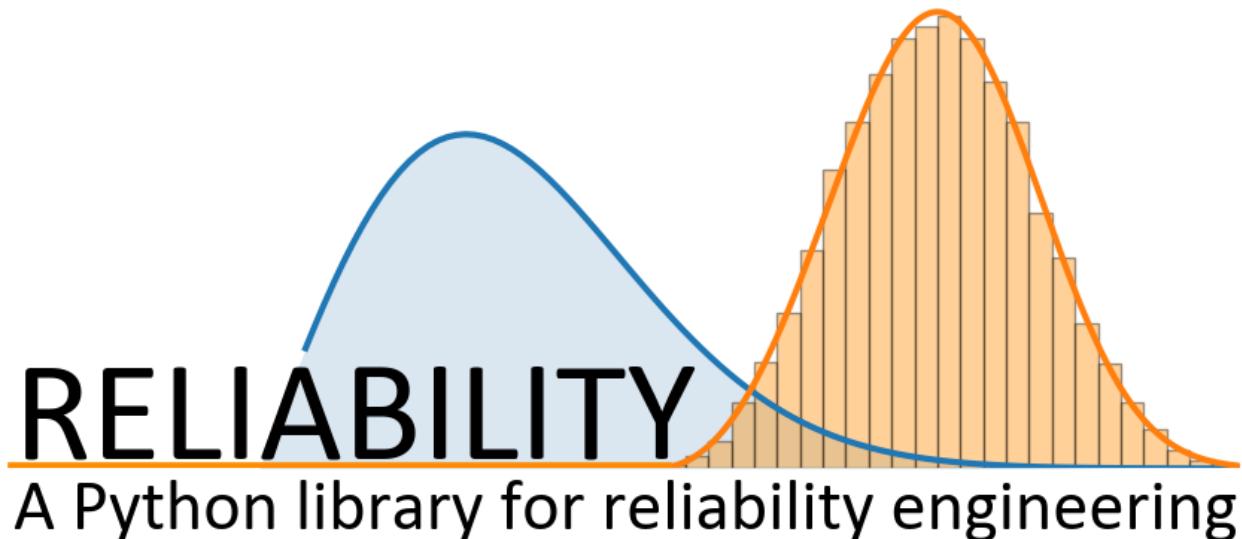
- **a** (*float*) – The fitted parameter from the Exponential model
- **b** (*float*) – The fitted parameter from the Exponential model
- **beta** (*float*) – The fitted Weibull_2P beta parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **b_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **b_upper** (*float*) – The upper CI estimate of the parameter
- **b_lower** (*float*) – The lower CI estimate of the parameter
- **beta_upper** (*float*) – The upper CI estimate of the parameter
- **beta_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **alpha_at_use_stress** (*float*) – The equivalent Weibull alpha parameter at the use level stress (only provided if use_level_stress is provided).

- **distribution_at_use_stress** (*object*) – The Weibull distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)
```

```
static logR(t, T, a, b, beta)
```

```
static logf(t, T, a, b, beta)
```



71.1.23 Fit_Weibull_Eyring

```
class reliability. ALT_fitters.Fit_Weibull_Eyring(failures, failure_stress, right_censored=None,
                                                 right_censored_stress=None,
                                                 use_level_stress=None, CI=0.95, optimizer=None,
                                                 show_probability_plot=True,
                                                 show_life_stress_plot=True, print_results=True)
```

This function will Fit the Weibull-Eyring life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with temperature. It is recommended that you ensure your temperature data are in Kelvin.

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.

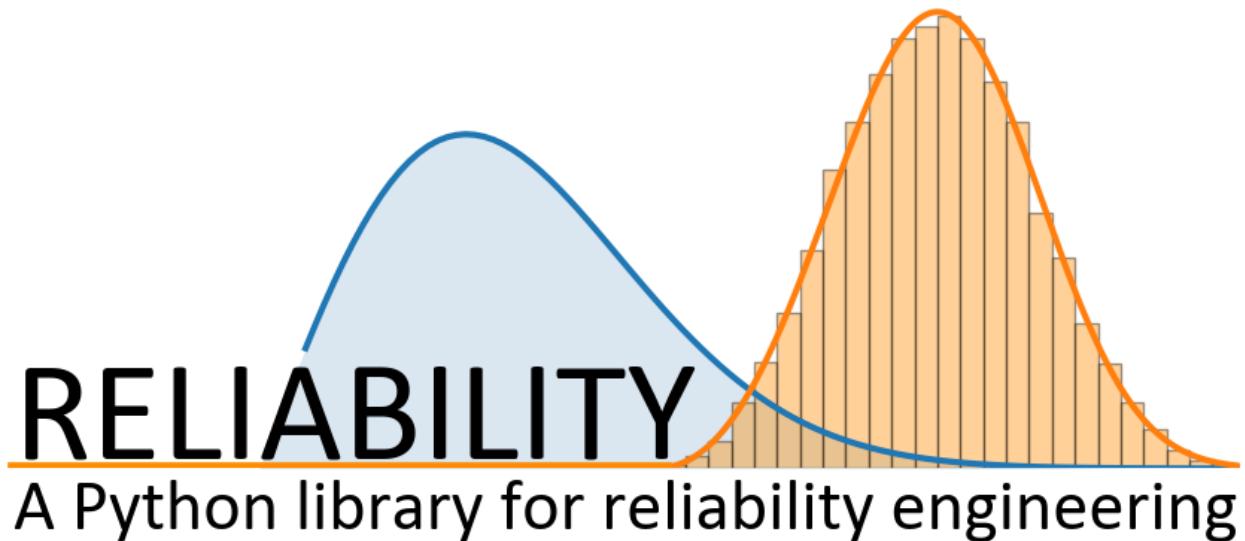
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

- **a** (*float*) – The fitted parameter from the Eyring model
- **c** (*float*) – The fitted parameter from the Eyring model
- **beta** (*float*) – The fitted Weibull_2P beta parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **beta_upper** (*float*) – The upper CI estimate of the parameter
- **beta_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)

- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **alpha_at_use_stress** (*float*) – The equivalent Weibull alpha parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Weibull distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)
static logR(t, T, a, c, beta)
static logf(t, T, a, c, beta)
```



71.1.24 Fit_Weibull_Power

```
class reliability. ALT_fitters.Fit_Weibull_Power(failures, failure_stress, right_censored=None,
                                                right_censored_stress=None, use_level_stress=None,
                                                CI=0.95, optimizer=None,
                                                show_probability_plot=True,
                                                show_life_stress_plot=True, print_results=True)
```

This function will Fit the Weibull-Power life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with non-thermal stresses (typically in fatigue applications).

Parameters

- **failures** (*array, list*) – The failure data.
- **failure_stress** (*array, list*) – The corresponding stresses (such as temperature) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress** (*array, list, optional*) – The corresponding stresses (such as temperature) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*int, float, optional*) – The use level stress at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

Returns

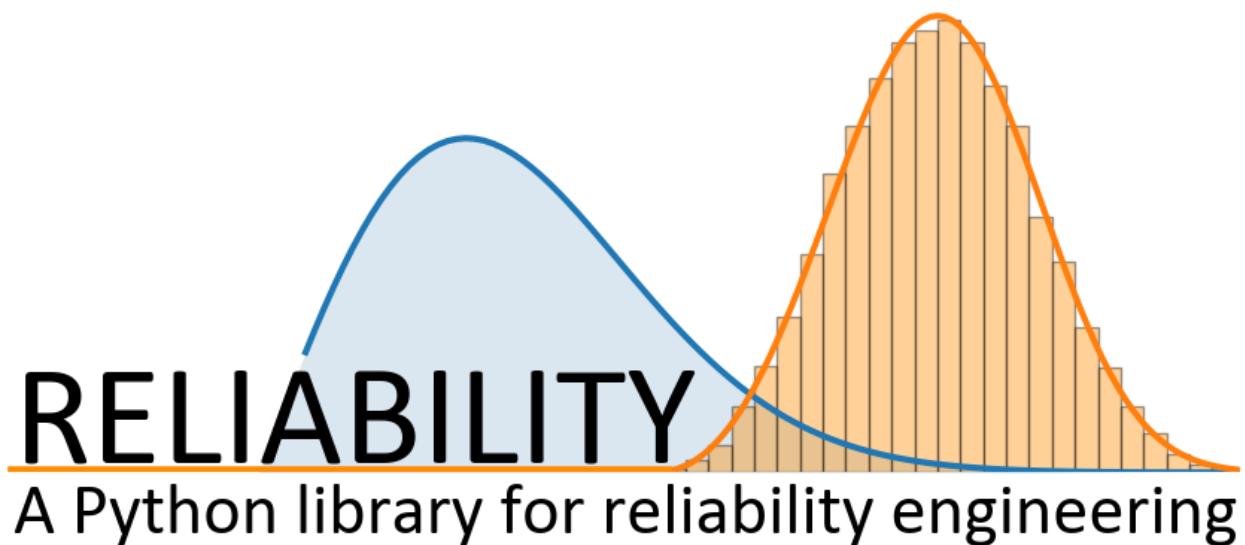
- **a** (*float*) – The fitted parameter from the Power model
- **n** (*float*) – The fitted parameter from the Power model
- **beta** (*float*) – The fitted Weibull_2P beta parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **beta_upper** (*float*) – The upper CI estimate of the parameter

- **beta_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **alpha_at_use_stress** (*float*) – The equivalent Weibull alpha parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Weibull distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, T_f, T_rc)
```

```
static logR(t, T, a, n, beta)
```

```
static logf(t, T, a, n, beta)
```



71.1.25 Fit_Weibull_Power_Exponential

```
class reliability. ALT_fitters.Fit_Weibull_Power_Exponential(failures, failure_stress_1,  
                                                               failure_stress_2,  
                                                               right_censored=None,  
                                                               right_censored_stress_1=None,  
                                                               right_censored_stress_2=None,  
                                                               use_level_stress=None, CI=0.95,  
                                                               optimizer=None,  
                                                               show_probability_plot=True,  
                                                               show_life_stress_plot=True,  
                                                               print_results=True)
```

This function will Fit the Weibull_Power_Exponential life-stress model to the data provided. Please see the online documentation for the equations of this model.

This model is most appropriate to model a life-stress relationship with thermal and non-thermal stresses. It is essential that you ensure your thermal stress is stress_1 (as this will be modeled by the Exponential) and your non-thermal stress is stress_2 (as this will be modeled by the Power). Also ensure that your temperature data are in Kelvin.

Parameters

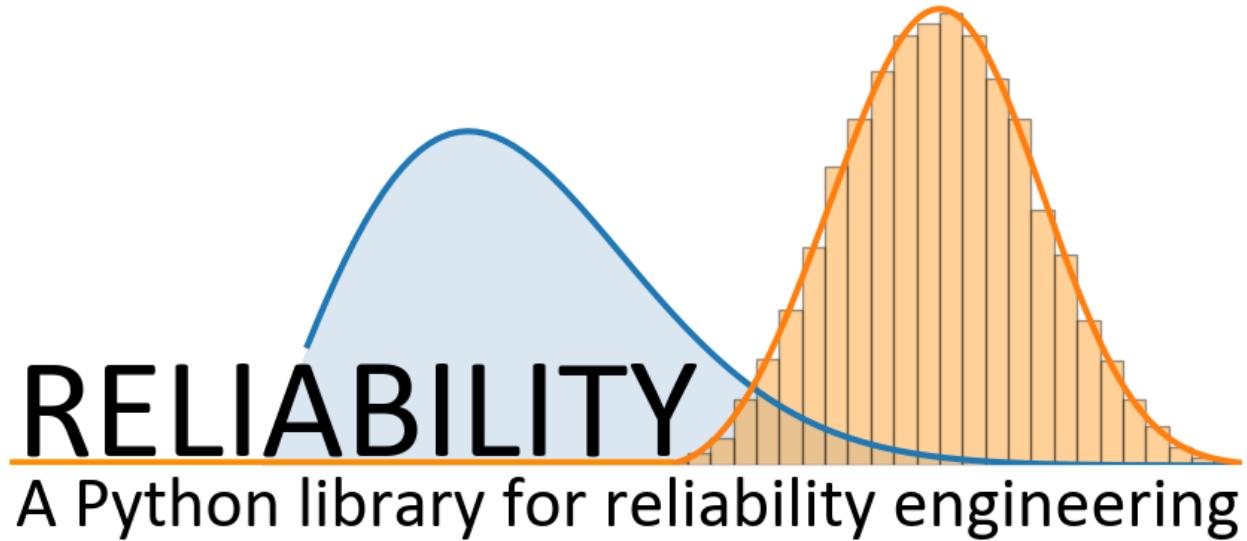
- **failures** (*array, list*) – The failure data.
- **failure_stress_1** (*array, list*) – The corresponding stress 1 (thermal stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **failure_stress_2** (*array, list*) – The corresponding stress 2 (non-thermal stress) at which each failure occurred. This must match the length of failures as each failure is tied to a failure stress.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **right_censored_stress_1** (*array, list, optional*) – The corresponding stress 1 (thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **right_censored_stress_2** (*array, list, optional*) – The corresponding stress 1 (non-thermal stress) at which each right_censored data point was obtained. This must match the length of right_censored as each right_censored value is tied to a right_censored stress. Conditionally optional input. This must be provided if right_censored is provided.
- **use_level_stress** (*array, list optional*) – A two element list or array of the use level stresses in the form [stress_1, stress_2] at which you want to know the mean life. Optional input.
- **print_results** (*bool, optional*) – True/False. Default is True. Prints the results to the console.
- **show_probability_plot** (*bool, object, optional*) – True/False. Default is True. Provides a probability plot of the fitted ALT model. If an axes object is passed it will be used.
- **show_life_stress_plot** (*bool, str, object, optional*) – If True the life-stress plot will be shown. To hide the life-stress plot use False. To swap the axes and show a stress-life plot use ‘swap’. If an axes handle is passed it will be used. Default is True.
- **CI** (*float, optional*) – Confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’,

‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

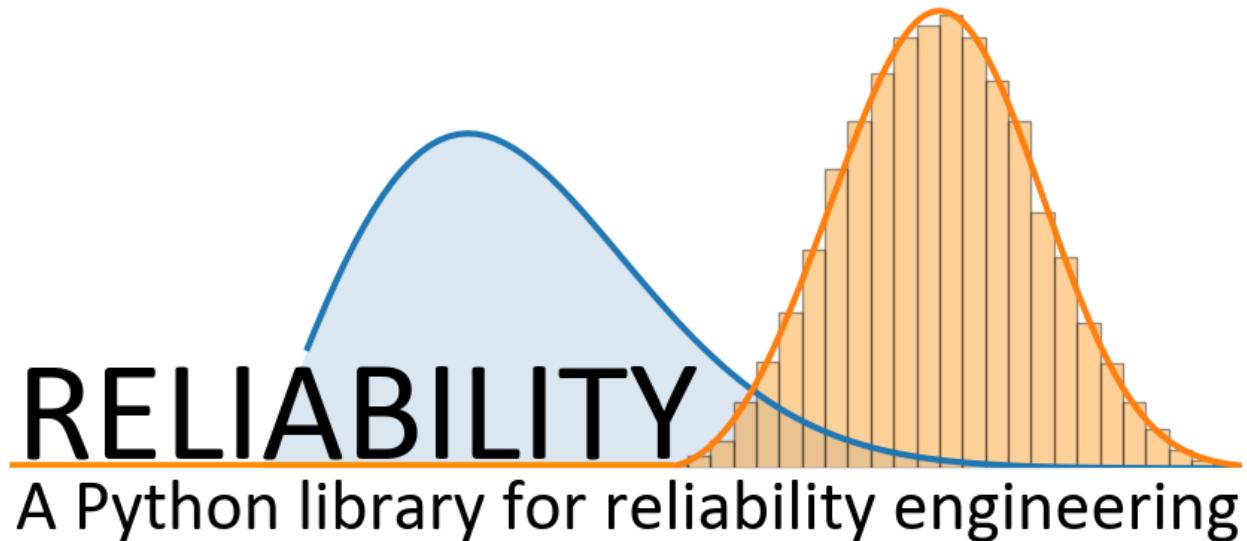
Returns

- **a** (*float*) – The fitted parameter from the Power_Exponential model
- **c** (*float*) – The fitted parameter from the Power_Exponential model
- **n** (*float*) – The fitted parameter from the Power_Exponential model
- **beta** (*float*) – The fitted Weibull_2P beta parameter
- **loglik2** (*float*) – Log Likelihood*-2 (as used in JMP Pro)
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **a_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **c_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **n_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – The standard error (sqrt(variance)) of the parameter
- **a_upper** (*float*) – The upper CI estimate of the parameter
- **a_lower** (*float*) – The lower CI estimate of the parameter
- **c_upper** (*float*) – The upper CI estimate of the parameter
- **c_lower** (*float*) – The lower CI estimate of the parameter
- **n_upper** (*float*) – The upper CI estimate of the parameter
- **n_lower** (*float*) – The lower CI estimate of the parameter
- **beta_upper** (*float*) – The upper CI estimate of the parameter
- **beta_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – A dataframe of the goodness of fit criterion (Log-likelihood, AICc, BIC)
- **change_of_parameters** (*dataframe*) – A dataframe showing the change of the parameters (alpha and beta) at each stress level.
- **mean_life** (*float*) – The mean life at the use_level_stress (only provided if use_level_stress is provided).
- **alpha_at_use_stress** (*float*) – The equivalent Weibull alpha parameter at the use level stress (only provided if use_level_stress is provided).
- **distribution_at_use_stress** (*object*) – The Weibull distribution at the use level stress (only provided if use_level_stress is provided).
- **probability_plot** (*object*) – The figure object from the probability plot (only provided if show_probability_plot is True).
- **life_stress_plot** (*object*) – The figure object from the life-stress plot (only provided if show_life_stress_plot is True).

```
static LL(params, t_f, t_rc, S1_f, S2_f, S1_rc, S2_rc)
static logR(t, S1, S2, a, c, n, beta)
static logf(t, S1, S2, a, c, n, beta)
```



71.2 Convert_data



71.2.1 FNRN_to_FR

```
class reliability.Convert_data.FNRN_to_FR(failures, num_failures, right_censored=None,
                                           num_right_censored=None)
```

Converts data from FNRN format to FR format

Parameters

- **failures** (*array, list*) – The failure times
- **num_failures** (*array, list*) – The number of failures are each failure time. Length must match length of failures.
- **right_censored** (*array, list, optional*) – The right censored times
- **num_right_censored** (*array, list, optional*) – The number of values at each right_censored time. Length must match length of right_censored.

Returns

- **failures** (*array*) – The failure times
- **right_censored** (*array*) – The right censored times

Notes

Example usage:

```
FR = FNRN_to_FR(failures=[1,2,3], num_failures=[1,1,2], right_censored=[9,8,7], num_
→right_censored=[5,4,4])
print(FR.failures)
>>> [1. 2. 3. 3.]
print(FR.right_censored)
>>> [9. 9. 9. 9. 9. 8. 8. 8. 8. 7. 7. 7. 7.]
FR.print()
>>> Data (FR format)
    failures  right censored
        1          9
        2          9
        3          9
        3          9
                    9
                    8
                    8
                    8
                    7
                    7
                    7
                    7
                    7
```

print()

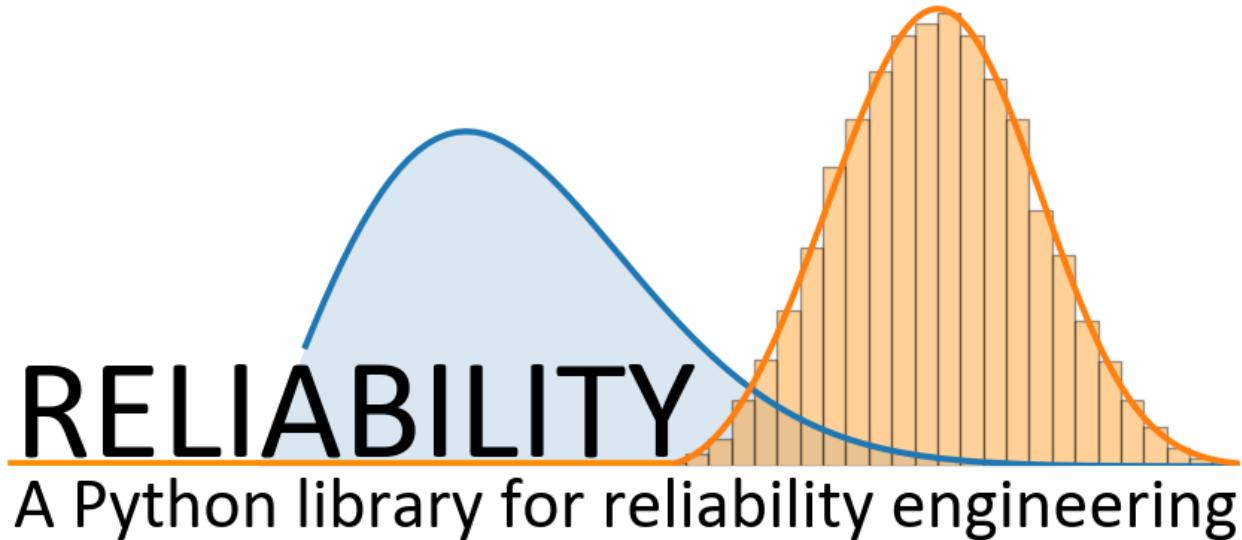
This will print a dataframe of the data in FR format to the console

write_to_xlsx(path, **kwargs)

This will export the data in FR format to an xlsx file at the specified path.

Parameters

- **path** (*str*) – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas



71.2.2 FNRN_to_XCN

```
class reliability.Convert_data.FNRN_to_XCN(failures, num_failures, right_censored=None, num_right_censored=None, censor_code='C', failure_code='F')
```

Converts data from FNRN format to XCN format.

Parameters

- **failures** (*array, list*) – The failure times
- **num_failures** (*array, list*) – The number of failures for each failure time. Length must match length of failures.
- **right_censored** (*array, list, optional*) – The right censored times
- **num_right_censored** (*array, list, optional*) – The number of right censored for each right censored time. Length must match length of right_censored.
- **censor_code** (*str, int, optional*) – The code to use for the censored items. Default is 'C'
- **failure_code** (*str, int, optional*) – The code to use for the failed items. Default is 'F'

Returns

- **X** (*array*) – The event times
- **C** (*array*) – The censor codes
- **N** (*array*) – The number of events at each event time

Notes

Example usage:

```
XCN = FNRN_to_XCN(failures=[1, 2, 3], num_failures=[2, 2, 1], right_censored=[9, 8, 7], num_right_censored=[3, 2, 1])
print(XCN.X)
>>> [1. 2. 3. 7. 8. 9.]
```

(continues on next page)

(continued from previous page)

```

print(XCN.C)
>>> ['F' 'F' 'F' 'C' 'C' 'C']
print(XCN.N)
>>> [2 2 1 1 2 3]
XCN.print()
>>> Data (XCN format)
  event time censor code  number of events
    1      F            2
    2      F            2
    3      F            1
    7      C            1
    8      C            2
    9      C            3

```

print()

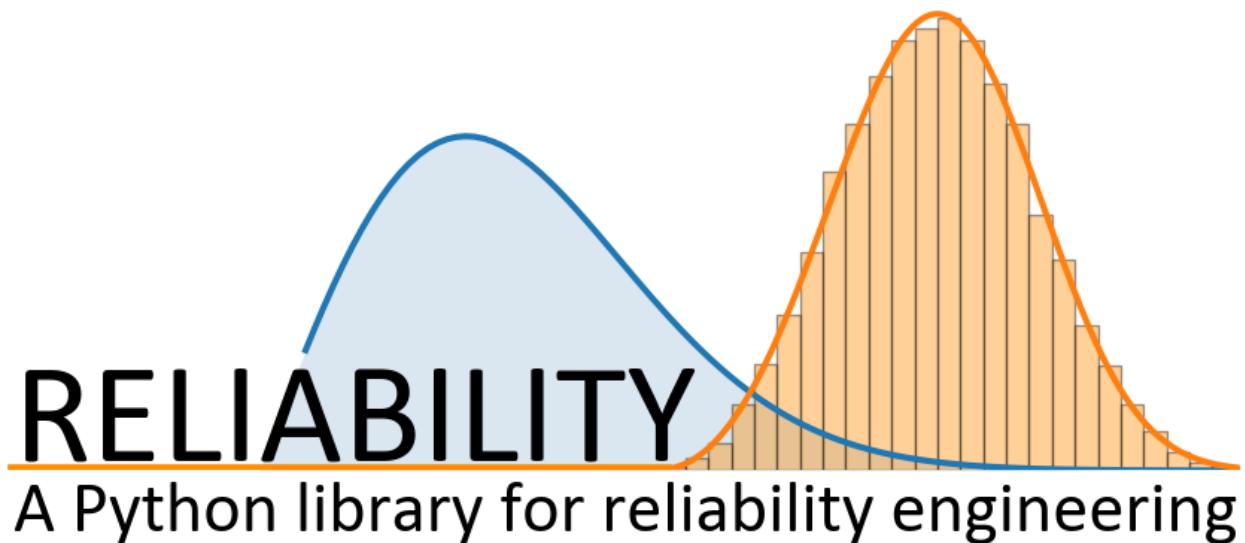
This will print a dataframe of the data in XCN format to the console

write_to_xlsx(path, **kwargs)

This will export the data in XCN format to an xlsx file at the specified path.

Parameters

- **path** (str) – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas



71.2.3 FR_to_FNRR

```
class reliability.Convert_data.FR_to_FNRR(failures, right_censored=None)
```

Converts data from FR format to FNRR format

Parameters

- **failures** (array, list) – The failure times
- **right censored** (array, list, optional) – The right censored times

Returns

- **failures** (*array*) – The failure times
- **num_failures** (*array*) – The number of failures are each failure time
- **right_censored** (*array*) – The right censored times
- **num_right_censored** (*array*) – The number of values at each right_censored time

Notes

Example usage:

```
FNRN = FR_to_FNRN(failures=[1,1,2,2,3], right_censored=[9,9,9,9,8,8,7])
print(FNRN.failures)
>>> [1 2 3]
print(FNRN.num_failures)
>>> [2 2 1]
print(FNRN.right_censored)
>>> [7 8 9]
print(FNRN.num_right_censored)
>>> [1 2 4]
FNRN.print()
>>> Data (FNRN format)
      failures  number of failures  right censored  number of right censored
          1                  2                  7                  1
          2                  2                  8                  2
          3                  1                  9                  4
```

print()

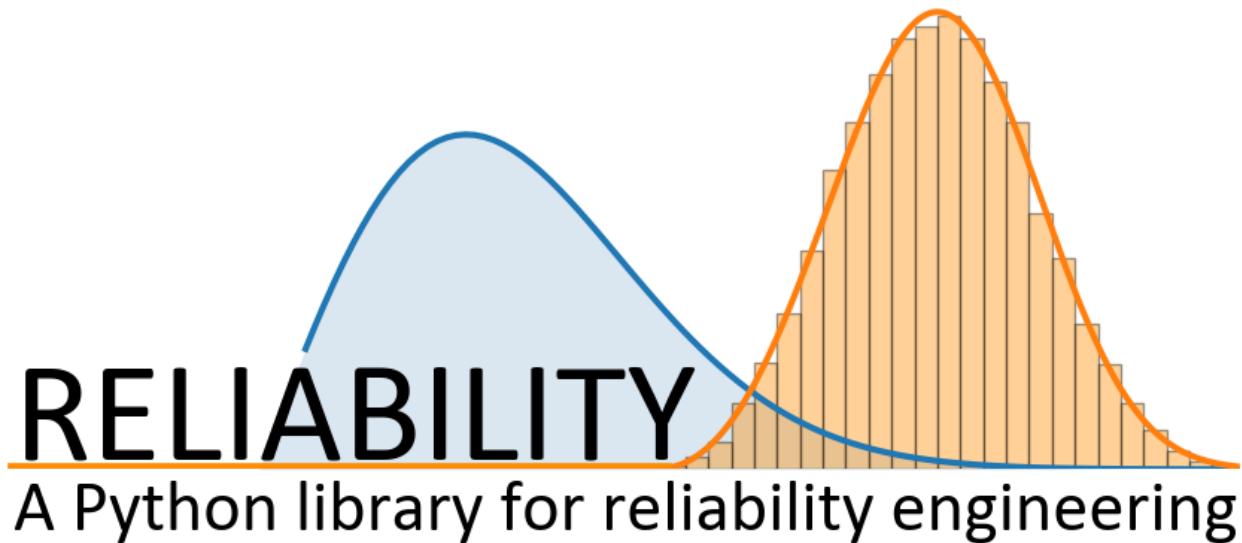
This will print a dataframe of the data in FNRN format to the console

write_to_xlsx(*path*, ***kwargs*)

This will export the data in FNRN format to an xlsx file at the specified path.

Parameters

- **path** (*str*) – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas



71.2.4 FR_to_XCN

```
class reliability.Convert_data.FR_to_XCN(failures, right_censored=None, censor_code='C',
                                         failure_code='F')
```

Converts data from FR format to XCN format.

Parameters

- **failures** (*array, list*) – The failure times
- **right_censored** (*array, list, optional*) – The right censored times
- **censor_code** (*str, int, optional*) – The code to use for the censored items. Default is ‘C’
- **failure_code** (*str, int, optional*) – The code to use for the failed items. Default is ‘F’

Returns

- **X** (*array*) – The event times
- **C** (*array*) – The censor codes
- **N** (*array*) – The number of events at each event time

Notes

Example usage:

```
XCN = FR_to_XCN(failures=[1,1,2,2,3], right_censored=[9,9,9,9,8,8,7])
print(XCN.X)
    >>> [1 2 3 7 8 9]
print(XCN.C)
    >>> ['F' 'F' 'F' 'C' 'C' 'C']
print(XCN.N)
    >>> [2 2 1 1 2 4]
XCN.print()
    >>> Data (XCN format)
        event time censor code  number of events
```

(continues on next page)

(continued from previous page)

1	F	2
2	F	2
3	F	1
7	C	1
8	C	2
9	C	4

print()

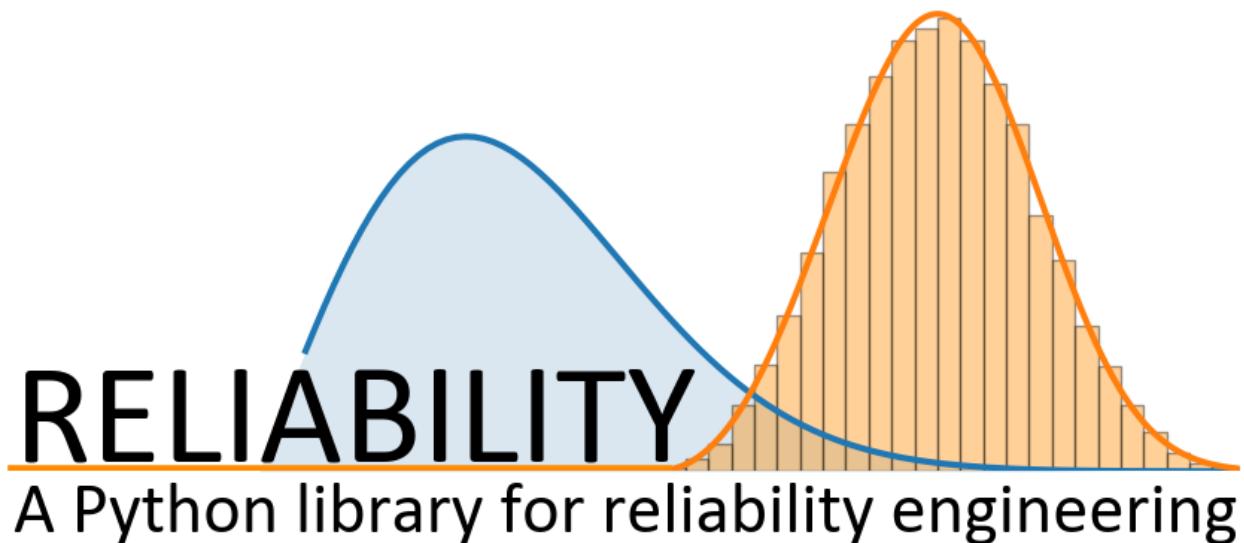
This will print a dataframe of the data in XCN format to the console

write_to_xlsx(path, **kwargs)

This will export the data in XCN format to an xlsx file at the specified path.

Parameters

- **path (str)** – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas

**71.2.5 XCN_to_FNRRN**

```
class reliability.Convert_data.XCN_to_FNRRN(X, C, N=None, censor_code=None, failure_code=None)
```

Converts data from XCN format to FNRRN format.

Parameters

- **X (list, array)** – The failure or right_censored time.
- **C (list, array)** – The censoring code for each X. The default censor codes that will be recognised (not case sensitive) as right censored values are are R, 'RC', 'RIGHT CENS', 'RIGHT CENSORED', 'C', 'CENSORED', 'CENS', 'S', 'SUSP', 'SUSPENSION', 'SUSPENDED', 'UF', 'UNFAILED', 'UNFAIL', 'NF', 'NO FAIL', 'NO FAILURE', 'NOT FAILED', 1. The default failure codes that will be recognised (not case sensitive) as failures are 'F', 'FAIL', 'FAILED', 'FAILURE', 0.
- **N (list, array, optional)** – The quantity for each X. If omitted all items are assumed to have quantity (N) of 1.

- **censor_code** (*str, int, optional*) – The censor code you have used if it does not appear in the defaults listed above.
- **failure_code** (*str, int, optional*) – The failure code you have used if it does not appear in the defaults listed above.

Returns

- **failures** (*array*) – failure times
- **num_failures** (*array*) – the number of failures for each failure time
- **right_censored** (*array*) – right censored times
- **num_right_censored** (*array*) – the number of right censored for each right censored time

Notes

Example usage:

```
FNRM = XCN_to_FNRM(X=[1,2,3,7,8,9], C=['f','f','f','c','c','c'], N=[1,2,2,3,2,1])
print(FNRM.failures)
>>> [1 2 3]
print(FNRM.num_failures)
>>> [1 2 2]
print(FNRM.right_censored)
>>> [7 8 9]
print(FNRM.num_right_censored)
>>> [3 2 1]
FNRM.print()
>>> Data (FNRM format)
      failures  number of failures  right censored  number of right censored
          1                  1                  7                  3
          2                  2                  8                  2
          3                  2                  9                  1
```

`print()`

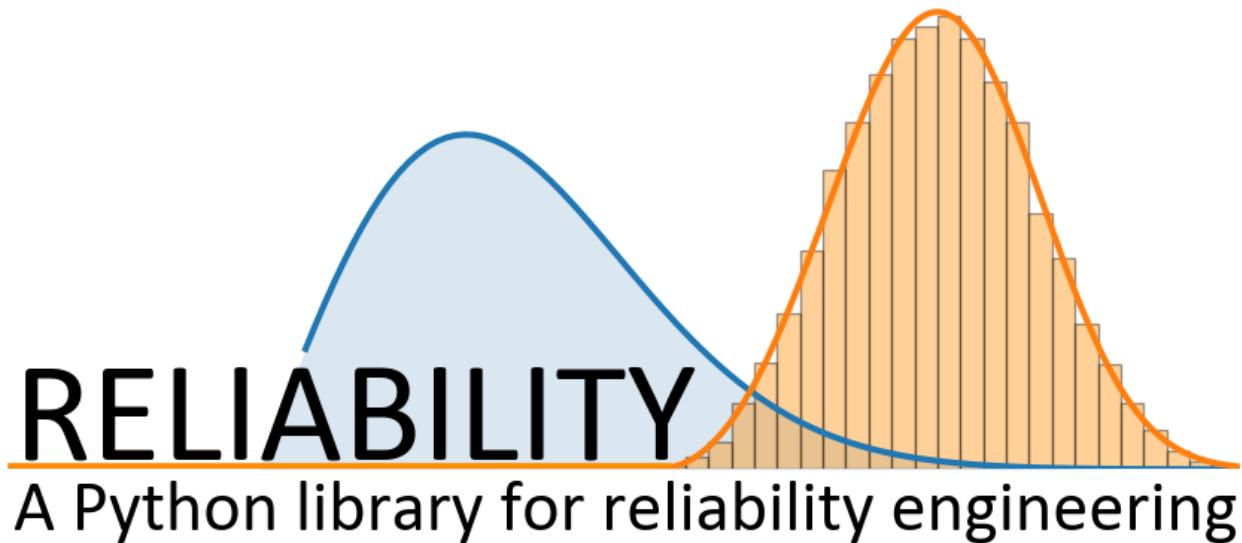
This will print a dataframe of the data in FNRM format to the console

`write_to_xlsx(path, **kwargs)`

This will export the data in FNRM format to an xlsx file at the specified path.

Parameters

- **path** (*str*) – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas



71.2.6 XCN_to_FR

```
class reliability.Convert_data.XCN_to_FR(X, C=None, censor_code=None, failure_code=None)
```

Converts data from XCN format to FR format.

Parameters

- **X** (*list, array*) – The failure or right_censored time.
- **C** (*list, array*) – The censoring code for each X. The default censor codes that will be recognised (not case sensitive) as right censored values are are R, ‘RC’, ‘RIGHT CENS’, ‘RIGHT CENSORED’, ‘C’, ‘CENSORED’, ‘CENS’, ‘S’, ‘SUSP’, ‘SUSPENSION’, ‘SUSPENDED’, ‘UF’, ‘UNFAILED’, ‘UNFAIL’, ‘NF’, ‘NO FAIL’, ‘NO FAILURE’, ‘NOT FAILED’, 1. The default failure codes that will be recognised (not case sensitive) as failures are ‘F’, ‘FAIL’, ‘FAILED’, ‘FAILURE’, 0.
- **N** (*list, array, optional*) – The quantity for each X. If omitted all items are assumed to have quantity (N) of 1.
- **censor_code** (*str, int, optional*) – The censor code you have used if it does not appear in the defaults listed above.
- **failure_code** (*str, int, optional*) – The failure code you have used if it does not appear in the defaults listed above.

Returns

- **failures** (*array*) – failure times
- **right_censored** (*array*) – right censored times

Notes

Example usage:

```
FR = XCN_to_FR(X=[1,2,3,7,8,9], C=['f','f','f','c','c','c'], N=[1,2,2,3,2,1])
print(FR.failures)
>>> [1 2 2 3 3]
```

(continues on next page)

(continued from previous page)

```
print(FR.right_censored)
>>> [7 7 7 8 8 9]
FR.print()
>>> Data (FR format)
failures  right censored
1          7
2          7
2          7
3          8
3          8
9          9
```

print()

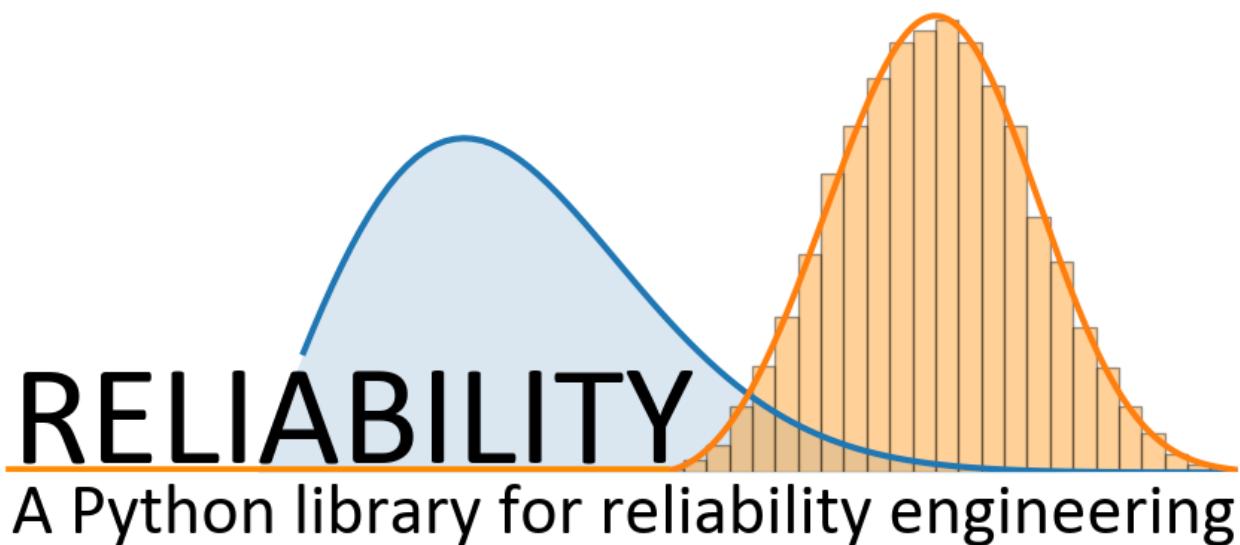
This will print a dataframe of the data in FR format to the console

write_to_xlsx(path, **kwargs)

This will export the data in FR format to an xlsx file at the specified path.

Parameters

- **path** (*str*) – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas

**71.2.7 xlsx_to_FNRM****class reliability.Convert_data.xlsx_to_FNRM(path, **kwargs)**

Converts data from xlsx format into FNRM format. The xlsx format is a Microsoft Excel xlsx file.

Parameters

- **path** (*str*) – The filepath for the xlsx file. Note that you must prefix this with r to specify it as raw text.

Returns

- **failures** (*array*) – failure times

- **num_failures** (array) – the number of failures for each failure time
- **right_censored** (array) – right censored times
- **num_right_censored** (array) – the number of right censored for each right censored time

Notes

For example usage, please see the [online documentation](#).

The function is expecting the xlsx file to have columns in FNRRN format. If they are in another format (FR, XCN) then you will need to use the appropriate function for that format.

A reduced form (FN) is accepted and all values will be assumed to be failures.

`print()`

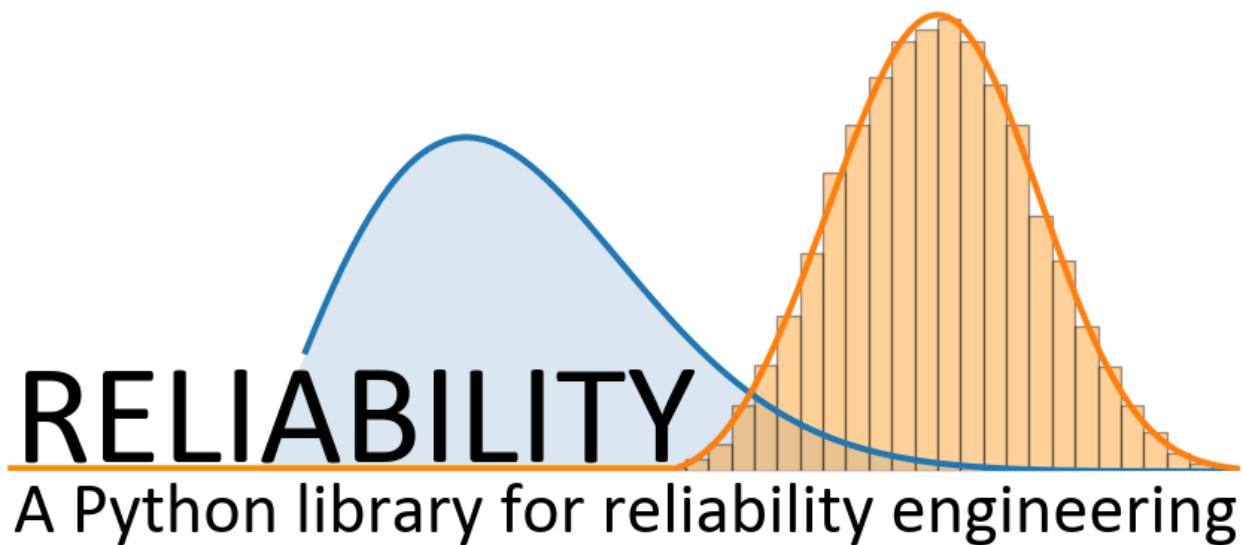
This will print a dataframe of the data in FNRRN format to the console

`write_to_xlsx(path, **kwargs)`

This will export the data in FNRRN format to an xlsx file at the specified path.

Parameters

- **path** (str) – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas



71.2.8 `xlsx_to_FR`

`class reliability.Convert_data.xlsx_to_FR(path, **kwargs)`

Converts data from xlsx format into FR format. The xlsx format is a Microsoft Excel xlsx file.

Parameters

- path** (str) – The filepath for the xlsx file. Note that you must prefix this with r to specify it as raw text.

Returns

- **failures** (array) – failure times
- **right_censored** (array) – right censored times

Notes

For example usage, please see the [online documentation](#).

The function is expecting the xlsx file to have columns in FR format. If they are in another format (XCN, FNRM) then you will need to use the appropriate function for that format.

A reduced form (F) is accepted and all values will be assumed to be failures.

`print()`

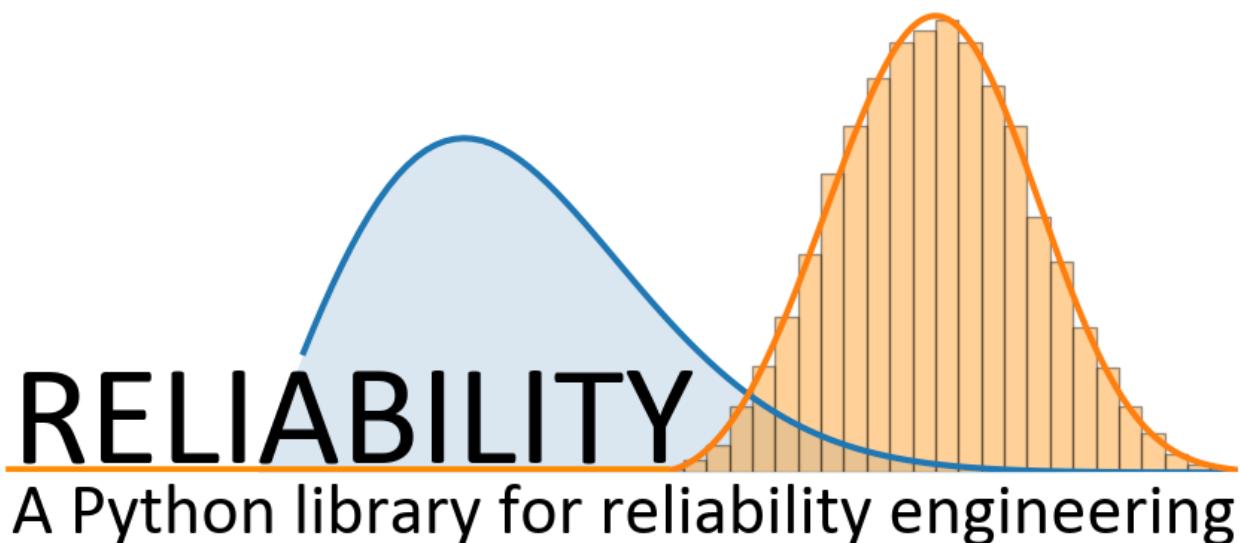
This will print a dataframe of the data in FR format to the console

`write_to_xlsx(path, **kwargs)`

This will export the data in FR format to an xlsx file at the specified path.

Parameters

- **path (str)** – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas



71.2.9 `xlsx_to_XCN`

```
class reliability.Convert_data.xlsx_to_XCN(path, censor_code_in_xlsx=None,
                                             failure_code_in_xlsx=None, censor_code_in_XCN='C',
                                             failure_code_in_XCN='F', **kwargs)
```

Converts data from xlsx format into XCN format. The xlsx format is a Microsoft Excel xlsx file.

Parameters

- **path (str)** – The filepath for the xlsx file. Note that you must prefix this with r to specify it as raw text.
- **censor_code_in_xlsx (str, int optional)** – The censor code you have used if it does not appear in the defaults. The default censor codes that will be recognised (not case sensitive) are 'R', 'RC', 'RIGHT CENS', 'RIGHT CENSORED', 'C', 'CENSORED', 'CENS', 'S', 'SUSP', 'SUSPENSION', 'SUSPENDED', 'UF', 'UNFAILED', 'UNFAIL', 'NF', 'NO FAIL', 'NO FAILURE', 'NOT FAILED', 1

- **failure_code_in_xlsx** (*str, int, optional*) – The failure code you have used if it does not appear in the defaults. The default failure codes that will be recognised (not case sensitive) are ‘F’, ‘FAIL’, ‘FAILED’, ‘FAILURE’, 0
- **censor_code_in_XCN** (*str, int, optional*) – The censor code to be used in XCN format. Default is ‘C’
- **failure_code_in_XCN** (*str, int, optional*) – The failure code to be used in XCN format. Default is ‘F’

Returns

- **X** (*array*) – event times
- **C** (*array*) – censor codes
- **N** (*array*) – number of events at each event time

Notes

For example usage, please see the [online documentation](#).

The function is expecting the xlsx file to have columns in XCN format. If they are in another format (FR, FNRN) then you will need to use the appropriate function for that format.

A reduced form (XC) is accepted and all values will be assumed to have a quantity (N) of 1.

`print()`

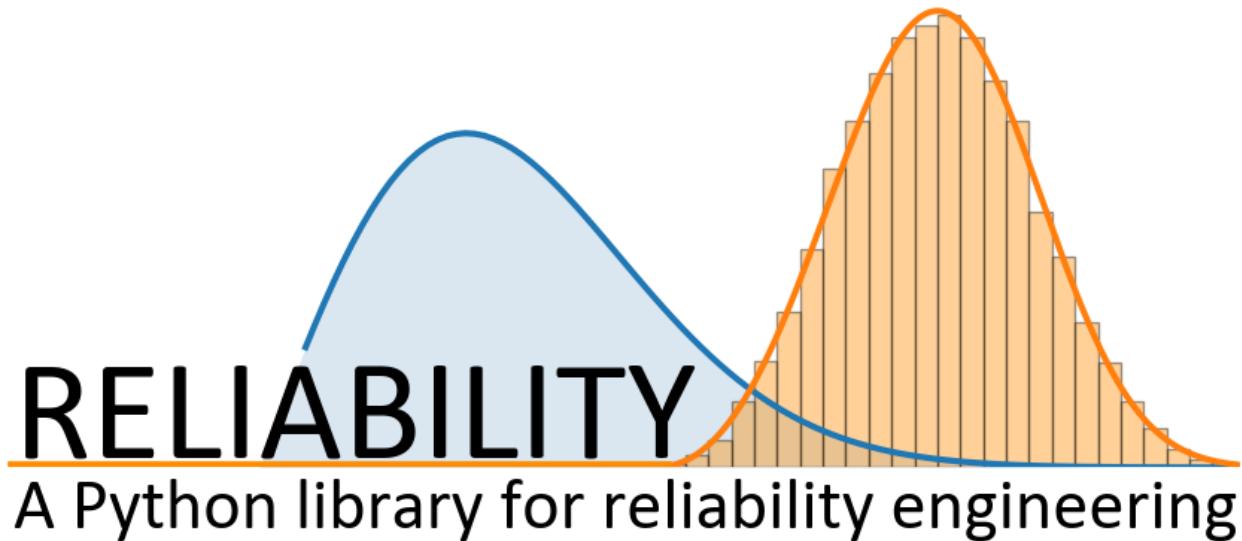
This will print a dataframe of the data in XCN format to the console

`write_to_xlsx(path, **kwargs)`

This will export the data in XCN format to an xlsx file at the specified path.

Parameters

- **path** (*str*) – The file path of the xlsx file to be written
- **kwargs** – Keyword arguments passed directly to pandas



71.3 Datasets

Datasets

This file contains several datasets that are useful for testing and experimenting.

Please see the [online documentation](#) for detailed examples.

```
class reliability.Datasets. ALT_load
```

This is a single stress accelerated life test (ALT) dataset conducted at 3 loads. This dataset contains 20 failure times and no censoring.

Sourced from Dr. Mohammad Modarres, University of Maryland

Returns

- **failures** (*list*) – The failure times
- **failure_stresses** (*list*) – The stress corresponding to each failure time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability. ALT_fitters import Fit_Weibull_Power
from reliability. Datasets import ALT_load
Fit_Weibull_Power(failures=ALT_load().failures, failure_stress=ALT_load().failure_
→stresses)
```

```
class reliability.Datasets. ALT_load2
```

This is a single stress accelerated life test (ALT) dataset conducted at 3 loads. This dataset contains 18 values, 5 of which are censored.

Sourced from Dr. Mohammad Modarres, University of Maryland

Returns

- **failures** (*list*) – The failure times
- **failure_stresses** (*list*) – The stress corresponding to each failure time
- **right_censored** (*list*) – The right censored times
- **right_censored_stresses** (*list*) – The stress corresponding to each right censored time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability. ALT_fitters import Fit_Weibull_Power
from reliability. Datasets import ALT_load2
Fit_Weibull_Power(failures=ALT_load2().failures, failure_stress=ALT_load2().failure_
→stresses, right_censored=ALT_load2().right_censored, right_censored_stress=ALT_
→load2().right_censored_stresses)
```

class reliability.Datasets. ALT_temperature

This is a single stress accelerated life test (ALT) dataset conducted at 3 temperatures. The dataset contains mostly censored data but is easily fitted by several ALT models.

Sourced from Dr. Mohammad Modarres, University of Maryland.

Returns

- **failures** (*list*) – The failure times
- **failure_stresses** (*list*) – The stress corresponding to each failure time
- **right_censored** (*list*) – The right censored times
- **right_censored_stresses** (*list*) – The stress corresponding to each right censored time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability. ALT_fitters import Fit_Weibull_Exponential
from reliability. Datasets import ALT_temperature
Fit_Weibull_Exponential(failures=ALT_temperature().failures, failure_stress=ALT_
    .temperature().failure_stresses, right_censored=ALT_temperature().right_censored,_
    .right_censored_stress=ALT_temperature().right_censored_stresses)
```

class reliability.Datasets. ALT_temperature2

This is a single stress accelerated life test (ALT) dataset conducted at 4 temperatures. This dataset contains 40 values, 20 of which are censored.

Sourced from Dr. Mohammad Modarres, University of Maryland

Returns

- **failures** (*list*) – The failure times
- **failure_stresses** (*list*) – The stress corresponding to each failure time
- **right_censored** (*list*) – The right censored times
- **right_censored_stresses** (*list*) – The stress corresponding to each right censored time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability. ALT_fitters import Fit_Weibull_Exponential
from reliability. Datasets import ALT_temperature2
Fit_Weibull_Exponential(failures=ALT_temperature2().failures, failure_stress=ALT_
    .temperature2().failure_stresses, right_censored=ALT_temperature2().right_censored,_
    .right_censored_stress=ALT_temperature2().right_censored_stresses)
```

class reliability.Datasets. ALT_temperature3

This is a single stress accelerated life test (ALT) dataset conducted at 3 temperatures. This dataset contains 30 values and no censoring.

Returns

- **failures** (*list*) – The failure times
- **failure_stresses** (*list*) – The stress corresponding to each failure time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability. ALT_fitters import Fit_Weibull_Exponential
from reliability. Datasets import ALT_temperature3
Fit_Weibull_Exponential(failures=ALT_temperature3().failures, failure_stress=ALT_
    ↴temperature3().failure_stresses)
```

class reliability.Datasets. ALT_temperature4

This is a single stress accelerated life test (ALT) dataset conducted at 3 temperatures. This dataset contains 20 values and no censoring.

Returns

- **failures** (*list*) – The failure times
- **failure_stresses** (*list*) – The stress corresponding to each failure time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability. ALT_fitters import Fit_Weibull_Exponential
from reliability. Datasets import ALT_temperature4
Fit_Weibull_Exponential(failures=ALT_temperature4().failures, failure_stress=ALT_
    ↴temperature4().failure_stresses)
```

class reliability.Datasets. ALT_temperature_humidity

This is a dual stress accelerated life test (ALT) dataset conducted at 2 different temperatures and 2 different humidities. The dataset contains 12 failures and no censoring.

Returns

- **failures** (*list*) – The failure times
- **failure_stress_temp** (*list*) – The temperature stress corresponding to each failure time
- **failure_stress_humidity** (*list*) – The humidity stress corresponding to each failure time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability. Datasets import ALT_temperature_humidity
from reliability. ALT_fitters import Fit_Normal_Dual_Exponential
data = ALT_temperature_humidity()
Fit_Normal_Dual_Exponential(failures=data.failures, failure_stress_1=data.failure_
    ↴stress_temp, failure_stress_2=data.failure_stress_humidity)
```

class reliability.Datasets. ALT_temperature_voltage

This is a dual stress accelerated life test (ALT) dataset conducted at 2 different temperatures and 2 different voltages. The dataset contains 12 failures and no censoring.

Returns

- **failures** (*list*) – The failure times
- **failure_stress_temp** (*list*) – The temperature stress corresponding to each failure time
- **failure_stress_voltage** (*list*) – The voltage stress corresponding to each failure time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import ALT_temperature_voltage
from reliability. ALT_fitters import Fit_Normal_Dual_Exponential
data = ALT_temperature_voltage()
Fit_Normal_Dual_Exponential(failures=data.failures, failure_stress_1=data.failure_
+stress_temp, failure_stress_2=data.failure_stress_voltage)
```

class reliability.Datasets. ALT_temperature_voltage2

This is a dual stress accelerated life test (ALT) dataset conducted at 3 different temperatures and 2 different voltages. There are 18 failures and no censoring. Note that there is stress-pair that contains only a single failure.

Returns

- **failures** (*list*) – The failure times
- **failure_stress_temp** (*list*) – The temperature stress corresponding to each failure time
- **failure_stress_voltage** (*list*) – The voltage stress corresponding to each failure time
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import ALT_temperature_voltage2
from reliability. ALT_fitters import Fit_Normal_Dual_Exponential
data = ALT_temperature_voltage2()
Fit_Normal_Dual_Exponential(failures=data.failures, failure_stress_1=data.failure_
+stress_temp, failure_stress_2=data.failure_stress_voltage)
```

class reliability.Datasets. MCF_1

This dataset is formatted for use with the Mean Cumulative Function (MCF_parametric or MCF_nonparametric). It consists of failure times for five systems. It exhibits a fairly constant failure rate, appearing to be slightly increasing (beta > 1).

Returns

- **times** (*list*) – A list of lists. Each sublist contains the failure times for each system.
- **number_of_systems** (*int*) – The number of systems in the dataset (len(times))

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Repairable_systems import MCF_nonparametric
from reliability.Datasets import MCF_1
MCF_nonparametric(data=MCF_1().times)
```

class reliability.Datasets.MCF_2

This dataset is formatted for use with the Mean Cumulative Function (MCF_parametric or MCF_nonparametric). It consists of failure times for 56 systems. It exhibits an increasing failure rate at the start and a decreasing failure rate near the end. Due to this shape it is not fitted well by the power law model used in MCF parametric.

Returns

- **times** (*list*) – A list of lists. Each sublist contains the failure times for each system.
- **number_of_systems** (*int*) – The number of systems in the dataset (len(times))

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Repairable_systems import MCF_nonparametric
from reliability.Datasets import MCF_2
MCF_nonparametric(data=MCF_2().times)
```

class reliability.Datasets.automotive

This dataset is relatively small and a challenging task to fit with any distribution due to its size and shape. It also includes mostly right censored data which makes fitting more difficult.

Sourced (with permission) from: V.V. Krivtsov and J. W. Case (1999), Peculiarities of Censored Data Analysis in Automotive Industry Applications - SAE Technical Paper Series, # 1999-01-3220

Returns

- **failures** (*list*) – The failure times
- **right_censored** (*list*) – The right censored times
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import automotive
from reliability.Fitters import Fit_Weibull_2P
Fit_Weibull_2P(failures=automotive().failures, right_censored=automotive().right_censored)
```

class reliability.Datasets.defective_sample

This dataset is heavily right censored with intermixed multiply censored data (not all censored values are greater than the largest failure). It exhibits the behavior of a defective sample (aka. Limited fraction defective). Thanks to Alexander Davis for providing this dataset.

Returns

- **failures** (*list*) – The failure times

- **right_censored** (*list*) – The right censored times
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import defective_sample
from reliability.Fitters import Fit_Weibull_DS
Fit_Weibull_DS(failures=defective_sample().failures, right_censored=defective_
sample().right_censored)
```

class reliability.Datasets.electronics

This dataset is heavily right censored without intermixed censoring (all censored values are greater than the largest failure). Thanks to Jiwon Cha for providing this dataset.

Returns

- **dataframe** (*dataframe*) – A dataframe with columns of time, quantity, category (this is data in XCN format).
- **failures** (*list*) – The failure times
- **right_censored** (*list*) – The right censored times
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

This dataset is in the correct format for use in Fit_Weibull_2P_grouped. When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import electronics
from reliability.Fitters import Fit_Weibull_2P_grouped
Fit_Weibull_2P_grouped(dataframe=electronics().dataframe)
```

class reliability.Datasets.mileage

This dataset is simple to fit. It contains 100 values with no right censoring. The data appears to be from a Normal Distribution.

Sourced from Example 2.31 (page 63) of Reliability Engineering and Risk analysis 3rd Edition by M. Modarres, M. Kaminskiy, and V.V. Krivtsov

Returns

- **failures** (*list*) – The failure times
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import mileage
from reliability.Fitters import Fit_Weibull_2P
Fit_Weibull_2P(failures=mileage().failures)
```

class reliability.Datasets.mixture

This dataset is from a mixture model with heavy censoring (97.90622% right censored). It is best modelled using a Weibull Mixture Model.

Returns

- **failures** (*list*) – The failure times
- **right_censored** (*list*) – The right censored times
- **info** (*dataframe*) – Descriptive statistics about the dataset

Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import mixture
from reliability.Fitters import Fit_Weibull_Mixture
Fit_Weibull_Mixture(failures=automotive().failures, right_censored=automotive() .
    ↴right_censored)
```

class reliability.Datasets.system_growth

This dataset is contains 22 values with no right censoring. The data is from a system that has an increasing MTBF.

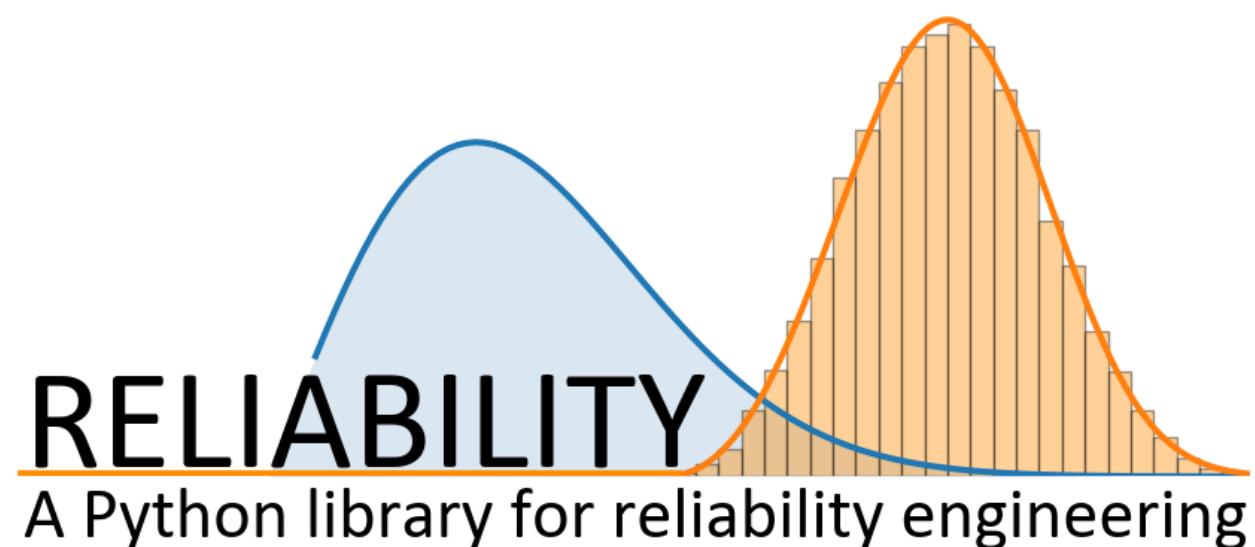
Returns

- **failures** (*list*) – The failure times
- **info** (*dataframe*) – Descriptive statistics about the dataset

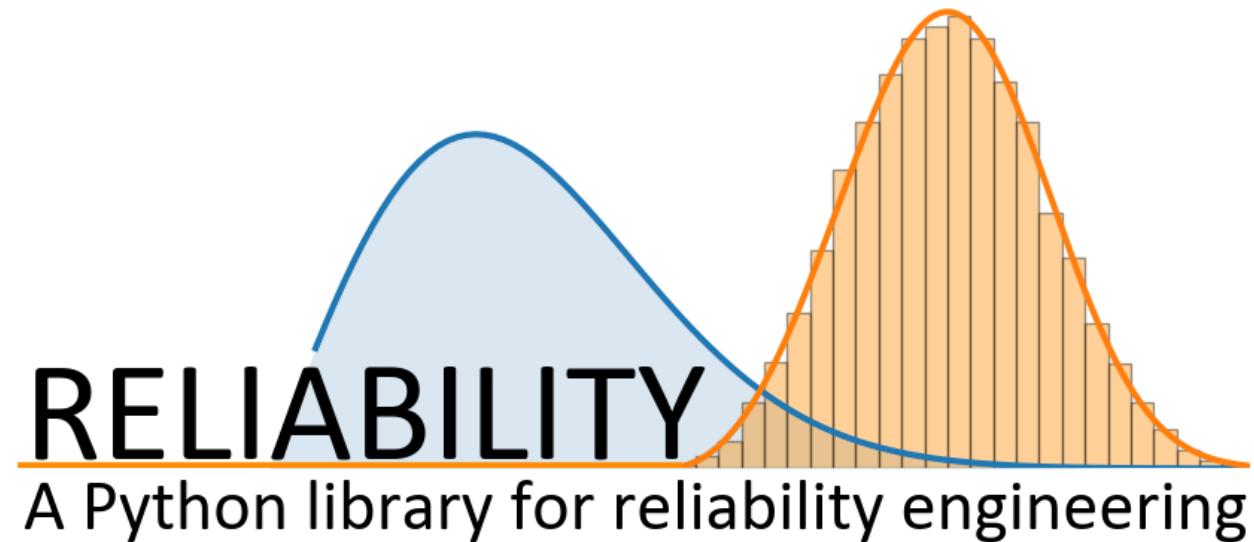
Notes

When importing the dataset, ensure it is called using the brackets after the name. Example Usage:

```
from reliability.Datasets import system_growth
from reliability.Fitters import Fit_Weibull_2P
Fit_Weibull_2P(failures=system_growth().failures)
```



71.4 Distributions



71.4.1 Beta_Distribution

```
class reliability.Distributions.Beta_Distribution(alpha=None, beta=None)
```

Beta probability distribution. Creates a probability distribution object.

Parameters

- **alpha** (*float, int*) – Shape parameter 1. Must be > 0
- **beta** (*float, int*) – Shape parameter 2. Must be > 0

Returns

- **name** (*str*) – ‘Beta’
- **name2** (*‘str’*) – ‘Beta_2P’
- **param_title_long** (*str*) – ‘Beta Distribution (=5,=2)’
- **param_title** (*str*) – ‘=5,=2’
- **parameters** (*list*) – [alpha,beta]
- **alpha** (*float*)
- **beta** (*float*)
- **gamma** (*float*)
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)

- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

kwargs are not accepted

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

CHF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, ***kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the SF (survival function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters

q (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the SF at `q`.

mean_residual_life(*t*)

Mean Residual Life calculator

Parameters

t (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

MRL (*float*) – The mean residual life

plot(*xvals=None, xmin=None, xmax=None*)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use `plt.show()`. If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

quantile(*q*)

Quantile calculator

Parameters

q (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the CDF at `q`. This is the probability that a random variable from the distribution is $< q$

random_samples(*number_of_samples*, *seed=None*)

Draws random samples from the probability distribution

Parameters

- **number_of_samples** (*int*) – The number of samples to be drawn. Must be greater than 0.
- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

samples (*array*) – The random samples

Notes

This is the same as rvs in scipy.stats

stats()

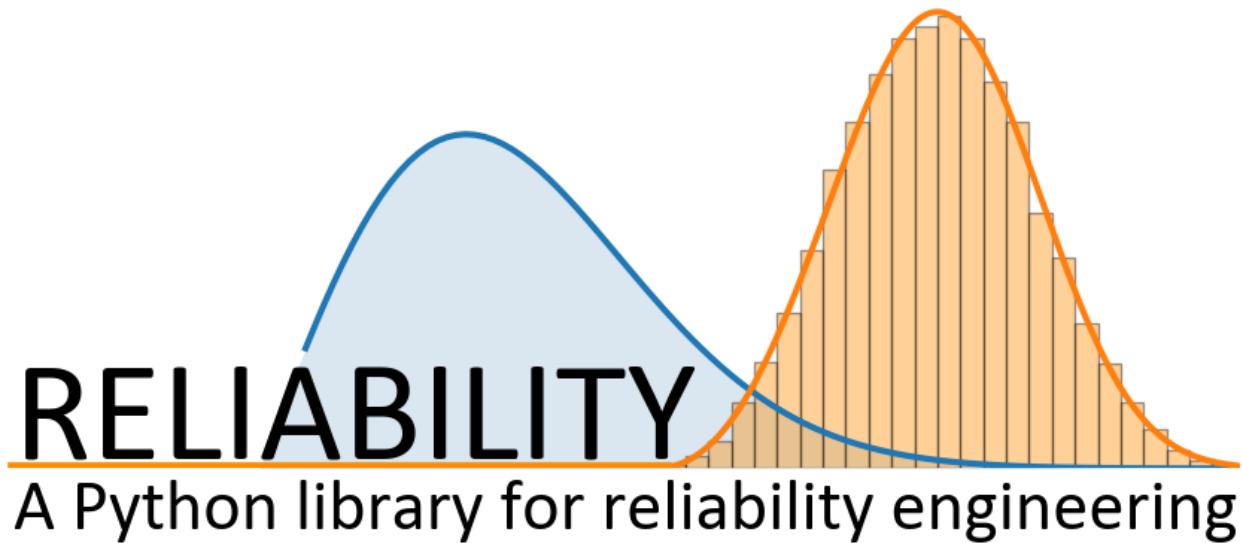
Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters

None

Returns

None



71.4.2 Competing_Risks_Model

class reliability.Distributions.Competing_Risks_Model(*distributions*)

The competing risks model is used to model the effect of multiple risks (expressed as probability distributions) that act on a system over time. The model is obtained using the product of the survival functions:

$$SF_{total} = SF_1 SF_2 SF_3 \dots SF_n$$

The output API is similar to the other probability distributions (Weibull, Normal, etc.) as shown below.

Parameters

distributions (*list, array*) – a list or array of probability distribution objects used to construct the model

Returns

- **name** (*str*) – ‘Competing risks’
- **name2** (*str*) – ‘Competing risks using 3 distributions’. The exact name depends on the number of distributions used
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

An equivalent form of this model is to sum the hazard or cumulative hazard functions which will give the same result. In this way, we see the CDF, HF, and CHF of the overall model being equal to or higher than any of the constituent distributions. Similarly, the SF of the overall model will always be equal to or lower than any of the constituent distributions. The PDF occurs earlier in time since the earlier risks cause the population to fail sooner leaving less to fail due to the later risks.

This model should be used when a data set has been divided by failure mode and each failure mode has been modelled separately. The competing risks model can then be used to recombine the constituent distributions into a single model. Unlike the mixture model, there are no proportions as the risks are competing to cause failure rather than being mixed.

As this process is multiplicative for the survival function, and may accept many distributions of different types, the mathematical formulation quickly gets complex. For this reason, the algorithm combines the models numerically rather than empirically so there are no simple formulas for many of the descriptive statistics (mean, median, etc.). Also, the accuracy of the model is dependent on *xvals*. If the *xvals* array is small (<100 values) then the answer will be ‘blocky’ and inaccurate. The variable *xvals* is only accepted for PDF, CDF, SF, HF, CHF. The other methods (like random samples) use the default *xvals* for maximum accuracy. The default number of values generated when *xvals* is not given is 1000. Consider this carefully when specifying *xvals* in order to avoid inaccuracies in the results.

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_components=False*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default) and `len(xvals) >= 2`.

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

CHF (*xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default) and `len(xvals) >= 2`.

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF (*xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default) and `len(xvals) >= 2`.

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(`xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs`)

Plots the PDF (probability density function)

Parameters

- `show_plot` (`bool, optional`) – True or False. Default = True
- `plot_components` (`bool`) – Option to plot the components of the model. True or False. Default = False.
- `xvals` (`array, list, optional`) – x-values for plotting
- `xmin` (`int, float, optional`) – minimum x-value for plotting
- `xmax` (`int, float, optional`) – maximum x-value for plotting
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

`yvals` (`array, float`) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default) and `len(xvals) >= 2`.

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(`xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs`)

Plots the SF (survival function)

Parameters

- `show_plot` (`bool, optional`) – True or False. Default = True
- `plot_components` (`bool`) – Option to plot the components of the model. True or False. Default = False.
- `xvals` (`array, list, optional`) – x-values for plotting
- `xmin` (`int, float, optional`) – minimum x-value for plotting
- `xmax` (`int, float, optional`) – maximum x-value for plotting
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

`yvals` (`array, float`) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default) and `len(xvals) >= 2`.

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters

q (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the SF at *q*.

mean_residual_life(*t*)

Mean Residual Life calculator

Parameters

t (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

MRL (*float*) – The mean residual life

plot(*xvals=None, xmin=None, xmax=None*)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use plt.show(). If *xvals* is specified, it will be used. If *xvals* is not specified but *xmin* and/or *xmax* are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

quantile(*q*)

Quantile calculator

Parameters

q (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the CDF at *q*. This is the probability that a random variable from the distribution is < *q*

random_samples(*number_of_samples, seed=None*)

Draws random samples from the probability distribution

Parameters

- **number_of_samples** (*int*) – The number of samples to be drawn. Must be greater than 0.
- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

samples (*array*) – The random samples

Notes

This is the same as rvs in `scipy.stats`

stats()

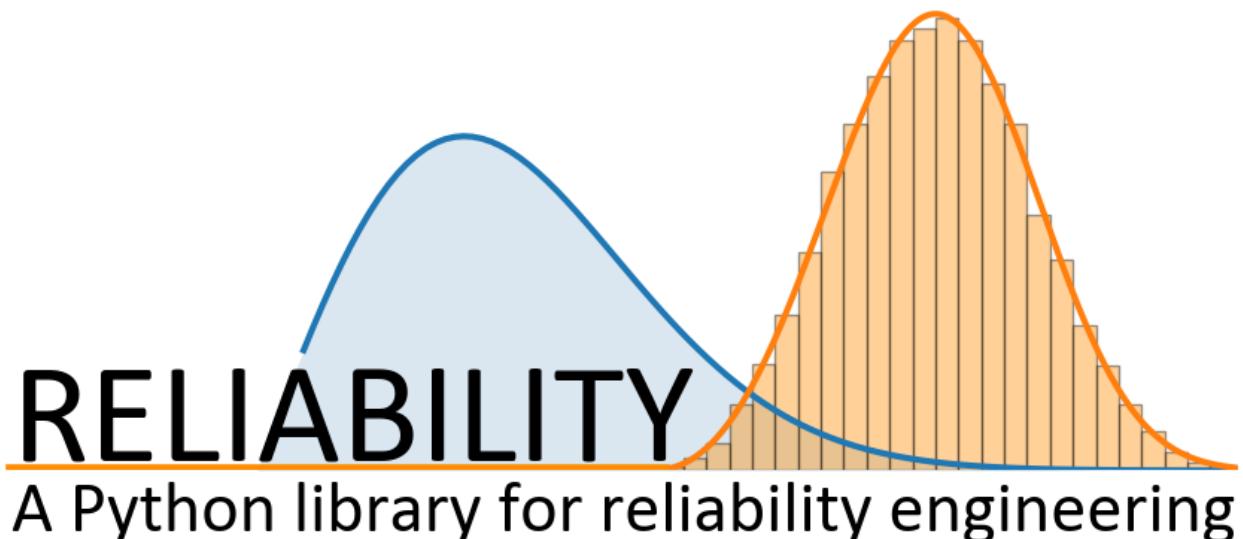
Descriptive statistics of the probability distribution. These are the same as the statistics shown using `.plot()` but printed to the console.

Parameters

`None`

Returns

`None`



71.4.3 DSZI Model

```
class reliability.Distributions.DSZI_Model(distribution, DS=None, ZI=None)
```

Defective Subpopulation Zero Inflated Model. This model should be used when there are failures at $t=0$ (“dead on arrival”) creating a zero inflated (ZI) distribution and/or many right censored failures creating a defective subpopulation (DS) model. The parameters DS and ZI represent the maximum and minimum of the CDF respectively. Their default values are 1 and 0 which is equivalent to a non-DS and non-ZI model. Leaving one as the default and specifying the other can be used to create a DS or ZI model, while specifying both parameters creates a DSZI model.

The output API is similar to the other probability distributions (Weibull, Normal, etc.) as shown below.

Parameters

- **distribution** (*object*) – A probability distribution object representing the base distribution to which the DS and ZI transformations are made.
- **DS** (*float, optional*) – The defective subpopulation fraction. Must be between 0 and 1. Must be greater than ZI. This is the maximum of the CDF. Default is 1 which is equivalent to a non-DS CDF (ie. everything fails eventually).
- **ZI** (*float, optional*) – The zero inflated fraction. Must be between 0 and 1. Must be less than DS. This is the minimum of the CDF. Default is 0 which is equivalent to a non-ZI CDF (ie. no failures at $t=0$).

Returns

- **DS** (*float*)
- **ZI** (*float*)
- **name** (*str*) – ‘DSZI’
- **name2** (*str*) – ‘Defective Subpopulation Zero Inflated Weibull’. Exact string depends on the values of DS and ZI, and the type of base distribution.
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

DS and ZI are optional but at least one of them must be specified. Leaving them both unspecified is equivalent to the base distribution specified in the “distribution” parameter.

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if *show_plot* is True (which it is by default).

If *xvals* is specified, it will be used. If *xvals* is not specified but *xmin* and/or *xmax* are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution’s parameters.

CHF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, ***kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(`xvals=None`, `xmin=None`, `xmax=None`, `show_plot=True`, `**kwargs`)

Plots the SF (survival function)

Parameters

- `show_plot` (*bool, optional*) – True or False. Default = True
- `xvals` (*array, list, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting
- `xmax` (*int, float, optional*) – maximum x-value for plotting
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

`yvals` (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(`q`)

Inverse survival function calculator

Parameters

- `q` (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

`x` (*float*) – The inverse of the SF at `q`.

mean_residual_life(`t`)

Mean Residual Life calculator

Parameters

- `t` (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

`MRL` (*float*) – The mean residual life.

Notes

If `DS < 1` the `MRL` will return `np.inf`

plot(`xvals=None`, `xmin=None`, `xmax=None`)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- `xvals` (*list, array, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting

- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use plt.show(). If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

quantile(*q*)

Quantile calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between ZI and DS. If $q < ZI$ or $q > DS$ then a ValueError will be raised.

Returns

- **x** (*float*) – The inverse of the CDF at *q*. This is the probability that a random variable from the distribution is $< q$

random_samples(*number_of_samples*, *right_censored_time=None*, *seed=None*)

Draws random samples from the probability distribution

Parameters

- **number_of_samples** (*int*) – The number of samples to be drawn. Must be greater than 0.
- **right_censored_time** (*float*) – The time to use as the right censored value. Only required if DS is not 1. The right_censored_time can be thought of as the end of the observation period.
- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

- **failures, right_censored** (*array, array*) – failures is an array of the random samples of the failure times. right_censored is an array of the right censored random samples. failures will contain zeros if $ZI > 0$. right_censored will be empty if $DS = 1$, otherwise all of the All the right_censored samples will be equal to the parameter right_censored_time.

Notes

If any of the failure times exceed the right_censored_time, these failures will be converted to right_censored_time and moved into the right_censored array. A warning will be printed if this occurs. To prevent this from occurring, select a higher right_censored_time.

stats()

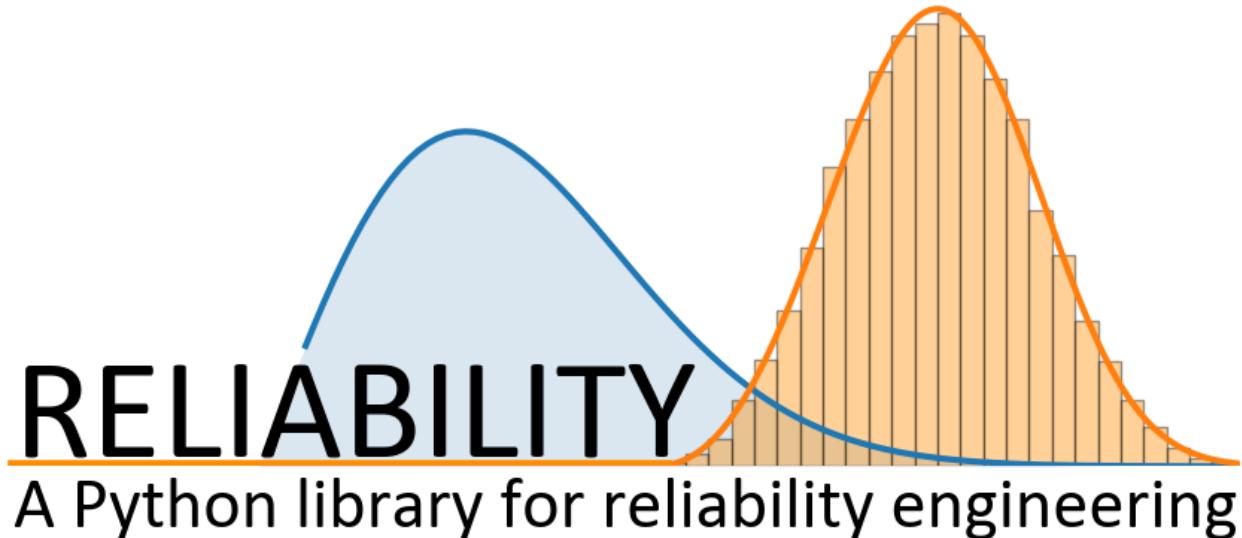
Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters

None

Returns

None



71.4.4 Exponential_Distribution

```
class reliability.Distributions.Exponential_Distribution(Lambda=None, gamma=0, **kwargs)
```

Exponential probability distribution. Creates a probability distribution object.

Parameters

- **Lambda** (*float, int*) – Scale parameter. Must be > 0
- **gamma** (*float, int, optional*) – threshold (offset) parameter. Must be ≥ 0 . Default = 0

Returns

- **name** (*str*) – ‘Exponential’
- **name2** (*‘str’*) – ‘Exponential_1P’ or ‘Exponential_2P’ depending on the value of the gamma parameter
- **param_title_long** (*str*) – ‘Exponential Distribution (=5)’
- **param_title** (*str*) – ‘=5’
- **parameters** (*list*) – [Lambda,gamma]
- **Lambda** (*float*)
- **gamma** (*float*)
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)

- **b5** (float)
- **b95** (float)

Notes

kwargs are used internally to generate the confidence intervals

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **xvals** (array, list, optional) – x-values for plotting
- **xmin** (int, float, optional) – minimum x-value for plotting
- **xmax** (int, float, optional) – maximum x-value for plotting
- **show_plot** (bool, optional) – True or False. Default = True
- **plot_CI** (bool, optional) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI** (float, optional) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (list, array, optional) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and *CI_type*='time'.
- **CI_x** (list, array, optional) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and *CI_type*='reliability'.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (array, float) – The y-values of the plot. Only returned if *CI_x* and *CI_y* are not specified.
- **lower_estimate**, **point_estimate**, **upper_estimate** (tuple) – A tuple of arrays or floats of the confidence interval estimates based on *CI_x* or *CI_y*. Only returned if *CI_x* or *CI_y* is specified and the confidence intervals are available. If *CI_x* is specified, the point estimate is the y-values from the distribution at *CI_x*. If *CI_y* is specified, the point estimate is the x-values from the distribution at *CI_y*.

Notes

The plot will be shown if *show_plot* is True (which it is by default).

If *xvals* is specified, it will be used. If *xvals* is not specified but *xmin* and/or *xmax* are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

CHF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **xvals** (array, list, optional) – x-values for plotting
- **xmin** (int, float, optional) – minimum x-value for plotting

- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and CI_type='time'.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and CI_type='reliability'.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if CI_x and CI_y are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on CI_x or CI_y. Only returned if CI_x or CI_y is specified and the confidence intervals are available. If CI_x is specified, the point estimate is the y-values from the distribution at CI_x. If CI_y is specified, the point estimate is the x-values from the distribution at CI_y.

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, ***kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if *show_plot* is True (which it is by default).

If *xvals* is specified, it will be used. If *xvals* is not specified but *xmin* and/or *xmax* are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the SF (survival function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and *CI_type*='time'.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and *CI_type*='reliability'.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if *CI_x* and *CI_y* are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on *CI_x* or *CI_y*. Only returned if *CI_x* or *CI_y* is specified and the confidence intervals are available. If *CI_x* is specified, the point estimate is the y-values from the distribution at *CI_x*. If *CI_y* is specified, the point estimate is the x-values from the distribution at *CI_y*.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

`inverse_SF(q)`

Inverse survival function calculator

Parameters

`q (float, list, array)` – Quantile to be calculated. Must be between 0 and 1.

Returns

`x (float)` – The inverse of the SF at `q`.

`mean_residual_life(t)`

Mean Residual Life calculator

Parameters

`t (int, float)` – Time (x-value) at which mean residual life is to be evaluated

Returns

`MRL (float)` – The mean residual life

`plot(xvals=None, xmin=None, xmax=None)`

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- `xvals (list, array, optional)` – x-values for plotting
- `xmin (int, float, optional)` – minimum x-value for plotting
- `xmax (int, float, optional)` – maximum x-value for plotting

Returns

`None`

Notes

The plot will be shown. No need to use `plt.show()`. If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

`quantile(q)`

Quantile calculator

Parameters

`q (float, list, array)` – Quantile to be calculated. Must be between 0 and 1.

Returns

`x (float)` – The inverse of the CDF at `q`. This is the probability that a random variable from the distribution is $< q$

`random_samples(number_of_samples, seed=None)`

Draws random samples from the probability distribution

Parameters

- `number_of_samples (int)` – The number of samples to be drawn. Must be greater than 0.

- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

samples (*array*) – The random samples

Notes

This is the same as rvs in scipy.stats

stats()

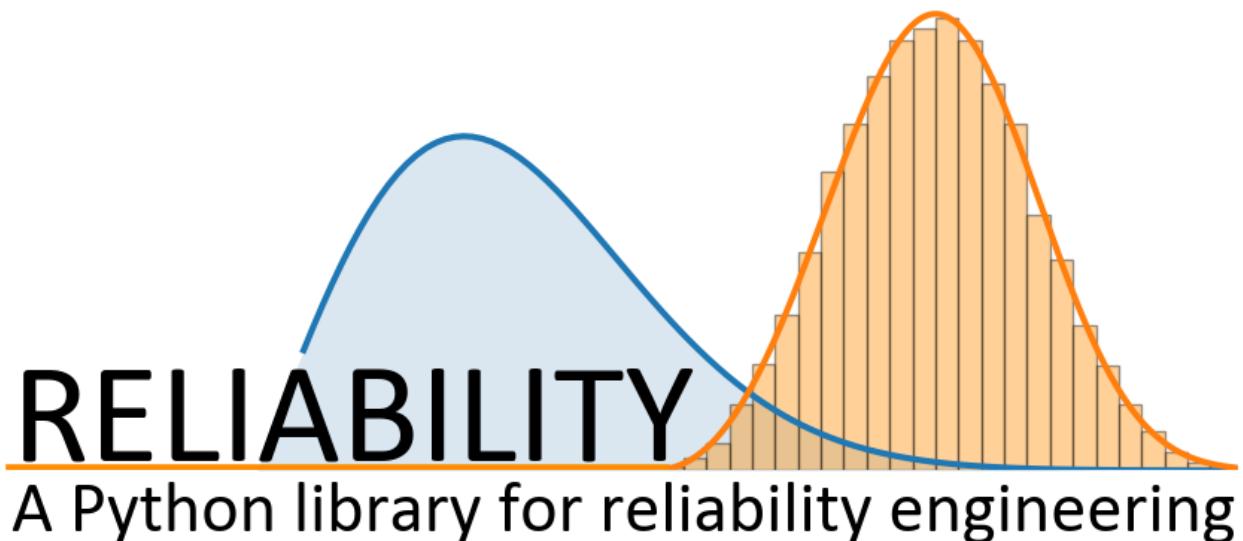
Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters

None

Returns

None



71.4.5 Gamma_Distribution

```
class reliability.Distributions.Gamma_Distribution(alpha=None, beta=None, gamma=0, **kwargs)
```

Gamma probability distribution. Creates a probability distribution object.

Parameters

- **alpha** (*float, int*) – Scale parameter. Must be > 0
- **beta** (*float, int*) – Shape parameter. Must be > 0
- **gamma** (*float, int, optional*) – threshold (offset) parameter. Must be ≥ 0 . Default = 0

Returns

- **name** (*str*) – ‘Gamma’
- **name2** (*‘str’*) – ‘Gamma_2P’ or ‘Gamma_3P’ depending on the value of the gamma parameter
- **param_title_long** (*str*) – ‘Gamma Distribution (=5,=2)’

- **param_title** (*str*) – ‘=5,=2’
- **parameters** (*list*) – [alpha,beta,gamma]
- **alpha** (*float*)
- **beta** (*float*)
- **gamma** (*float*)
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

kwargs are used internally to generate the confidence intervals

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI_type=None*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if **CI_x** and **CI_y** are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on **CI_x** or **CI_y**. Only returned if **CI_x** or **CI_y** is specified and the confidence intervals are available. If **CI_x** is specified, the point estimate is the y-values from the distribution at **CI_x**. If **CI_y** is specified, the point estimate is the x-values from the distribution at **CI_y**.

Notes

The plot will be shown if **show_plot** is True (which it is by default).

If **xvals** is specified, it will be used. If **xvals** is not specified but **xmin** and/or **xmax** are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

CHF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and **CI_type**=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and **CI_type**=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if **CI_x** and **CI_y** are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on **CI_x** or **CI_y**. Only returned if **CI_x** or **CI_y** is specified and the confidence intervals are available. If **CI_x** is specified, the point estimate is the y-values from the distribution at **CI_x**. If **CI_y** is specified, the point estimate is the x-values from the distribution at **CI_y**.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF(`xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs`)

Plots the HF (hazard function)

Parameters

- `show_plot` (*bool, optional*) – True or False. Default = True
- `xvals` (*array, list, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting
- `xmax` (*int, float, optional*) – maximum x-value for plotting
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

`yvals` (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(`xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs`)

Plots the PDF (probability density function)

Parameters

- `show_plot` (*bool, optional*) – True or False. Default = True
- `xvals` (*array, list, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting
- `xmax` (*int, float, optional*) – maximum x-value for plotting
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

`yvals` (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(`xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs`)

Plots the SF (survival function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and CI_type=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and CI_type=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if CI_x and CI_y are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on CI_x or CI_y. Only returned if CI_x or CI_y is specified and the confidence intervals are available. If CI_x is specified, the point estimate is the y-values from the distribution at CI_x. If CI_y is specified, the point estimate is the x-values from the distribution at CI_y.

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution’s parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

- **x** (*float*) – The inverse of the SF at q.

mean_residual_life(*t*)

Mean Residual Life calculator

Parameters

- **t** (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

- **MRL** (*float*) – The mean residual life

plot(xvals=None, xmin=None, xmax=None)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use plt.show(). If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

quantile(q)

Quantile calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the CDF at q. This is the probability that a random variable from the distribution is < q

random_samples(number_of_samples, seed=None)

Draws random samples from the probability distribution

Parameters

- **number_of_samples** (*int*) – The number of samples to be drawn. Must be greater than 0.
- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

samples (*array*) – The random samples

Notes

This is the same as rvs in scipy.stats

stats()

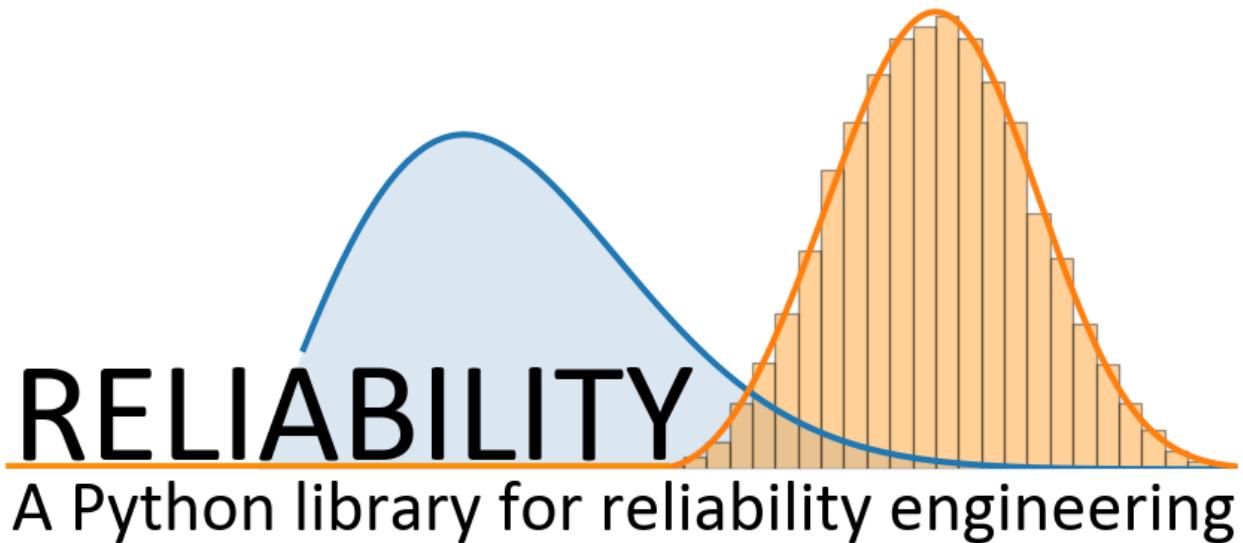
Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters

None

Returns

None



71.4.6 Gumbel_Distribution

```
class reliability.Distributions.Gumbel_Distribution(mu=None, sigma=None, **kwargs)
```

Gumbel probability distribution. Creates a probability distribution object.

Parameters

- **mu** (*float, int*) – Location parameter
- **sigma** (*float, int*) – Scale parameter. Must be > 0

Returns

- **name** (*str*) – ‘Gumbel’
- **name2** (*‘str’*) – ‘Gumbel_2P’
- **param_title_long** (*str*) – ‘Gumbel Distribution (=5,=2)’
- **param_title** (*str*) – ‘=5,=2’
- **parameters** (*list*) – [mu,sigma]
- **mu** (*float*)
- **sigma** (*float*)
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)

- **b95** (*float*)

Notes

kwargs are used internally to generate the confidence intervals

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI_type=None*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if *CI_x* and *CI_y* are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on *CI_x* or *CI_y*. Only returned if *CI_x* or *CI_y* is specified and the confidence intervals are available. If *CI_x* is specified, the point estimate is the y-values from the distribution at *CI_x*. If *CI_y* is specified, the point estimate is the x-values from the distribution at *CI_y*.

Notes

The plot will be shown if *show_plot* is True (which it is by default).

If *xvals* is specified, it will be used. If *xvals* is not specified but *xmin* and/or *xmax* are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution’s parameters.

CHF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI_type=None*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting

- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and CI_type='time'.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and CI_type='reliability'.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if CI_x and CI_y are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on CI_x or CI_y. Only returned if CI_x or CI_y is specified and the confidence intervals are available. If CI_x is specified, the point estimate is the y-values from the distribution at CI_x. If CI_y is specified, the point estimate is the x-values from the distribution at CI_y.

Notes

The plot will be shown if show_plot is True (which it is by default).

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution’s parameters.

HF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(`xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs`)

Plots the PDF (probability density function)

Parameters

- `show_plot` (*bool, optional*) – True or False. Default = True
- `xvals` (*array, list, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting
- `xmax` (*int, float, optional*) – maximum x-value for plotting
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

`yvals` (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(`xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs`)

Plots the SF (survival function)

Parameters

- `xvals` (*array, list, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting
- `xmax` (*int, float, optional*) – maximum x-value for plotting
- `show_plot` (*bool, optional*) – True or False. Default = True
- `plot_CI` (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- `CI_type` (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- `CI` (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- `CI_y` (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- `CI_x` (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is `True` (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

`inverse_SF(q)`

Inverse survival function calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

- **x** (*float*) – The inverse of the SF at `q`.

`mean_residual_life(t)`

Mean Residual Life calculator

Parameters

- **t** (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

- **MRL** (*float*) – The mean residual life

`plot(xvals=None, xmin=None, xmax=None)`

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

- **None**

Notes

The plot will be shown. No need to use `plt.show()`. If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

`quantile(q)`

Quantile calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

`x (float)` – The inverse of the CDF at `q`. This is the probability that a random variable from the distribution is $< q$

random_samples(*number_of_samples*, *seed=None*)

Draws random samples from the probability distribution

Parameters

- `number_of_samples (int)` – The number of samples to be drawn. Must be greater than 0.
- `seed (int, optional)` – The random seed passed to numpy. Default = None

Returns

`samples (array)` – The random samples

Notes

This is the same as `rvs` in `scipy.stats`

stats()

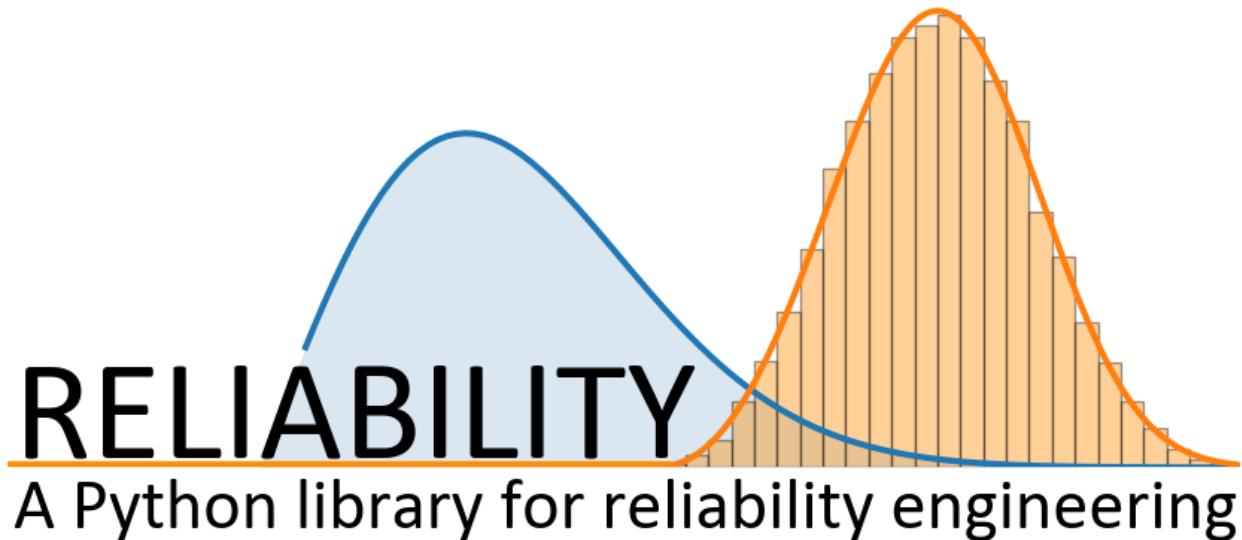
Descriptive statistics of the probability distribution. These are the same as the statistics shown using `.plot()` but printed to the console.

Parameters

`None`

Returns

`None`



71.4.7 Loglogistic_Distribution

```
class reliability.Distributions.Loglogistic_Distribution(alpha=None, beta=None, gamma=0,  
**kwargs)
```

Loglogistic probability distribution. Creates a probability distribution object.

Parameters

- `alpha (float, int)` – Scale parameter. Must be > 0

- **beta** (*float, int*) – Shape parameter. Must be > 0
- **gamma** (*float, int, optional*) – threshold (offset) parameter. Must be ≥ 0 . Default = 0

Returns

- **name** (*str*) – ‘Loglogistic’
- **name2** (*‘str*) – ‘Loglogistic_2P’ or ‘Loglogistic_3P’ depending on the value of the gamma parameter
- **param_title_long** (*str*) – ‘Loglogistic Distribution (=5,=2)’
- **param_title** (*str*) – ‘=5,=2’
- **parameters** (*list*) – [alpha,beta,gamma]
- **alpha** (*float*)
- **beta** (*float*)
- **gamma** (*float*)
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

kwargs are used internally to generate the confidence intervals

`CDF(xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs)`

Plots the CDF (cumulative distribution function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.

- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

CHF(`xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI=None, CI_y=None, CI_x=None, **kwargs`)

Plots the CHF (cumulative hazard function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is `True` (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = `True`
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to `matplotlib` (e.g. `color`, `linestyle`)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is `True` (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = `True`
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to `matplotlib` (e.g. `color`, `linestyle`)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(`xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs`)

Plots the SF (survival function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the SF at q.

mean_residual_life(*t*)

Mean Residual Life calculator

Parameters

t (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

MRL (*float*) – The mean residual life

plot(*xvals=None, xmin=None, xmax=None*)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use plt.show(). If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

quantile(*q*)

Quantile calculator

Parameters

q (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the CDF at q. This is the probability that a random variable from the distribution is < q

random_samples(*number_of_samples, seed=None*)

Draws random samples from the probability distribution

Parameters

- **number_of_samples** (*int*) – The number of samples to be drawn. Must be greater than 0.
- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

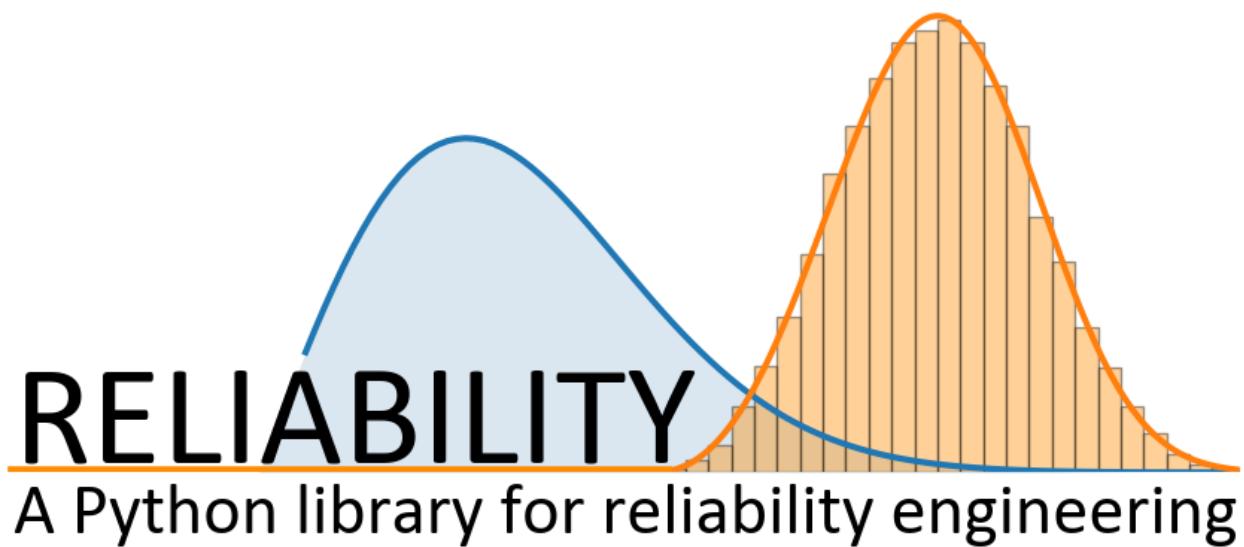
samples (*array*) – The random samples

Notes

This is the same as rvs in scipy.stats

stats()

Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters`None`**Returns**`None`

71.4.8 Lognormal_Distribution

```
class reliability.Distributions.Lognormal_Distribution(mu=None, sigma=None, gamma=0,  
**kwargs)
```

Lognormal probability distribution. Creates a probability distribution object.

Parameters

- **mu** (*float, int*) – Location parameter
- **sigma** (*float, int*) – Scale parameter. Must be > 0
- **gamma** (*float, int, optional*) – threshold (offset) parameter. Must be ≥ 0 . Default = 0

Returns

- **name** (*str*) – ‘Lognormal’
- **name2** (*str*) – ‘Lognormal_2P’ or ‘Lognormal_3P’ depending on the value of the gamma parameter
- **param_title_long** (*str*) – ‘Lognormal Distribution (=5,=2)’
- **param_title** (*str*) – ‘=5,=2’
- **parameters** (*list*) – [mu,sigma,gamma]
- **mu** (*float*)
- **sigma** (*float*)
- **gamma** (*float*)
- **mean** (*float*)
- **variance** (*float*)

- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

kwargs are used internally to generate the confidence intervals

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI_type=None*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if *CI_x* and *CI_y* are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on *CI_x* or *CI_y*. Only returned if *CI_x* or *CI_y* is specified and the confidence intervals are available. If *CI_x* is specified, the point estimate is the y-values from the distribution at *CI_x*. If *CI_y* is specified, the point estimate is the x-values from the distribution at *CI_y*.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

`CHF(xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs)`

Plots the CHF (cumulative hazard function)

Parameters

- `xvals` (*array, list, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting
- `xmax` (*int, float, optional*) – maximum x-value for plotting
- `show_plot` (*bool, optional*) – True or False. Default = True
- `plot_CI` (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- `CI_type` (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- `CI` (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- `CI_y` (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- `CI_x` (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- `yvals` (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- `lower_estimate, point_estimate, upper_estimate` (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

`HF(xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs)`

Plots the HF (hazard function)

Parameters

- `show_plot` (*bool, optional*) – True or False. Default = True

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs*)

Plots the SF (survival function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.

- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is `True` (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

- **x** (*float*) – The inverse of the SF at `q`.

mean_residual_life(*t*)

Mean Residual Life calculator

Parameters

- **t** (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

- **MRL** (*float*) – The mean residual life

plot(*xvals=None, xmin=None, xmax=None*)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use plt.show(). If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

`quantile(q)`

Quantile calculator

Parameters

`q (float, list, array)` – Quantile to be calculated. Must be between 0 and 1.

Returns

`x (float)` – The inverse of the CDF at `q`. This is the probability that a random variable from the distribution is $< q$

`random_samples(number_of_samples, seed=None)`

Draws random samples from the probability distribution

Parameters

- `number_of_samples (int)` – The number of samples to be drawn. Must be greater than 0.
- `seed (int, optional)` – The random seed passed to numpy. Default = None

Returns

`samples (array)` – The random samples

Notes

This is the same as `rvs` in `scipy.stats`

`stats()`

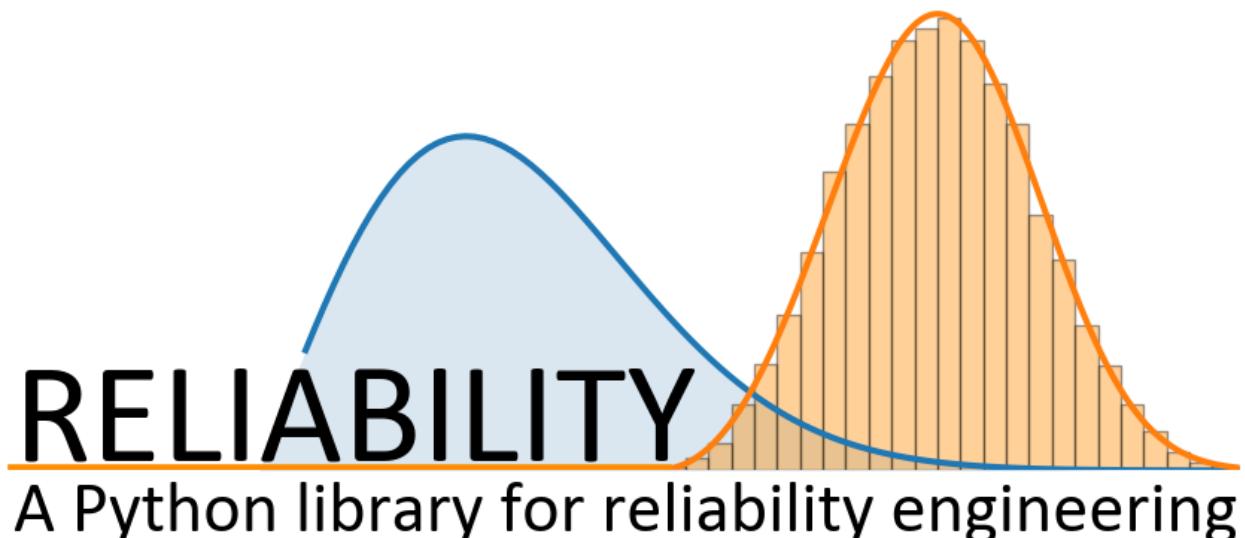
Descriptive statistics of the probability distribution. These are the same as the statistics shown using `.plot()` but printed to the console.

Parameters

`None`

Returns

`None`



71.4.9 Mixture_Model

```
class reliability.Distributions.Mixture_Model(distributions, proportions=None)
```

The mixture model is used to create a distribution that contains parts from multiple distributions. This allows for a more complex model to be constructed as the sum of other distributions, each multiplied by a proportion (where the proportions sum to 1). The model is obtained using the sum of the cumulative distribution functions:

$$CDF_{total} = (CDF_1 p_1) + (CDF_2 p_2) + (CDF_3 p_3) + \dots + (CDF_n p_n)$$

The output API is similar to the other probability distributions (Weibull, Normal, etc.) as shown below.

Parameters

- **distributions** (*list, array*) – List or array of probability distribution objects used to construct the model.
- **proportions** (*list, array*) – List or array of floats specifying how much of each distribution to add to the mixture. The sum of proportions must always be 1.

Returns

- **name** (*str*) – ‘Mixture’
- **name2** (*str*) – ‘Mixture using 3 distributions’. The exact name depends on the number of distributions used.
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

An equivalent form of this model is to sum the PDF. SF is obtained as 1-CDF. Note that you cannot simply sum the HF or CHF as this method would be equivalent to the competing risks model. In this way, we see the mixture model will always lie somewhere between the constituent models.

This model should be used when a data set cannot be modelled by a single distribution, as evidenced by the shape of the PDF, CDF or probability plot (points do not form a straight line). Unlike the competing risks model, this model requires the proportions to be supplied.

As this process is additive for the survival function, and may accept many distributions of different types, the mathematical formulation quickly gets complex. For this reason, the algorithm combines the models numerically rather than empirically so there are no simple formulas for many of the descriptive statistics (mean, median, etc.). Also, the accuracy of the model is dependent on *xvals*. If the *xvals* array is small (<100 values) then the answer will be ‘blocky’ and inaccurate. The variable *xvals* is only accepted for PDF, CDF, SF, HF, CHF. The other

methods (like random samples) use the default xvals for maximum accuracy. The default number of values generated when xvals is not given is 1000. Consider this carefully when specifying xvals in order to avoid inaccuracies in the results.

CDF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default) and len(xvals) ≥ 2 .

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

CHF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default) and len(xvals) ≥ 2 .

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default) and len(xvals) ≥ 2 .

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if show_plot is True (which it is by default) and len(xvals) ≥ 2 .

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_components=False, **kwargs*)

Plots the SF (survival function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True

- **plot_components** (*bool*) – Option to plot the components of the model. True or False. Default = False.
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns**yvals** (*array, float*) – The y-values of the plot**Notes**

The plot will be shown if show_plot is True (which it is by default) and len(xvals) ≥ 2 .

If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters**q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.**Returns****x** (*float*) – The inverse of the SF at *q*.**mean_residual_life(*t*)**

Mean Residual Life calculator

Parameters**t** (*int, float*) – Time (x-value) at which mean residual life is to be evaluated**Returns****MRL** (*float*) – The mean residual life**plot(*xvals=None, xmin=None, xmax=None*)**

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns*None***Notes**

The plot will be shown. No need to use plt.show(). If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

quantile(*q*)

Quantile calculator

Parameters

q (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the CDF at *q*. This is the probability that a random variable from the distribution is < *q*

random_samples(*number_of_samples*, *seed=None*)

Draws random samples from the probability distribution

Parameters

- **number_of_samples** (*int*) – The number of samples to be drawn. Must be greater than 0.
- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

samples (*array*) – The random samples

Notes

This is the same as rvs in scipy.stats

stats()

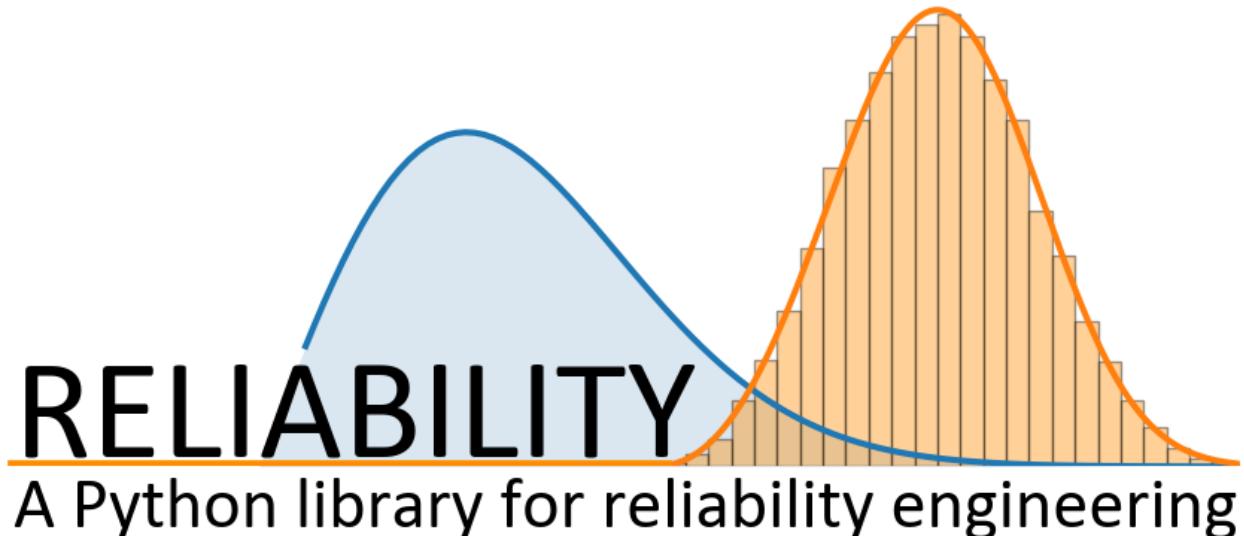
Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters

None

Returns

None



71.4.10 Normal_Distribution

```
class reliability.Distributions.Normal_Distribution(mu=None, sigma=None, **kwargs)
```

Normal probability distribution. Creates a probability distribution object.

Parameters

- **mu** (*float, int*) – Location parameter
- **sigma** (*float, int*) – Scale parameter. Must be > 0

Returns

- **name** (*str*) – ‘Normal’
- **name2** (*‘str’*) – ‘Normal_2P’
- **param_title_long** (*str*) – ‘Normal Distribution (=5,=2)’
- **param_title** (*str*) – ‘=5,=2’
- **parameters** (*list*) – [mu,sigma]
- **mu** (*float*)
- **sigma** (*float*)
- **mean** (*float*)
- **variance** (*float*)
- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

kwargs are used internally to generate the confidence intervals

```
CDF(xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs)
```

Plots the CDF (cumulative distribution function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.

- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and **CI_type**=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and **CI_type**=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if **CI_x** and **CI_y** are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on **CI_x** or **CI_y**. Only returned if **CI_x** or **CI_y** is specified and the confidence intervals are available. If **CI_x** is specified, the point estimate is the y-values from the distribution at **CI_x**. If **CI_y** is specified, the point estimate is the x-values from the distribution at **CI_y**.

Notes

The plot will be shown if **show_plot** is True (which it is by default).

If **xvals** is specified, it will be used. If **xvals** is not specified but **xmin** and/or **xmax** are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution’s parameters.

CHF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs*)

Plots the CHF (cumulative hazard function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and **CI_type**=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and **CI_type**=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is `True` (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

HF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the HF (hazard function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = `True`
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is `True` (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = `True`
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(`xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs`)

Plots the SF (survival function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the SF at q.

mean_residual_life(*t*)

Mean Residual Life calculator

Parameters

t (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

MRL (*float*) – The mean residual life

plot(*xvals=None, xmin=None, xmax=None*)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use plt.show(). If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

quantile(*q*)

Quantile calculator

Parameters

q (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

x (*float*) – The inverse of the CDF at q. This is the probability that a random variable from the distribution is < q

random_samples(*number_of_samples, seed=None*)

Draws random samples from the probability distribution

Parameters

- **number_of_samples** (*int*) – The number of samples to be drawn. Must be greater than 0.
- **seed** (*int, optional*) – The random seed passed to numpy. Default = None

Returns

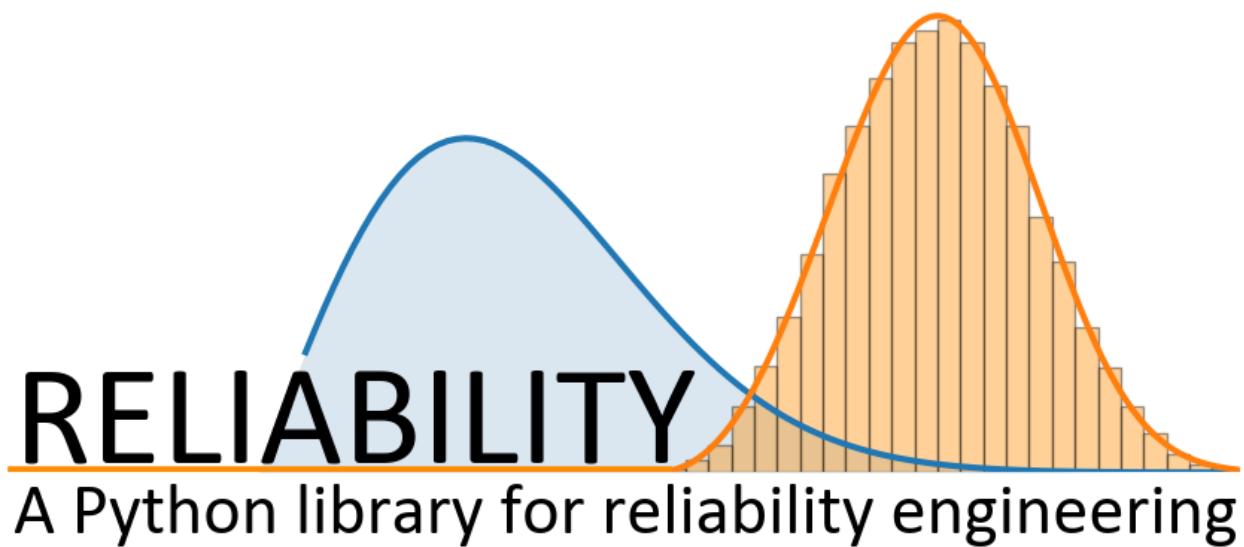
samples (*array*) – The random samples

Notes

This is the same as rvs in scipy.stats

stats()

Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters`None`**Returns**`None`

71.4.11 Weibull_Distribution

```
class reliability.Distributions.Weibull_Distribution(alpha=None, beta=None, gamma=0,  
**kwargs)
```

Weibull probability distribution. Creates a probability distribution object.

Parameters

- **alpha** (*float, int*) – Scale parameter. Must be > 0
- **beta** (*float, int*) – Shape parameter. Must be > 0
- **gamma** (*float, int, optional*) – threshold (offset) parameter. Must be ≥ 0 . Default = 0

Returns

- **name** (*str*) – ‘Weibull’
- **name2** (*‘str’*) – ‘Weibull_2P’ or ‘Weibull_3P’ depending on the value of the gamma parameter
- **param_title_long** (*str*) – ‘Weibull Distribution (=5,=2)’
- **param_title** (*str*) – ‘=5,=2’
- **parameters** (*list*) – [alpha,beta,gamma]
- **alpha** (*float*)
- **beta** (*float*)
- **gamma** (*float*)
- **mean** (*float*)
- **variance** (*float*)

- **standard_deviation** (*float*)
- **skewness** (*float*)
- **kurtosis** (*float*)
- **excess_kurtosis** (*float*)
- **median** (*float*)
- **mode** (*float*)
- **b5** (*float*)
- **b95** (*float*)

Notes

kwargs are used internally to generate the confidence intervals

CDF(*xvals=None*, *xmin=None*, *xmax=None*, *show_plot=True*, *plot_CI=True*, *CI_type=None*, *CI=None*, *CI_y=None*, *CI_x=None*, ***kwargs*)

Plots the CDF (cumulative distribution function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’time’.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and *CI_type*=’reliability’.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if *CI_x* and *CI_y* are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on *CI_x* or *CI_y*. Only returned if *CI_x* or *CI_y* is specified and the confidence intervals are available. If *CI_x* is specified, the point estimate is the y-values from the distribution at *CI_x*. If *CI_y* is specified, the point estimate is the x-values from the distribution at *CI_y*.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

`CHF(xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs)`

Plots the CHF (cumulative hazard function)

Parameters

- `xvals` (*array, list, optional*) – x-values for plotting
- `xmin` (*int, float, optional*) – minimum x-value for plotting
- `xmax` (*int, float, optional*) – maximum x-value for plotting
- `show_plot` (*bool, optional*) – True or False. Default = True
- `plot_CI` (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- `CI_type` (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.
- `CI` (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- `CI_y` (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- `CI_x` (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- `yvals` (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- `lower_estimate, point_estimate, upper_estimate` (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

`HF(xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs)`

Plots the HF (hazard function)

Parameters

- `show_plot` (*bool, optional*) – True or False. Default = True

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

PDF(*xvals=None, xmin=None, xmax=None, show_plot=True, **kwargs*)

Plots the PDF (probability density function)

Parameters

- **show_plot** (*bool, optional*) – True or False. Default = True
- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

yvals (*array, float*) – The y-values of the plot

Notes

The plot will be shown if `show_plot` is True (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

SF(*xvals=None, xmin=None, xmax=None, show_plot=True, plot_CI=True, CI_type=None, CI=None, CI_y=None, CI_x=None, **kwargs*)

Plots the SF (survival function)

Parameters

- **xvals** (*array, list, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting
- **show_plot** (*bool, optional*) – True or False. Default = True
- **plot_CI** (*bool, optional*) – True or False. Default = True. Only used if the distribution object was created by Fitters.
- **CI_type** (*str, optional*) – Must be either “time”, “reliability”, or “none”. Default is “time”. Only used if the distribution object was created by Fitters.

- **CI** (*float, optional*) – The confidence interval between 0 and 1. Only used if the distribution object was created by Fitters.
- **CI_y** (*list, array, optional*) – The confidence interval y-values to trace. Only used if the distribution object was created by Fitters and `CI_type='time'`.
- **CI_x** (*list, array, optional*) – The confidence interval x-values to trace. Only used if the distribution object was created by Fitters and `CI_type='reliability'`.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, linestyle)

Returns

- **yvals** (*array, float*) – The y-values of the plot. Only returned if `CI_x` and `CI_y` are not specified.
- **lower_estimate, point_estimate, upper_estimate** (*tuple*) – A tuple of arrays or floats of the confidence interval estimates based on `CI_x` or `CI_y`. Only returned if `CI_x` or `CI_y` is specified and the confidence intervals are available. If `CI_x` is specified, the point estimate is the y-values from the distribution at `CI_x`. If `CI_y` is specified, the point estimate is the x-values from the distribution at `CI_y`.

Notes

The plot will be shown if `show_plot` is `True` (which it is by default).

If `xvals` is specified, it will be used. If `xvals` is not specified but `xmin` and/or `xmax` are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters.

inverse_SF(*q*)

Inverse survival function calculator

Parameters

- **q** (*float, list, array*) – Quantile to be calculated. Must be between 0 and 1.

Returns

- **x** (*float, array*) – The inverse of the SF at `q`.

mean_residual_life(*t*)

Mean Residual Life calculator

Parameters

- **t** (*int, float*) – Time (x-value) at which mean residual life is to be evaluated

Returns

- **MRL** (*float*) – The mean residual life

plot(*xvals=None, xmin=None, xmax=None*)

Plots all functions (PDF, CDF, SF, HF, CHF) and descriptive statistics in a single figure

Parameters

- **xvals** (*list, array, optional*) – x-values for plotting
- **xmin** (*int, float, optional*) – minimum x-value for plotting
- **xmax** (*int, float, optional*) – maximum x-value for plotting

Returns

None

Notes

The plot will be shown. No need to use plt.show(). If xvals is specified, it will be used. If xvals is not specified but xmin and/or xmax are specified then an array with 200 elements will be created using these limits. If nothing is specified then the range will be based on the distribution's parameters. No plotting keywords are accepted.

`quantile(q)`

Quantile calculator

Parameters

`q (float, list, array)` – Quantile to be calculated. Must be between 0 and 1.

Returns

`x (float, array)` – The inverse of the CDF at `q`. This is the probability that a random variable from the distribution is $< q$

`random_samples(number_of_samples, seed=None)`

Draws random samples from the probability distribution

Parameters

- `number_of_samples (int)` – The number of samples to be drawn. Must be greater than 0.
- `seed (int, optional)` – The random seed passed to numpy. Default = None

Returns

`samples (array)` – The random samples

Notes

This is the same as rvs in scipy.stats

`stats()`

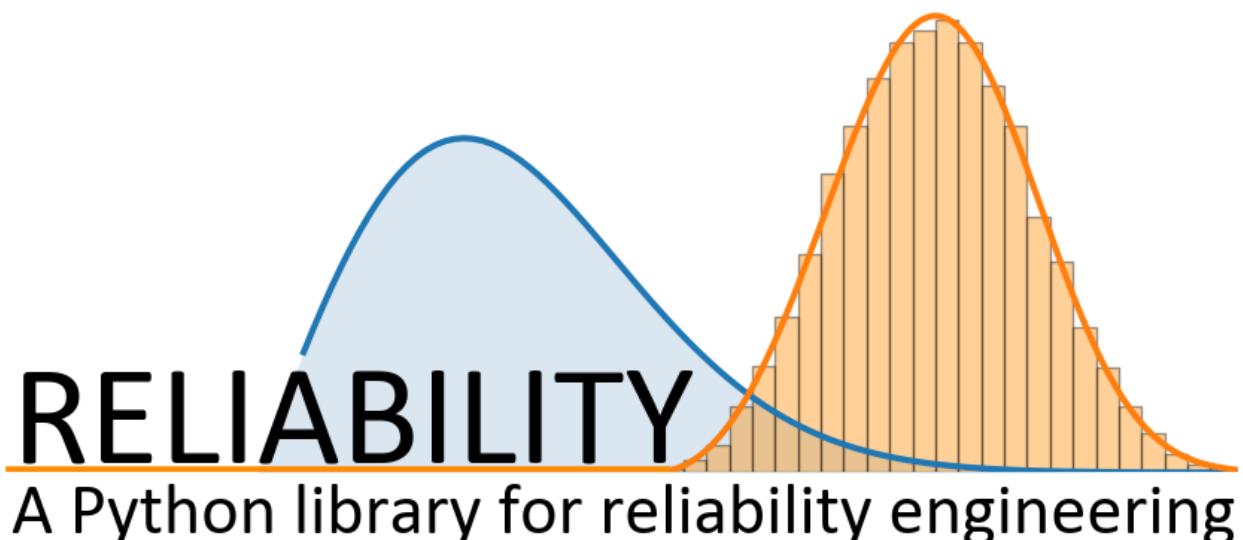
Descriptive statistics of the probability distribution. These are the same as the statistics shown using .plot() but printed to the console.

Parameters

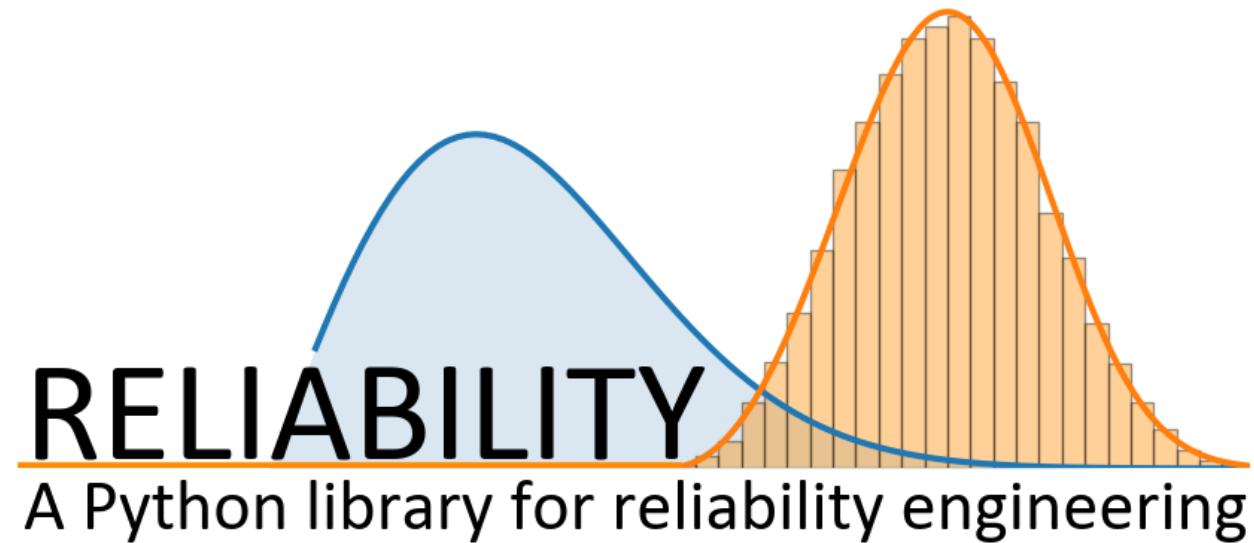
`None`

Returns

`None`



71.5 Fitters



71.5.1 Fit_Beta_2P

```
class reliability.Fitters.Fit_Beta_2P(failures=None, right_censored=None,  
                                      show_probability_plot=True, print_results=True, CI=0.95,  
                                      quantiles=None, method='MLE', optimizer=None,  
                                      downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Beta distribution (alpha,beta) to the data provided. All data must be in the range $0 < x < 1$.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with point estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Beta_2P alpha parameter
- **beta** (*float*) – the fitted Beta_2P beta parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Beta_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

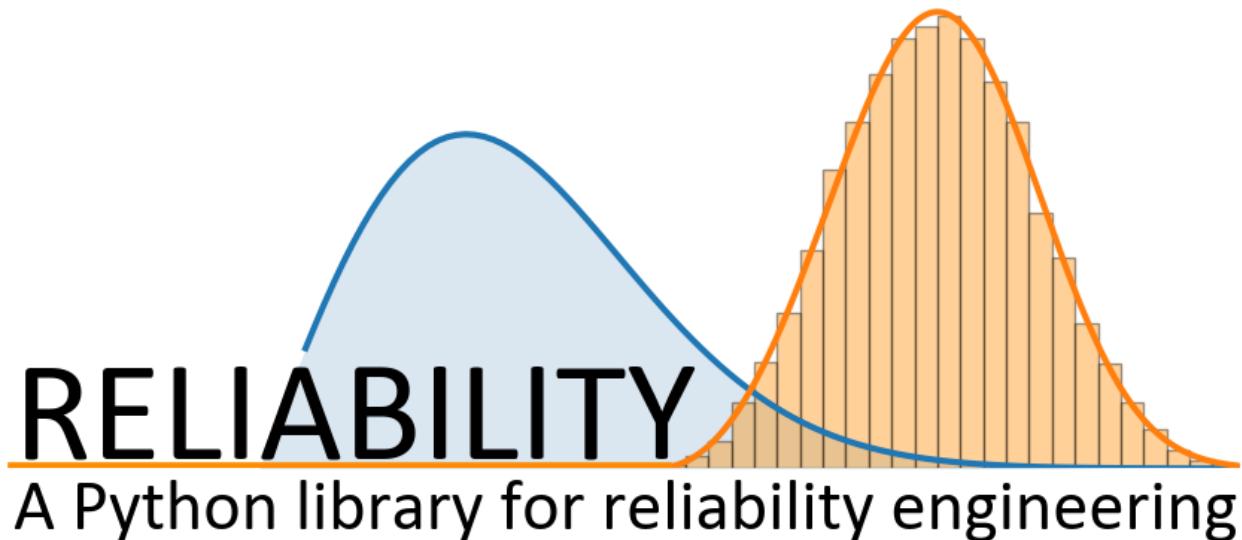
If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

Confidence intervals on the plots are not provided.

```
static LL(params, T_f, T_rc)
```

```
static logR(t, a, b)
```

```
static logf(t, a, b)
```



71.5.2 Fit_Everything

```
class reliability.Fitters.Fit_Everything(failures=None, right_censored=None, exclude=None,
                                         sort_by='BIC', method='MLE', optimizer=None,
                                         print_results=True, show_histogram_plot=True,
                                         show_PP_plot=True, show_probability_plot=True,
                                         show_best_distribution_probability_plot=True,
                                         downsample_scatterplot=True)
```

This function will fit all available distributions to the data provided. The only distributions not fitted are Weibull_DSZI and Weibull_ZI. The Beta_2P distribution will only be fitted if the data are between 0 and 1.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements for all the 2 parameter distributions to be fitted and 3 elements for all distributions to be fitted.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **sort_by** (*str*) – Goodness of fit test to sort results by. Must be ‘BIC’, ‘AICc’, ‘AD’, or ‘Log-likelihood’. Default is BIC.
- **show_probability_plot** (*bool, optional*) – Provides a probability plot of each of the fitted distributions. True or False. Default = True

- **show_histogram_plot** (*bool, optional*) – True or False. Default = True. Will show a histogram (scaled to account for censored data) with the PDF and CDF of each fitted distribution.
- **show_PP_plot** (*bool, optional*) – Provides a comparison of parametric vs non-parametric fit using Probability-Probability (PP) plot. True or False. Default = True.
- **show_best_distribution_probability_plot** (*bool, optional*) – Provides a probability plot in a new figure of the best fitting distribution. True or False. Default = True.
- **exclude** (*list, array, optional*) – List or array of strings specifying which distributions to exclude. Default is None. Options are Weibull_2P, Weibull_3P, Weibull_CR, Weibull_Mixture, Weibull_DS, Normal_2P, Gamma_2P, Loglogistic_2P, Gamma_3P, Lognormal_2P, Lognormal_3P, Loglogistic_3P, Gumbel_2P, Exponential_2P, Exponential_1P, Beta_2P.
- **print_results** (*bool, optional*) – Will show the results of the fitted parameters and the goodness of fit tests in a dataframe. True/False. Defaults to True.
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.

Returns

- **results** (*dataframe*) – a pandas dataframe of results. Fitted parameters in this dataframe may be accessed by name. See below example in Notes.
- **best_distribution** (*object*) – a reliability.Distributions object created based on the parameters of the best fitting distribution.
- **best_distribution_name** (*str*) – the name of the best fitting distribution. E.g. ‘Weibull_3P’
- **parameters and goodness of fit results** (*float*) – This is provided for each fitted distribution. For example, the Weibull_3P distribution values are Weibull_3P_alpha, Weibull_3P_beta, Weibull_3P_gamma, Weibull_3P_BIC, Weibull_3P_AICc, Weibull_3P_AD, Weibull_3P_loglik
- **excluded_distributions** (*list*) – a list of strings of the excluded distributions.
- **probability_plot** (*object*) – The figure handle from the probability plot (only provided if show_probability_plot is True).
- **best_distribution_probability_plot** (*object*) – The figure handle from the best distribution probability plot (only provided if show_best_distribution_probability_plot is True).
- **histogram_plot** (*object*) – The figure handle from the histogram plot (only provided if show_histogram_plot is True).

- **PP_plot** (*object*) – The figure handle from the probability-probability plot (only provided if `show_PP_plot` is True).

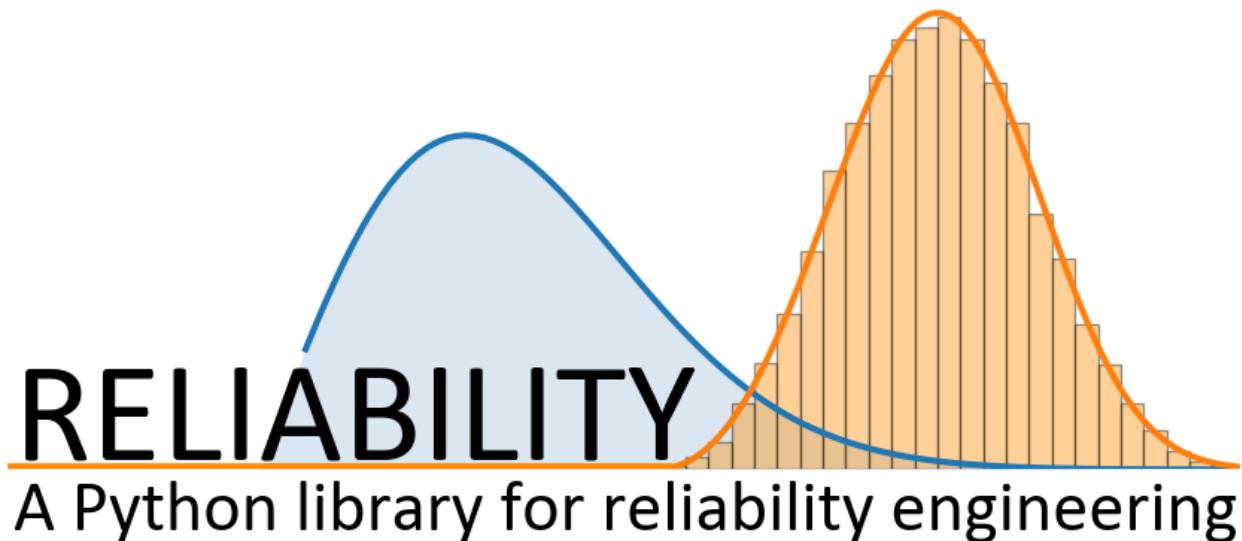
Notes

All parametric models have the number of parameters in the name. For example, `Weibull_2P` uses `alpha` and `beta`, whereas `Weibull_3P` uses `alpha`, `beta`, and `gamma`. This is applied even for `Normal_2P` for consistency in naming conventions. From the results, the distributions are sorted based on their goodness of fit test results, where the smaller the goodness of fit value, the better the fit of the distribution to the data.

If the data provided contains only 2 failures, the three parameter distributions will automatically be excluded.

Example Usage:

```
X = [5,3,8,6,7,4,5,4,2]
output = Fit_Everything(X)
print('Weibull Alpha =',output.Weibull_2P_alpha)
```



71.5.3 Fit_Exponential_1P

```
class reliability.Fitters.Fit_Exponential_1P(failures=None, right_censored=None,
                                              show_probability_plot=True, print_results=True,
                                              CI=0.95, quantiles=None, method='MLE',
                                              optimizer=None, downsample_scatterplot=True,
                                              **kwargs)
```

Fits a one parameter Exponential distribution (Lambda) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 1 element.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = `None`.
- **show_probability_plot** (*bool, optional*) – True or False. Default = `True`
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for the model's parameter. True or False. Default = `True`

- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **Lambda** (*float*) – the fitted Exponential_1P Lambda parameter
- **Lambda_inv** (*float*) – the inverse of the fitted Exponential_1P Lambda parameter
- **Lambda_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Lambda_SE_inv** (*float*) – the standard error (sqrt(variance)) of the inverse of the parameter
- **Lambda_upper** (*float*) – the upper CI estimate of the parameter
- **Lambda_lower** (*float*) – the lower CI estimate of the parameter
- **Lambda_upper_inv** (*float*) – the upper CI estimate of the inverse of the parameter
- **Lambda_lower_inv** (*float*) – the lower CI estimate of the inverse of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Exponential_Distribution object with the parameter of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)

- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

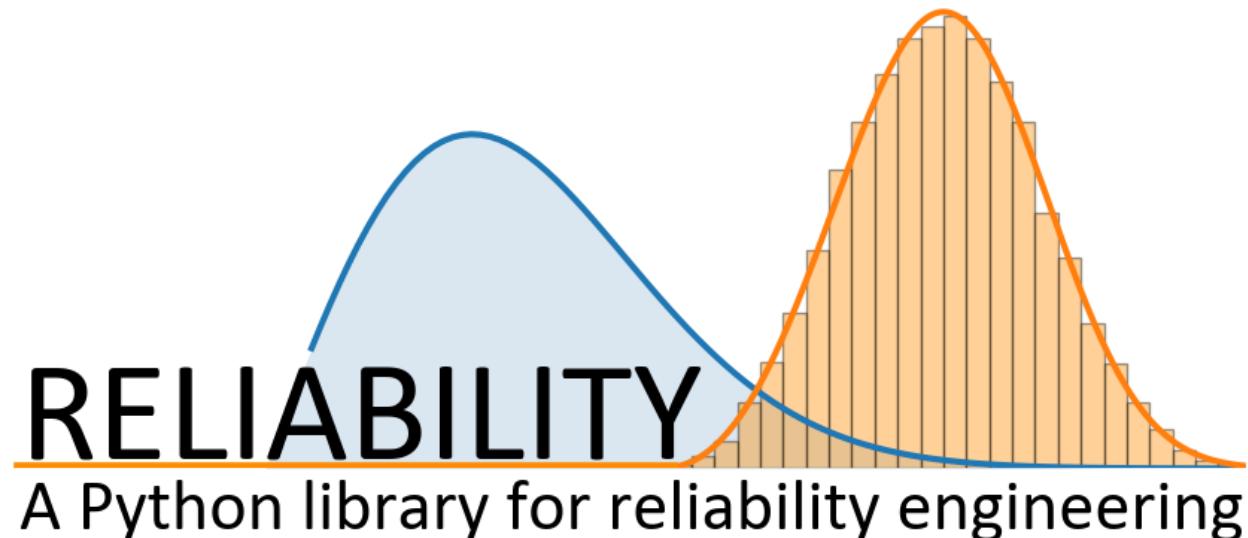
This is a one parameter distribution, but the results provide both the parameter (Lambda) as well as the inverse (1/Lambda). This is provided for convenience as some other software (Minitab and scipy.stats) use 1/Lambda instead of Lambda. Lambda_SE_inv, Lambda_upper_inv, and Lambda_lower_inv are also provided for convenience.

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)
```

```
static logR(t, L)
```

```
static logf(t, L)
```



71.5.4 Fit_Exponential_2P

```
class reliability.Fitters.Fit_Exponential_2P(failures=None, right_censored=None, show_probability_plot=True, print_results=True, CI=0.95, quantiles=None, method='MLE', optimizer=None, downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Exponential distribution (Lambda, gamma) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 1 element.

- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for the model's parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either 'MLE' (maximum likelihood estimation), 'LS' (least squares estimation), 'RRX' (Rank regression on X), or 'RRY' (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is 'MLE'.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either 'TNC', 'L-BFGS-B', 'nelder-mead', or 'powell'. Specifying the optimizer will result in that optimizer being used. To use all of these specify 'best' and the best result will be returned. The default behaviour is to try each optimizer in order ('TNC', 'L-BFGS-B', 'nelder-mead', and 'powell') and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1,..., 0.95, 0.99] set quantiles as either 'auto', True, 'default', 'on'. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **Lambda** (*float*) – the fitted Exponential_1P Lambda parameter
- **Lambda_inv** (*float*) – the inverse of the fitted Exponential_1P Lambda parameter
- **gamma** (*float*) – the fitted Exponential_2P gamma parameter
- **Lambda_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Lambda_SE_inv** (*float*) – the standard error (sqrt(variance)) of the inverse of the parameter
- **gamma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Lambda_upper** (*float*) – the upper CI estimate of the parameter
- **Lambda_lower** (*float*) – the lower CI estimate of the parameter
- **Lambda_upper_inv** (*float*) – the upper CI estimate of the inverse of the parameter
- **Lambda_lower_inv** (*float*) – the lower CI estimate of the inverse of the parameter
- **gamma_upper** (*float*) – the upper CI estimate of the parameter
- **gamma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)

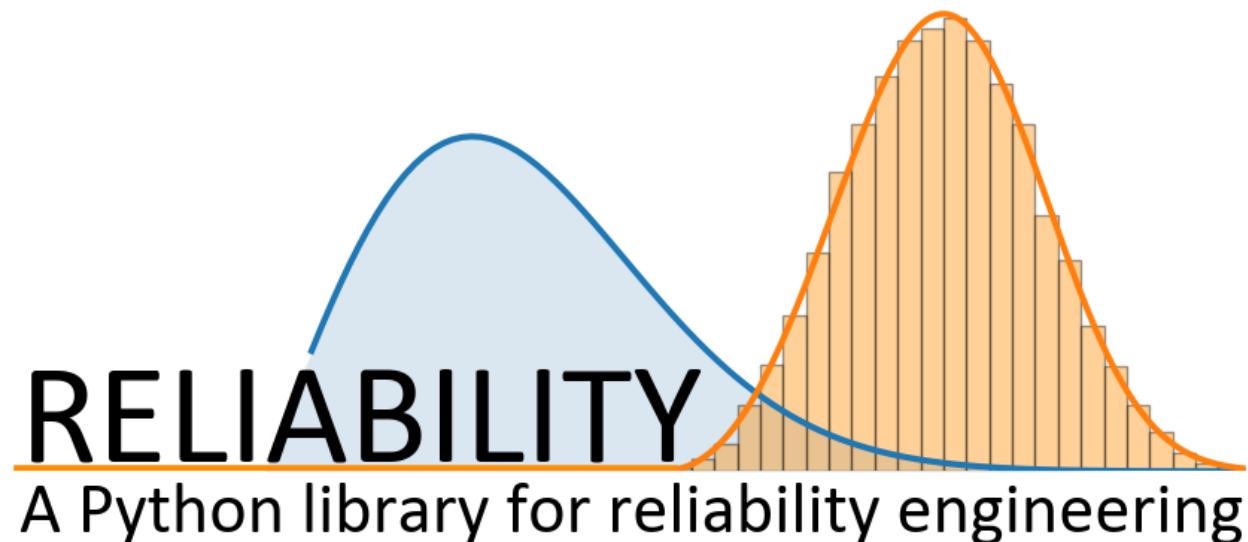
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Exponential_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

This is a two parameter distribution (Lambda, gamma), but the results provide both Lambda as well as the inverse (1/Lambda). This is provided for convenience as some other software (Minitab and scipy.stats) use 1/Lambda instead of Lambda. Lambda_SE_inv, Lambda_upper_inv, and Lambda_lower_inv are also provided for convenience.

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)  
static LL_inv(params, T_f, T_rc)  
static logR(t, L, g)  
static logf(t, L, g)
```



71.5.5 Fit_Gamma_2P

```
class reliability.Fitters.Fit_Gamma_2P(failures=None, right_censored=None,
                                         show_probability_plot=True, print_results=True, CI=0.95,
                                         method='MLE', optimizer=None, quantiles=None,
                                         CI_type='time', downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Gamma distribution (alpha,beta) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1,..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Gamma_2P alpha parameter
- **beta** (*float*) – the fitted Gamma_2P beta parameter

- **mu** (*float*) – mu = ln(alpha). Alternate parametrisation (mu, beta) used for the confidence intervals.
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **mu_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **Cov_mu_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **mu_upper** (*float*) – the upper CI estimate of the parameter
- **mu_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Gamma_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

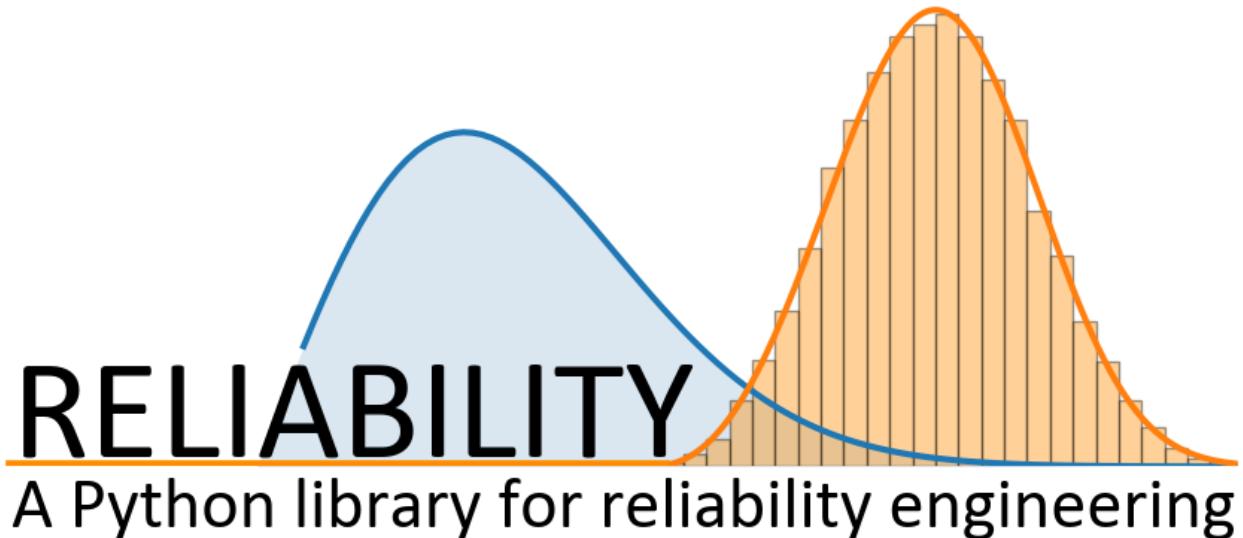
If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

This is a two parameter distribution but it has two parametrisations. These are alpha,beta and mu,beta. The alpha,beta parametrisation is reported in the results table while the mu,beta parametrisation is accessible from the results by name. The reason for this is because the most common parametrisation (alpha,beta) should be reported while the less common parametrisation (mu,beta) is used by some other software so is provided for convenience of comparison. The mu = ln(alpha) relationship is simple but this relationship does not extend to the variances or covariances so additional calculations are required to find both solutions. The mu,beta parametrisation is used for the confidence intervals as it is more stable.

```

static LL_ab(params, T_f, T_rc)
static LL_mb(params, T_f, T_rc)
static logR_ab(t, a, b)
static logR_mb(t, m, b)
static logf_ab(t, a, b)
static logf_mb(t, m, b)

```



71.5.6 Fit_Gamma_3P

```
class reliability.Fitters.Fit_Gamma_3P(failures=None, right_censored=None,
                                         show_probability_plot=True, print_results=True, CI=0.95,
                                         optimizer=None, method='MLE', quantiles=None,
                                         CI_type='time', downsample_scatterplot=True, **kwargs)
```

Fits a three parameter Gamma distribution (alpha,beta,gamma) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 3 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), or ‘LS’ (least squares estimation). Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’,

‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).

- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Gamma_3P alpha parameter
- **beta** (*float*) – the fitted Gamma_3P beta parameter
- **mu** (*float*) – mu = ln(alpha). Alternate parametrisation (mu, beta) used for the confidence intervals.
- **gamma** (*float*) – the fitted Gamma_3P gamma parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **mu_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **gamma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **Cov_mu_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **mu_upper** (*float*) – the upper CI estimate of the parameter
- **mu_lower** (*float*) – the lower CI estimate of the parameter
- **gamma_upper** (*float*) – the upper CI estimate of the parameter
- **gamma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)

- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Gamma_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

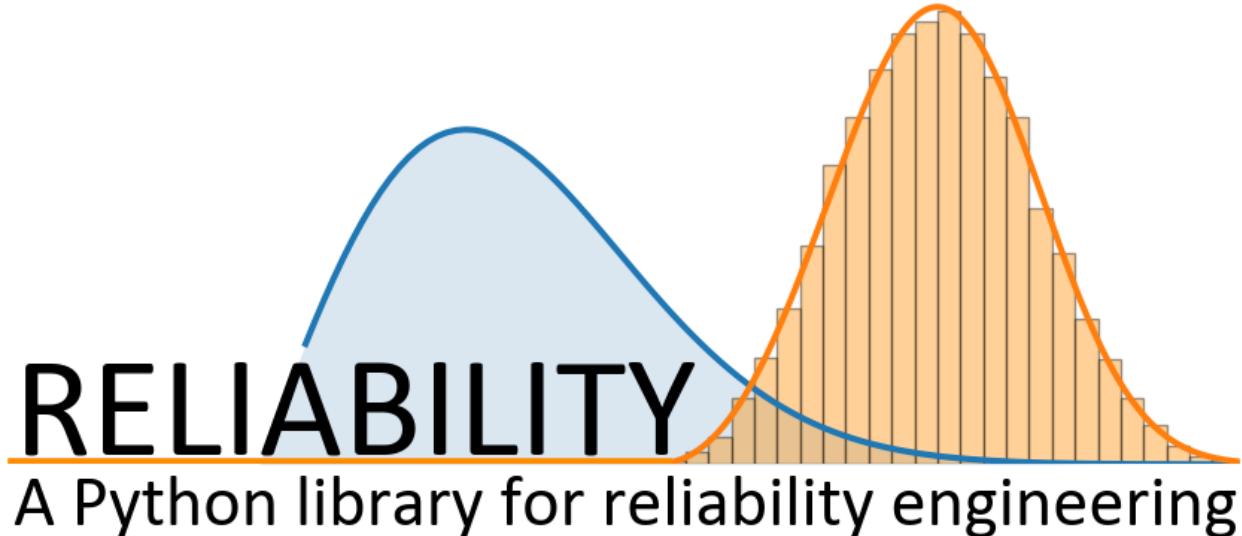
Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

If the fitted gamma parameter is less than 0.01, the Gamma_3P results will be discarded and the Gamma_2P distribution will be fitted. The returned values for gamma and gamma_SE will be 0.

This is a three parameter distribution but it has two parametrisations. These are alpha,beta,gamma and mu,beta,gamma. The alpha,beta,gamma parametrisation is reported in the results table while the mu,beta,gamma parametrisation is accessible from the results by name. The reason for this is because the most common parametrisation (alpha,beta,gamma) should be reported while the less common parametrisation (mu,beta,gamma) is used by some other software so is provided for convenience of comparison. The mu = ln(alpha) relationship is simple but this relationship does not extend to the variances or covariances so additional calculations are required to find both solutions. The mu,beta,gamma parametrisation is used for the confidence intervals as it is more stable.

```
static LL_abg(params, T_f, T_rc)
static LL_mbg(params, T_f, T_rc)
static logR_abg(t, a, b, g)
static logR_mbg(t, m, b, g)
static logf_abg(t, a, b, g)
static logf_mbg(t, m, b, g)
```



71.5.7 Fit_Gumbel_2P

```
class reliability.Fitters.Fit_Gumbel_2P(failures=None, right_censored=None,  
                                         show_probability_plot=True, print_results=True, CI=0.95,  
                                         quantiles=None, CI_type='time', method='MLE',  
                                         optimizer=None, downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Gumbel distribution (μ, σ) to the data provided. Note that it will return a fit that may be partially in the negative domain ($x < 0$). If you need an entirely positive distribution that is similar to Gumbel then consider using Weibull.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.

- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1,..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle).

Returns

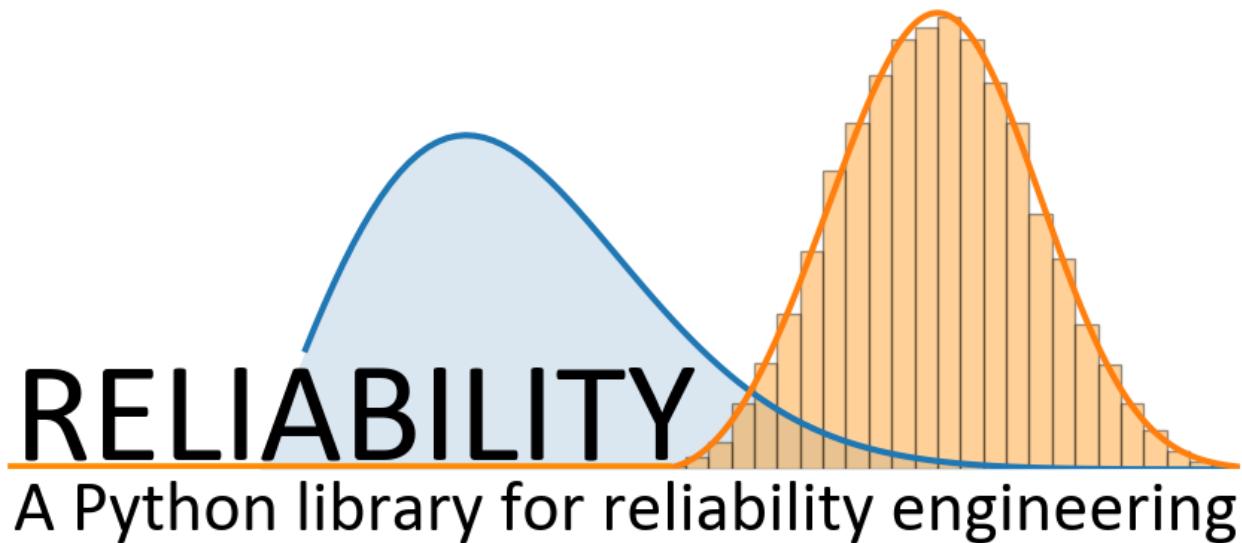
- **mu** (*float*) – the fitted Gumbel_2P mu parameter
- **sigma** (*float*) – the fitted Gumbel_2P sigma parameter
- **mu_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_mu_sigma** (*float*) – the covariance between the parameters
- **mu_upper** (*float*) – the upper CI estimate of the parameter
- **mu_lower** (*float*) – the lower CI estimate of the parameter
- **sigma_upper** (*float*) – the upper CI estimate of the parameter
- **sigma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Gumbel_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

The Gumbel Distribution is similar to the Normal Distribution, with mu controlling the peak of the distribution between $-\infty < \mu < \infty$.

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)  
static logR(t, mu, sigma)  
static logf(t, mu, sigma)
```



71.5.8 Fit_Loglogistic_2P

```
class reliability.Fitters.Fit_Loglogistic_2P(failures=None, right_censored=None,  
                                              show_probability_plot=True, print_results=True,  
                                              CI=0.95, quantiles=None, CI_type='time',  
                                              method='MLE', optimizer=None,  
                                              downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Loglogistic distribution (alpha,beta) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either 'MLE' (maximum likelihood estimation), 'LS' (least squares estimation), 'RRX' (Rank regression

on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.

- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Loglogistic_2P alpha parameter
- **beta** (*float*) – the fitted Loglogistic_2P beta parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Loglogistic_Distribution object with the parameters of the fitted distribution

- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

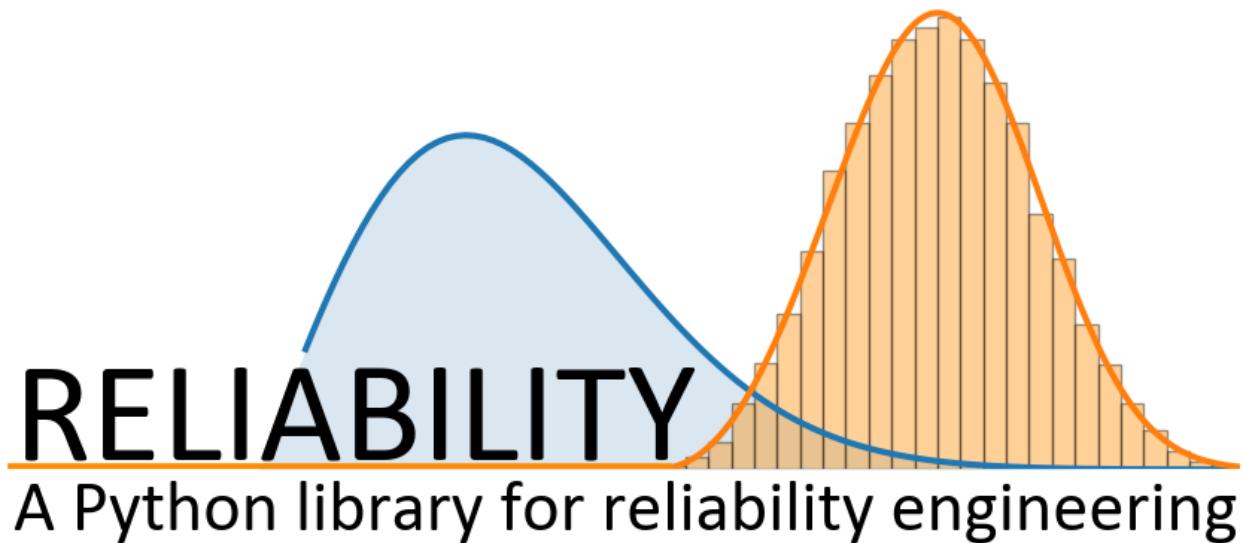
Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)
```

```
static logR(t, a, b)
```

```
static logf(t, a, b)
```



71.5.9 Fit_Loglogistic_3P

```
class reliability.Fitters.Fit_Loglogistic_3P(failures=None, right_censored=None,  
                                              show_probability_plot=True, print_results=True,  
                                              CI=0.95, CI_type='time', optimizer=None,  
                                              method='MLE', quantiles=None,  
                                              downsample_scatterplot=True, **kwargs)
```

Fits a three parameter Loglogistic distribution (alpha,beta,gamma) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 3 elements.

- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a data frame of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), or ‘LS’ (least squares estimation). Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Loglogistic_3P alpha parameter
- **beta** (*float*) – the fitted Loglogistic_3P beta parameter
- **gamma** (*float*) – the fitted Loglogistic_3P gamma parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **gamma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter

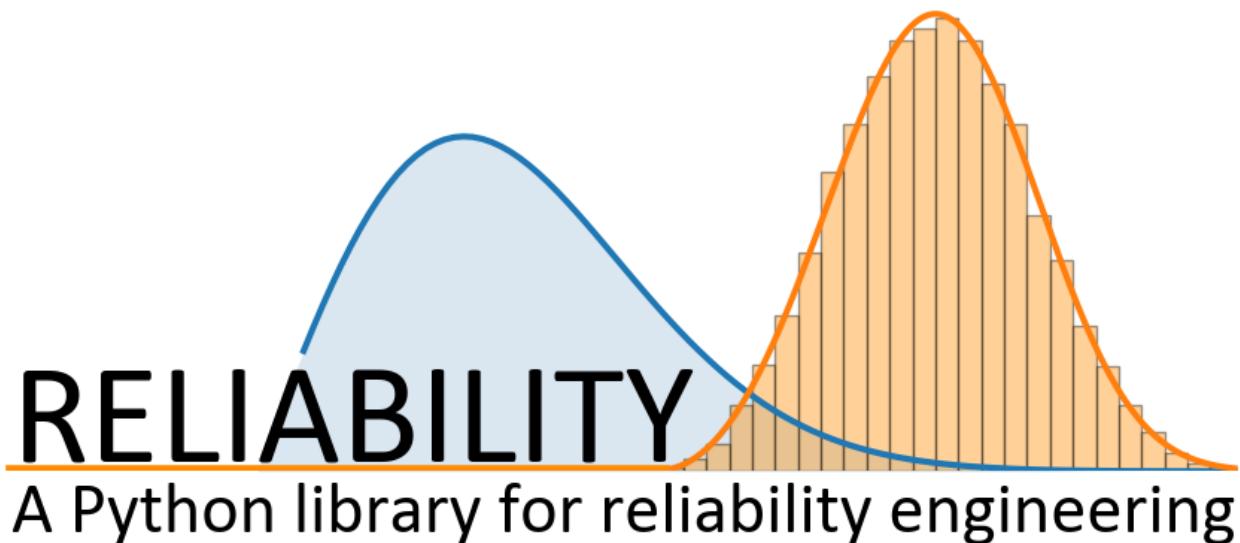
- **gamma_upper** (*float*) – the upper CI estimate of the parameter
- **gamma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Loglogistic_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

If the fitted gamma parameter is less than 0.01, the Loglogistic_3P results will be discarded and the Loglogistic_2P distribution will be fitted. The returned values for gamma and gamma_SE will be 0.

```
static LL(params, T_f, T_rc)  
static logR(t, a, b, g)  
static logf(t, a, b, g)
```



71.5.10 Fit_Lognormal_2P

```
class reliability.Fitters.Fit_Lognormal_2P(failures=None, right_censored=None,
                                             show_probability_plot=True, print_results=True, CI=0.95,
                                             quantiles=None, optimizer=None, CI_type='time',
                                             method='MLE', force_sigma=None,
                                             downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Lognormal distribution (mu,sigma) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements if force_sigma is not specified or at least 1 element if force_sigma is specified.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.

- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **force_sigma** (*float, int, optional*) – Used to specify the sigma value if you need to force sigma to be a certain value. Used in ALT probability plotting. Optional input. If specified it must be > 0 .
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle).

Returns

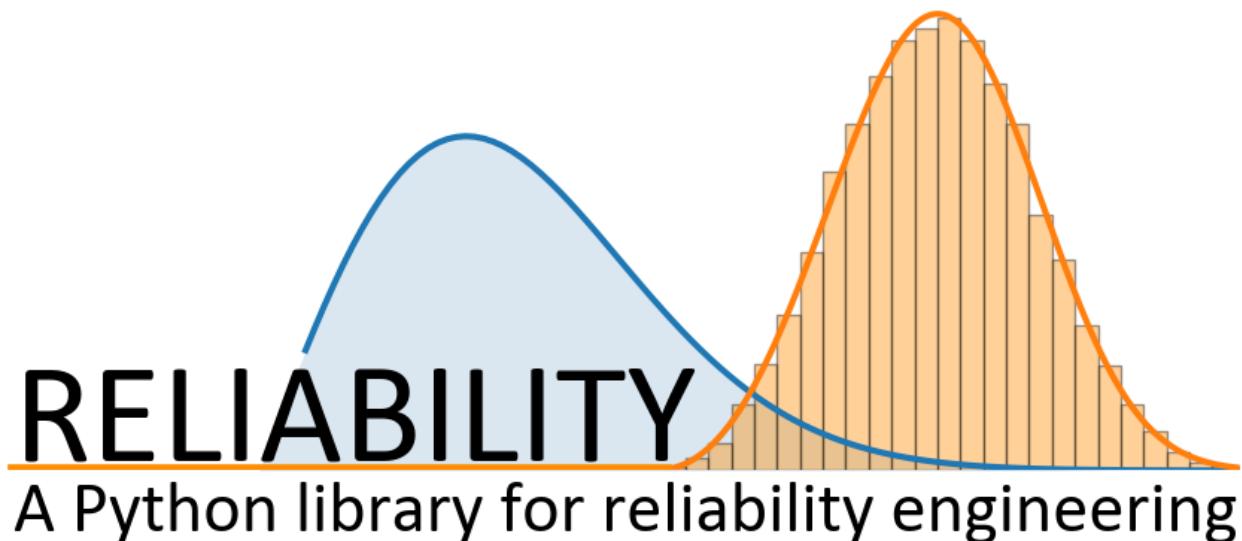
- **mu** (*float*) – the fitted Lognormal_2P alpha parameter
- **sigma** (*float*) – the fitted Lognormal_2P beta parameter
- **mu_SE** (*float*) – the standard error ($\text{sqrt}(\text{variance})$) of the parameter
- **sigma_SE** (*float*) – the standard error ($\text{sqrt}(\text{variance})$) of the parameter
- **Cov_mu_sigma** (*float*) – the covariance between the parameters
- **mu_upper** (*float*) – the upper CI estimate of the parameter
- **mu_lower** (*float*) – the lower CI estimate of the parameter
- **sigma_upper** (*float*) – the upper CI estimate of the parameter
- **sigma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Lognormal_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).

- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)
static LL_fs(params, T_f, T_rc, force_sigma)
static logR(t, mu, sigma)
static logf(t, mu, sigma)
```



71.5.11 Fit_Lognormal_3P

```
class reliability.Fitters.Fit_Lognormal_3P(failures=None, right_censored=None,
                                             show_probability_plot=True, print_results=True, CI=0.95,
                                             quantiles=None, CI_type='time', optimizer=None,
                                             method='MLE', downsample_scatterplot=True, **kwargs)
```

Fits a three parameter Lognormal distribution (μ, σ, γ) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 3 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True

- **print_results** (*bool, optional*) – Prints a data frame of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), or ‘LS’ (least squares estimation). Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle).

Returns

- **mu** (*float*) – the fitted Lognormal_3P mu parameter
- **sigma** (*float*) – the fitted Lognormal_3P sigma parameter
- **gamma** (*float*) – the fitted Lognormal_3P gamma parameter
- **mu_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **gamma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_mu_sigma** (*float*) – the covariance between the parameters
- **mu_upper** (*float*) – the upper CI estimate of the parameter
- **mu_lower** (*float*) – the lower CI estimate of the parameter
- **sigma_upper** (*float*) – the upper CI estimate of the parameter
- **sigma_lower** (*float*) – the lower CI estimate of the parameter
- **gamma_upper** (*float*) – the upper CI estimate of the parameter
- **gamma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)

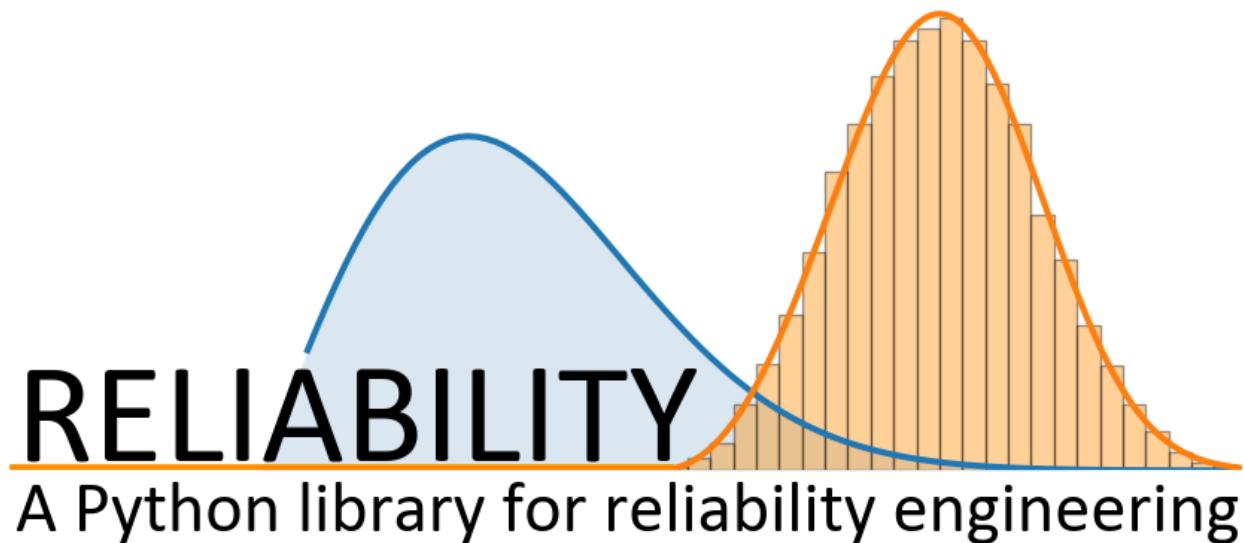
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Lognormal_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

If the fitted gamma parameter is less than 0.01, the Lognormal_3P results will be discarded and the Lognormal_2P distribution will be fitted. The returned values for gamma and gamma_SE will be 0.

```
static LL(params, T_f, T_rc)
static logR(t, mu, sigma, gamma)
static logf(t, mu, sigma, gamma)
```



71.5.12 Fit_Normal_2P

```
class reliability.Fitters.Fit_Normal_2P(failures=None, right_censored=None,  
                                         show_probability_plot=True, print_results=True, CI=0.95,  
                                         quantiles=None, optimizer=None, CI_type='time',  
                                         method='MLE', force_sigma=None,  
                                         downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Normal distribution (μ, σ) to the data provided. Note that it will return a fit that may be partially in the negative domain ($x < 0$). If you need an entirely positive distribution that is similar to Normal then consider using Weibull.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements if force_sigma is not specified or at least 1 element if force_sigma is specified.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **force_sigma** (*float, int, optional*) – Used to specify the beta value if you need to force sigma to be a certain value. Used in ALT probability plotting. Optional input. If specified it must be > 0 .
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.

- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

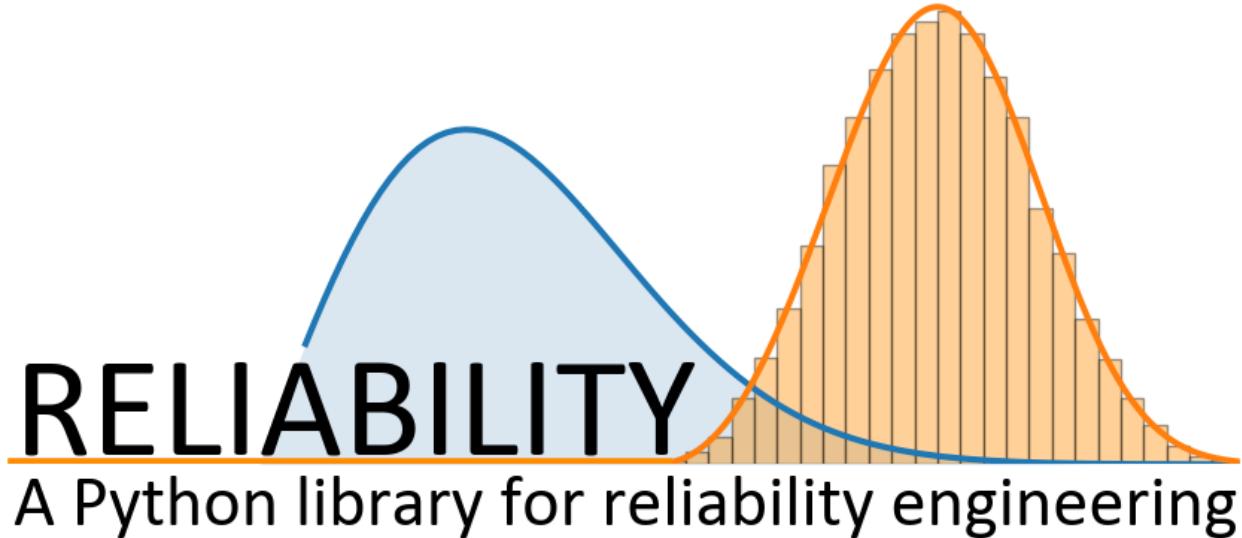
Returns

- **mu** (*float*) – the fitted Normal_2P mu parameter
- **sigma** (*float*) – the fitted Normal_2P sigma parameter
- **mu_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **sigma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_mu_sigma** (*float*) – the covariance between the parameters
- **mu_upper** (*float*) – the upper CI estimate of the parameter
- **mu_lower** (*float*) – the lower CI estimate of the parameter
- **sigma_upper** (*float*) – the upper CI estimate of the parameter
- **sigma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Normal_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)
static LL_fs(params, T_f, T_rc, force_sigma)
static logR(t, mu, sigma)
static logf(t, mu, sigma)
```



71.5.13 Fit_Weibull_2P

```
class reliability.Fitters.Fit_Weibull_2P(failures=None, right_censored=None, show_probability_plot=True, print_results=True, CI=0.95, quantiles=None, CI_type='time', method='MLE', optimizer=None, force_beta=None, downsample_scatterplot=True, **kwargs)
```

Fits a two parameter Weibull distribution (alpha,beta) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements if force_beta is not specified or at least 1 element if force_beta is specified.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.

- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **force_beta** (*float, int, optional*) – Used to specify the beta value if you need to force beta to be a certain value. Used in ALT probability plotting. Optional input. If specified it must be > 0 .
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [1, 5, 10, ..., 95, 99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

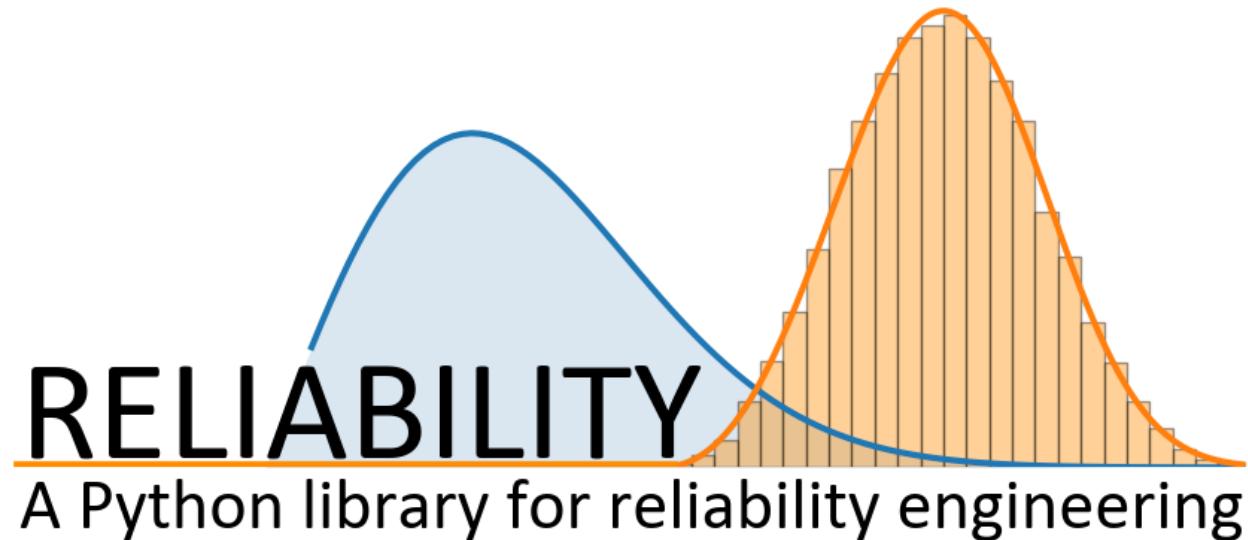
- **alpha** (*float*) – the fitted Weibull_2P alpha parameter
- **beta** (*float*) – the fitted Weibull_2P beta parameter
- **alpha_SE** (*float*) – the standard error ($\text{sqrt}(\text{variance})$) of the parameter
- **beta_SE** (*float*) – the standard error ($\text{sqrt}(\text{variance})$) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Weibull_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).

- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)  
static LL_fb(params, T_f, T_rc, force_beta)  
static logR(t, a, b)  
static logf(t, a, b)
```



71.5.14 Fit_Weibull_2P_grouped

```
class reliability.Fitters.Fit_Weibull_2P_grouped(dataframe=None, show_probability_plot=True,  
                                                print_results=True, CI=0.95, force_beta=None,  
                                                quantiles=None, method='MLE', optimizer=None,  
                                                CI_type='time', downsample_scatterplot=True,  
                                                **kwargs)
```

Fits a two parameter Weibull distribution (alpha,beta) to the data provided. This function is similar to `Fit_Weibull_2P` however it accepts a dataframe which allows for efficient handling of grouped (repeated) data.

Parameters

- **dataframe** (*dataframe*) – a pandas dataframe of the appropriate format. See the example in Notes.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True

- **print_results** (*bool, optional*) – Prints a data frame of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), ‘LS’ (least squares estimation), ‘RRX’ (Rank regression on X), or ‘RRY’ (Rank regression on Y). LS will perform both RRX and RRY and return the better one. Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. The default optimizer is ‘TNC’. The option to use all these optimizers is not available (as it is in all the other Fitters). If the optimizer fails, the initial guess will be returned.
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **force_beta** (*float, int, optional*) – Used to specify the beta value if you need to force beta to be a certain value. Used in ALT probability plotting. Optional input. If specified it must be > 0 .
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1, ..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Weibull_2P alpha parameter
- **beta** (*float*) – the fitted Weibull_2P beta parameter
- **alpha_SE** (*float*) – the standard error ($\text{sqrt}(\text{variance})$) of the parameter
- **beta_SE** (*float*) – the standard error ($\text{sqrt}(\text{variance})$) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)

- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Weibull_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if show_probability_plot = True

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

Requirements of the input dataframe: The column titles MUST be ‘category’, ‘time’, ‘quantity’ The category values MUST be ‘F’ for failure or ‘C’ for censored (right censored). The time values are the failure or right censored times. The quantity is the number of items at that time. This must be specified for all values even if the quantity is 1.

Example of the input dataframe:

category	time	quantity
F	24	1
F	29	1
F	34	1
F	39	2
F	40	1
F	42	3
F	44	1
C	50	3
C	55	5
C	60	10

This is easiest to achieve by importing data from excel. An example of this is:

```
import pandas as pd
from reliability.Fitters import Fit_Weibull_2P_grouped
filename = 'C:\Users\Current User\Desktop\data.xlsx'
df = pd.read_excel(io=filename)
Fit_Weibull_2P_grouped(dataframe=df)
```

static LL(*params*, *T_f*, *T_rc*, *Q_f*, *Q_rc*)

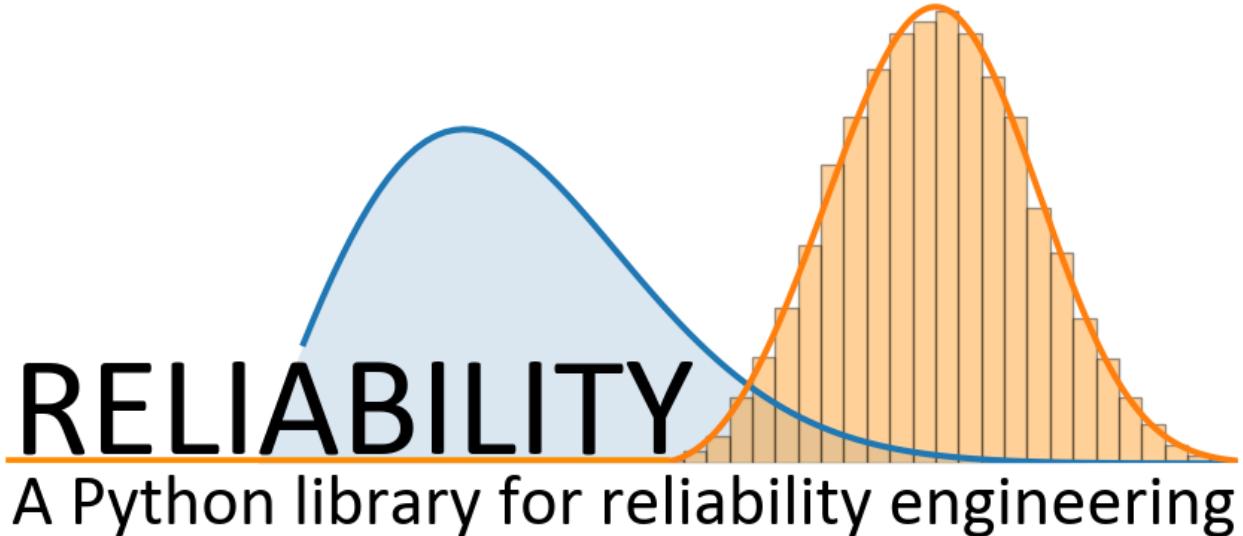
```

static LL_fb(params, T_f, T_rc, Q_f, Q_rc, force_beta)

static logR(t, a, b)

static logf(t, a, b)

```



71.5.15 Fit_Weibull_3P

```

class reliability.Fitters.Fit_Weibull_3P(failures=None, right_censored=None,
                                             show_probability_plot=True, print_results=True, CI=0.95,
                                             quantiles=None, CI_type='time', optimizer=None,
                                             method='MLE', downsample_scatterplot=True, **kwargs)

```

Fits a three parameter Weibull distribution (alpha,beta,gamma) to the data provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 3 elements
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **method** (*str, optional*) – The method used to fit the distribution. Must be either ‘MLE’ (maximum likelihood estimation), or ‘LS’ (least squares estimation). Default is ‘MLE’.
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.

- **CI_type** (*str, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use ‘none’ to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or ‘none’. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **quantiles** (*bool, str, list, array, None, optional*) – quantiles (y-values) to produce a table of quantiles failed with lower, point, and upper estimates. Default is None which results in no output. To use default array [0.01, 0.05, 0.1,..., 0.95, 0.99] set quantiles as either ‘auto’, True, ‘default’, ‘on’. If an array or list is specified then it will be used instead of the default array. Any array or list specified must contain values between 0 and 1.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Weibull_3P alpha parameter
- **beta** (*float*) – the fitted Weibull_3P beta parameter
- **gamma** (*float*) – the fitted Weibull_3P gamma parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **gamma_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **Cov_alpha_beta** (*float*) – the covariance between the parameters
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **gamma_upper** (*float*) – the upper CI estimate of the parameter
- **gamma_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Weibull_Distribution object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).

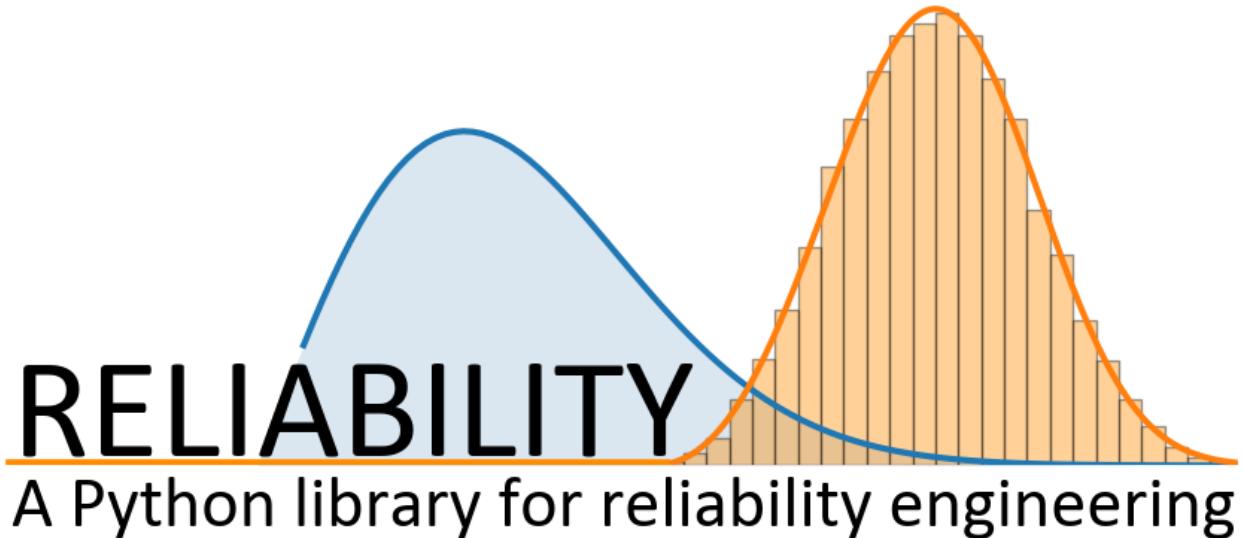
- **quantiles** (*dataframe*) – a pandas dataframe of the quantiles with bounds on time. This is only produced if quantiles is not None. Since quantiles defaults to None, this output is not normally produced.
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

If the fitted gamma parameter is less than 0.01, the Weibull_3P results will be discarded and the Weibull_2P distribution will be fitted. The returned values for gamma and gamma_SE will be 0.

```
static LL(params, T_f, T_rc)
static logR(t, a, b, g)
static logf(t, a, b, g)
```



71.5.16 Fit_Weibull_CR

```
class reliability.Fitters.Fit_Weibull_CR(failures=None, right_censored=None,
                                         show_probability_plot=True, print_results=True, CI=0.95,
                                         optimizer=None, downsample_scatterplot=True, **kwargs)
```

Fits a Weibull Competing Risks Model consisting of two Weibull_2P distributions (this does not fit the gamma parameter). Similar to the mixture model, you can use this model when you think there are multiple failure modes acting to create the failure data.

Parameters

- **failures** (*array, list*) – An array or list of the failure data. There must be at least 4 failures, but it is highly recommended to use another model if you have less than 20 failures.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.

- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha_1** (*float*) – the fitted Weibull_2P alpha parameter for the first distribution
- **beta_1** (*float*) – the fitted Weibull_2P beta parameter for the first distribution
- **alpha_2** (*float*) – the fitted Weibull_2P alpha parameter for the second distribution
- **beta_2** (*float*) – the fitted Weibull_2P beta parameter for the second distribution
- **alpha_1_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_1_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **alpha_2_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_2_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **alpha_1_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_1_lower** (*float*) – the lower CI estimate of the parameter
- **alpha_2_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_2_lower** (*float*) – the lower CI estimate of the parameter
- **beta_1_upper** (*float*) – the upper CI estimate of the parameter
- **beta_1_lower** (*float*) – the lower CI estimate of the parameter
- **beta_2_upper** (*float*) – the upper CI estimate of the parameter
- **beta_2_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)

- **distribution** (*object*) – a Competing_Risks_Model object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

Notes

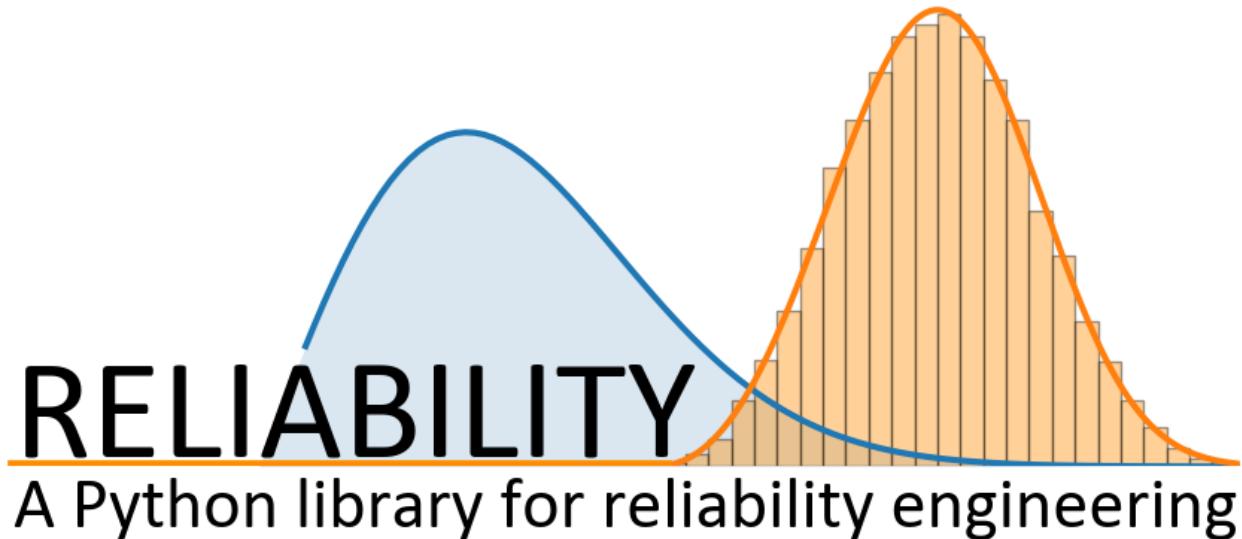
This is different to the Weibull Mixture model as the overall Survival Function is the product of the individual Survival Functions rather than being the sum as is the case in the Weibull Mixture Model.

Mixture Model: $SF_{model} = (proportion_1 SF_1) + ((1 - proportion_1) SF_2)$

Competing Risks Model: $SF_{model} = SF_1 SF_2$

Whilst some failure modes may not be fitted as well by a Weibull distribution as they may be by another distribution, it is unlikely that data from a competing risks model will be fitted noticeably better by other types of competing risks models than would be achieved by a Weibull Competing Risks model. For this reason, other types of competing risks models are not implemented.

```
static LL(params, T_f, T_rc)
static logR(t, a1, b1, a2, b2)
static logf(t, a1, b1, a2, b2)
```



71.5.17 Fit_Weibull_DS

```
class reliability.Fitters.Fit_Weibull_DS(failures=None, right_censored=None,
                                         show_probability_plot=True, print_results=True, CI=0.95,
                                         optimizer=None, downsample_scatterplot=True, **kwargs)
```

Fits a Weibull Defective Subpopulation (DS) distribution to the data provided. This is a 3 parameter distribution (alpha, beta, DS).

Parameters

- **failures** (*array, list*) – An array or list of the failure data. There must be at least 2 failures.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

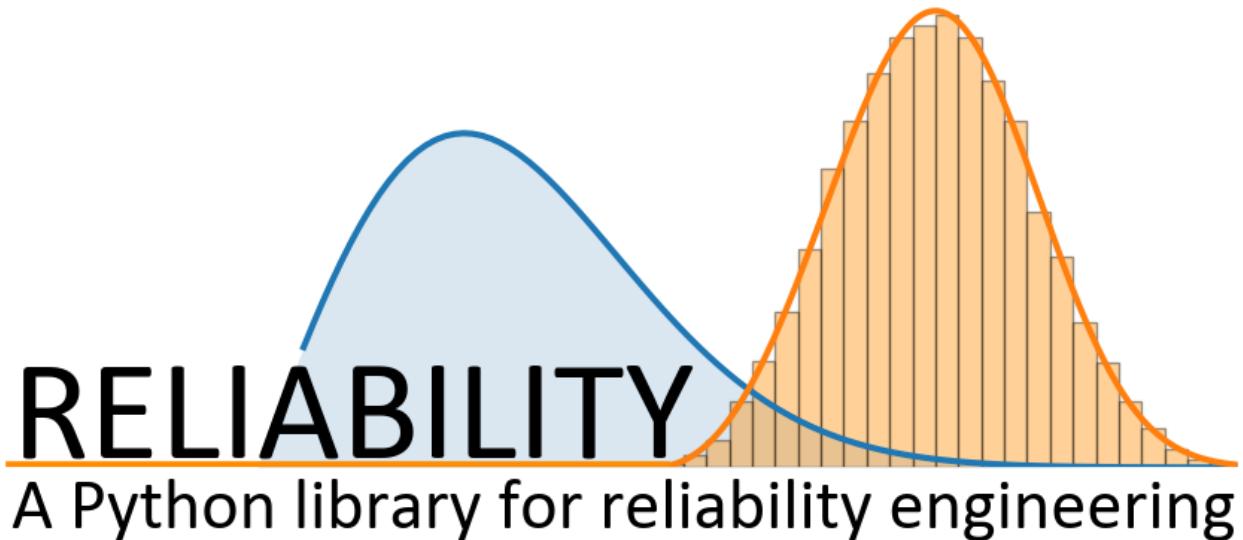
- **alpha** (*float*) – the fitted Weibull_DS alpha parameter
- **beta** (*float*) – the fitted Weibull_DS beta parameter
- **DS** (*float*) – the fitted Weibull_DS DS parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **DS_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **DS_upper** (*float*) – the upper CI estimate of the parameter
- **DS_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)

- **distribution** (*object*) – a DSZI_Model object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)
static logR(t, a, b, ds)
static logf(t, a, b, ds)
```



71.5.18 Fit_Weibull_DSZI

```
class reliability.Fitters.Fit_Weibull_DSZI(failures=None, right_censored=None,
                                             show_probability_plot=True, print_results=True, CI=0.95,
                                             optimizer=None, downsample_scatterplot=True, **kwargs)
```

Fits a Weibull Defective Subpopulation Zero Inflated (DSZI) distribution to the data provided. This is a 4 parameter distribution (alpha, beta, DS, ZI).

Parameters

- **failures** (*array, list*) – An array or list of the failure data. There must be at least 2 non-zero failures.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.

- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

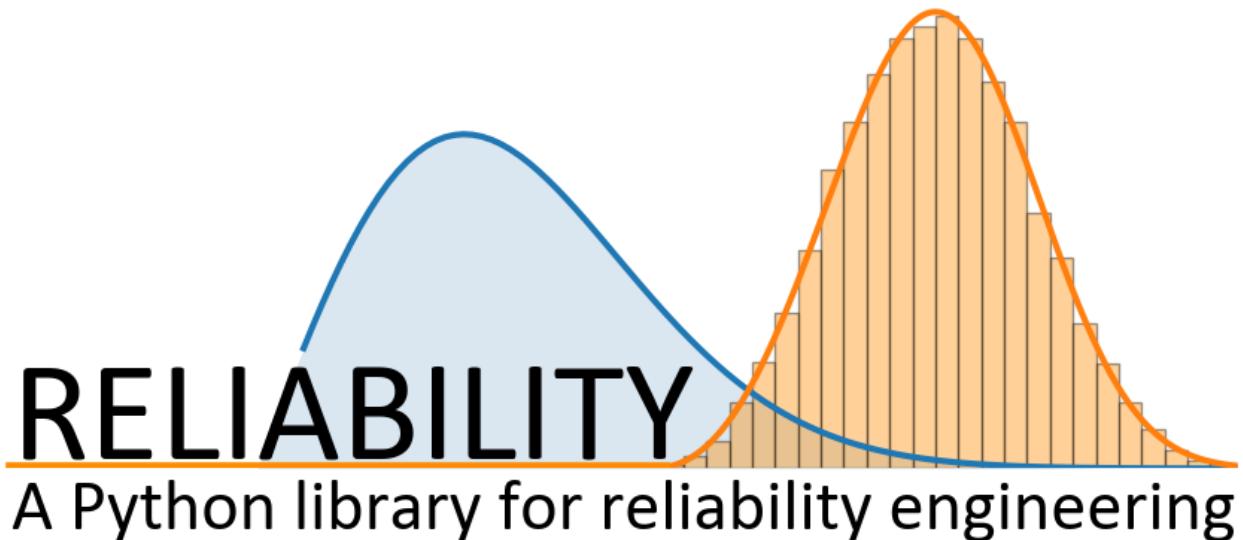
- **alpha** (*float*) – the fitted Weibull_DSZI alpha parameter
- **beta** (*float*) – the fitted Weibull_DSZI beta parameter
- **DS** (*float*) – the fitted Weibull_DSZI DS parameter
- **ZI** (*float*) – the fitted Weibull_DSZI ZI parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **DS_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **ZI_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **DS_upper** (*float*) – the upper CI estimate of the parameter
- **DS_lower** (*float*) – the lower CI estimate of the parameter
- **ZI_upper** (*float*) – the upper CI estimate of the parameter
- **ZI_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)

- **distribution** (*object*) – a DSZI_Model object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_0, T_f, T_rc)
static logR(t, a, b, ds, zi)
static logf(t, a, b, ds, zi)
```



71.5.19 Fit_Weibull_Mixture

```
class reliability.Fitters.Fit_Weibull_Mixture(failures=None, right_censored=None,
                                              show_probability_plot=True, print_results=True,
                                              CI=0.95, optimizer=None,
                                              downsample_scatterplot=True, **kwargs)
```

Fits a mixture of two Weibull_2P distributions (this does not fit the gamma parameter). Right censoring is supported, though care should be taken to ensure that there still appears to be two groups when plotting only the failure data. A second group cannot be made from a mostly or totally censored set of samples. Use this model when you think there are multiple failure modes acting to create the failure data.

Parameters

- **failures** (*array, list*) – An array or list of the failure data. There must be at least 4 failures, but it is highly recommended to use another model if you have less than 20 failures.

- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a datafram of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha_1** (*float*) – the fitted Weibull_2P alpha parameter for the first (left) group
- **beta_1** (*float*) – the fitted Weibull_2P beta parameter for the first (left) group
- **alpha_2** (*float*) – the fitted Weibull_2P alpha parameter for the second (right) group
- **beta_2** (*float*) – the fitted Weibull_2P beta parameter for the second (right) group
- **proportion_1** (*float*) – the fitted proportion of the first (left) group
- **proportion_2** (*float*) – the fitted proportion of the second (right) group. Same as 1-proportion_1
- **alpha_1_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_1_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **alpha_2_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_2_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **proportion_1_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **alpha_1_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_1_lower** (*float*) – the lower CI estimate of the parameter
- **alpha_2_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_2_lower** (*float*) – the lower CI estimate of the parameter
- **beta_1_upper** (*float*) – the upper CI estimate of the parameter
- **beta_1_lower** (*float*) – the lower CI estimate of the parameter
- **beta_2_upper** (*float*) – the upper CI estimate of the parameter
- **beta_2_lower** (*float*) – the lower CI estimate of the parameter

- **proportion_1_upper** (*float*) – the upper CI estimate of the parameter
- **proportion_1_lower** (*float*) – the lower CI estimate of the parameter
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a Mixture_Model object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

Notes

This is different to the Weibull Competing Risks as the overall Survival Function is the sum of the individual Survival Functions multiplied by a proportion rather than being the product as is the case in the Weibull Competing Risks Model.

Mixture Model: $SF_{model} = (proportion_1 SF_1) + ((1 - proportion_1) SF_2)$

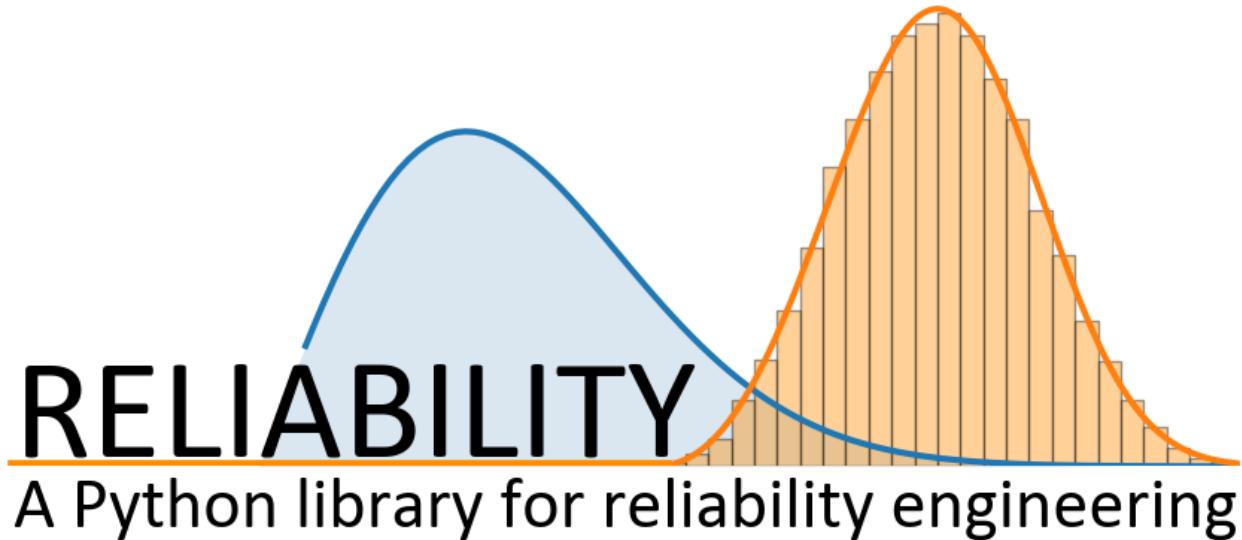
Competing Risks Model: $SF_{model} = SF_1 SF_2$

Similar to the competing risks model, you can use this model when you think there are multiple failure modes acting to create the failure data.

Whilst some failure modes may not be fitted as well by a Weibull distribution as they may be by another distribution, it is unlikely that a mixture of data from two distributions (particularly if they are overlapping) will be fitted noticeably better by other types of mixtures than would be achieved by a Weibull mixture. For this reason, other types of mixtures are not implemented.

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

```
static LL(params, T_f, T_rc)
static logR(t, a1, b1, a2, b2, p)
static logf(t, a1, b1, a2, b2, p)
```



71.5.20 Fit_Weibull_ZI

```
class reliability.Fitters.Fit_Weibull_ZI(failures=None, right_censored=None,  
                                         show_probability_plot=True, print_results=True, CI=0.95,  
                                         optimizer=None, downsample_scatterplot=True, **kwargs)
```

Fits a Weibull Zero Inflated (ZI) distribution to the data provided. This is a 3 parameter distribution (alpha, beta, ZI).

Parameters

- **failures** (*array, list*) – An array or list of the failure data. There must be at least 2 non-zero failures.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_probability_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the point estimate, standard error, Lower CI and Upper CI for each parameter. True or False. Default = True
- **optimizer** (*str, optional*) – The optimization algorithm used to find the solution. Must be either ‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, or ‘powell’. Specifying the optimizer will result in that optimizer being used. To use all of these specify ‘best’ and the best result will be returned. The default behaviour is to try each optimizer in order (‘TNC’, ‘L-BFGS-B’, ‘nelder-mead’, and ‘powell’) and stop once one of the optimizers finds a solution. If the optimizer fails, the initial guess will be returned. For more detail see the [documentation](#).
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is True. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the probability plot (e.g. color, label, linestyle)

Returns

- **alpha** (*float*) – the fitted Weibull_ZI alpha parameter
- **beta** (*float*) – the fitted Weibull_ZI beta parameter
- **ZI** (*float*) – the fitted Weibull_ZI ZI parameter
- **alpha_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **beta_SE** (*float*) – the standard error (sqrt(variance)) of the parameter
- **ZI_SE** (*float*) – the standard error (sqrt(variance)) of the parameter.
- **alpha_upper** (*float*) – the upper CI estimate of the parameter
- **alpha_lower** (*float*) – the lower CI estimate of the parameter
- **beta_upper** (*float*) – the upper CI estimate of the parameter
- **beta_lower** (*float*) – the lower CI estimate of the parameter
- **ZI_upper** (*float*) – the upper CI estimate of the parameter.
- **ZI_lower** (*float*) – the lower CI estimate of the parameter.
- **loglik** (*float*) – Log Likelihood (as used in Minitab and Reliasoft)
- **loglik2** (*float*) – LogLikelihood*-2 (as used in JMP Pro)
- **AICc** (*float*) – Akaike Information Criterion
- **BIC** (*float*) – Bayesian Information Criterion
- **AD** (*float*) – the Anderson Darling (corrected) statistic (as reported by Minitab)
- **distribution** (*object*) – a DSZI_Model object with the parameters of the fitted distribution
- **results** (*dataframe*) – a pandas dataframe of the results (point estimate, standard error, lower CI and upper CI for each parameter)
- **goodness_of_fit** (*dataframe*) – a pandas dataframe of the goodness of fit values (Log-likelihood, AICc, BIC, AD).
- **probability_plot** (*object*) – the axes handle for the probability plot. This is only returned if `show_probability_plot = True`

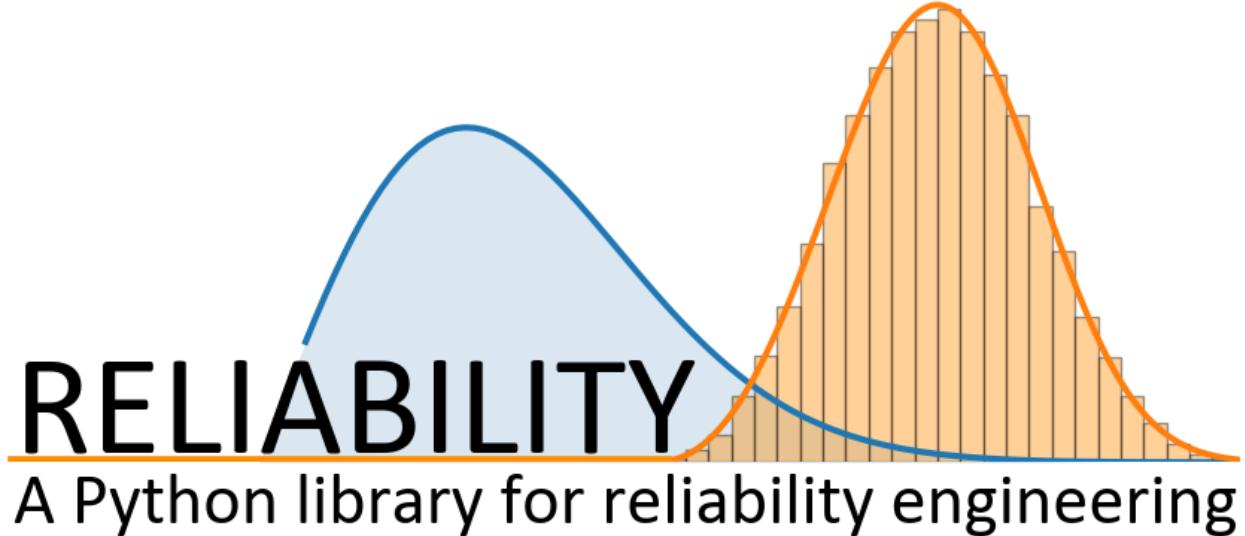
Notes

If the fitting process encounters a problem a warning will be printed. This may be caused by the chosen distribution being a very poor fit to the data or the data being heavily censored. If a warning is printed, consider trying a different optimizer.

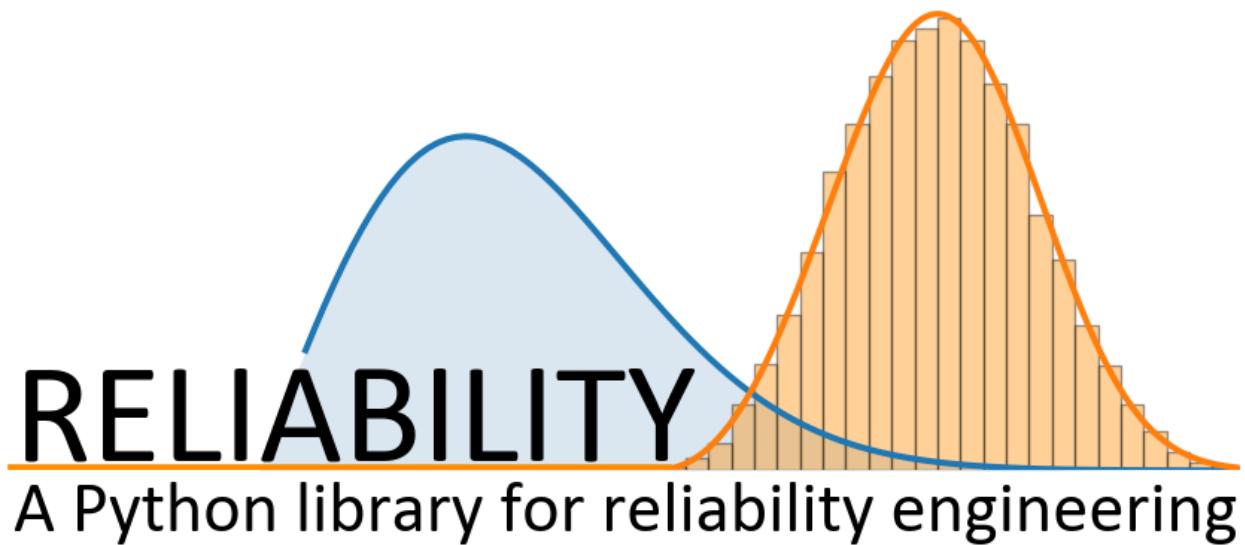
```
static LL(params, T_0, T_f, T_rc)

static logR(t, a, b, zi)

static logf(t, a, b, zi)
```



71.6 Nonparametric



71.6.1 KaplanMeier

```
class reliability.Nonparametric.KaplanMeier(failures=None, right_censored=None, show_plot=True, print_results=True, plot_CI=True, CI=0.95, plot_type='SF', **kwargs)
```

Uses the Kaplan-Meier estimation method to calculate the reliability from failure data. Right censoring is supported and confidence bounds are provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.

- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the results. True or False. Default = True
- **plot_type** (*str*) – Must be either ‘SF’, ‘CDF’, or ‘CHF’. Default is SF.
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **plot_CI** (*bool*) – Shades the upper and lower confidence interval. True or False. Default = True
- **kwarg**s – Plotting keywords that are passed directly to matplotlib for the plot (e.g. color, label, linestyle)

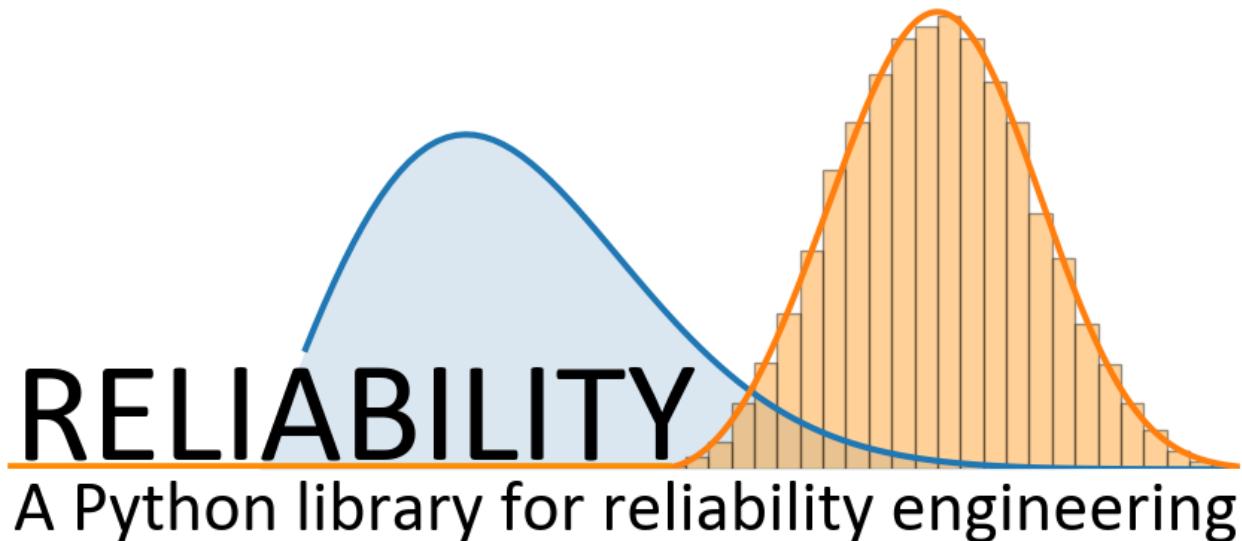
Returns

- **results** (*dataframe*) – A pandas dataframe of results for the SF
- **KM** (*array*) – The Kaplan-Meier Estimate column from results dataframe. This column is the non-parametric estimate of the Survival Function (reliability function).
- **xvals** (*array*) – the x-values to plot the stepwise plot as seen when show_plot=True
- **SF** (*array*) – survival function stepwise values (these differ from the KM values as there are extra values added in to make the plot into a step plot)
- **CDF** (*array*) – cumulative distribution function stepwise values
- **CHF** (*array*) – cumulative hazard function stepwise values
- **SF_lower** (*array*) – survival function stepwise values for lower CI
- **SF_upper** (*array*) – survival function stepwise values for upper CI
- **CDF_lower** (*array*) – cumulative distribution function stepwise values for lower CI
- **CDF_upper** (*array*) – cumulative distribution function stepwise values for upper CI
- **CHF_lower** (*array*) – cumulative hazard function stepwise values for lower CI
- **CHF_upper** (*array*) – cumulative hazard function stepwise values for upper CI
- **data** (*array*) – the failures and right_censored values sorted. Same as ‘Failure times’ column from results dataframe
- **censor_codes** (*array*) – the censoring codes (0 or 1) from the sorted data. Same as ‘Censoring code (censored=0)’ column from results dataframe

Notes

The confidence bounds are calculated using the Greenwood formula with Normal approximation, which is the same as featured in Minitab.

The Kaplan-Meier method provides the SF. The CDF and CHF are obtained from transformations of the SF. It is not possible to obtain a useful version of the PDF or HF as the derivative of a stepwise function produces discontinuous (jagged) functions.



71.6.2 NelsonAalen

```
class reliability.Nonparametric.NelsonAalen(failures=None, right_censored=None, show_plot=True, print_results=True, plot_CI=True, CI=0.95, plot_type='SF', **kwargs)
```

Uses the Nelson-Aalen estimation method to calculate the reliability from failure data. Right censoring is supported and confidence bounds are provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the results. True or False. Default = True
- **plot_type** (*str*) – Must be either ‘SF’, ‘CDF’, or ‘CHF’. Default is SF.
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **plot_CI** (*bool*) – Shades the upper and lower confidence interval. True or False. Default = True
- **kwargs** – Plotting keywords that are passed directly to matplotlib for the plot (e.g. color, label, linestyle)

Returns

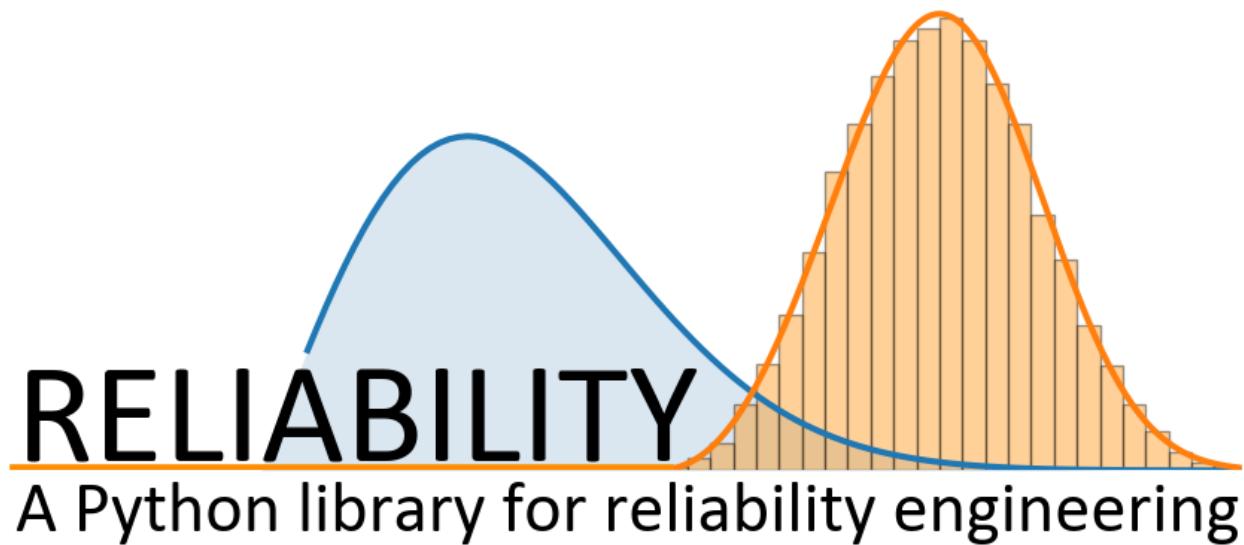
- **results** (*dataframe*) – A pandas dataframe of results for the SF
- **NA** (*array*) – The Nelson-Aalen Estimate column from results dataframe. This column is the non-parametric estimate of the Survival Function (reliability function).
- **xvals** (*array*) – the x-values to plot the stepwise plot as seen when show_plot=True

- **SF (array)** – survival function stepwise values (these differ from the NA values as there are extra values added in to make the plot into a step plot)
- **CDF (array)** – cumulative distribution function stepwise values
- **CHF (array)** – cumulative hazard function stepwise values
- **SF_lower (array)** – survival function stepwise values for lower CI
- **SF_upper (array)** – survival function stepwise values for upper CI
- **CDF_lower (array)** – cumulative distribution function stepwise values for lower CI
- **CDF_upper (array)** – cumulative distribution function stepwise values for upper CI
- **CHF_lower (array)** – cumulative hazard function stepwise values for lower CI
- **CHF_upper (array)** – cumulative hazard function stepwise values for upper CI
- **data (array)** – the failures and right_censored values sorted. Same as ‘Failure times’ column from results dataframe
- **censor_codes (array)** – the censoring codes (0 or 1) from the sorted data. Same as ‘Censoring code (censored=0)’ column from results dataframe

Notes

The confidence bounds are calculated using the Greenwood formula with Normal approximation, which is the same as featured in Minitab.

The Nelson-Aalen method provides the SF. The CDF and CHF are obtained from transformations of the SF. It is not possible to obtain a useful version of the PDF or HF as the derivative of a stepwise function produces discontinuous (jagged) functions. Nelson-Aalen does obtain the HF directly which is then used to obtain the CHF, but this function is not smooth and is of little use.



71.6.3 RankAdjustment

```
class reliability.Nonparametric.RankAdjustment(failures=None, right_censored=None,
                                              print_results=True, a=None, show_plot=True,
                                              plot_CI=True, CI=0.95, plot_type='SF', **kwargs)
```

Uses the rank-adjustment estimation method to calculate the reliability from failure data. Right censoring is supported and confidence bounds are provided.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_plot** (*bool, optional*) – True or False. Default = True
- **print_results** (*bool, optional*) – Prints a dataframe of the results. True or False. Default = True
- **plot_type** (*str*) – Must be either ‘SF’, ‘CDF’, or ‘CHF’. Default is SF.
- **CI** (*float, optional*) – confidence interval for estimating confidence limits on parameters. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **plot_CI** (*bool*) – Shades the upper and lower confidence interval. True or False. Default = True
- **a - int,float,optional** – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Optional input. Default is $a=0.3$ which is the median rank method (same as the default in Minitab). Must be in the range 0 to 1. For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **kwparams** – Plotting keywords that are passed directly to matplotlib for the plot (e.g. color, label, linestyle)

Returns

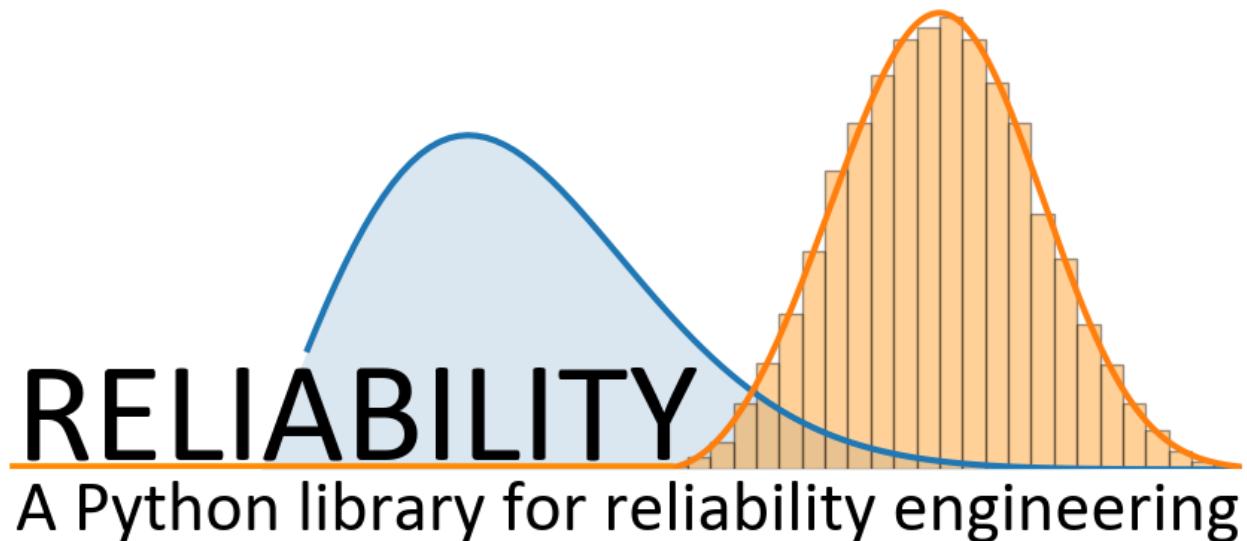
- **results** (*dataframe*) – A pandas dataframe of results for the SF
- **RA** (*array*) – The Rank Adjustment Estimate column from results dataframe. This column is the non-parametric estimate of the Survival Function (reliability function).
- **xvals** (*array*) – the x-values to plot the stepwise plot as seen when show_plot=True
- **SF** (*array*) – survival function stepwise values (these differ from the RA values as there are extra values added in to make the plot into a step plot)
- **CDF** (*array*) – cumulative distribution function stepwise values
- **CHF** (*array*) – cumulative hazard function stepwise values
- **SF_lower** (*array*) – survival function stepwise values for lower CI
- **SF_upper** (*array*) – survival function stepwise values for upper CI
- **CDF_lower** (*array*) – cumulative distribution function stepwise values for lower CI
- **CDF_upper** (*array*) – cumulative distribution function stepwise values for upper CI
- **CHF_lower** (*array*) – cumulative hazard function stepwise values for lower CI
- **CHF_upper** (*array*) – cumulative hazard function stepwise values for upper CI
- **data** (*array*) – the failures and right_censored values sorted. Same as ‘Failure times’ column from results dataframe
- **censor_codes** (*array*) – the censoring codes (0 or 1) from the sorted data. Same as ‘Censoring code (censored=0)’ column from results dataframe

Notes

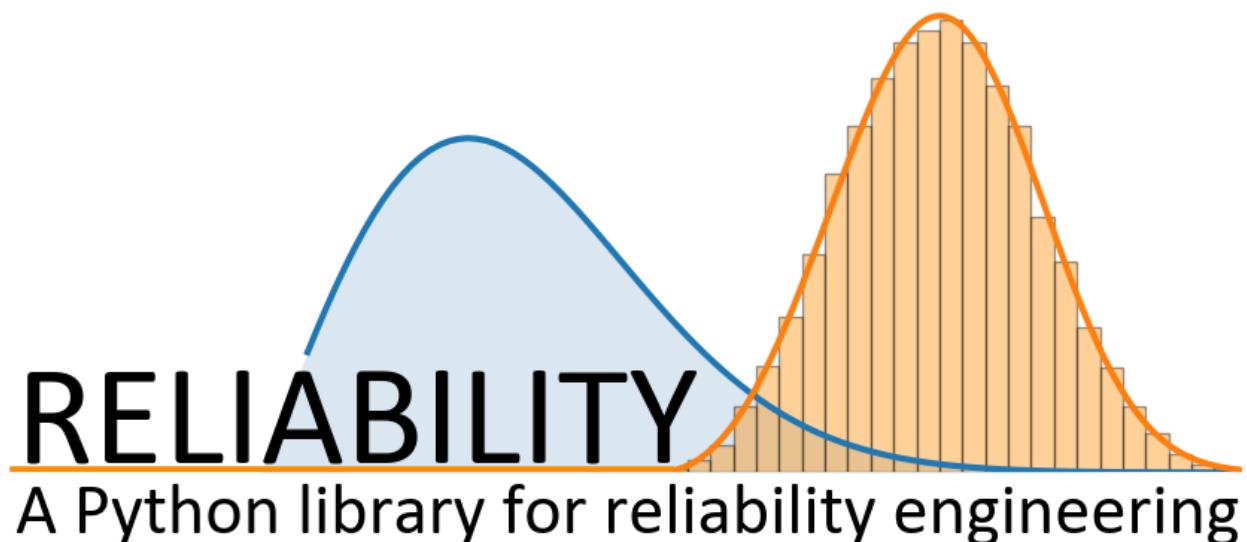
The confidence bounds are calculated using the Greenwood formula with Normal approximation, which is the same as featured in Minitab.

The rank-adjustment method provides the SF. The CDF and CHF are obtained from transformations of the SF. It is not possible to obtain a useful version of the PDF or HF as the derivative of a stepwise function produces discontinuous (jagged) functions.

The Rank-adjustment algorithm is the same as is used in `Probability_plotting.plotting_positions` to obtain y-values for the scatter plot. As with `plotting_positions`, the heuristic constant “a” is accepted, with the default being 0.3 for median ranks.



71.7 Other_functions



71.7.1 crosshairs

```
class reliability.Other_functions.crosshairs(xlabel=None, ylabel=None, decimals=2,
                                             dateformat=None, **kwargs)
```

Adds interactive crosshairs to matplotlib plots

Parameters

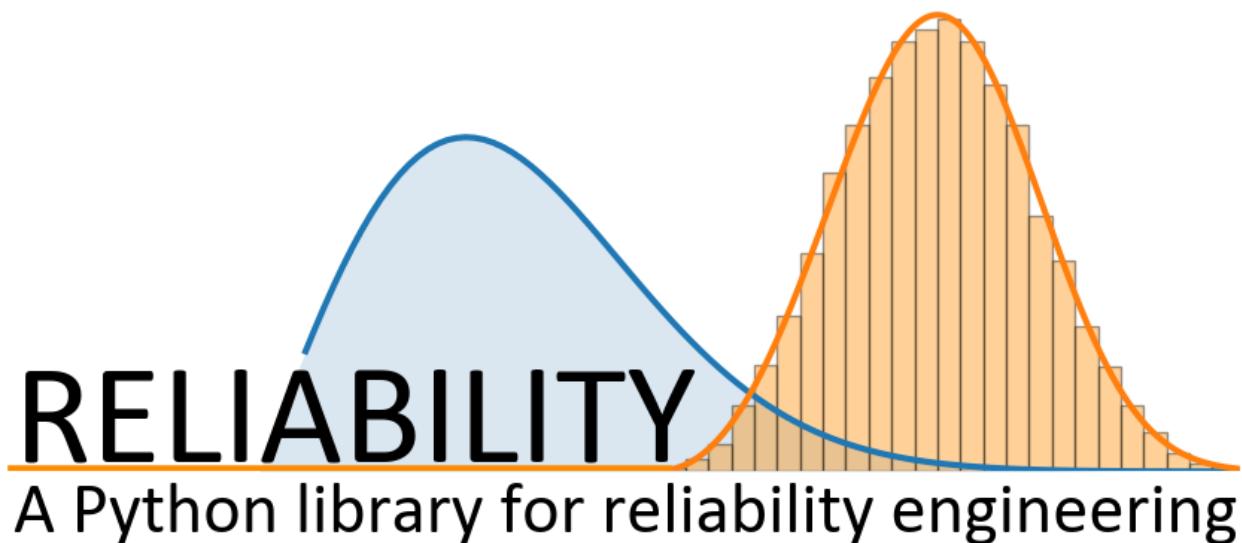
- **xlabel** (*str, optional*) – The xlabel for annotations. Default is ‘x’.
- **ylabel** (*str, optional*) – The ylabel for annotations. Default is ‘y’.
- **decimals** (*int, optional*) – The number of decimals for rounding. Default is 2.
- **dateformat** (*str, optional*) – The datetime format. If specified the x crosshair and label will be formatted as a date using the format provided. Default is None which results in no date format being used on x.
- **kwargs** (*optional*) – plotting kwargs to change the style of the crosshairs (eg. color, linestyle, etc.).

Returns

None

Notes

Ensure this is used after you plot everything as anything plotted after crosshairs() is called will not be recognised by the snap-to feature. For a list of acceptable dateformat strings see <https://docs.python.org/3/library/datetime.html#strftime-and-strptime-format-codes>



71.7.2 distribution_explorer

```
class reliability.Other_functions.distribution_explorer
```

Generates an interactive plot of PDF, CDF, SF, HF, CHF for the selected distribution. Parameters can be changed using slider widgets. Distributions can be changed using radio button widget.

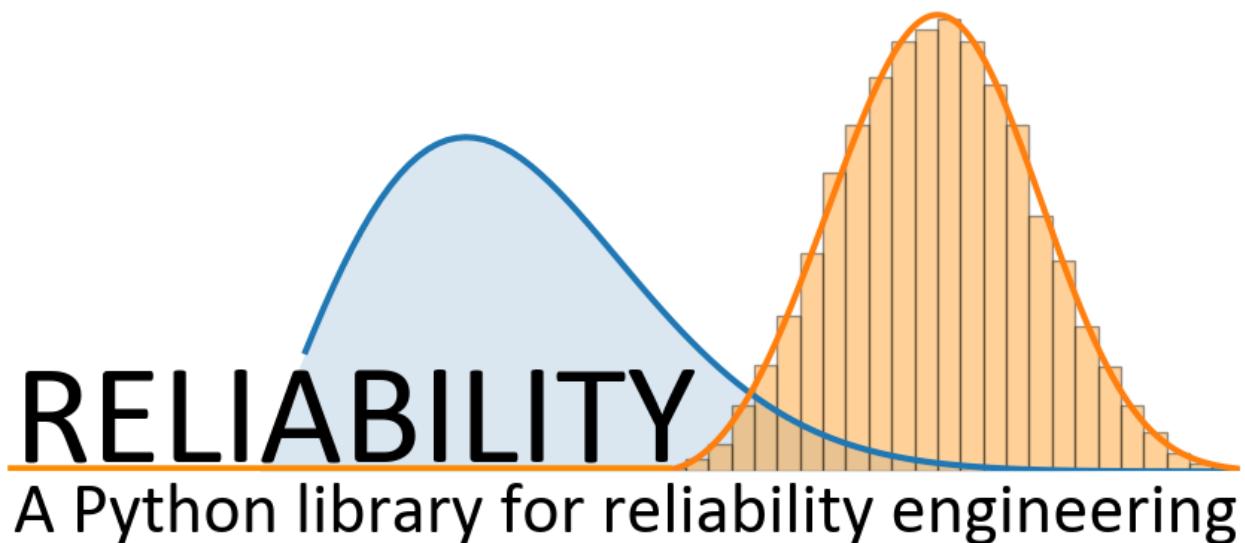
Parameters

None

Returns*None***Notes**

Example usage:

```
from reliability.Other_functions import distribution_explorer
distribution_explorer()
```

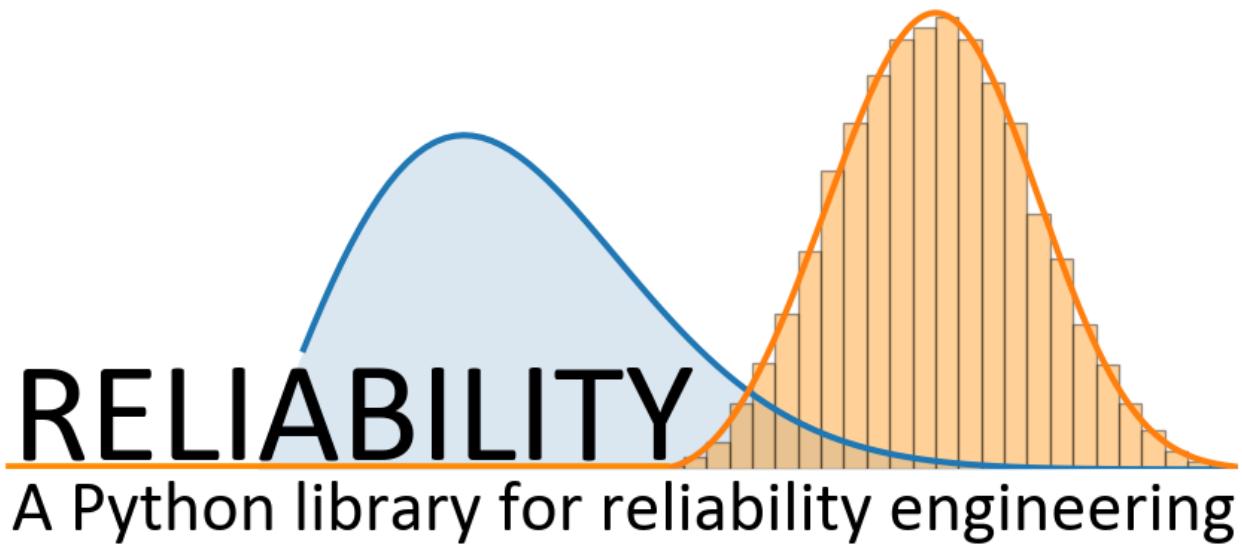
**71.7.3 histogram**

```
class reliability.Other_functions.histogram(data, white_above=None, bins=None, density=True,
                                           cumulative=False, **kwargs)
```

Plots a histogram using the data specified. This is similar to plt.hist except that it sets better defaults and also shades the bins white above a specified value (white_above). This is useful for representing complete data as right censored data in a histogram.

Parameters

- **data** (*array, list*) – The data to plot in the histogram.
- **white_above** (*float, int, optional*) – Bins above this value will be shaded white to represent right censored data. Default = None.
- **bins** (*array, string, optional*) – An array of bin edges or a string to specify how to calculate the bin edges. Acceptable strings are ‘auto’, ‘fd’, ‘doane’, ‘scott’, ‘stone’, ‘rice’, ‘sturges’, ‘sqrt’. Default = ‘auto’. For more information on these methods, see the numpy documentation: https://numpy.org/doc/stable/reference/generated/numpy.histogram_bin_edges.html
- **density** (*bool, optional*) – Determines whether to plot a density histogram or a count histogram. Default = True which is required when plotting a PDF or CDF.
- **cumulative** (*bool, optional*) – Use False for PDF and True for CDF. Default = False.
- **kwargs** – Plotting kwargs for the histogram (color, alpha, etc.) which are passed to matplotlib.

Returns*None*

71.7.4 make_ALT_data

```
class reliability.Other_functions.make_ALT_data(distribution, life_stress_model, stress_1,  
                                              stress_2=None, a=None, b=None, c=None, n=None,  
                                              m=None, beta=None, sigma=None,  
                                              use_level_stress=None, number_of_samples=100,  
                                              fraction_censored=0.5, seed=None)
```

Generates Accelerated Life Test (ALT) data based on model parameters. This function is primarily used when testing the functions in ALT_fitters.

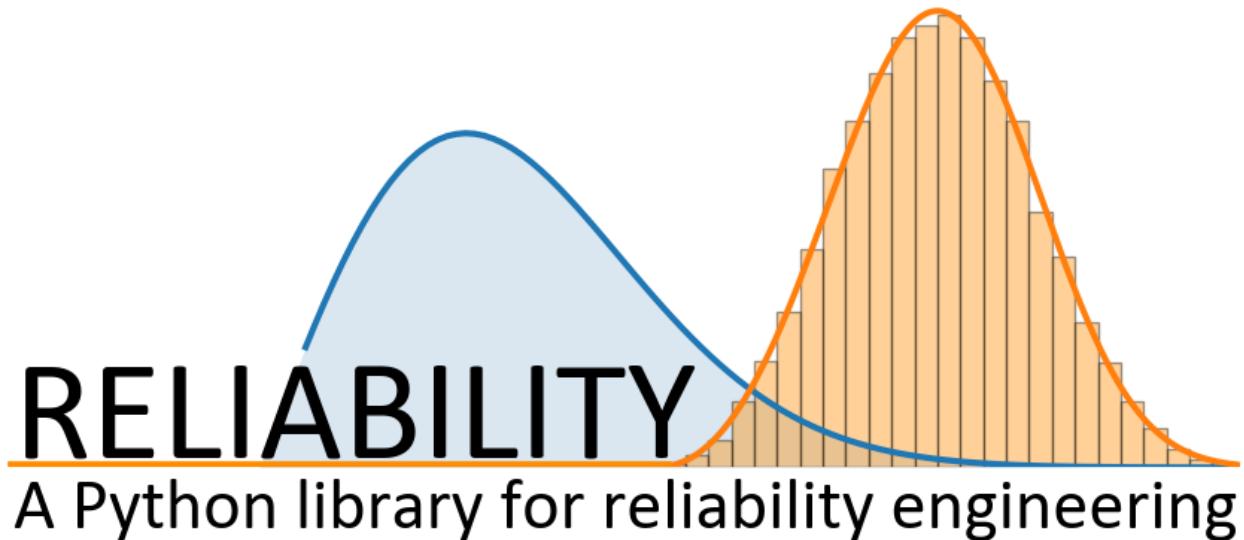
Parameters

- **distribution** (*str*) – Must be either “Weibull”, “Exponential”, “Lognormal”, or “Normal”.
- **life_stress_model** (*str*) – Must be either “Exponential”, “Eyring”, “Power”, “Dual_Exponential”, “Power_Exponential”, or “Dual_Power”
- **stress_1** (*array, list*) – The stresses for the ALT data. eg. [100,50,10].
- **stress_2** (*array, list*) – The stresses for the ALT data. eg. [0.8,0.6,0.4]. Required only if using a dual stress model. Must match the length of stress_1.
- **a** (*float, int*) – Parameter from all models.
- **b** (*float, int, optional*) – Parameter from Exponential and Dual_Exponential models.
- **c** (*float, int, optional*) – Parameter from Eyring, Dual_Exponential, Power_Exponential, and Dual_Power models.
- **n** (*float, int, optional*) – Parameter from Power, Power_Exponential, and Dual_Power models.
- **m** (*float, int, optional*) – Parameter from Dual_Power model.
- **beta** (*float, int, optional*) – Shape parameter for Weibull distribution.
- **sigma** (*float, int, optional*) – Shape parameter for Normal or Lognormal distributions.

- **use_level_stress** (*float, int, list, array, optional*) – A float or int (if single stress) or a list or array (if dual stress). Optional input. Default = None.
- **number_of_samples** (*int, optional*) – The number of samples to generate for each stress. Default = 100. The total data points will be equal to the number of samples x number of stress levels
- **fraction_censored** (*int, float, optional*) – Use 0 for no censoring or specify a float between 0 and 1 for right censoring. Censoring is “multiply censored” meaning that there is no threshold above which all the right censored values will occur. Default = 0.5.
- **seed** (*int, optional*) – The random seed for repeatability. Default = None.

Returns

- **failures** (*list*) – The failure data.
- **failure_stresses** (*list*) – The failure stresses that are paired with the failure data. Only provided if using a single stress model.
- **failure_stresses_1** (*list*) – The failure stresses for stress_1 that are paired with the failure data. Only provided if using a dual stress model.
- **failure_stresses_2** (*list*) – The failure stresses for stress_2 that are paired with the failure data. Only provided if using a dual stress model.
- **right_censored** (*list*) – The right censored data. This is only provided if fraction_censored > 0.
- **right_censored_stresses** (*list*) – The failure stresses that are paired with the right censored data. This is only provided if fraction_censored > 0. Only provided if using a single stress model.
- **right_censored_stresses_1** (*list*) – The failure stresses that are paired with the right censored data. This is only provided if fraction_censored > 0. Only provided if using a dual stress model.
- **right_censored_stresses_2** (*list*) – The failure stresses that are paired with the right censored data. This is only provided if fraction_censored > 0. Only provided if using a dual stress model.
- **mean_life_at_use_stress** (*float*) – This is only provided if use_level_stress is provided.



71.7.5 make_right_censored_data

```
class reliability.Other_functions.make_right_censored_data(data, threshold=None,  
fraction_censored=None, seed=None)
```

This function is used to create right censored data from complete data. It will right censor the data based on a specified threshold or fraction to censor.

Parameters

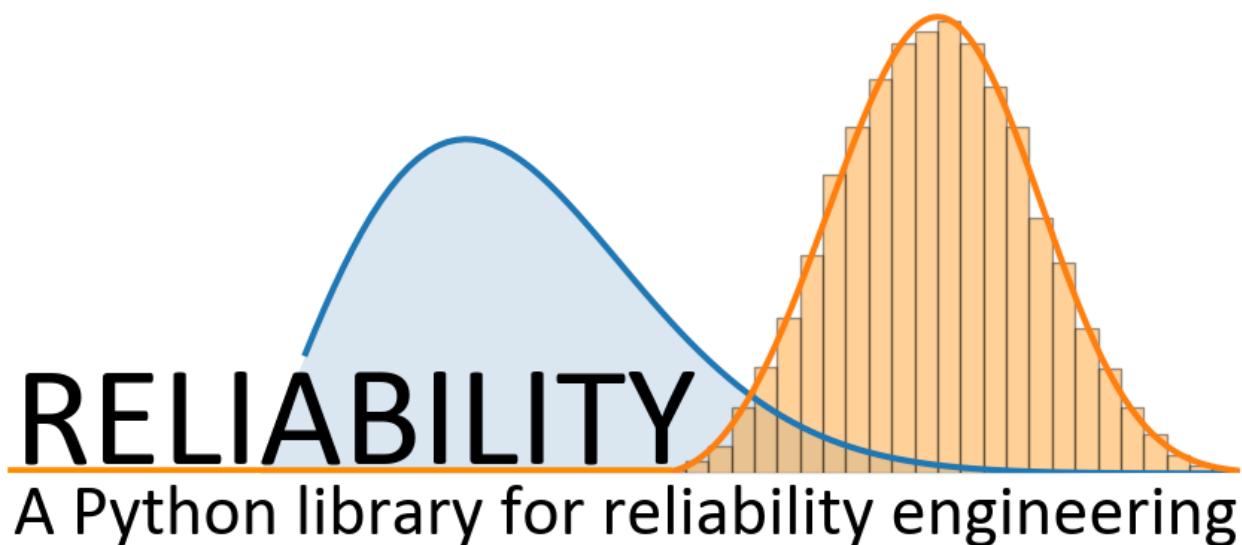
- **data** (*list, array*) – The complete data.
- **threshold** (*int, float, optional*) – This is the point to right censor (right censoring is done if $\text{data} > \text{threshold}$). This is known as “singly censored data” as everything is censored at a single point. Default is `None` in which case the `fraction_censored` will be used. See the notes below.
- **fraction_censored** (*int, float, optional*) – Must be ≥ 0 and < 1 . Default = `0.5`. Censoring is done randomly. This is known as “multiply censored data” as there are multiple times at which censoring occurs. See the notes below.
- **seed** (*int, optional*) – Sets the random seed. This is used for multiply censored data (i.e. when `threshold` is `None`). The data is shuffled to remove censoring bias that may be caused by any pre-sorting. Specifying the seed ensures a repeatable random shuffle. Default is `None` which will result in a different censoring each time. The seed is only used when `threshold` is not specified and the data is being multiply censored based on the `fraction_censored`.

Returns

- **failures** (*array*) – The array of failure data
- **right_censored** (*array*) – The array of right censored data

Notes

If both `threshold` and `fraction_censored` are `None`, `fraction_censored` will default to `0.5` to produce multiply censored data. If both `threshold` and `fraction_censored` are specified, an error will be raised since these methods conflict.



71.7.6 similar_distributions

```
class reliability.Other_functions.similar_distributions(distribution,
                                                       include_location_shifted=True,
                                                       show_plot=True, print_results=True,
                                                       number_of_distributions_to_show=3)
```

This is a tool to find similar distributions when given an input distribution. It is useful to see how similar one distribution is to another. For example, you may look at a Weibull distribution and think it looks like a Normal distribution. Using this tool you can determine the parameters of the Normal distribution that most closely matches your Weibull distribution.

Parameters

- **distribution** (*object*) – A distribution object created using the reliability.Distributions module.
- **include_location_shifted** (*bool, optional*) – When set to True it will include Weibull_3P, Lognormal_3P, Gamma_3P, Exponential_2P, and Loglogistic_3P. Default = True
- **show_plot** (*bool, optional*) – If True it will show the PDF and CDF of the input distributions and the most similar distributions. Default = True.
- **print_results** (*bool, optional*) – If True the results will be printed to the console. Default = True.
- **number_of_distributions_to_show** (*int, optional*) – The number of similar distributions to show. Default = 3. If the number specified exceeds the number available (typically 10), then the number specified will automatically be reduced. Must be > 1.

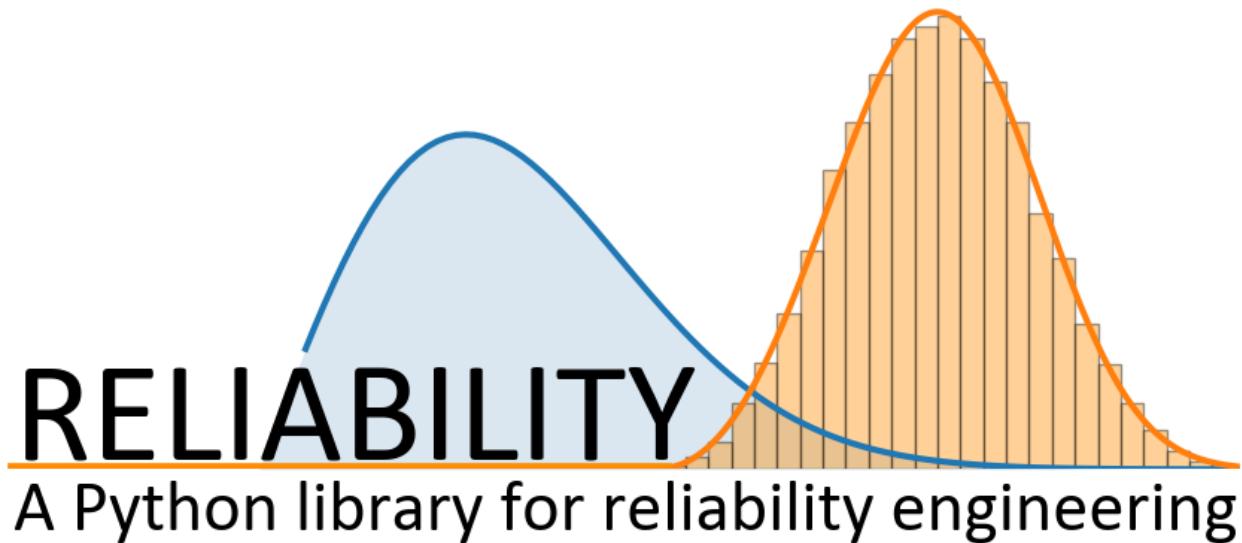
Returns

- **results** (*array*) – An array of distributions objects ranked in order of best fit.
- **most_similar_distribution** (*object*) – A distribution object. This is the first item from results.

Notes

The following example shows the distributions most similar to the input Weibull Distribution.

```
from reliability.Distributions import Weibull_Distribution
from reliability.Other_functions import similar_distributions
dist = Weibull_Distribution(alpha=50, beta=3.3)
similar_distributions(distribution=dist)
```



71.7.7 stress_strength

```
class reliability.Other_functions.stress_strength(stress, strength, show_plot=True,  
                                                print_results=True, warn=True)
```

Given the probability distributions for stress and strength, this module will find the probability of failure due to stress-strength interference. Failure is defined as when stress>strength. The calculation is achieved using numerical integration.

Parameters

- **stress** (*object*) – A probability distribution from the Distributions module
- **strength** (*object*) – A probability distribution from the Distributions module
- **show_plot** (*bool, optional*) – If True the distribution plot will be shown. Default = True.
- **print_results** (*bool, optional*) – If True, the results will be printed to console. Default = True.
- **warn** (*bool, optional*) – A warning will be issued if both stress and strength are Normal as you should use stress_strength_normal. You can suppress this using warn=False. A warning will be issued if the median stress > median strength as the user may have assigned the distributions to the wrong variables. You can suppress this using warn=False. Default = True.

Returns

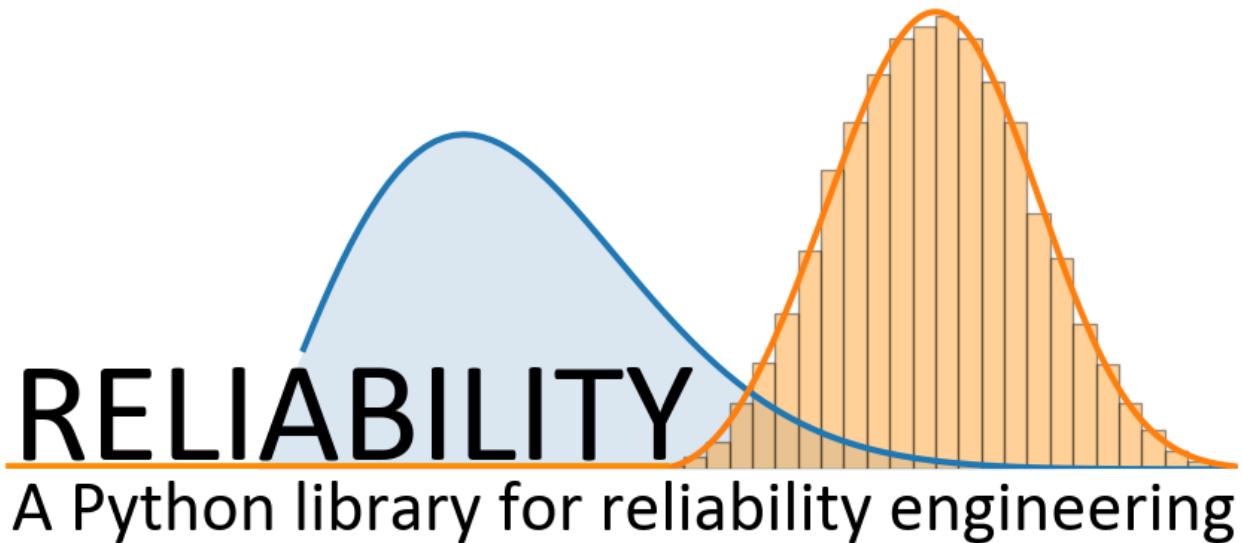
probability_of_failure (*float*) – The probability of failure due to stress-strength interference

Notes

Example usage:

```
from reliability.Distributions import Weibull_Distribution, Gamma_Distribution  
stress = Weibull_Distribution(alpha=2, beta=3, gamma=1)  
strength = Gamma_Distribution(alpha=2, beta=3, gamma=3)  
stress_strength(stress=stress, strength=strength)
```

A warning will be issued if probability_of_failure > 1. This can occur when one of the distributions asymptotes. This warning can not be suppressed.



71.7.8 `stress_strength_normal`

```
class reliability.Other_functions.stress_strength_normal(stress, strength, show_plot=True,
                                                       print_results=True, warn=True)
```

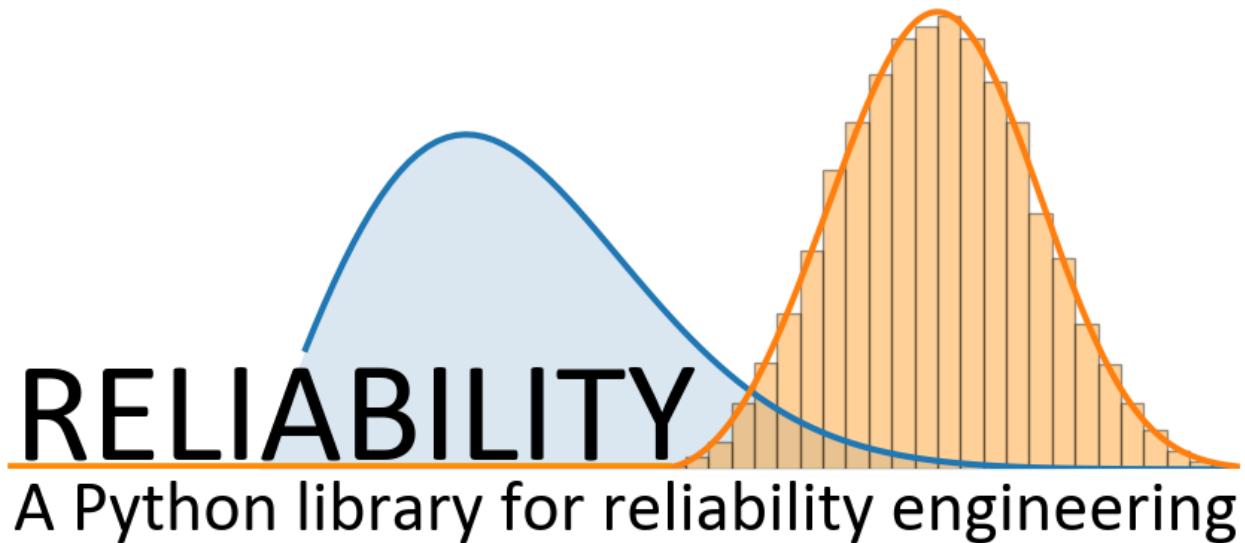
Given the probability distributions for stress and strength, this module will find the probability of failure due to stress-strength interference. Failure is defined as when stress>strength. Uses the exact formula method which is only valid for two Normal Distributions. If you have distributions that are not both Normal Distributions, use the function `stress_strength`.

Parameters

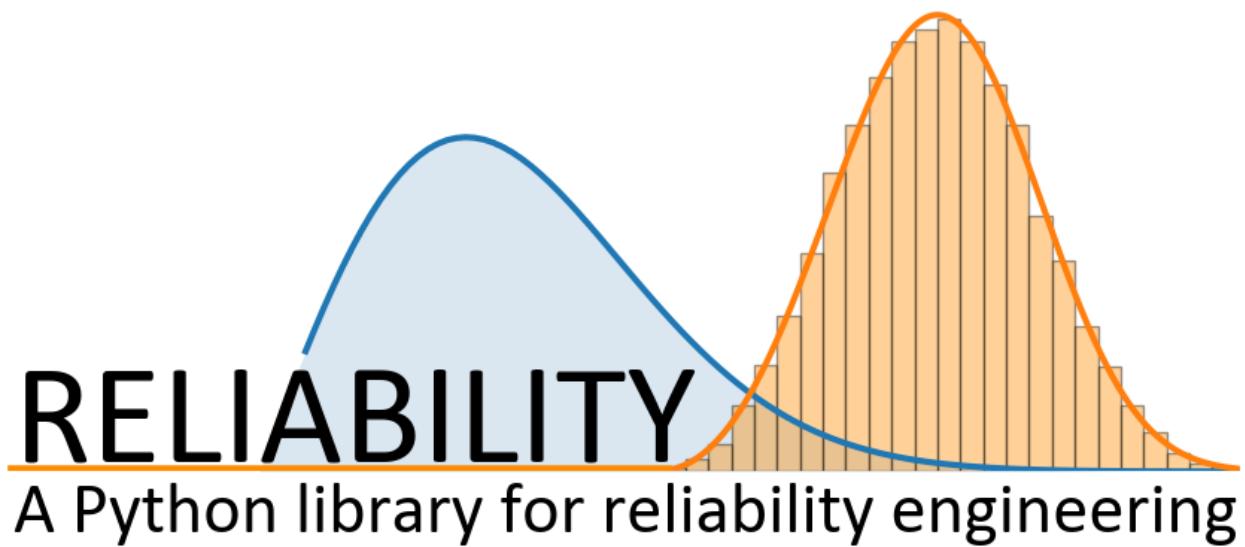
- **stress** (*object*) – A probability distribution from the Distributions module
- **strength** (*object*) – A probability distribution from the Distributions module
- **show_plot** (*bool, optional*) – If True the distribution plot will be shown. Default = True.
- **print_results** (*bool, optional*) – If True, the results will be printed to console. Default = True.
- **warn** (*bool, optional*) – A warning will be issued if the `stress.mean > strength.mean` as the user may have assigned the distributions to the wrong variables. You can suppress this using `warn=False`. Default = True

Returns

probability_of_failure (*float*) – The probability of failure due to stress-strength interference



71.8 PoF



71.8.1 SN_diagram

```
class reliability.PoF.SN_diagram(stress, cycles, stress_runout=None, cycles_runout=None, xscale='log',
                                  stress_trace=None, cycles_trace=None, show_endurance_limit=None,
                                  method_for_bounds='statistical', CI=0.95, **kwargs)
```

This function will plot the stress vs number of cycles (S-N) diagram when supplied with data from a series of fatigue tests.

Parameters

- **stress** (*array, list*) – The stress values at failure.
- **cycles** (*array, list*) – The cycles values at failure. Must match the length of stress.

- **stress_runout** (array, list, optional) – The stress values that did not result in failure. Optional input.
- **cycles_runout** (array, list, optional) – The cycles values that did not result in failure. Optional input. If supplied, Must match the length of stress_runout.
- **xscale** (str, optional) – The scale for the x-axis. Must be ‘log’ or ‘linear’. Default is ‘log’.
- **stress_trace** (array, list, optional) – The stress values to be traced across to cycles values on the plot. Optional input.
- **cycles_trace** (array, list, optional) – The cycles values to be traced across to stress values on the plot. Optional input.
- **show_endurance_limit** (bool, optional) – This will adjust all lines of best fit to be greater than or equal to the average stress_runout. Defaults to False if stress_runout is not specified. Defaults to True if stress_runout is specified.
- **method_for_bounds** (str, None, optional) – The method for the confidence bounds. Must be ‘statistical’, ‘residual’, or None. Defaults to ‘statistical’. If set to ‘statistical’ the CI value is used, otherwise it is not used for the ‘residual’ method. Residual uses the maximum residual datapoint for symmetric bounds. Setting the method for bounds to None will turn off the confidence bounds.
- **CI** (float, optional) – The confidence interval. Must be between 0 and 1. Default is 0.95 for 95% confidence interval. Only used if method_for_bounds is ‘statistical’.
- **kwarg**s – Other plotting keywords (eg. color, linestyle, etc) are accepted and passed to matplotlib for the line of best fit.

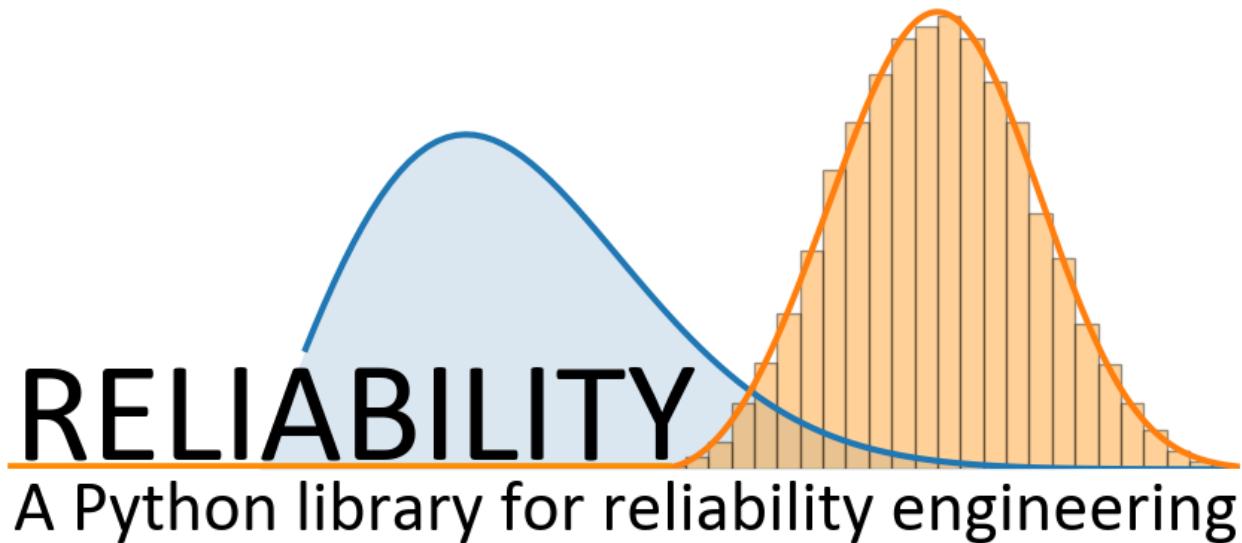
Returns

None – The plot is the only output. All calculated values are shown on the plot.

Notes

Example usage:

```
from reliability.PoF import SN_diagram
import matplotlib.pyplot as plt
stress = [340, 300, 290, 275, 260, 255, 250, 235, 230, 220, 215, 210]
cycles = [15000, 24000, 36000, 80000, 177000, 162000, 301000, 290000, 361000, ↴
    881000, 1300000, 2500000]
stress_runout = [210, 210, 205, 205, 205]
cycles_runout = [10 ** 7, 10 ** 7, 10 ** 7, 10 ** 7, 10 ** 7]
SN_diagram(stress=stress, cycles=cycles, stress_runout=stress_runout, cycles_ ↴
    runout=cycles_runout, method_for_bounds='residual', cycles_trace=[5 * 10 ** 5], ↴
    stress_trace=[260])
plt.show()
```



71.8.2 `acceleration_factor`

```
class reliability.PoF.acceleration_factor(AF=None, T_use=None, T_acc=None, Ea=None, print_results=True)
```

This function uses the Arrhenius model for Acceleration factor to determine the relationship between temperature and reaction rate.

Please see the [online documentation](#) for more details and formulas.

This function accepts `T_use` as a mandatory input and the user may specify any two of the three other variables, and the third variable will be found.

Parameters

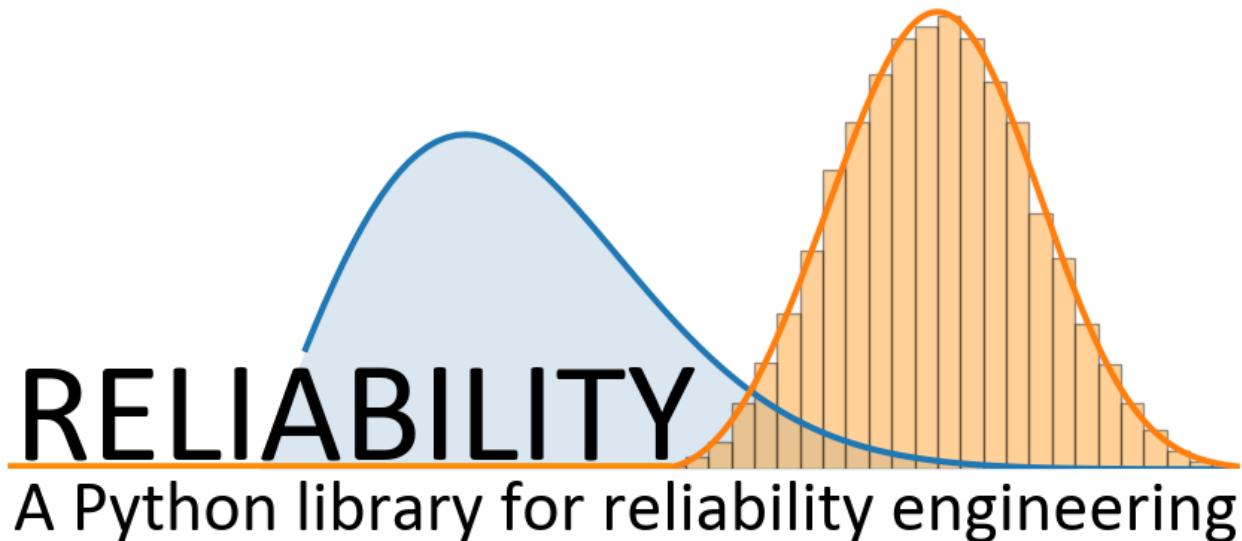
- `T_use` (*float, int*) – Temperature of usage (Celsius)
- `T_acc` (*float, int, optional*) – Temperature of acceleration (Celsius)
- `Ea` (*float, int, optional*) – Activation energy (eV)
- `AF` (*float, int, optional*) – Acceleration factor
- `print_results` (*bool, optional*) – Default is True. If True the results will be printed to the console.

Returns

- `AF` (*float*) – Acceleration Factor
- `T_acc` (*float*) – Accelerated temperature
- `T_use` (*float*) – Use temperature
- `Ea` (*float*) – Activation energy (eV)

Notes

Two of the three optional inputs must be specified and the third one will be found.



71.8.3 creep_failure_time

```
class reliability.PoF.creep_failure_time(temp_low, temp_high, time_low, C=20, print_results=True)
```

This function uses the Larson-Miller relation to find the time to failure due to creep.

The method uses a known failure time (time_low) at a lower failure temperature (temp_low) to find the unknown failure time at the higher temperature (temp_high).

This relation requires the input temperatures in Fahrenheit. To convert Celsius to Fahrenheit use $F = C1.8 + 32$

Note that the conversion between Fahrenheit and Rankine used in this calculation is $R = F + 459.67$

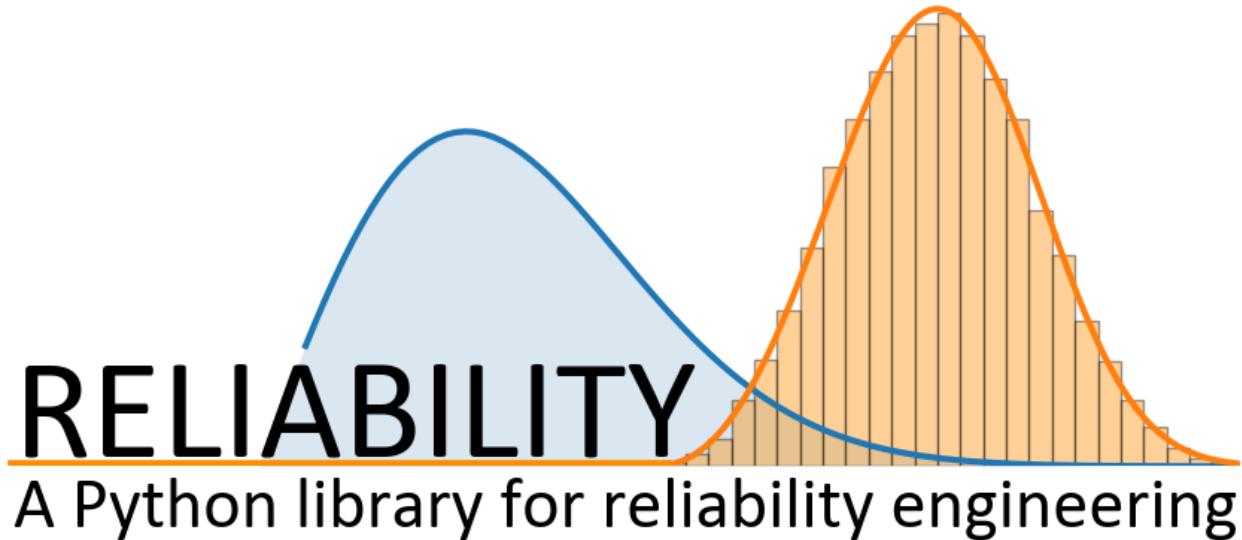
For more information see [Wikipedia](#).

Parameters

- **temp_low** (*float, int*) – The temperature (in degrees Fahrenheit) where the time_low is known
- **temp_high** (*float, int*) – The temperature (in degrees Fahrenheit) which time_high is unknown and will be found by this function
- **time_low** (*float, int*) – The time to failure at temp_low
- **C** (*float, int, optional*) – The creep constant. The default is 20. Typically 20-22 for metals.
- **print_results** (*bool, optional*) – If print_results is True, the output will also be printed to the console.

Returns

time_high (*float*) – The time to failure at the higher temperature.



71.8.4 `creep_rupture_curves`

```
class reliability.PoF.creep_rupture_curves(temp_array, stress_array, TTF_array, stress_trace=None, temp_trace=None)
```

This function plots the creep rupture curves for a given set of creep data. It also fits the lines of best fit to each temperature.

The time to failure for a given temperature can be found by specifying `stress_trace` and `temp_trace`.

Parameters

- `temp_array` (*array, list*) – The temperatures
- `stress_array` (*array, list*) – The stresses
- `TTF_array` (*array, list*) – The times to failure at the given temperatures and stresses
- `stress_trace` (*float, int, optional*) – The stress value used to determine the time to failure. Both `stress_trace` and `temp_trace` must be provided to calculate the time to failure.
- `temp_trace` (*float, int*) – The temperature value used to determine the time to failure. Both `stress_trace` and `temp_trace` must be provided to calculate the time to failure.

Returns

None – The plot is the only output. Use `plt.show()` to show it.

Notes

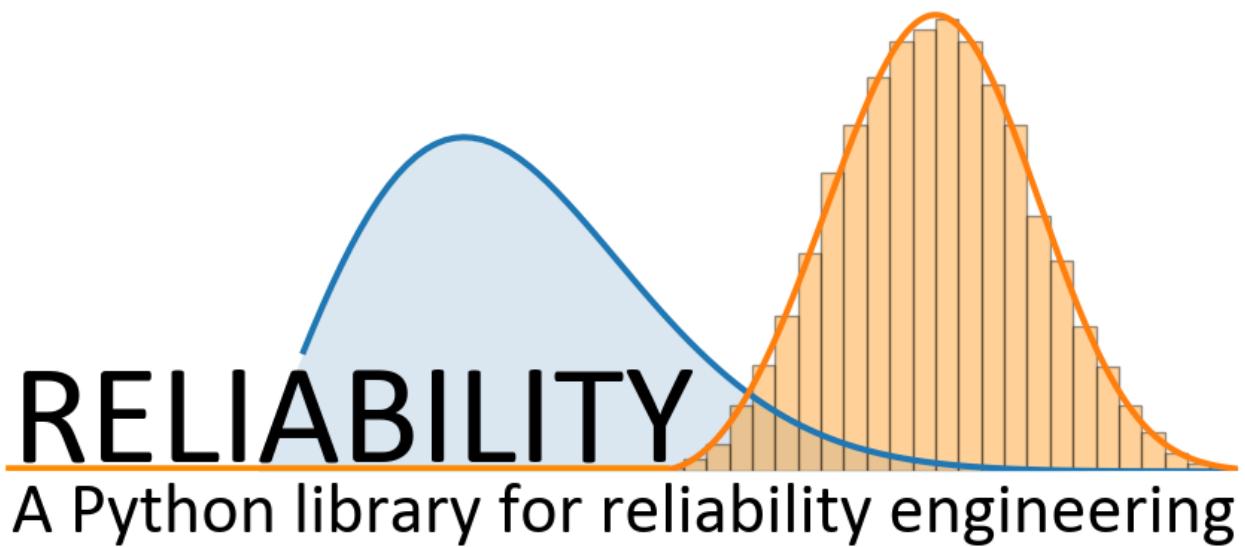
Example Usage:

```
from reliability.PoF import creep_rupture_curves
import matplotlib.pyplot as plt
TEMP = [900, 900, 900, 900, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000, 1100, 1100, 1100, 1100, 1100, 1200, 1200, 1200, 1200, 1350, 1350, 1350]
STRESS = [90, 82, 78, 70, 80, 75, 68, 60, 56, 49, 43, 38, 60.5, 50, 40, 29, 22, 40, 30, 25, 20, 20, 15, 10]
TTF = [37, 975, 3581, 9878, 7, 17, 213, 1493, 2491, 5108, 7390, 10447, 18, 167, 615, 2220, 6637, 19, 102, 125, 331, 3.7, 8.9, 31.8]
creep_rupture_curves(temp_array=TEMP, stress_array=STRESS, TTF_array=TTF, stress_trace=100, temp_trace=100)
```

(continues on next page)

(continued from previous page)

```
↳ trace=70, temp_trace=1100)
plt.show()
```



71.8.5 fracture_mechanics_crack_growth

```
class reliability.PoF.fracture_mechanics_crack_growth(Kc, C, m, P, W, t, Kt=1.0, a_initial=1.0,
                                                       D=None, a_final=None, crack_type='edge',
                                                       print_results=True, show_plot=True)
```

This function uses the principles of fracture mechanics to find the number of cycles required to grow a crack from an initial length until a final length. The final length (a_{final}) may be specified, but if not specified then a_{final} will be set as the critical crack length (a_{crit}) which causes failure due to rapid fracture.

This function performs the same calculation using two methods; simplified and iterative. The simplified method assumes that the geometry factor ($f(g)$), the stress (S_{net}), and the critical crack length (a_{crit}) are constant. This method is the way most textbooks show these problems solved as they can be done in a few steps. The iterative method does not make the assumptions that the simplified method does and as a result, the parameters $f(g)$, S_{net} and a_{crit} must be recalculated based on the current crack length at every cycle.

This function is applicable only to thin plates with an edge crack or a centre crack (which is to be specified using the parameter `crack_type`).

You may also use this function for notched components by specifying the parameters Kt and D which are based on the geometry of the notch. For any notched components, this method assumes the notched component has a “shallow notch” where the notch depth (D) is much less than the plate width (W). The value of Kt for notched components may be found using the [efatigue website](#). In the case of notched components, the local stress concentration from the notch will often cause slower crack growth. In these cases, the crack length is calculated in two parts (stage 1 and stage 2) which can clearly be seen on the plot using the iterative method. The only geometry this function is designed for is unnotched and notched thin flat plates. No centre holes are allowed.

Parameters

- **Kc** (*float, int*) – fracture toughness
- **Kt** (*float, int, optional*) – The stress concentration factor. Default is 1 for no notch.
- **D** (*float, int, None, optional*) – Depth of the notch. Default is None for no notch. A notched specimen is assumed to be doubly-notched (equal notches on both sides).

- **C** (*float, int*) – Material constant (sometimes referred to as A).
- **m** (*float, int*) – Material constant (sometimes referred to as n). This value must not be 2 due to the way the formula works.
- **P** (*float, int*) – External load on the material (MPa)
- **t** (*float, int*) – Plate thickness (mm)
- **W** (*float, int*) – Plate width (mm)
- **a_initial** (*float, int, optional*) – Initial crack length (mm). Default is 1 mm.
- **a_final** (*float, int, None, optional*) – Final crack length (mm) - default is None in which case a_final is assumed to be a_crit (length at failure). It is useful to be able to enter a_final in cases where there are multiple loading regimes over time.
- **crack_type** (*str, optional*) – Must be either ‘edge’ or ‘center’. Default is ‘edge’. The geometry factor used for each of these in the simplified method is 1.12 for edge and 1.0 for center. The iterative method calculates these values exactly using a_initial and W (plate width).
- **print_results** (*bool, optional*) – Default is True. If True, the results will be printed to the console.
- **show_plot** (*bool, optional*) – Default is True. If True the iterative method’s crack growth will be plotted.

Returns

- **Nf_stage_1_simplified** (*float*) – Number of cycles in stage 1 of crack growth using the simplified method.
- **Nf_stage_2_simplified** (*float*) – Number of cycles in stage 2 of crack growth using the simplified method.
- **Nf_total_simplified** (*float*) – Total number of cycles (**Nf_stage_1_simplified** + **Nf_stage_2_simplified**)
- **final_crack_length_simplified** (*float*) – Final crack length using the simplified method
- **transition_length_simplified** (*float*) – The transition length (stage 1 - 2 interface) using the simplified method.
- **Nf_stage_1_iterative** (*float*) – Number of cycles in stage 1 of crack growth using the iterative method.
- **Nf_stage_2_iterative** (*float*) – Number of cycles in stage 2 of crack growth using the iterative method.
- **Nf_total_iterative** (*float*) – Total number of cycles (**Nf_stage_1_iterative** + **Nf_stage_2_iterative**)
- **final_crack_length_iterative** (*float*) – Final crack length using the iterative method
- **transition_length_iterative** (*float*) – The transition length (stage 1 - 2 interface) using the iterative method.

Notes

Example usage:

```
from reliability.PoF import fracture_mechanics_crack_growth
fracture_mechanics_crack_growth(Kc=66,C=6.91*10**-12,m=3,P=0.15,W=100,t=5,Kt=2.41,a_
initial=1,D=10,crack_type='edge')
```

(continues on next page)

(continued from previous page)

```

print('')
fracture_mechanics_crack_growth(Kc=66,C=3.81*10**-12,m=3,P=0.103,W=100,t=5,crack_
→type='center')

"""

Results from fracture_mechanics_crack_growth:
SIMPLIFIED METHOD (keeping f(g), S_max, and a_crit as constant):
Crack growth was found in two stages since the transition length ( 2.08 mm ) due to
→the notch, was greater than the initial crack length ( 1 mm ).  

Stage 1 (a_initial to transition length): 6802 cycles  

Stage 2 (transition length to a_final): 1133 cycles  

Total cycles to failure: 7935 cycles.  

Critical crack length to cause failure was found to be: 7.86 mm.

ITERATIVE METHOD (recalculating f(g), S_max, and a_crit for each cycle):
Crack growth was found in two stages since the transition length ( 2.45 mm ) due to
→the notch, was greater than the initial crack length ( 1 mm ).  

Stage 1 (a_initial to transition length): 7576 cycles  

Stage 2 (transition length to a_final): 671 cycles  

Total cycles to failure: 8247 cycles.  

Critical crack length to cause failure was found to be: 6.39 mm.

Results from fracture_mechanics_crack_growth:
SIMPLIFIED METHOD (keeping f(g), S_max, and a_crit as constant):
Crack growth was found in a single stage since the transition length ( 0.0 mm ) was
→less than the initial crack length 1.0 mm.  

Total cycles to failure: 281359 cycles.  

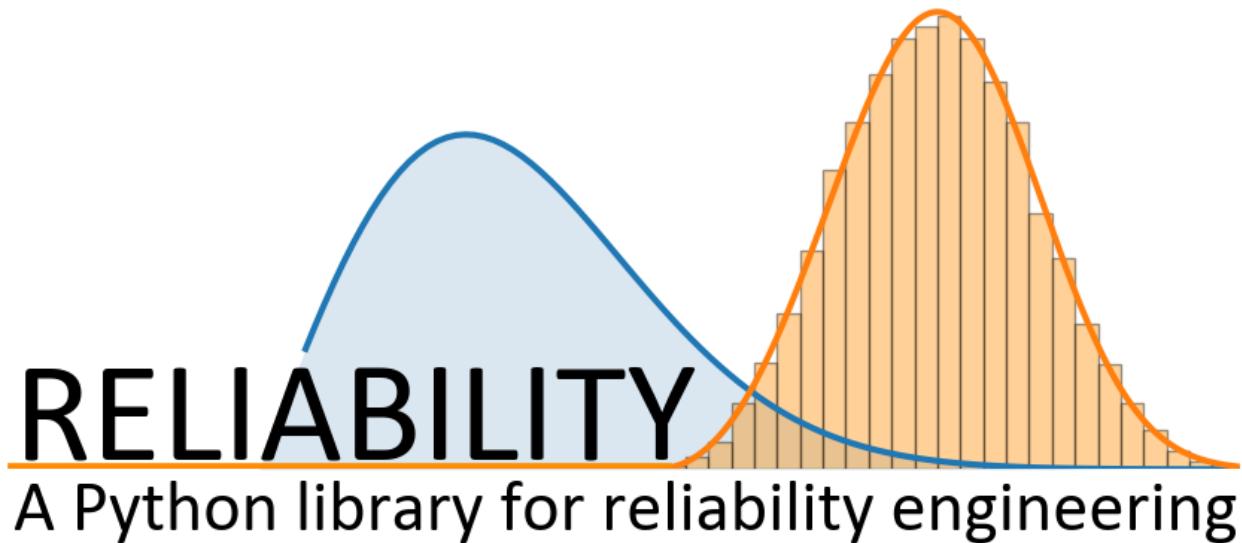
Critical crack length to cause failure was found to be: 32.67 mm.

ITERATIVE METHOD (recalculating f(g), S_max, and a_crit for each cycle):
Crack growth was found in a single stage since the transition length ( 0.0 mm ) was
→less than the initial crack length 1.0 mm.  

Total cycles to failure: 225827 cycles.  

Critical crack length to cause failure was found to be: 18.3 mm.
"""

```



71.8.6 fracture_mechanics_crack_initiation

```
class reliability.PoF.fracture_mechanics_crack_initiation(P, A, Sy, E, K, n, b, c, sigma_f,
                                                               epsilon_f, Kt=1.0, q=1.0,
                                                               mean_stress_correction_method='modified_morrow',
                                                               print_results=True)
```

This function uses the material properties, the local cross sectional area, and force applied to the component to determine how many cycles until crack initiation (of a 1mm crack).

Units should always be in MPa (and mm² for area). This function may be used for an un-notched or notched component. If the component is un-notched, the parameters q and Kt may be left as their default values of 1.

While there are formulas to find the parameters q and Kt, these formulas have not been included here so that the function is reasonably generic to different materials and geometries. Resources for finding some of these parameters if they are not given to you:

- $q = 1/(1+a/r)$ Where r is the notch radius of curvature (in mm), and a is $0.025^*(2070/S_u)$.
- S_u is the ultimate strength in MPa. This only applies to high strength steels where $S_u > 550$ MPa.
- Kt can be calculated using the [efatigue website](#). This website will provide you with the appropriate Kt for your notched geometry.

Parameters

- **P** (*float, int*) – Force applied on the component [units of MPa].
- **A** (*float, int*) – Cross sectional area of the component (at the point of crack initiation) [units of mm²].
- **Sy** (*float, int*) – Yield strength of the material [units of MPa].
- **E** (*float, int*) – Elastic modulus (Young's modulus) [units of MPa]
- **K** (*float, int*) – Strength coefficient of the material
- **n** (*float, int*) – Strain hardening exponent of the material
- **b** (*float, int*) – Elastic strain exponent of the material

- **c** (*float, int*) – Plastic strain exponent of the material
- **sigma_f** (*float, int*) – Fatigue strength coefficient of the material
- **epsilon_f** (*float, int*) – Fatigue strain coefficient of the material
- **q** (*float, int, optional*) – Notch sensitivity factor. Default is 1 for no notch.
- **Kt** (*float, int, optional*) – Stress concentration factor. Default is 1 for no notch.
- **mean_stress_correction_method** (*str, optional*) – Must be either ‘morrow’, ‘modified_morrow’, or ‘SWT’. Default is ‘modified_morrow’ as this is the same as the uncorrected Coffin-Manson relationship when mean stress is zero.
- **print_results** (*bool, optional*) – The results will be printed to the console if `print_results` is True.

Returns

- **sigma_max** (*float*) – The maximum stress
- **sigma_min** (*float*) – The minimum stress
- **sigma_mean** (*float*) – The mean stress
- **epsilon_max** (*float*) – The maximum strain
- **epsilon_min** (*float*) – The minimim strain
- **epsilon_mean** (*float*) – The mean strain
- **cycles_to_failure** (*float*) – The number of cycles until failure due to fatigue

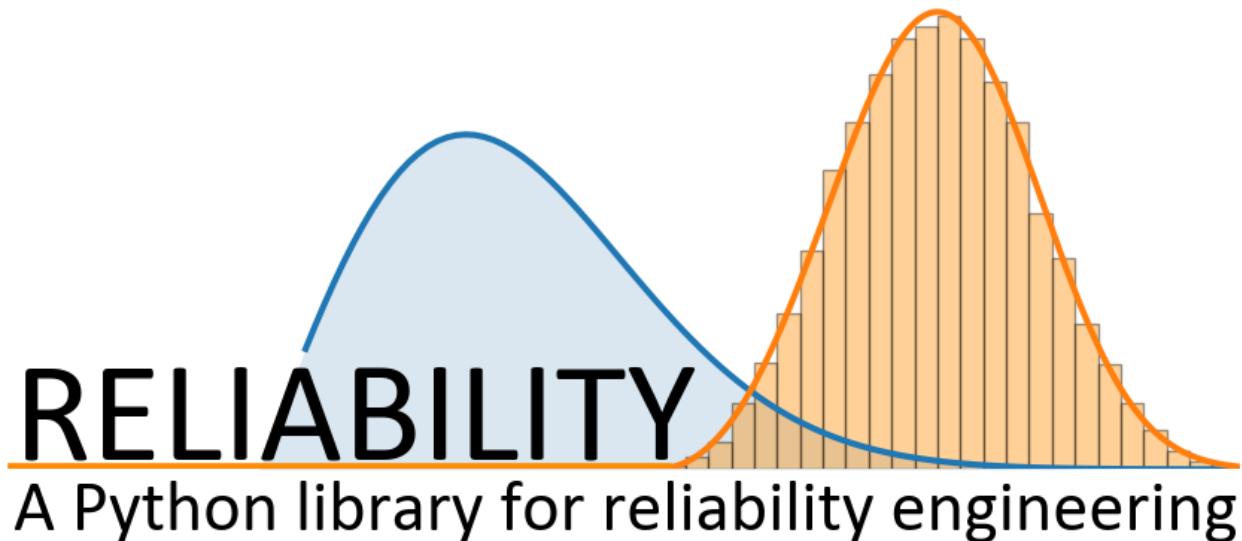
Notes

Example usage:

```
from reliability.PoF import fracture_mechanics_crack_initiation
fracture_mechanics_crack_initiation(P=0.15, A=5*80, Kt=2.41, q=0.9857, Sy=690,_
→E=210000, K=1060, n=0.14, b=-0.081, c=-0.65, sigma_f=1160, epsilon_f=1.1,mean_
→stress_correction_method='SWT')

"""

Results from fracture_mechanics_crack_initiation:
A crack of 1 mm will be formed after: 2919.91 cycles (5839.82 reversals).
Stresses in the component: Min = -506.7291 MPa , Max = 506.7291 MPa , Mean = -5.
→684341886080802e-14 MPa.
Strains in the component: Min = -0.0075 , Max = 0.0075 , Mean = 8.673617379884035e-
→19
Mean stress correction method used: SWT
""
```



71.8.7 `palmgren_miner_linear_damage`

```
class reliability.PoF.palmgren_miner_linear_damage(rated_life, time_at_stress, stress)
```

Uses the Palmgren-Miner linear damage hypothesis to calculate the outputs.

Parameters

- **rated life** (*array, list*) – How long the component will last at a given stress level.
- **time_at_stress** (*array, list*) – How long the component is subjected to the stress that gives the specified rated_life
- **stress** (*float, int*) – What stress the component is subjected to. This is not used in the calculation but is required for printing the output. Ensure that the time_at_stress and rated life are in the same units as the answer will also be in those units

Returns

None – The printed results are the only output.

Notes

The output will print the results to the console. The printed results include:

- Fraction of life consumed per load cycle
- service life of the component
- Fraction of damage caused at each stress level

Example usage: Ball bearings are fail after 50000 hrs, 6500 hrs, and 1000 hrs, after being subjected to a stress of 1kN, 2kN, and 4kN respectively. If each load cycle involves 40 mins at 1kN, 15 mins at 2kN, and 5 mins at 4kN, how long will the ball bearings last?

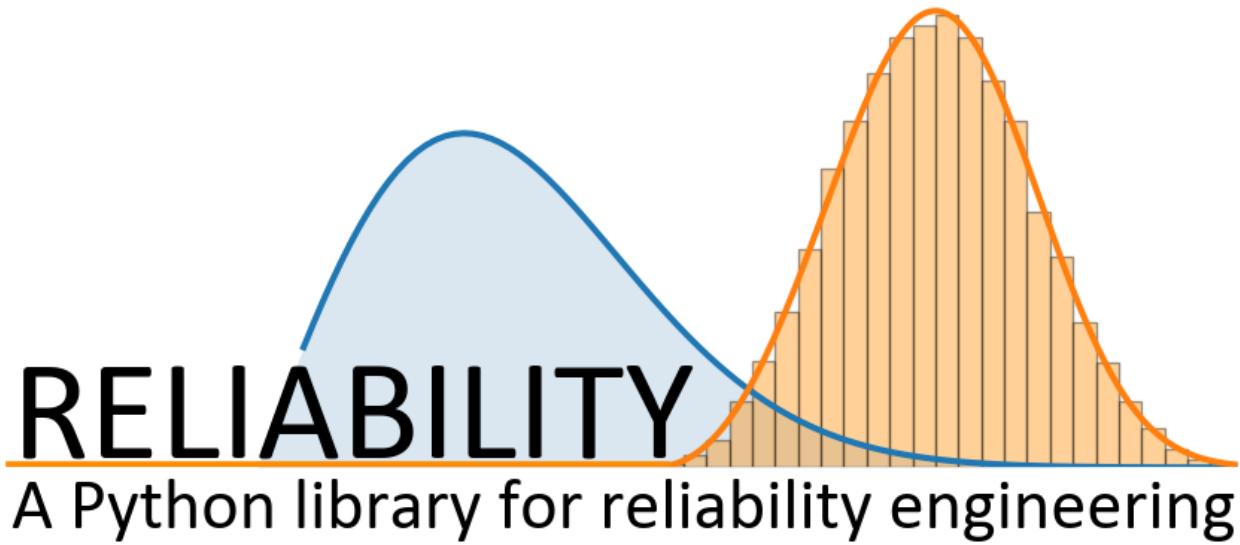
```
from reliability.PoF import palmgren_miner_linear_damage
palmgren_miner_linear_damage(rated_life=[50000, 6500, 1000], time_at_stress=[40/60, ↴15/60, 5/60], stress=[1, 2, 4])
```

```
""
```

(continues on next page)

(continued from previous page)

Palmgren-Miner Linear Damage Model results:
Each load cycle uses 0.01351 % of the components life.
The service life of the component is 7400.37951 load cycles.
The amount of damage caused at each stress level is:
Stress = 1 , Damage fraction = 9.86717 %.
Stress = 2 , Damage fraction = 28.463 %.
Stress = 4 , Damage fraction = 61.66983 %.
""



71.8.8 strain_life_diagram

```
class reliability.PoF.strain_life_diagram(E, sigma_f, epsilon_f, b, c, K=None, n=None,
                                           mean_stress_correction_method='SWT', max_stress=None,
                                           min_stress=None, max_strain=None, min_stress=None,
                                           print_results=True, show_plot=True)
```

This function plots the strain-life diagram.

If you do not have the parameters `sigma_f`, `epsilon_f`, `b`, `c`, but you do have stress, strain, and cycles data then you can use the function ‘`stress_strain_life_parameters_from_data`’ to find these parameters.

Parameters

- `E` (`float, int`) – The modulus of elasticity. Ensure this is in the same units for which `K` and `n` were obtained (typically MPa)
- `sigma_f` (`float, int`) – The fatigue strength coefficient.
- `epsilon_f` (`float, int`) – The fatigue strain coefficient.
- `b` (`float, int`) – The elastic strain exponent.
- `c` (`float, int`) – The plastic strain exponent.
- `K` (`float, int, optional`) – The cyclic strength coefficient. Optional input. Only required if you specify `max_stress` or `max_strain`.
- `n` (`float, int, optional`) – The strain hardening exponent. Optional input. Only required if you specify `max_stress` or `max_strain`.

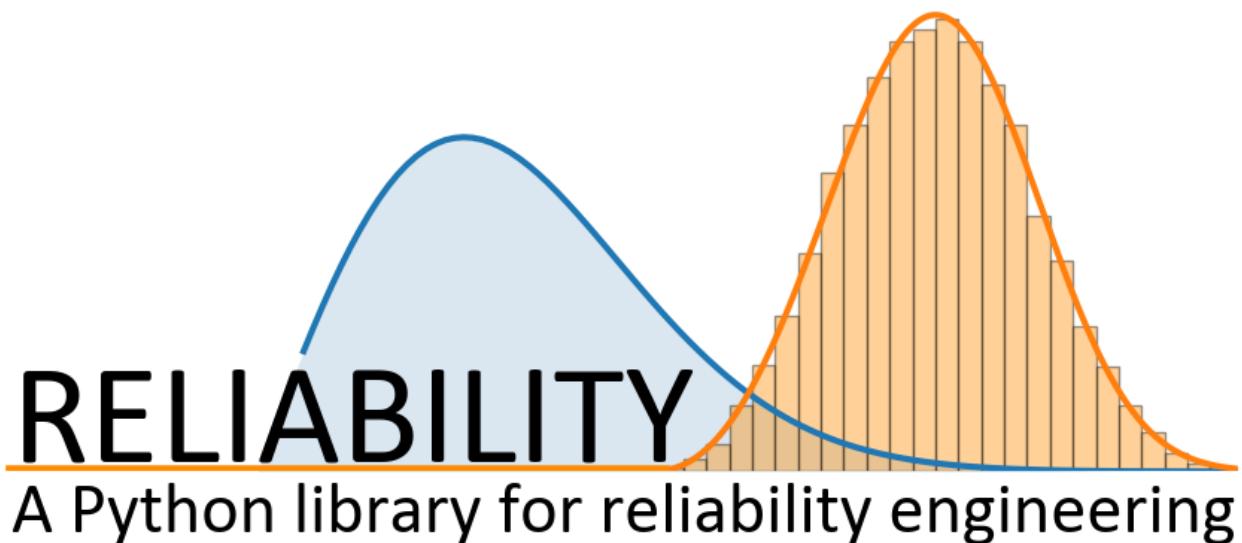
- **mean_stress_correction_method** (*str, optional*) – Must be either ‘morrow’, ‘modified_morrow’, or ‘SWT’. Default is ‘SWT’. Only used if `mean_stress` is found to be non-zero.
- **max_stress** (*float, int, optional*) – Specify the `max_stress` if you want cycles to failure. If specified, you will need to also specify `K` and `n`.
- **max_strain** (*float, int, optional*) – Specify the `max_strain` if you want cycles to failure.
- **min_stress** (*float, int, optional*) – If this is not `-max_stress` then specify it here. Optional input.
- **min_strain** (*float, int, optional*) – If this is not `-max_strain` then specify it here. Optional input.
- **print_results** (*bool, optional*) – Default is True. The cycles to failure will only be printed if `max_stress` OR `max_strain` is specified.
- **show_plot** (*bool, optional*) – Default is True. The strain-life plot will be generated if `show_plot` = True. Use `plt.show()` to show it.

Returns

- **cycles_to_failure** (*float*) – The number of cycles until fatigue failure
- **max_stress** (*float*) – The maximum stress
- **max_strain** (*float*) – The maximum strain
- **min_stress** (*float*) – The minimum stress
- **min_strain** (*float, int, optional*) – The minimum strain

Notes

When specifying min and max stress or strain, do not specify both stress and strain as the corresponding value will be automatically calculated. Only specify the min if it is not -max.



71.8.9 stress_strain_diagram

```
class reliability.PoF.stress_strain_diagram(K, n, E, max_strain=None, max_stress=None,
                                             min_stress=None, min_strain=None, print_results=True,
                                             initial_load_direction='tension')
```

This function plots the stress-strain diagram.

If you do not have the parameters K, n, but you do have stress and strain data then you can use the function 'stress_strain_life_parameters_from_data' to calculate K and n.

Parameters

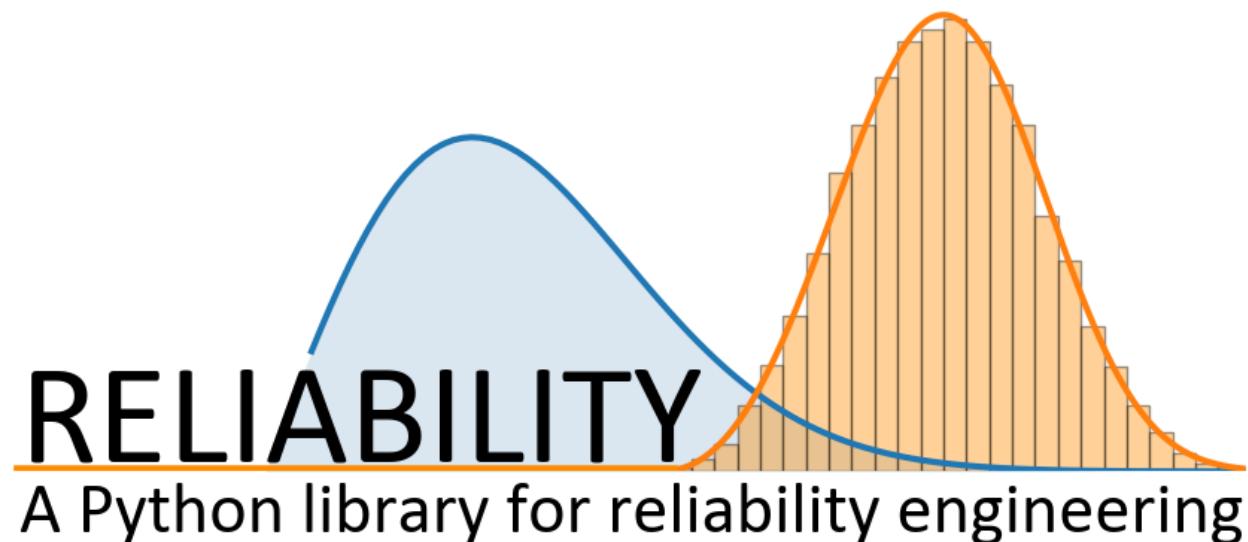
- **K** (*float, int*) – The cyclic strength coefficient
- **n** (*float, int*) – The strain hardening exponent
- **E** (*float, int*) – The modulus of elasticity. Ensure this is in the same units for which K and n were obtained (typically MPa).
- **max_strain** (*float, int*) – The maximum strain to use for cyclic loading when plotting the hysteresis loop.
- **max_stress** (*float, int*) – The maximum stress to use for cyclic loading when plotting the hysteresis loop.
- **min_strain** (*float, int, optional*) – If this is not -max_strain then specify it here. Optional input.
- **min_stress** (*float, int, optional*) – If this is not -max_stress then specify it here. Optional input.
- **initial_load_direction** (*str, optional*) – Must be ‘tension’ or ‘compression’. Default is tension.

Returns

None – The plot is the only output. All calculated values are shown on the plot.

Notes

When specifying min and max stress or strain, Do not specify both stress and strain as the corresponding value will be automatically calculated. Only specify the min if it is not -max.



71.8.10 `stress_strain_life_parameters_from_data`

```
class reliability.PoF.stress_strain_life_parameters_from_data(strain, stress, E, cycles=None,  
                                                               print_results=True,  
                                                               show_plot=True)
```

This function will use stress and strain data to calculate the stress-strain parameters: K, n. If cycles is provided it will also calculate the strain-life parameters: sigma_f, epsilon_f, b, c.

It is not possible to calculate the strain-life parameters without stress because stress is needed to find elastic strain.

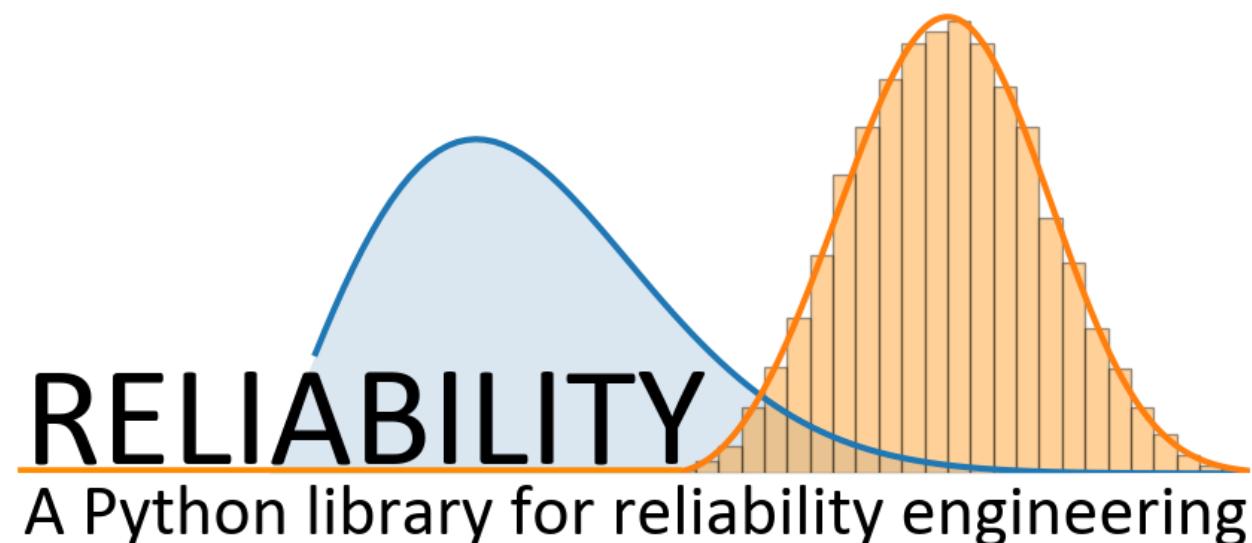
If you already have the parameters K, n, sigma_f, epsilon_f, b, c, then you can use the function 'stress_strain_diagram' for the plot.

Parameters

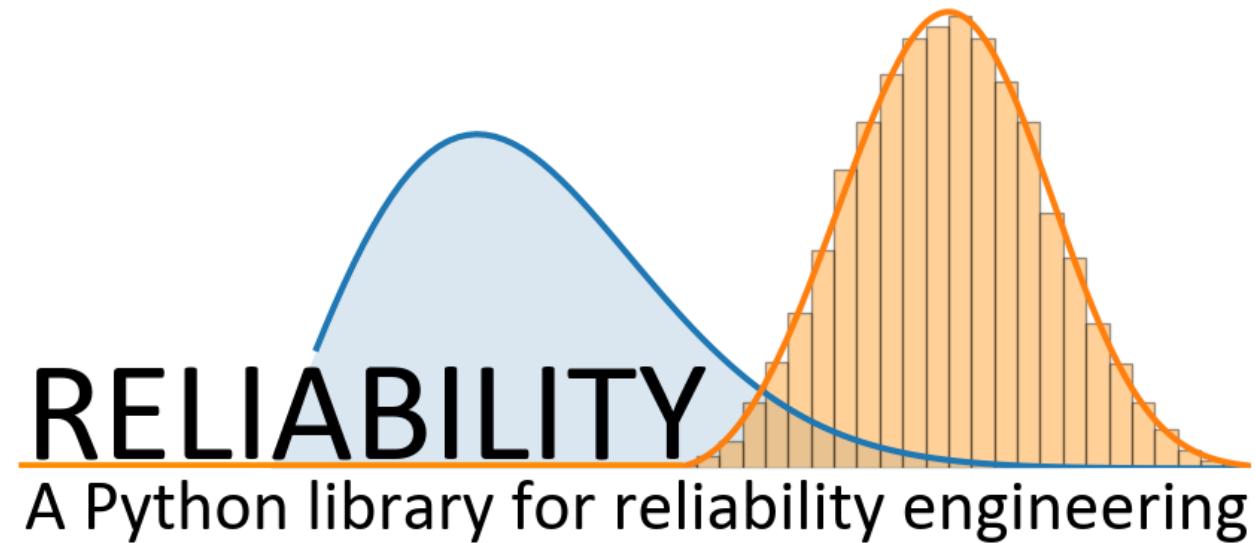
- **strain** (*array, list*) – The strain values
- **stress** (*array, list*) – The stress values
- **E** (*int, float*) – The modulus of elasticity. Ensure this is in the same units as stress (typically MPa).
- **cycles** (*array, list, optional*) – The number of cycles to failure. Optional input. This is required if you want to obtain the parameters sigma_f, epsilon_f, b, c
- **print_results** (*bool, optional*) – If True the results will be printed to console. Default is True.
- **show_plot** (*bool, optional*) – If True the stress strain diagram will be produced. Default is True. Use plt.show() to show it.

Returns

- **K** (*float*) – The cyclic strength coefficient
- **n** (*float*) – The cyclic strain hardening exponent
- **sigma_f** (*float*) – The fatigue strength coefficient. This is only generated if cycles is provided.
- **epsilon_f** (*float*) – The fatigue strain coefficient. This is only generated if cycles is provided.
- **b** (*float*) – The elastic strain exponent. This is only generated if cycles is provided.
- **c** (*float*) – The plastic strain exponent. This is only generated if cycles is provided.



71.9 Probability_plotting



71.9.1 Beta_probability_plot

```
class reliability.Probability_plotting.Beta_probability_plot(failures=None,
                                                               right_censored=None,
                                                               _fitted_dist_params=None,
                                                               a=None, CI=0.95,
                                                               show_fitted_distribution=True,
                                                               show_scatter_points=True,
                                                               downsample_scatterplot=False,
                                                               **kwargs)
```

Generates a probability plot on Beta scaled probability paper so that the CDF of the distribution appears linear.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations. times. Defaults = True.

- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

figure (*object*) – The figure handle of the probability plot is returned as an object

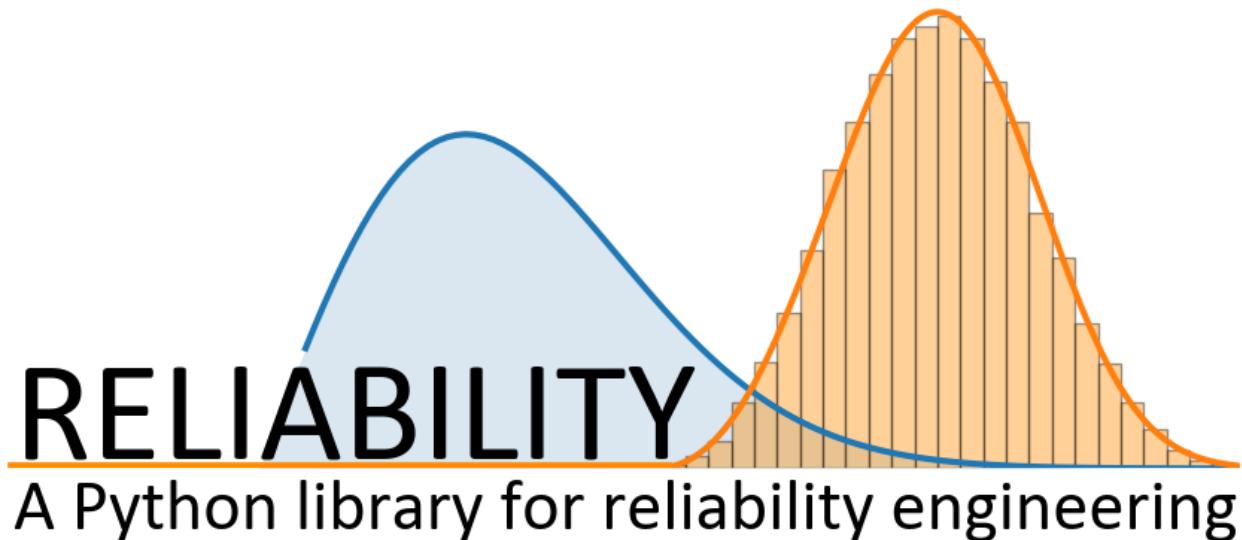
Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1. The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

Both parameters of a Beta Distribution affect the axes scaling such that when two different Beta Distributions are plotted on the same Beta probability paper, one of them will always appear curved.

If your plot does not appear automatically, use `plt.show()` to show it.

Confidence intervals are not included for the Beta distribution.



71.9.2 Exponential_probability_plot

```
class reliability.Probability_plotting.Exponential_probability_plot(failures=None,  
                                         right_censored=None,  
                                         fit_gamma=False, __fitted_dist_params=None,  
                                         a=None, CI=0.95,  
                                         show_fitted_distribution=True,  
                                         show_scatter_points=True,  
                                         downsample_scatterplot=False,  
                                         **kwargs)
```

Generates a probability plot on Exponentially scaled probability paper so that the CDF of the distribution appears linear. This differs from the Exponential_probability_plot_Weibull_Scale as Exponential paper will make multiple distributions with different Lambda parameters appear as lines radiating from the origin rather than as parallel lines. The parallel form is more convenient so the Weibull Scale is more commonly used than the Exponential Scale when plotting Exponential Distributions. This function can be used to show Exponential_1P or Exponential_2P distributions.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 1 element.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **fit_gamma** (*bool, optional*) – Specify this as True in order to fit the Exponential_2P distribution and scale the x-axis to time - gamma. Default = False.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure times. Defaults = True.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

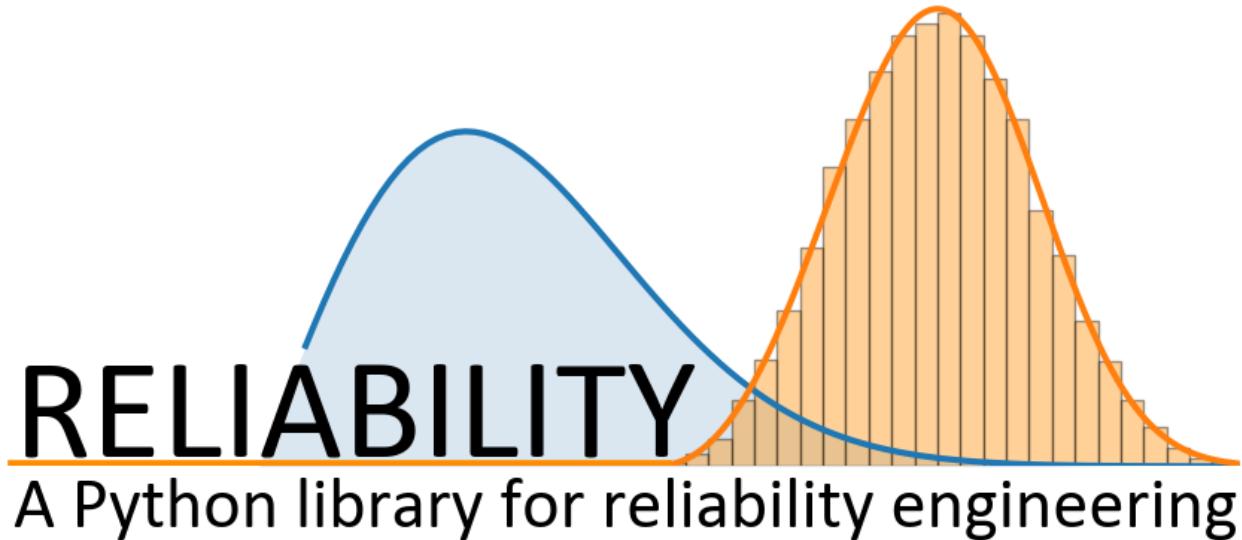
figure (*object*) – The figure handle of the probability plot is returned as an object

Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1. The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

`CI_type` is not required as the Exponential distribution has the same confidence interval bounds on both time and reliability.

If your plot does not appear automatically, use `plt.show()` to show it.



71.9.3 Exponential_probability_plot_Weibull_Scale

```
class reliability.Probability_plotting.Exponential_probability_plot_Weibull_Scale(failures=None,  
                                         right_censored=None,  
                                         fit_gamma=False,  
                                         fit=  
                                         fitted_dist_params=None,  
                                         a=None,  
                                         CI=0.95,  
                                         show_fitted_distribution=True,  
                                         show_scatter_points=True,  
                                         down=  
                                         sample_scatterplot=False,  
                                         **kwargs)
```

Generates a probability plot on Weibull scaled probability paper so that the CDF of the distribution appears linear. This differs from the Exponential probability plot on Exponential scaled probability paper as the Weibull paper will make multiple distributions with different Lambda parameters appear as parallel lines rather than as lines radiating from the origin. This change in scale has applications in ALT probability plotting. This function can be used to show Exponential_1P or Exponential_2P distributions.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 1 element.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **fit_gamma** (*bool, optional*) – Specify this as True in order to fit the Exponential_2P distribution and scale the x-axis to time - gamma. Default = False.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure times. Defaults = True.

- `downsample_scatterplot (bool, int, optional)` – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- `a (float, int, optional)` – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- `CI (float, optional)` – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- `kwargs` – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

`figure (object)` – The figure handle of the probability plot is returned as an object

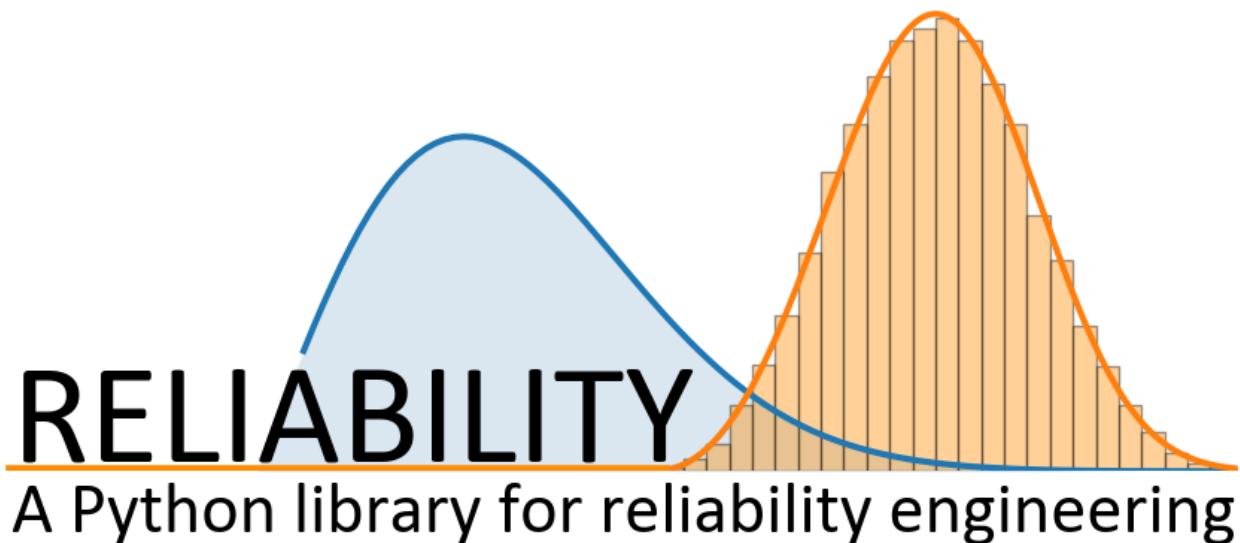
Notes

This function works because a Weibull Distribution with alpha = x and beta = 1 is identical to an Exponential Distribution with Lambda = 1/x.

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1. The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

`CI_type` is not required as the Exponential distribution has the same confidence interval bounds on both time and reliability.

If your plot does not appear automatically, use `plt.show()` to show it.



71.9.4 Gamma_probability_plot

```
class reliability.Probability_plotting.Gamma_probability_plot(failures=None,  
                                                               right_censored=None,  
                                                               fit_gamma=False,  
                                                               __fitted_dist_params=None,  
                                                               a=None, CI=0.95, CI_type='time',  
                                                               show_fitted_distribution=True,  
                                                               show_scatter_points=True,  
                                                               downsample_scatterplot=False,  
                                                               **kwargs)
```

Generates a probability plot on Gamma scaled probability paper so that the CDF of the distribution appears linear. This function can be used to show Gamma_2P or Gamma_3P distributions.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **fit_gamma** (*bool, optional*) – Specify this as True in order to fit the Gamma_3P distribution and scale the x-axis to time - gamma. Default = False.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure times. Defaults = True.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **CI_type** (*str; None, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use None to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or None. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

figure (*object*) – The figure handle of the probability plot is returned as an object

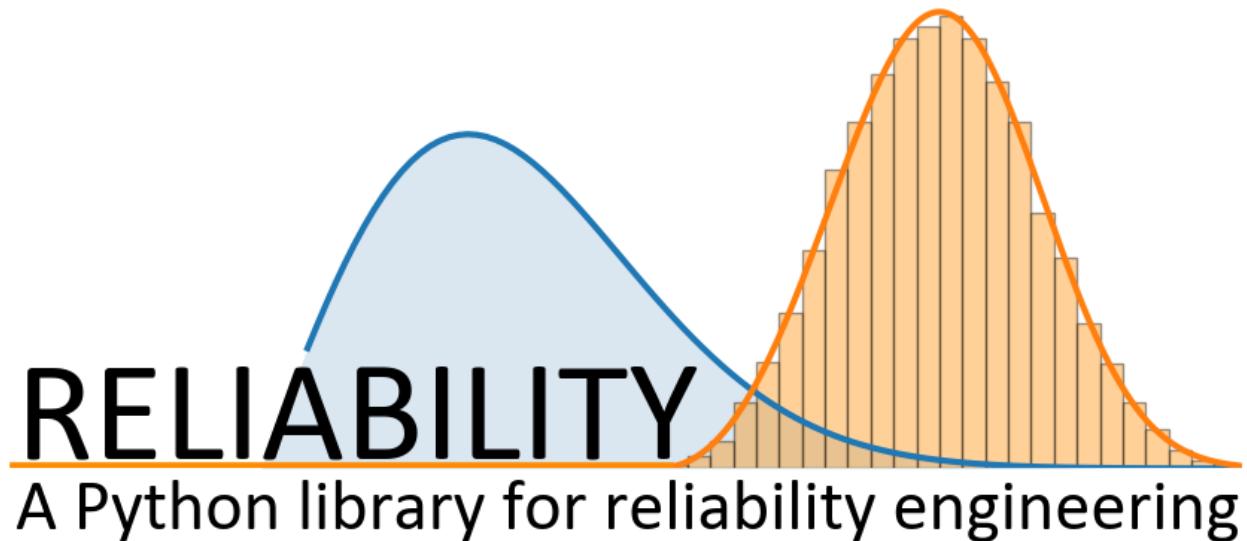
Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1.

The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

The beta parameter of a Gamma Distribution affects the axes scaling such that when two Gamma Distributions with different beta parameters are plotted on the same Gamma probability paper, one of them will always appear curved.

If your plot does not appear automatically, use plt.show() to show it.



71.9.5 Gumbel_probability_plot

```
class reliability.Probability_plotting.Gumbel_probability_plot(failures=None,
                                                               right_censored=None,
                                                               _fitted_dist_params=None,
                                                               a=None, CI=0.95,
                                                               CI_type='time',
                                                               show_fitted_distribution=True,
                                                               show_scatter_points=True,
                                                               downsample_scatterplot=False,
                                                               **kwargs)
```

Generates a probability plot on Gumbel scaled probability paper so that the CDF of the distribution appears linear.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure times. Defaults = True.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor

will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.

- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **CI_type** (*str, None, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use None to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or None. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **kwarg**s – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

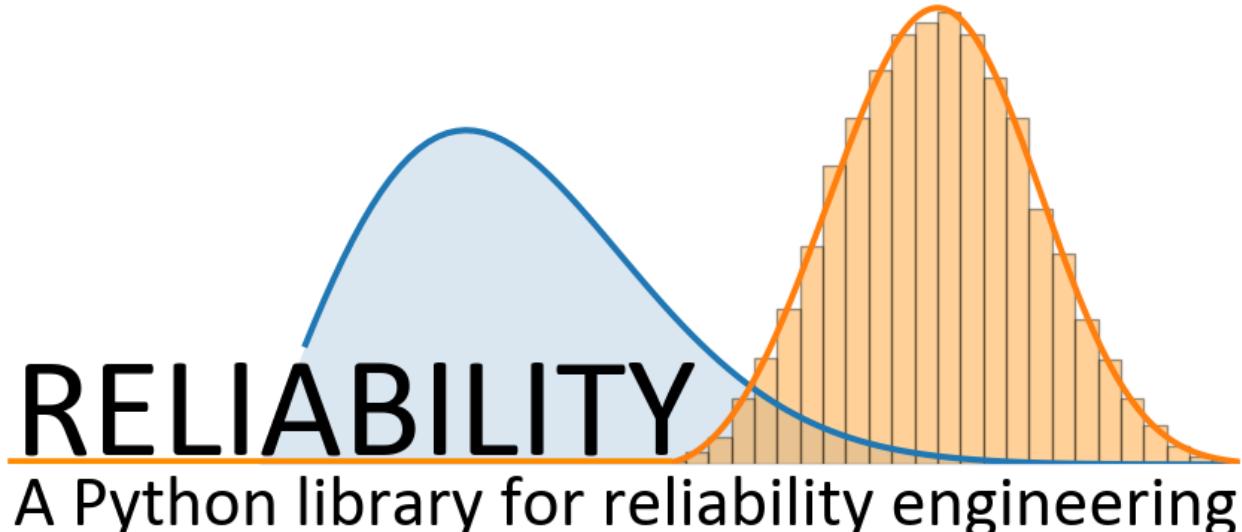
Returns

figure (*object*) – The figure handle of the probability plot is returned as an object

Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1. The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

If your plot does not appear automatically, use `plt.show()` to show it.



71.9.6 Loglogistic_probability_plot

```
class reliability.Probability_plotting.Loglogistic_probability_plot(failures=None,
                                                                    right_censored=None,
                                                                    fit_gamma=False, __fitted_dist_params=None,
                                                                    a=None, CI=0.95,
                                                                    CI_type='time',
                                                                    show_fitted_distribution=True,
                                                                    show_scatter_points=True,
                                                                    downsample_scatterplot=False,
                                                                    **kwargs)
```

Generates a probability plot on Loglogistically scaled probability paper so that the CDF of the distribution appears linear. This function can be used to show Loglogistic_2P or Loglogistic_3P distributions.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **fit_gamma** (*bool, optional*) – Specify this as True in order to fit the Loglogistic_3P distribution and scale the x-axis to time - gamma. Default = False.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure times. Defaults = True.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **CI_type** (*str, None, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use None to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or None. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

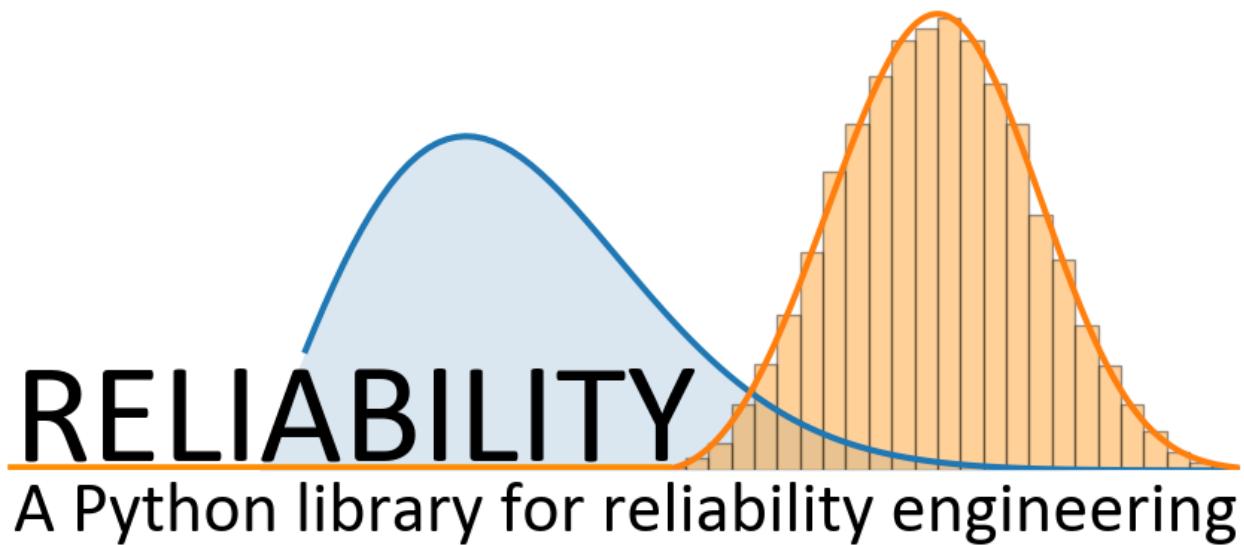
figure (*object*) – The figure handle of the probability plot is returned as an object

Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1.

The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

If your plot does not appear automatically, use plt.show() to show it.



71.9.7 Lognormal_probability_plot

```
class reliability.Probability_plotting.Lognormal_probability_plot(failures=None,  
                    right_censored=None,  
                    fit_gamma=False,  
                    _fitted_dist_params=None,  
                    a=None, CI=0.95,  
                    CI_type='time',  
                    show_fitted_distribution=True,  
                    show_scatter_points=True,  
                    downsample_scatterplot=False,  
                    **kwargs)
```

Generates a probability plot on Lognormal scaled probability paper so that the CDF of the distribution appears linear. This function can be used to show Lognormal_2P or Lognormal_3P distributions.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **fit_gamma** (*bool, optional*) – Specify this as True in order to fit the Lognormal_3P distribution and scale the x-axis to time - gamma. Default = False.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure times. Defaults = True.

- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: https://en.wikipedia.org/wiki/Q%20>93Q_plot#Heuristics
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **CI_type** (*str; None, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use None to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or None. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

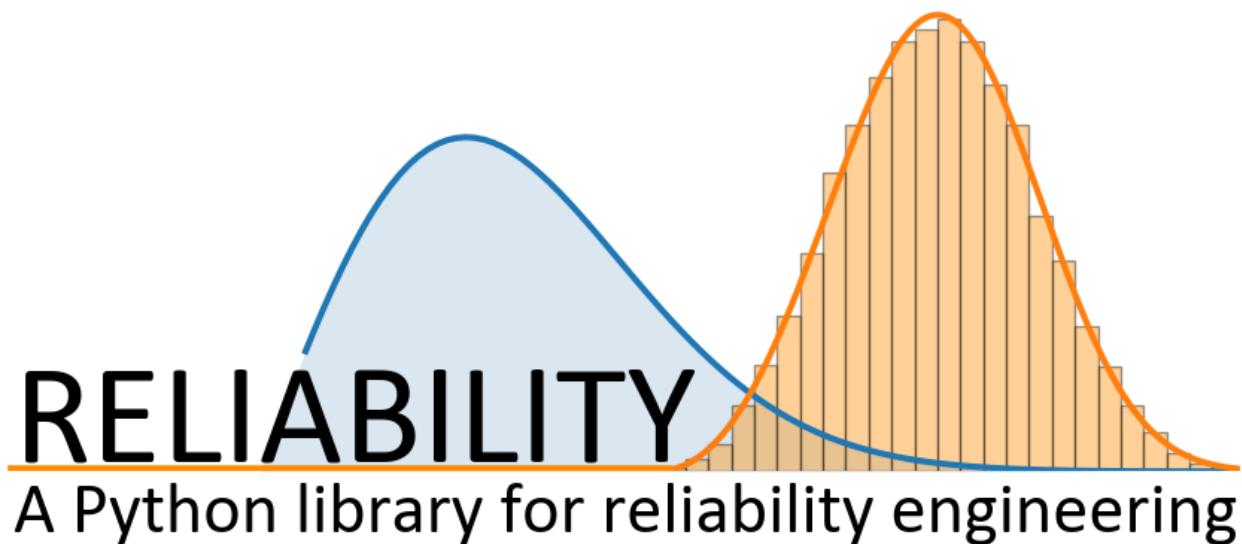
Returns

figure (*object*) – The figure handle of the probability plot is returned as an object

Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1. The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

If your plot does not appear automatically, use `plt.show()` to show it.



71.9.8 Normal_probability_plot

```
class reliability.Probability_plotting.Normal_probability_plot(failures=None,  
                                                               right_censored=None,  
                                                               __fitted_dist_params=None,  
                                                               a=None, CI=0.95,  
                                                               CI_type='time',  
                                                               show_fitted_distribution=True,  
                                                               show_scatter_points=True,  
                                                               downsample_scatterplot=False,  
                                                               **kwargs)
```

Generates a probability plot on Normal scaled probability paper so that the CDF of the distribution appears linear.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **show_fitted_distribution** (*bool, optional*) – If True, the fitted distribution will be plotted on the probability plot. Defaults = True.
- **show_scatter_points** (*bool, optional*) – If True, the plot will include the scatter points from the failure times. Defaults = True.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **CI_type** (*str, None, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use None to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or None. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

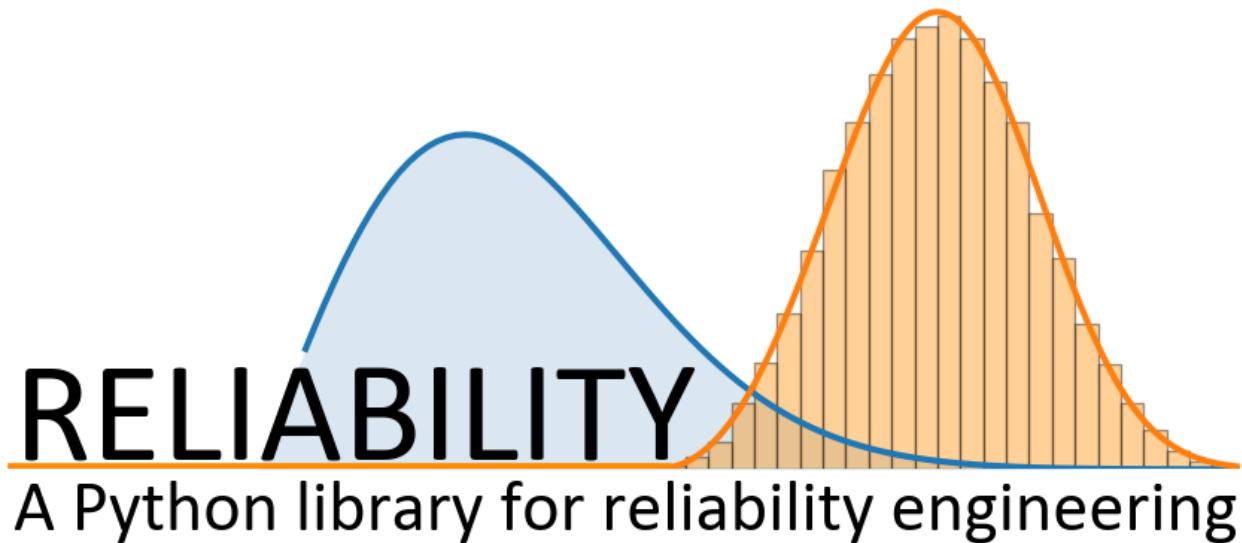
Returns

figure (*object*) – The figure handle of the probability plot is returned as an object

Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1. The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

If your plot does not appear automatically, use `plt.show()` to show it.



71.9.9 PP_plot_parametric

```
class reliability.Probability_plotting.PP_plot_parametric(X_dist=None, Y_dist=None,
                                                          y_quantile_lines=None,
                                                          x_quantile_lines=None,
                                                          show_diagonal_line=False,
                                                          downsample_scatterplot=False,
                                                          **kwargs)
```

The PP plot (probability-probability plot) consists of plotting the CDF of one distribution against the CDF of another distribution. If the distributions are similar, the PP plot will lie on the diagonal. This version of a PP plot is the fully parametric form in which we plot one distribution against another distribution. There is also a semi-parametric form offered in PP_plot_semiparametric.

Parameters

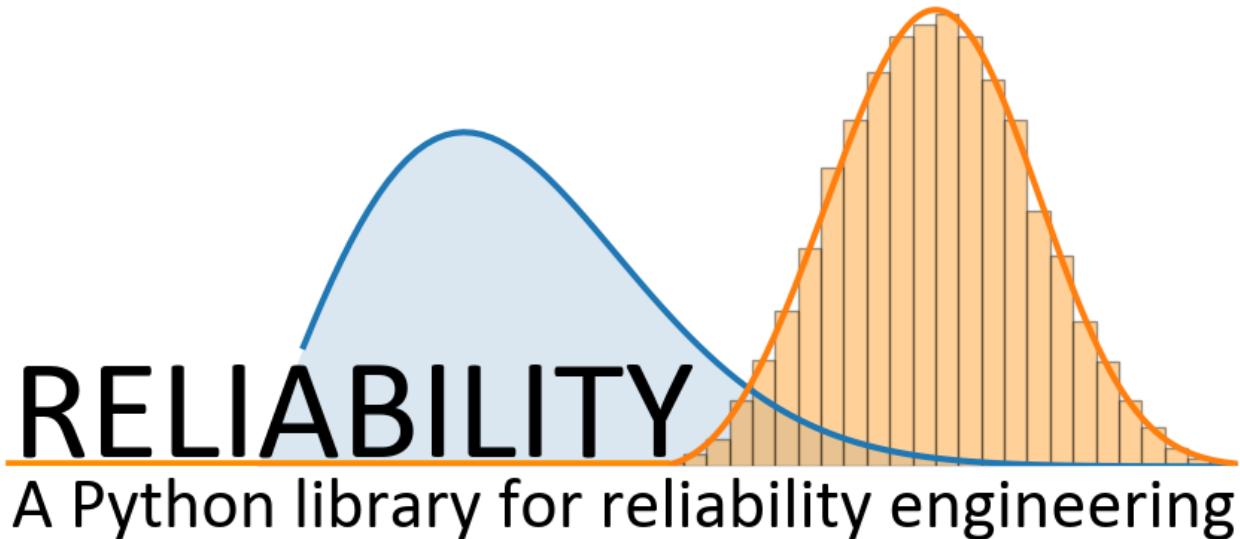
- **X_dist** (*object*) – A probability distribution object created using the reliability.Distributions module. The CDF of this distribution will be plotted along the X-axis.
- **Y_dist** (*object*) – A probability distribution object created using the reliability.Distributions module. The CDF of this distribution will be plotted along the Y-axis.
- **y_quantile_lines** (*array, list, optional*) – Starting points for the trace lines to find the X equivalent of the Y-quantile. Optional input. Default = None
- **x_quantile_lines** (*array, list, optional*) – Starting points for the trace lines to find the Y equivalent of the X-quantile. Optional input. Default = None
- **show_diagonal_line** (*bool, optional*) – If True the diagonal line will be shown on the plot. Default = False
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

figure (object) – The figure handle of the PP plot is returned as an object

Notes

If your plot does not appear automatically, use plt.show() to show it.



71.9.10 PP_plot_semiparametric

```
class reliability.Probability_plotting.PP_plot_semiparametric(X_data_failures=None,  
                                                               X_data_right_censored=None,  
                                                               Y_dist=None,  
                                                               show_diagonal_line=True,  
                                                               method='KM',  
                                                               downsample_scatterplot=False,  
                                                               **kwargs)
```

A PP plot (probability-probability plot) consists of plotting the CDF of one distribution against the CDF of another distribution. If we have both distributions we can use the function PP_plot_parametric. This function is for when we want to compare a fitted distribution to an empirical distribution for a given set of data. If the fitted distribution is a good fit the PP plot will lie on the diagonal line. The main purpose of this type of plot is to assess the goodness of fit in a graphical way. To create a semi-parametric PP plot, we must provide the failure data and the method ('KM' for Kaplan-Meier, 'NA' for Nelson-Aalen, 'RA' for Rank Adjustment) to estimate the empirical CDF, and we must also provide the parametric distribution for the parametric CDF. The failure times are the limiting values here so the parametric CDF is only calculated at the failure times since that is the result from the empirical CDF.

Parameters

- **X_data_failures** (*array, list*) – The failure times.
- **X_data_right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **Y_dist** (*object*) – A probability distribution created using the reliability.Distributions module. The CDF of this distribution will be plotted along the Y-axis.

- **method** (*str, optional*) – Must be ‘KM’, ‘NA’, or ‘RA’ for Kaplan-Meier, Nelson-Aalen, and Rank Adjustment respectively. Default = ‘KM’.
- **show_diagonal_line** (*bool*) – Default = True. If True the diagonal line will be shown on the plot.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

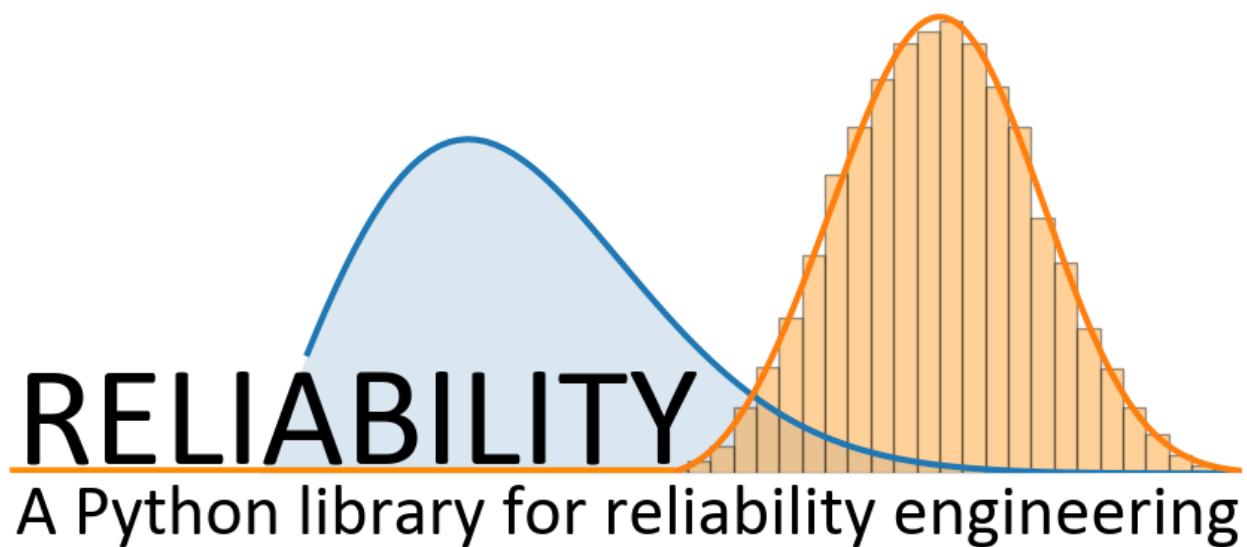
Returns

figure (*object*) – The figure handle of the PP plot is returned as an object

Notes

The empirical CDF also accepts X_data_right_censored just as Kaplan-Meier, Nelson-Aalen, and Rank Adjustment will also accept right censored data.

If your plot does not appear automatically, use plt.show() to show it.



71.9.11 QQ_plot_parametric

```
class reliability.Probability_plotting.QQ_plot_parametric(X_dist=None, Y_dist=None,
                                                       show_fitted_lines=True,
                                                       show_diagonal_line=False,
                                                       downsample_scatterplot=False,
                                                       **kwargs)
```

A QQ plot (quantile-quantile plot) consists of plotting failure units vs failure units for shared quantiles. A quantile is simply the fraction failing (ranging from 0 to 1). To generate this plot we calculate the failure units (these may be units of time, strength, cycles, landings, etc.) at which a certain fraction has failed (0.01, 0.02, 0.03... 0.99) for each distribution and plot them together. The time (or any other failure unit) at which a given fraction has failed is found using the inverse survival function. If the distributions are similar in shape, then the QQ plot should be

a reasonably straight line. By plotting the failure times at equal quantiles for each distribution we can obtain a conversion between the two distributions which is useful for Field-to-Test conversions that are necessary during accelerated life testing (ALT).

Parameters

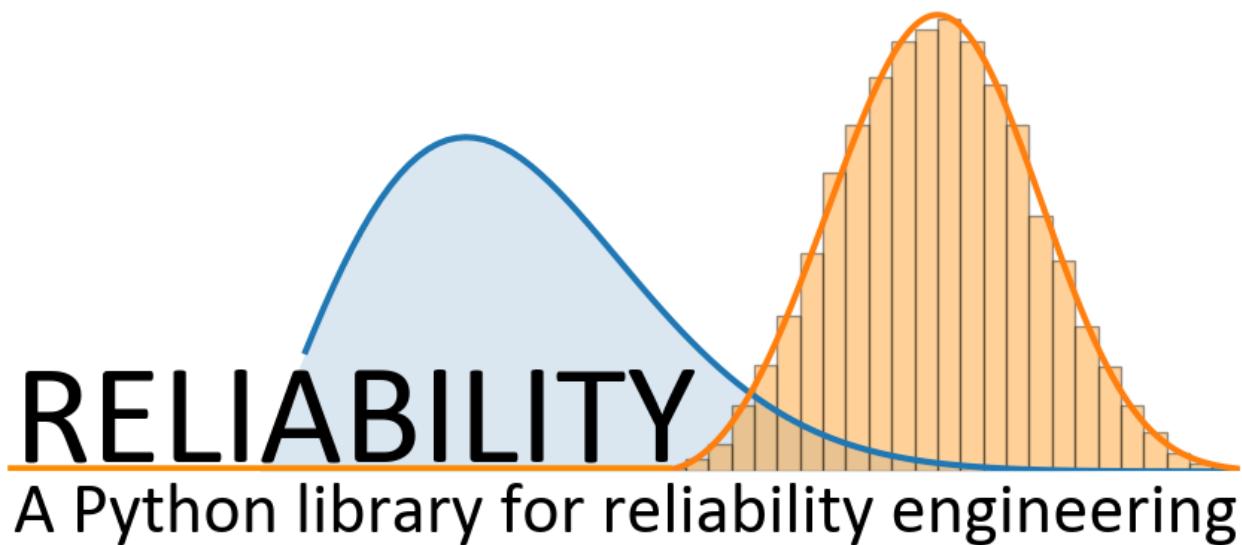
- **X_dist** (*object*) – A probability distribution object created using the reliability.Distributions module. The failure times at given quantiles from this distribution will be plotted along the X-axis.
- **Y_dist** (*object*) – A probability distribution object created using the reliability.Distributions module. The failure times at given quantiles from this distribution will be plotted along the Y-axis.
- **show_fitted_lines** (*bool*) – Default = True. These are the $Y=mX$ and $Y=mX+c$ lines of best fit.
- **show_diagonal_line** (*bool*) – Default = False. If True the diagonal line will be shown on the plot.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

model_parameters (*list*) – [m,m1,c1] - these are the values for the lines of best fit. m is used in $Y=mX$, and m1 and c1 are used in $Y=m1X+c1$

Notes

If your plot does not appear automatically, use plt.show() to show it.



71.9.12 QQ_plot_sempiparametric

```
class reliability.Probability_plotting.QQ_plot_sempiparametric(X_data_failures=None,
                                                               X_data_right_censored=None,
                                                               Y_dist=None,
                                                               show_fitted_lines=True,
                                                               show_diagonal_line=False,
                                                               method='KM',
                                                               downsample_scatterplot=False,
                                                               **kwargs)
```

A QQ plot (quantile-quantile plot) consists of plotting failure units vs failure units for shared quantiles. A quantile is simply the fraction failing (ranging from 0 to 1). When we have two parametric distributions we can plot the failure times for common quantiles against one another using `QQ_plot_parametric`. `QQ_plot_sempiparametric` is a semiparametric form of a QQ plot in which we obtain theoretical quantiles using a non-parametric estimate and a specified distribution. To generate this plot we begin with the failure units (these may be units of time, strength, cycles, landings, etc.). We then obtain an empirical CDF using either Kaplan-Meier, Nelson-Aalen, or Rank Adjustment. The empirical CDF gives us the quantiles we will use to equate the actual and theoretical failure times. Once we have the empirical CDF, we use the inverse survival function of the specified distribution to obtain the theoretical failure times and then plot the actual and theoretical failure times together. If the specified distribution is a good fit, then the QQ plot should be a reasonably straight line along the diagonal. The primary purpose of this plot is as a graphical goodness of fit test.

Parameters

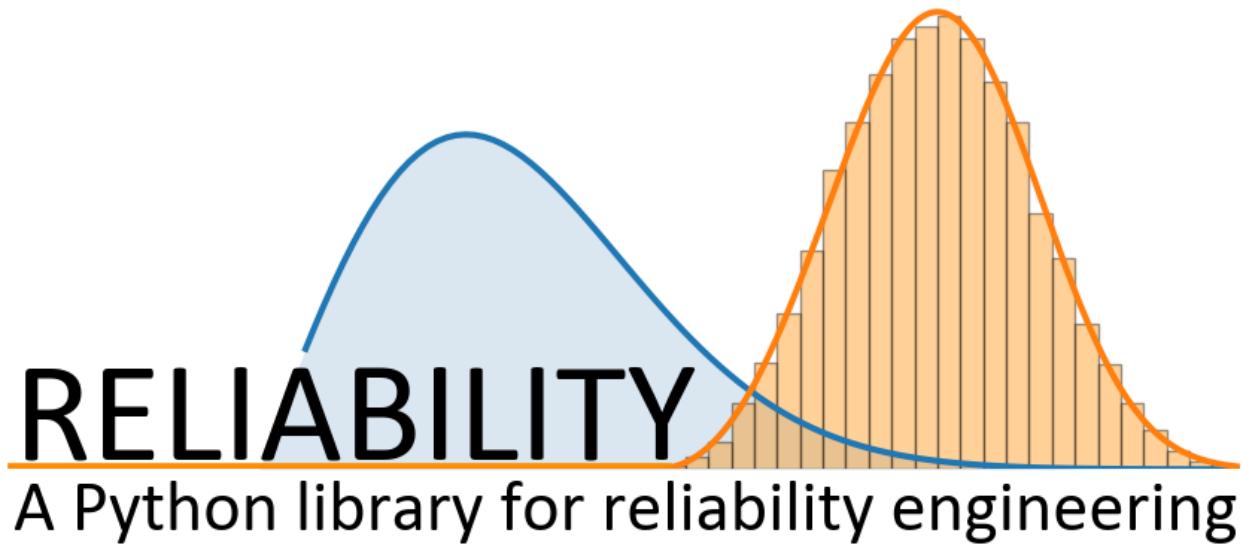
- **X_data_failures** (*list, array*) – The failure times. These will be plotted along the X-axis.
- **X_data_right_censored** (*list, array, optional*) – The right censored failure times. Optional input.
- **Y_dist** (*object*) – A probability distribution created using the `reliability.Distributions` module. The quantiles of this distribution will be plotted along the Y-axis.
- **method** (*str*) – Must be either ‘KM’, ‘NA’, or ‘RA’ for Kaplan-Meier, Nelson-Aalen, and Rank-Adjustment respectively. Default = ‘KM’.
- **show_fitted_lines** (*bool*) – Default = True. These are the $Y=m.X$ and $Y=m.X+c$ lines of best fit.
- **show_diagonal_line** (*bool*) – Default = False. If True the diagonal line will be shown on the plot.
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Plotting keywords that are passed directly to `matplotlib` (e.g. `color`, `label`, `linestyle`).

Returns

model_parameters (*list*) – $[m, m1, c1]$ - these are the values for the lines of best fit. m is used in $Y=m.X$, and $m1$ and $c1$ are used in $Y=m1.X+c1$

Notes

If your plot does not appear automatically, use `plt.show()` to show it.



71.9.13 Weibull_probability_plot

```
class reliability.Probability_plotting.Weibull_probability_plot(failures=None,  
                                                               right_censored=None,  
                                                               fit_gamma=False,  
                                                               _fitted_dist_params=None,  
                                                               a=None, CI=0.95,  
                                                               CI_type='time',  
                                                               show_fitted_distribution=True,  
                                                               show_scatter_points=True,  
                                                               downsample_scatterplot=False,  
                                                               **kwargs)
```

Generates a probability plot on Weibull scaled probability paper so that the CDF of the distribution appears linear. This function can be used to show Weibull_2P or Weibull_3P distributions.

Parameters

- **failures** (*array, list*) – The failure data. Must have at least 2 elements.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = `None`.
- **fit_gamma** (*bool, optional*) – Specify this as `True` in order to fit the Weibull_3P distribution and scale the x-axis to time - γ . Default = `False`.
- **show_fitted_distribution** (*bool, optional*) – If `True`, the fitted distribution will be plotted on the probability plot. Defaults = `True`.
- **show_scatter_points** (*bool, optional*) – If `True`, the plot will include the scatter points from the failure times. Defaults = `True`.
- **downsample_scatterplot** (*bool, int, optional*) – If `True` or `None`, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used

as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.

- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default = 0.3 which is the median rank method (same as the default in Minitab). For more heuristics, see: <https://en.wikipedia.org/wiki/Q%20plot#Heuristics>
- **CI** (*float, optional*) – The confidence interval for the bounds. Must be between 0 and 1. Optional input. Default = 0.95 for 95% CI.
- **CI_type** (*str, None, optional*) – This is the confidence bounds on time or reliability shown on the plot. Use None to turn off the confidence intervals. Must be either ‘time’, ‘reliability’, or None. Default is ‘time’. Some flexibility in names is allowed (eg. ‘t’, ‘time’, ‘r’, ‘rel’, ‘reliability’ are all valid).
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

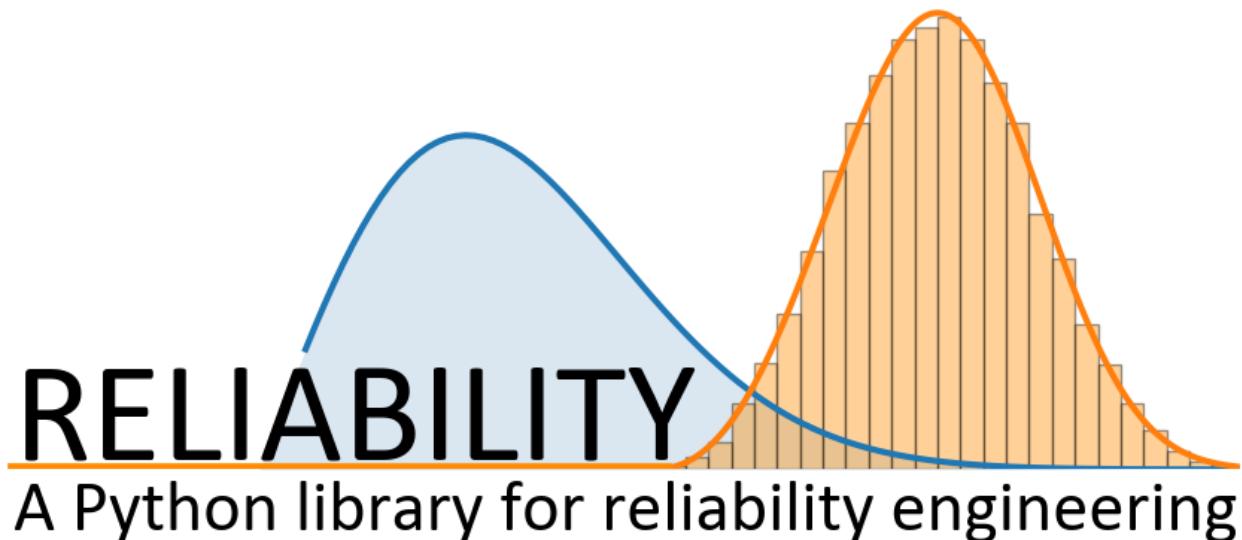
Returns

figure (*object*) – The figure handle of the probability plot is returned as an object

Notes

There is a hidden parameter called `__fitted_dist_params` which is used to specify the parameters of the distribution that has already been fitted. Passing a distribution object to this parameter will bypass the fitting process and use the parameters of the distribution provided. When this is done the minimum length of failures can be 1. The distribution object must contain the SE and Cov of the parameters so it needs to be generated by the Fitters module.

If your plot does not appear automatically, use `plt.show()` to show it.



71.9.14 `plot_points`

```
class reliability.Probability_plotting.plot_points(failures=None, right_censored=None,
                                                   func='CDF', a=None,
                                                   downsample_scatterplot=False, **kwargs)
```

Plots the failure points as a scatter plot based on the plotting positions. This is similar to a probability plot, just without the axes scaling or the fitted distribution. It may be used to overlay the failure points with a fitted

distribution on either the PDF, CDF, SF, HF, or CHF. If you choose to plot the points for PDF or HF the points will not form a smooth curve as this process requires integration of discrete points which leads to a discontinuous plot. The PDF and HF points are correct but not as useful as CDF, SF, and CHF.

Parameters

- **failures** (*array, list*) – The failure times. Minimum number of points allowed is 1.
- **right_censored** (*array, list, optional*) – The right censored failure times. Optional input.
- **func** (*str*) – The distribution function to plot. Choose either ‘PDF’, ‘CDF’, ‘SF’, ‘HF’, or ‘CHF’. Default = ‘CDF’.
- **a** (*float, int*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$. Default is $a=0.3$ which is the median rank method (same as the default in Minitab). For more heuristics, see: https://en.wikipedia.org/wiki/Q%20E2%80%93Q_plot#Heuristics
- **downsample_scatterplot** (*bool, int, optional*) – If True or None, and there are over 1000 points, then the scatterplot will be downsampled by a factor. The default downsample factor will seek to produce between 500 and 1000 points. If a number is specified, it will be used as the downsample factor. Default is False which will result in no downsampling. This functionality makes plotting faster when there are very large numbers of points. It only affects the scatterplot not the calculations.
- **kwargs** – Keyword arguments for the scatter plot. Defaults are set for `color='k'` and `marker='.'`. These defaults can be changed using `kwargs`.

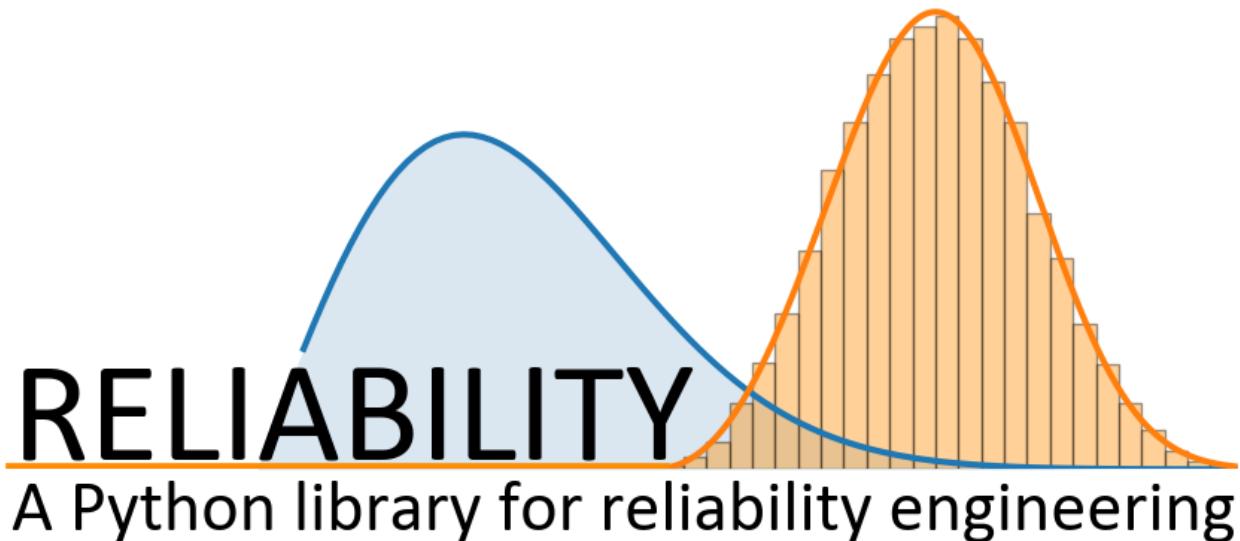
Returns

None

Notes

It is recommended that `plot_points` be used in conjunction with one of the plotting methods from a distribution (see the example below).

```
from reliability.Fitters import Fit_Lognormal_2P
from reliability.Probability_plotting import plot_points
import matplotlib.pyplot as plt
data = [8.0, 10.2, 7.1, 5.3, 8.5, 15.4, 17.7, 5.4, 5.8, 11.7, 4.4, 18.1, 8.5, 6.6, 9.7, 13.7, 8.2, 15.3, 2.9, 4.3]
fitted_dist = Fit_Lognormal_2P(failures=data, show_probability_plot=False, print_results=False) #fit the Lognormal distribution to the failure data
plot_points(failures=data, func='SF') #plot the failure points on the scatter plot
fitted_dist.distribution.SF() #plot the distribution
plt.show()
```



71.9.15 plotting_positions

```
class reliability.Probability_plotting.plotting_positions(failures=None, right_censored=None, a=None, sort=False)
```

Calculates the plotting positions for plotting on probability paper.

Parameters

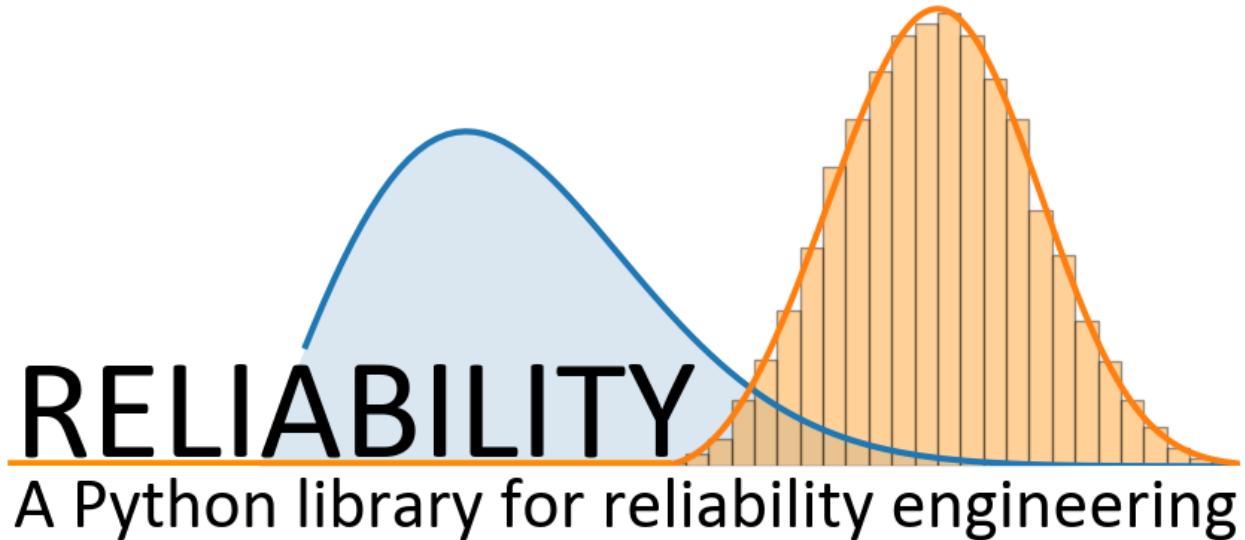
- **failures** (*array, list*) – The failure data. Must have at least 1 element.
- **right_censored** (*array, list, optional*) – The right censored failure data. Optional input. Default = None.
- **a** (*float, int, optional*) – The heuristic constant for plotting positions of the form $(k-a)/(n+1-2a)$ where k is the rank and n is the number of points. Optional input. Default is a = 0.3 which is the median rank method (same as the default in Minitab and Reliasoft). Must be in the range 0 to 1. For more heuristics, see: https://en.wikipedia.org/wiki/Quantile_plot#Heuristics
- **sort** (*bool, optional*) – Whether the output should be sorted. Default is False.

Returns

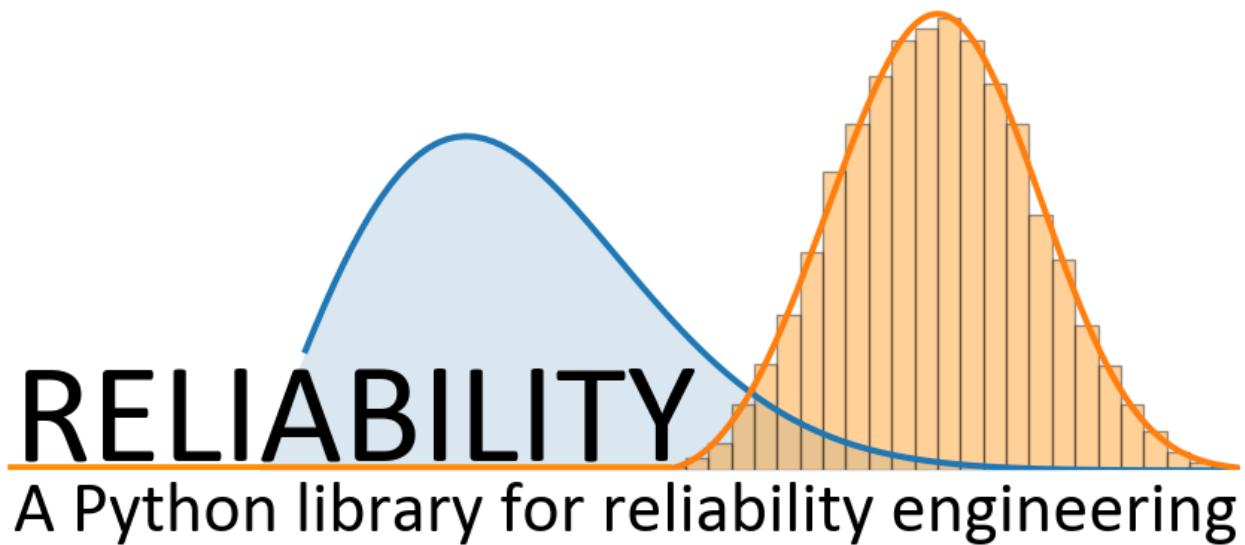
(array(x),array(y)) (*tuple*) – a tuple of two arrays. The arrays provide the x and y plotting positions. The x array will match the failures parameter while the y array will be the empirical estimate of the CDF at each of the failures.

Notes

This function is primarily used by the probability plotting functions.



71.10 Reliability_testing



71.10.1 KStest

```
class reliability.Reliability_testing.KStest(distribution, data, significance=0.05, print_results=True, show_plot=True)
```

Performs the Kolmogorov-Smirnov goodness of fit test to determine whether we can accept or reject the hypothesis that the data is from the specified distribution at the specified level of significance.

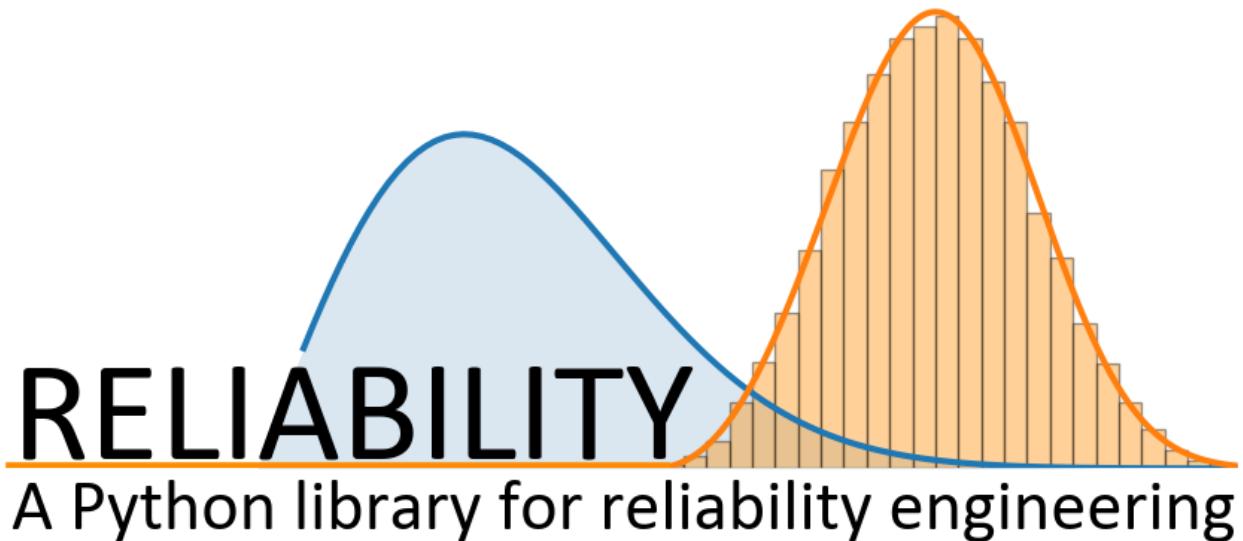
This method is not a means of comparing distributions (which can be done with AICc, BIC, and AD), but instead allows us to accept or reject a hypothesis that data come from a distribution.

Parameters

- **distribution** (*object*) – A distribution object created using the reliability.Distributions module.
- **data** (*array, list*) – The data that are hypothesised to come from the distribution.
- **significance** (*float*) – This is the complement of confidence. 0.05 significance is the same as 95% confidence. Must be between 0 and 0.5. Default = 0.05.
- **print_results** (*bool, optional*) – If True the results will be printed. Default = True
- **show_plot** (*bool, optional*) – If True a plot of the distribution CDF and empirical CDF will be generated. Default = True.

Returns

- **KS_statistic** (*float*) – The Kolmogorov-Smirnov statistic.
- **KS_critical_value** (*float*) – The Kolmogorov-Smirnov critical value.
- **hypothesis** (*string*) – ‘ACCEPT’ or ‘REJECT’. If KS_statistic < KS_critical_value then we can accept the hypothesis that the data is from the specified distribution.



71.10.2 chi2test

```
class reliability.Reliability_testing.chi2test(distribution, data, significance=0.05, bins=None, print_results=True, show_plot=True)
```

Performs the Chi-squared test for goodness of fit to determine whether we can accept or reject the hypothesis that the data is from the specified distribution at the specified level of significance.

This method is not a means of comparing distributions (which can be done with AICc, BIC, and AD), but instead allows us to accept or reject a hypothesis that data come from a distribution.

Parameters

- **distribution** (*object*) – A distribution object created using the reliability.Distributions module.
- **data** (*array, list*) – The data that are hypothesised to come from the distribution.
- **significance** (*float, optional*) – This is the complement of confidence. 0.05 significance is the same as 95% confidence. Must be between 0 and 0.5. Default = 0.05.

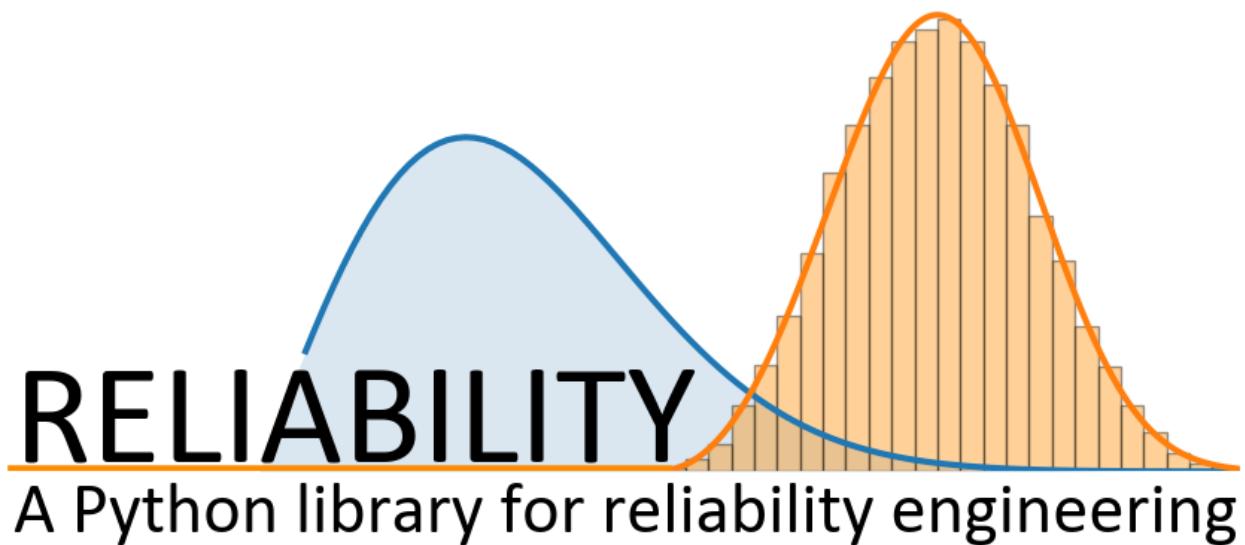
- **bins** (*array, list, string, optional*) – An array or list of the bin edges from which to group the data OR a string for the bin edge method from numpy. String options are ‘auto’, ‘fd’, ‘doane’, ‘scott’, ‘stone’, ‘rice’, ‘sturges’, or ‘sqrt’. Default = ‘auto’. For more information on these methods, see the numpy documentation: https://numpy.org/doc/stable/reference/generated/numpy.histogram_bin_edges.html
- **print_results** (*bool, optional*) – If True the results will be printed. Default = True
- **show_plot** (*bool, optional*) – If True a plot of the distribution and histogram will be generated. Default = True.

Returns

- **chisquared_statistic** (*float*) – The chi-squared statistic.
- **chisquared_critical_value** (*float*) – The chi-squared critical value.
- **hypothesis** (*string*) – ‘ACCEPT’ or ‘REJECT’. If chisquared_statistic < chisquared_critical_value then we can accept the hypothesis that the data is from the specified distribution
- **bin_edges** (*array*) – The bin edges used. If bins is a list or array then bin_edges = bins. If bins is a string then you can find the bin_edges that were calculated using this output.

Notes

The result is sensitive to the bins. For this reason, it is recommended to leave bins as the default value.



71.10.3 likelihood_plot

```
class reliability.Reliability_testing.likelihood_plot(distribution, failures, right_censored=None, CI=0.95, method='MLE', color=None)
```

Generates a likelihood contour plot. This is used for comparing distributions for statistically significant differences.

Parameters

- **distribution** (*str*) – The probability distribution to use for the likelihood plot. Must be one of Weibull, Normal, Gamma, Loglogistic, Lognormal, Gumbel, Beta.

- **failures** (*array, list*) – The failure data. Must have at least 2 failures.
- **right_censored** (*array, list, optional*) – The right censored data. Optional input. Default = None.
- **CI** (*float, array, list, optional*) – The confidence interval or an array of confidence intervals. All CI must be between 0.5 and 1. If an array of CI is specified there will be a contour at each CI. Default = 0.95 for 95% confidence intervals.
- **method** (*str, optional*) – The method used by the fitter. Must be either MLE, LS, RRX, RRY. Default is MLE.
- **color** (*str, optional*) – The color for the plot.

Returns

figure (*object*) – The figure handle of the likelihood plot is returned as an object.

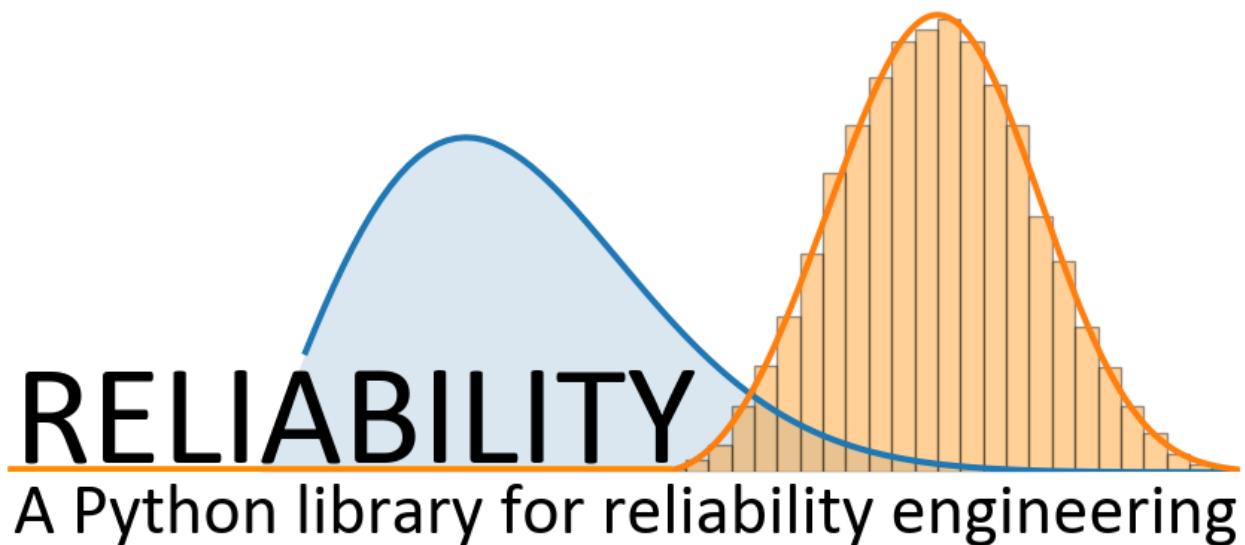
Notes

This plot is used to test for statistically significant differences between two or more distributions. If there is no overlap in the contours then there is a statistically significant difference between the distributions.

If your plot does not appear automatically, use plt.show() to show it.

Example usage:

```
from reliability.Reliability_testing import likelihood_plot import matplotlib.pyplot as plt
old_design = [2,9,23,38,67,2,11,28,40,76,3,17,33,45,90,4,17,34,55,115,6,19,34,56,126,9,21,37,57,197]
new_design=[15,116,32,148,61,178,67,181,75,183] likelihood_plot(distribution="Weibull",failures=old_design,CI=[0.9,0.95],likelihood_plot(distribution="Weibull",failures=new_design,CI=[0.9,0.95]) plt.show()
```



71.10.4 one_sample_proportion

```
class reliability.Reliability_testing.one_sample_proportion(trials=None, successes=None, CI=0.95, print_results=True)
```

Calculates the upper and lower bounds of reliability for a given number of trials and successes.

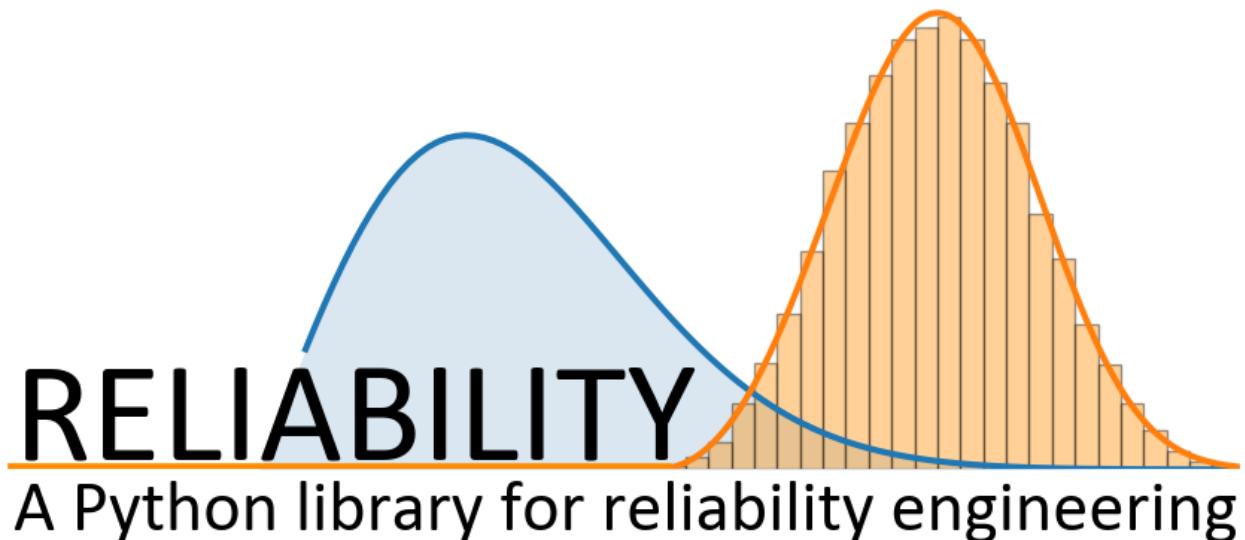
Parameters

- **trials** (*int*) – The number of trials which were conducted.

- **successes** (*int*) – The number of trials which were successful.
- **CI** (*float, optional*) – The desired confidence interval. Must be between 0 and 1. Default = 0.95 for 95% CI.
- **print_results** (*bool, optional*) – If True the results will be printed to the console. Default = True.

Returns

limits (*tuple*) – The confidence interval limits in the form (lower,upper).



71.10.5 reliability_test_duration

```
class reliability.Reliability_testing.reliability_test_duration(MTBF_required, MTBF_design,  
                                                               consumer_risk, producer_risk,  
                                                               one_sided=True,  
                                                               time_terminated=True,  
                                                               show_plot=True,  
                                                               print_results=True)
```

This function calculates the required duration for a reliability test to achieve the specified producers and consumers risks. This is done based on the specified MTBF required and MTBF design. For details please see the algorithm.

Parameters

- **MTBF_required** (*float, int*) – The required MTBF that the equipment must demonstrate during the test.
- **MTBF_design** (*float, int*) – The design target for the MTBF that the producer aims to achieve.
- **consumer_risk** (*float*) – The risk the consumer is accepting. This is the probability that a bad product will be accepted as a good product by the consumer.
- **producer_risk** (*float*) – The risk the producer is accepting. This is the probability that a good product will be rejected as a bad product by the consumer.
- **one_sided** (*bool, optional*) – The risk is analogous to the confidence interval, and the confidence interval can be one sided or two sided. Default = True.

- **time_terminated** (*bool, optional*) – Whether the test is time terminated or failure terminated. Typically it will be time terminated if the required test duration is sought. Default = True
- **show_plot** (*bool*) – If True, this will create a plot of the risk vs test duration. Default = True.
- **print_results** (*bool, optional*) – If True, this will print the results to the console. Default = True.

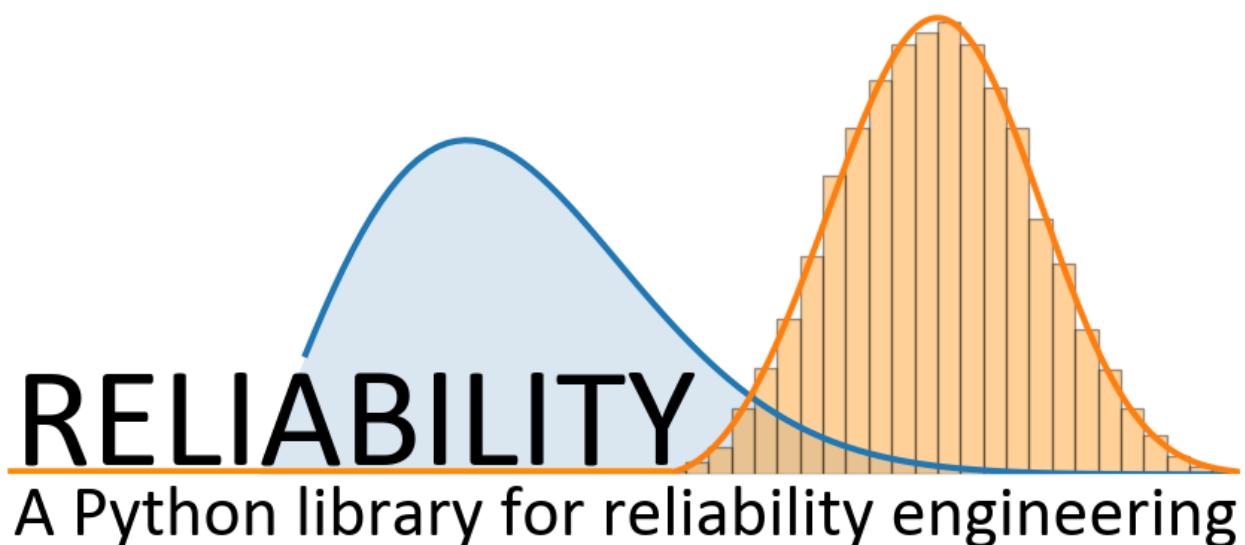
Returns

test_duration (*float*) – The required test duration to meet the input parameters.

Notes

The number of failures allowed is calculated but not provided by this function since the test will determine the actual number of failures so any prediction of number of failures ahead of time is not practical.

If the plot does not show automatically, use plt.show() to show it.



71.10.6 reliability_test_planner

```
class reliability.Reliability_testing.reliability_test_planner(MTBF=None,
                                                               number_of_failures=None,
                                                               CI=None, test_duration=None,
                                                               one_sided=True,
                                                               time_terminated=True,
                                                               print_results=True)
```

The function reliability_test_planner is used to solves for unknown test planning variables, given known variables. The Chi-squared distribution is used to find the lower confidence bound on MTBF for a given test duration, number of failures, and specified confidence interval.

The function will solve for any of the 4 variables, given the other 3. For example, you may want to know how many failures you are allowed to have in a given test duration to achieve a particular MTBF. The user must specify any 3 out of the 4 variables (not including one_sided, print_results, or time_terminated) and the remaining variable will be calculated.

Parameters

- **MTBF** (*float, int, optional*) – Mean Time Between Failures. This is the lower confidence bound on the MTBF. Units given in same units as the test_duration.
- **number_of_failures** (*int, optional*) – The number of failures recorded (or allowed) to achieve the MTBF. Must be ≥ 0 .
- **test_duration** (*float, int, optional*) – The amount of time on test required (or performed) to achieve the MTBF. May also be distance, rounds fires, cycles, etc. Units given in same units as MTBF.
- **CI** (*float, optional*) – The confidence interval at which the lower confidence bound on the MTBF is given. Must be between 0.5 and 1. For example, specify 0.95 for 95% confidence interval.
- **print_results** (*bool, optional*) – If True the results will be printed. Default = True.
- **one_sided** (*bool, optional*) – Use True for one-sided confidence interval and False for two-sided confidence interval. Default = True.
- **time_terminated** (*bool, optional*) – Use True for time-terminated test and False for failure-terminated test. Default = True.

Returns

- **MTBF** (*float*) – The lower bound on the MTBF.
- **number_of_failures** (*int*) – The number of failures allowed to achieve the MTBF at the specified CI and test_duration
- **test_duration** (*float*) – The required test duration
- **CI** (*float*) – The confidence interval.

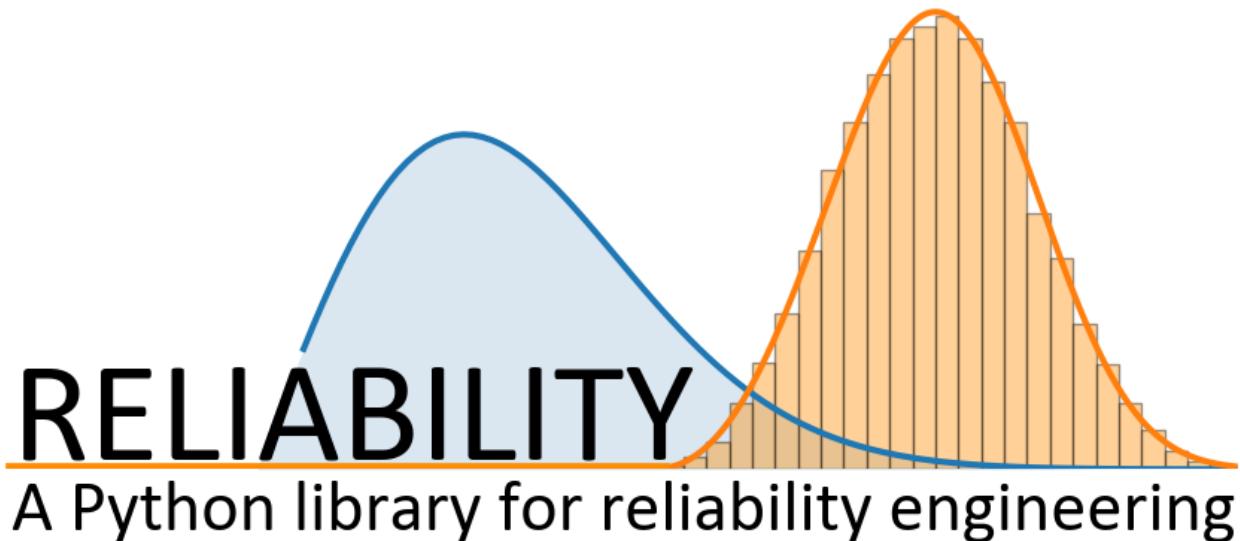
Notes

Please see the [documentation](#) for more detail on the equations used.

The returned values will match the input values with the exception of the input that was not provided.

The following example demonstrates how the MTBF is calculated:

```
from reliability.Reliability_testing import reliability_test_planner
reliability_test_planner(test_duration=19520, CI=0.8, number_of_failures=7)
>>> Reliability Test Planner results for time-terminated test
>>> Solving for MTBF
>>> Test duration: 19520
>>> MTBF (lower confidence bound): 1907.6398111904953
>>> Number of failures: 7
>>> Confidence interval (2 sided): 0.8
```



71.10.7 sample_size_no_failures

```
class reliability.Reliability_testing.sample_size_no_failures(reliability, CI=0.95, lifetimes=1,
                                                               weibull_shape=1,
                                                               print_results=True)
```

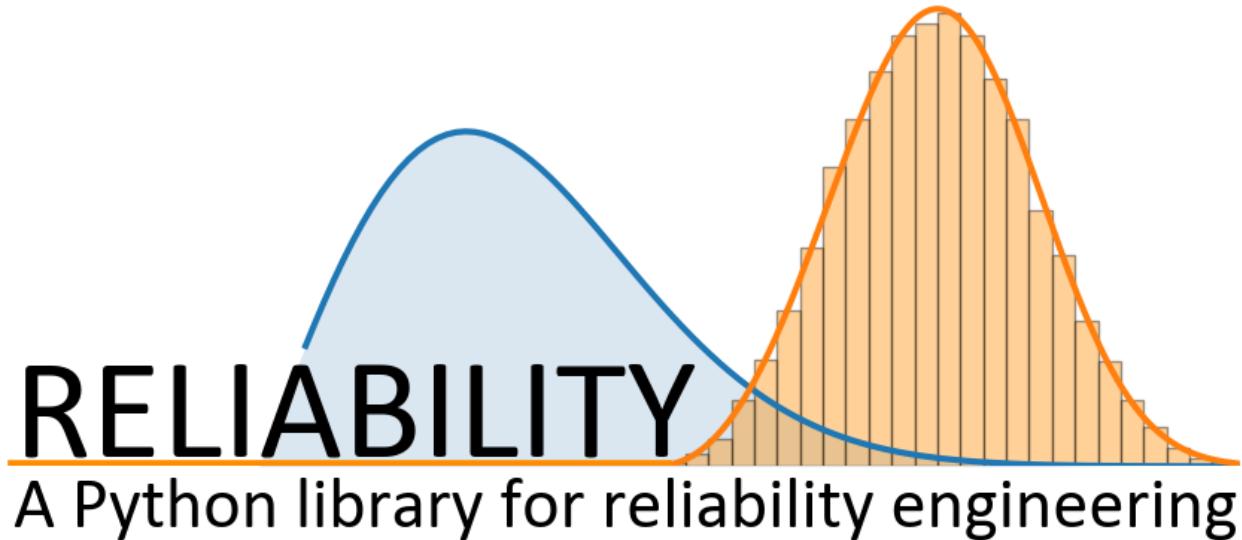
This is used to determine the sample size required for a test in which no failures are expected, and the desired outcome is the lower bound on the reliability based on the sample size and desired confidence interval.

Parameters

- **reliability** (*float*) – The lower bound on product reliability. Must be between 0 and 1.
- **CI** (*float, optional*) – The confidence interval of the result. Must be between 0.5 and 1 since a confidence less than 50% is not meaningful. Default = 0.95 for 95% CI.
- **lifetimes** (*int, float, optional*) – If testing the product for multiple lifetimes then more failures are expected so a smaller sample size will be required to demonstrate the desired reliability (assuming no failures). Conversely, if testing for less than one full lifetime then a larger sample size will be required. Default = 1. Must be greater than 0. No more than 5 is recommended due to test feasibility.
- **weibull_shape** (*int, float, optional*) – If the weibull shape (beta) of the failure mode is known, specify it here. Otherwise leave the default of 1 for the exponential distribution.
- **print_results** (*bool, optional*) – If True the results will be printed to the console. Default = True.

Returns

- n** (*int*) – The number of items required in the test. This will always be an integer (rounded up).



71.10.8 sequential_sampling_chart

```
class reliability.Reliability_testing.sequential_sampling_chart(p1, p2, alpha, beta,
                                                               show_plot=True,
                                                               print_results=True,
                                                               test_results=None,
                                                               max_samples=100)
```

This function plots the accept/reject boundaries for a given set of quality and risk levels. If supplied, the test results are also plotted on the chart.

A sequential sampling chart provides decision boundaries so that a success/failure test may be stopped as soon as there have been enough successes or enough failures to exceed the decision boundary. The decision boundary is calculated based on four parameters; producer's quality, consumer's quality, producer's risk, and consumer's risk. Producer's risk is the chance that the consumer rejects a batch when they should have accepted it. Consumer's risk is the chance that the consumer accepts a batch when they should have rejected it. We can also consider the producer's and consumer's quality to be the desired reliability of the sample, and the producer's and consumer's risk to be 1-confidence interval that the sample test result matches the population test result.

Parameters

- **p1** (*float*) – The producer's quality. This is the acceptable failure rate for the producer. Must be between 0 and 1 but is usually very small, typically around 0.01.
- **p2** (*float*) – The consumer's quality. This is the acceptable failure rate for the consumer. Must be between 0 and 1 but is usually very small, typically around 0.1.
- **alpha** (*float*) – The producer's risk. The probability of accepting a batch when it should have been rejected. Producer's CI = 1-alpha. Must be between 0 and 1 but is usually very small, typically 0.05.
- **beta** (*float*) – The consumer's risk. The probability of the consumer rejecting a batch when it should have been accepted. Consumer's CI = 1-beta. Must be between 0 and 1 but is usually very small, typically 0.1.
- **test_results** (*array, list, optional*) – The binary test results. eg. [0,0,0,1] represents 3 successes and 1 failure. Default=None. Use 0 for success and 1 for failure as this test is counting the number of failures.

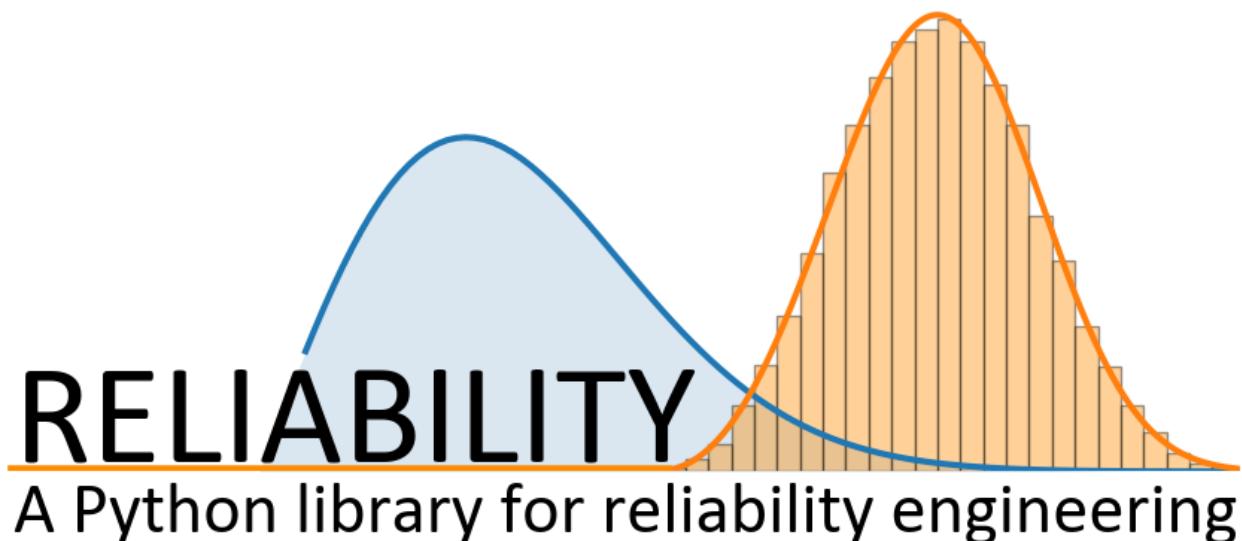
- **show_plot** (*bool, optional*) – If True the plot will be produced. Default = True.
- **print_results** (*bool, optional*) – If True the results will be printed to the console. Default = True.
- **max_samples** (*int, optional*) – The upper x-limit of the plot. Default = 100.

Returns

results (*dataframe*) – A dataframe of tabulated decision results with the columns “Samples”, “Failures to accept”, “Failures to reject”. This is independent of the test_results provided.

Notes

If show_plot is True, the sequential sampling chart with decision boundaries will be produced. The test_results are only plotted on the chart if provided as an input. The chart will display automatically so plt.show() is not required.



71.10.9 two_proportion_test

```
class reliability.Reliability_testing.two_proportion_test(sample_1_trials=None,
                                                       sample_1_successes=None,
                                                       sample_2_trials=None,
                                                       sample_2_successes=None, CI=0.95,
                                                       print_results=True)
```

Calculates whether the difference in test results between two samples is statistically significant.

For example, assume we have a poll of respondents in which 27/40 people agreed, and another poll in which 42/80 agreed. This test will determine if the difference is statistically significant for the given sample sizes at the specified confidence level.

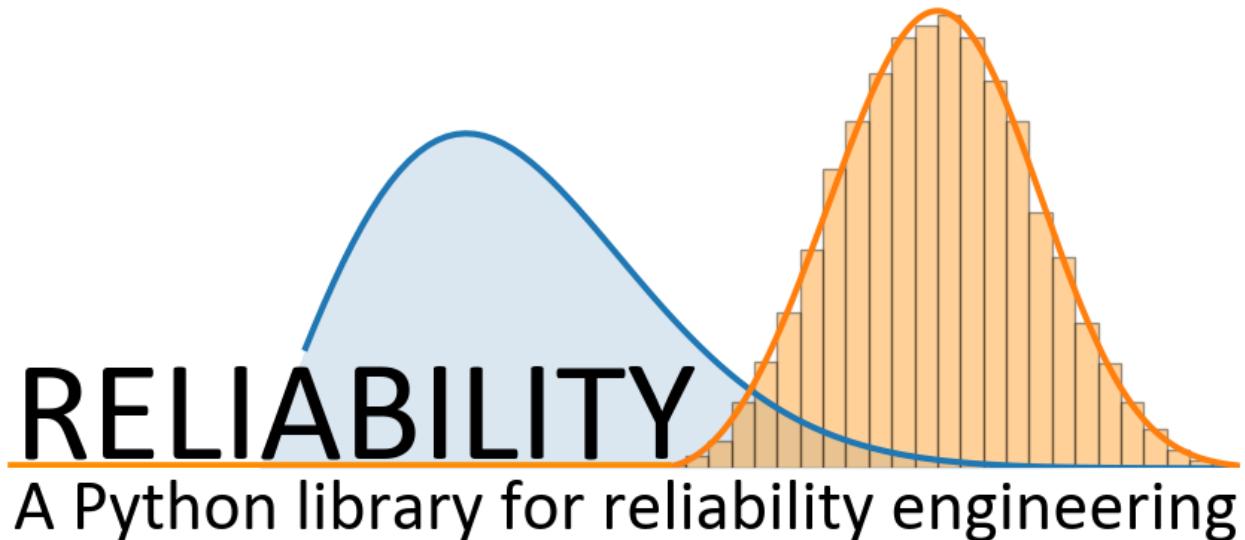
Parameters

- **sample_1_trials** (*int*) – The number of trials in the first sample.
- **sample_1_successes** (*int*) – The number of successes in the first sample.
- **sample_2_trials** (*int*) – The number of trials in the second sample.
- **sample_2_successes** (*int*) – The number of successes in the second sample.

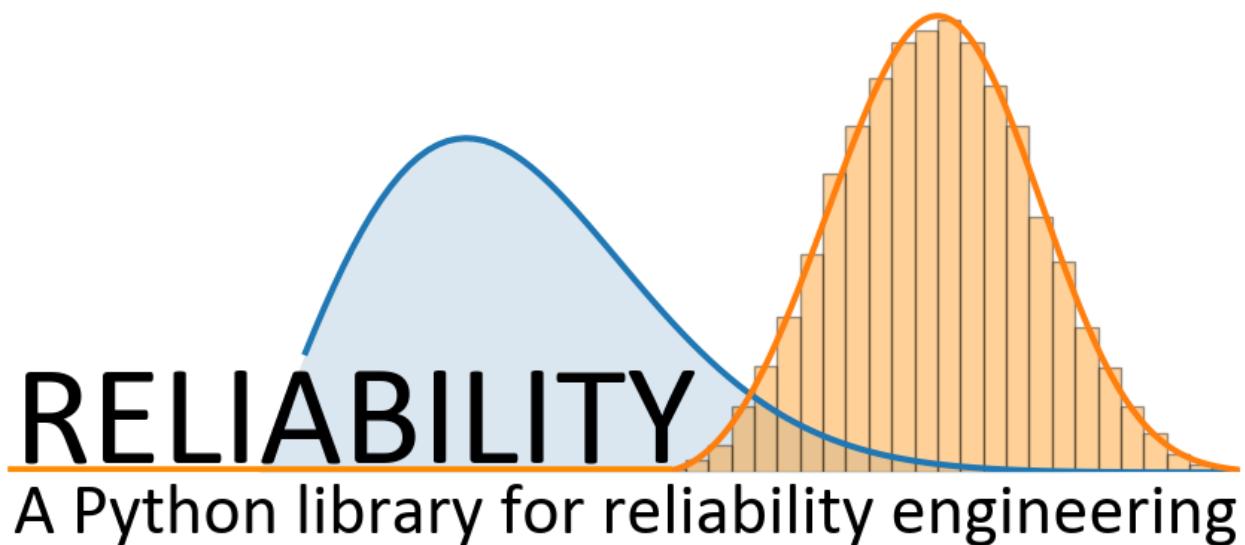
- **CI** (*float, optional*) – The desired confidence interval. Must be between 0 and 1. Default = 0.95 for 95% CI.
- **print_results** (*bool, optional*) – If True the results will be printed to the console. Default = True.

Returns

lower,upper,result (*tuple*) – The lower and upper are bounds on the difference. The result is either ‘significant’ or ‘non-significant’. If the bounds do not include 0 then it is a statistically significant difference.



71.11 Repairable_systems



71.11.1 MCF_nonparametric

```
class reliability.Repairable_systems.MCF_nonparametric(data, CI=0.95, print_results=True,
                                                       show_plot=True, plot_CI=True, **kwargs)
```

The Mean Cumulative Function (MCF) is a cumulative history function that shows the cumulative number of recurrences of an event, such as repairs over time. In the context of repairs over time, the value of the MCF can be thought of as the average number of repairs that each system will have undergone after a certain time. It is only applicable to repairable systems and assumes that each event (repair) is identical, but it does not assume that each system's MCF is identical (which is an assumption of the parametric MCF). The non-parametric estimate of the MCF provides both the estimate of the MCF and the confidence bounds at a particular time.

The shape of the MCF is a key indicator that shows whether the systems are improving, worsening, or staying the same over time. If the MCF is concave down (appearing to level out) then the system is improving. A straight line (constant increase) indicates it is staying the same. Concave up (getting steeper) shows the system is worsening as repairs are required more frequently as time progresses.

Parameters

- **data** (*list*) – The repair times for each system. Format this as a list of lists. eg. `data=[[4,7,9],[3,8,12]]` would be the data for 2 systems. The largest time for each system is assumed to be the retirement time and is treated as a right censored value. If the system was retired immediately after the last repair then you must include a repeated value at the end as this will be used to indicate a right censored value. eg. A system that had repairs at 4, 7, and 9 then was retired after the last repair would be entered as `data = [4,7,9,9]` since the last value is treated as a right censored value. If you only have data from 1 system you may enter the data in a single list as `data = [3,7,12]` and it will be nested within another list automatically.
- **print_results** (*bool, optional*) – Prints the table of MCF results (state, time, MCF_lower, MCF, MCF_upper, variance). Default = True.
- **CI** (*float, optional*) – Confidence interval. Must be between 0 and 1. Default = 0.95 for 95% CI (one sided).
- **show_plot** (*bool, optional*) – If True the plot will be shown. Default = True. Use `plt.show()` to show it.
- **plot_CI** (*bool, optional*) – If True, the plot will include the confidence intervals. Default = True. Set as False to remove the confidence intervals from the plot.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

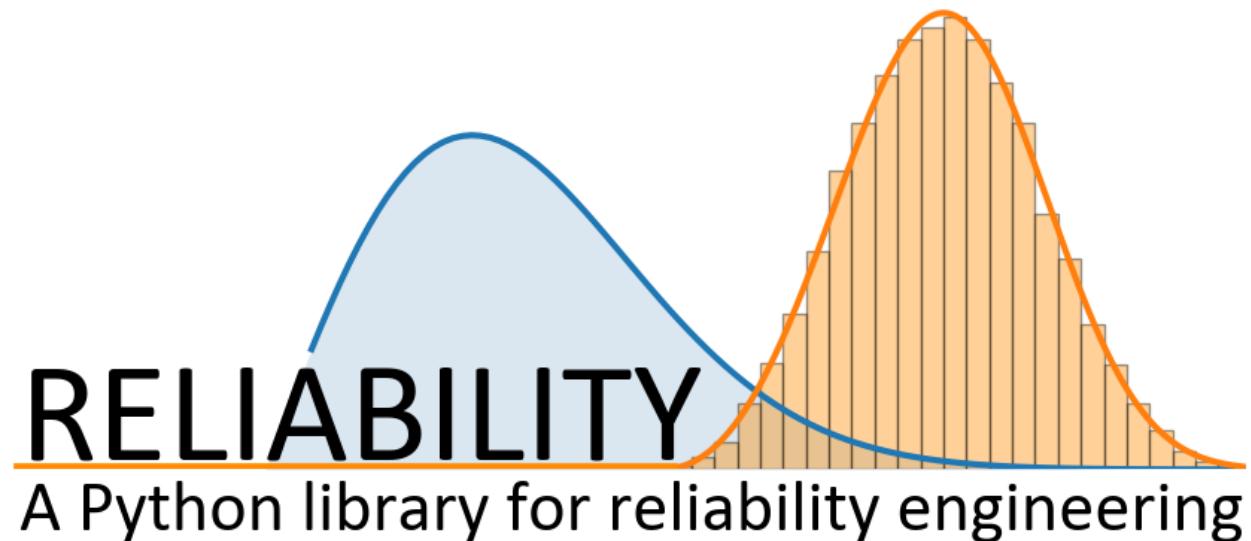
- **results** (*dataframe*) – This is a dataframe of the results that are printed. It includes the blank lines for censored values.
- **time** (*array*) – This is the time column from results. Blank lines for censored values are removed.
- **MCF** (*array*) – This is the MCF column from results. Blank lines for censored values are removed.
- **variance** (*array*) – This is the Variance column from results. Blank lines for censored values are removed.
- **lower** (*array*) – This is the MCF_lower column from results. Blank lines for censored values are removed.
- **upper** (*array*) – This is the MCF_upper column from results. Blank lines for censored values are removed

Notes

This example is taken from Reliasoft's example (available at http://reliawiki.org/index.php/Recurrent_Event_Data_Analysis). The failure times and retirement times (retirement time is indicated by +) of 5 systems are:

System	Times
1	5,10,15,17+
2	6,13,17,19+
3	12,20,25,26+
4	13,15,24+
5	16,22,25,28+

```
from reliability.Repairable_systems import MCF_nonparametric
times = [[5, 10, 15, 17], [6, 13, 17, 19], [12, 20, 25, 26], [13, 15, 24], [16, 22, 25, 28]]
MCF_nonparametric(data=times)
```



71.11.2 MCF_parametric

```
class reliability.Repairable_systems.MCF_parametric(data, CI=0.95, plot_CI=True,
                                                    print_results=True, show_plot=True, **kwargs)
```

The Mean Cumulative Function (MCF) is a cumulative history function that shows the cumulative number of recurrences of an event, such as repairs over time. In the context of repairs over time, the value of the MCF can be thought of as the average number of repairs that each system will have undergone after a certain time. It is only applicable to repairable systems and assumes that each event (repair) is identical. In the case of the fitted parametric MCF, it is assumed that each system's MCF is identical.

The shape (beta parameter) of the MCF is a key indicator that shows whether the systems are improving ($\beta < 1$), worsening ($\beta > 1$), or staying the same ($\beta = 1$) over time. If the MCF is concave down (appearing to level out) then the system is improving. A straight line (constant increase) indicates it is staying the same. Concave up (getting steeper) shows the system is worsening as repairs are required more frequently as time progresses.

Parameters

- **data** (*list*) – The repair times for each system. Format this as a list of lists. eg. `data=[[4,7,9],[3,8,12]]` would be the data for 2 systems. The largest time for each system is assumed to be the retirement time and is treated as a right censored value. If the system was retired immediately after the last repair then you must include a repeated value at the end as this will be used to indicate a right censored value. eg. A system that had repairs at 4, 7, and 9 then was retired after the last repair would be entered as `data = [4,7,9,9]` since the last value is treated as a right censored value. If you only have data from 1 system you may enter the data in a single list as `data = [3,7,12]` and it will be nested within another list automatically.
- **print_results** (*bool, optional*) – Prints the table of MCF results (state, time, MCF_lower, MCF, MCF_upper, variance). Default = True.
- **CI** (*float, optional*) – Confidence interval. Must be between 0 and 1. Default = 0.95 for 95% CI (one sided).
- **show_plot** (*bool, optional*) – If True the plot will be shown. Default = True. Use `plt.show()` to show it.
- **plot_CI** (*bool, optional*) – If True, the plot will include the confidence intervals. Default = True. Set as False to remove the confidence intervals from the plot.
- **kwarg**s – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

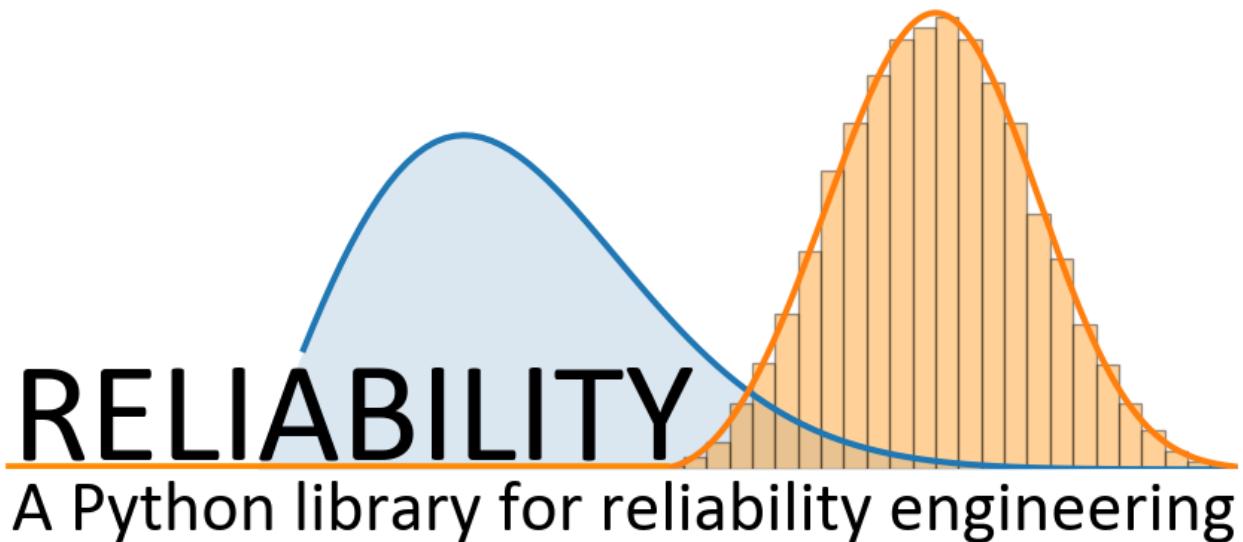
- **times** (*array*) – This is the times (x values) from the scatter plot. This value is calculated using `MCF_nonparametric`.
- **MCF** (*array*) – This is the MCF (y values) from the scatter plot. This value is calculated using `MCF_nonparametric`.
- **alpha** (*float*) – The calculated alpha parameter from $MCF = (t/\alpha)^{\beta}$
- **beta** (*float*) – The calculated beta parameter from $MCF = (t/\alpha)^{\beta}$
- **alpha_SE** (*float*) – The standard error in the alpha parameter
- **beta_SE** (*float*) – The standard error in the beta parameter
- **cov_alpha_beta** (*float*) – The covariance between the parameters
- **alpha_upper** (*float*) – The upper CI estimate of the parameter
- **alpha_lower** (*float*) – The lower CI estimate of the parameter
- **beta_upper** (*float*) – The upper CI estimate of the parameter
- **beta_lower** (*float*) – The lower CI estimate of the parameter
- **results** (*dataframe*) – A dataframe of the results (point estimate, standard error, Lower CI and Upper CI for each parameter)

Notes

This example is taken from Reliasoft's example (available at http://reliawiki.org/index.php/Recurrent_Event_Data_Analysis). The failure times and retirement times (retirement time is indicated by +) of 5 systems are:

System	Times
1	5,10,15,17+
2	6,13,17,19+
3	12,20,25,26+
4	13,15,24+
5	16,22,25,28+

```
from reliability.Repairable_systems import MCF_parametric
times = [[5, 10, 15, 17], [6, 13, 17, 19], [12, 20, 25, 26], [13, 15, 24], [16, 22, 25, 28]]
MCF_parametric(data=times)
```



71.11.3 ROCOF

```
class reliability.Repairable_systems.ROCOF(times_between_failures=None, failure_times=None, CI=0.95, test_end=None, show_plot=True, print_results=True, **kwargs)
```

Uses the failure times or failure interarrival times to determine if there is a trend in those times. The test for statistical significance is the Laplace test which compares the Laplace test statistic (U) with the z value (z_crit) from the standard normal distribution. If there is a statistically significant trend, the parameters of the model (Lambda_hat and Beta_hat) are calculated. By default the results are printed and a plot of the times and MTBF is plotted.

Parameters

- **times_between_failures** (*array, list, optional*) – The failure interarrival times. See the Notes below.
- **failure_times** (*array, list, optional*) – The actual failure times. See the Notes below.
- **test_end** (*int, float, optional*) – Use this to specify the end of the test if the test did not end at the time of the last failure. Default = None which will result in the last failure being treated as the end of the test.

- **CI** (*float*) – The confidence interval for the Laplace test. Must be between 0 and 1. Default is 0.95 for 95% CI.
- **show_plot** (*bool, optional*) – If True the plot will be produced. Default = True.
- **print_results** (*bool, optional*) – If True the results will be printed to console. Default = True.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

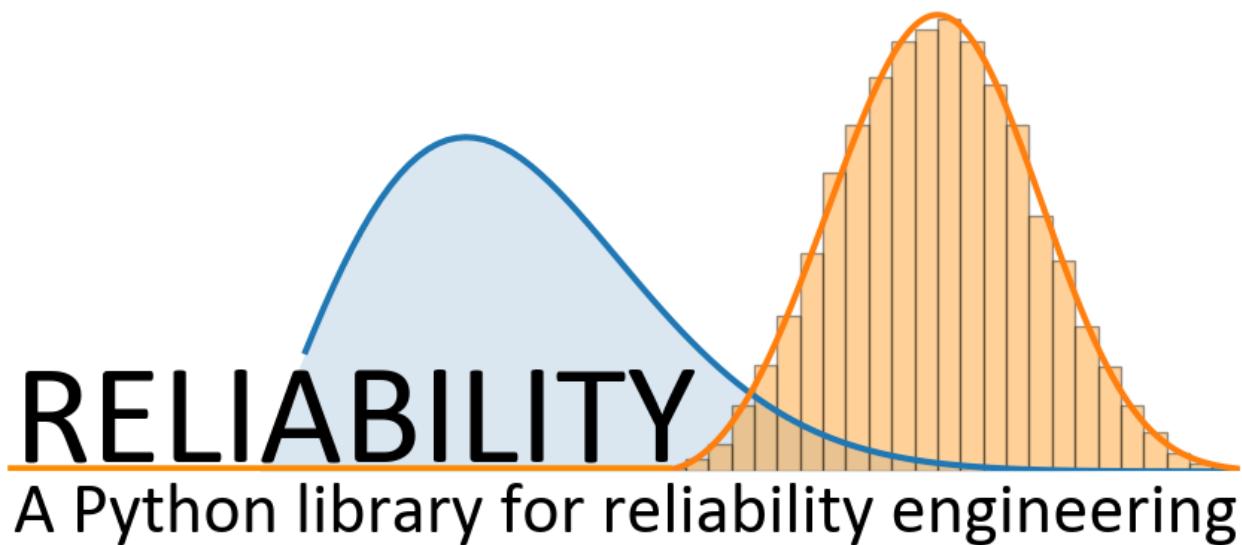
- **U** (*float*) – The Laplace test statistic
- **z_crit** (*tuple*) – (lower,upper) bound on z value. This is based on the CI.
- **trend** (*str*) – ‘improving’, ‘worsening’, ‘constant’. This is based on the comparison of U with z_crit
- **Beta_hat** (*float, str*) – The Beta parameter for the NHPP Power Law model. Only calculated if the trend is not constant, else a string is returned.
- **Lambda_hat** (*float, str*) – The Lambda parameter for the NHPP Power Law model. Only calculated if the trend is not constant.
- **ROCOF** (*float, str*) – The Rate of OCcurrence Of Failures. Only calculated if the trend is constant. If trend is not constant then ROCOF changes over time in accordance with Beta_hat and Lambda_hat. In this case a string will be returned.

Notes

You can specify either `times_between_failures` OR `failure_times` but not both. Both options are provided for convenience so the conversion between the two is done internally. `failure_times` should be the same as `np.cumsum(times_between_failures)`.

The repair time is assumed to be negligible. If the repair times are not negligibly small then you will need to manually adjust your input to factor in the repair times.

If `show_plot` is True, the ROCOF plot will be produced. Use `plt.show()` to show the plot.



71.11.4 optimal_replacement_time

```
class reliability.Repairable_systems.optimal_replacement_time(cost_PM, cost_CM, weibull_alpha,
                                                               weibull_beta,
                                                               show_time_plot=True,
                                                               show_ratio_plot=True,
                                                               print_results=True, q=0,
                                                               **kwargs)
```

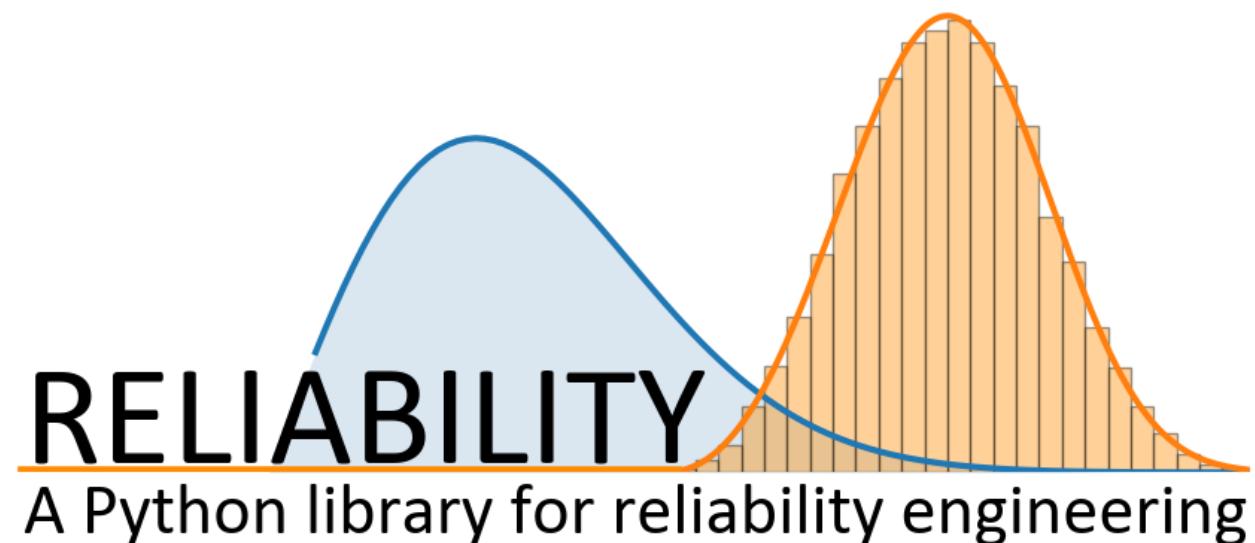
Calculates the cost model to determine how cost varies with replacement time. The cost model may be HPP (good as new replacement) or NHPP (as good as old replacement). Default is HPP.

Parameters

- **Cost_PM** (*int, float*) – The cost of preventative maintenance (must be smaller than Cost_CM)
- **Cost_CM** (*int, float*) – The cost of corrective maintenance (must be larger than Cost_PM)
- **weibull_alpha** (*int, float*) – The scale parameter of the underlying Weibull distribution.
- **weibull_beta** (*int, float*) – The shape parameter of the underlying Weibull distribution. Should be greater than 1 otherwise conducting PM is not economical.
- **q** (*int, optional*) – The restoration factor. Must be 0 or 1. Use q=1 for Power Law NHPP (as good as old) or q=0 for HPP (as good as new). Default is q=0 (as good as new).
- **show_time_plot** (*bool, axes, optional*) – If True the plot of replacement time vs cost per unit time will be produced in a new figure. If an axes subclass is passed then the plot be generated in that axes. If False then no plot will be generated. Default is True.
- **show_ratio_plot** (*bool, axes, optional*) – If True the plot of cost ratio vs replacement interval will be produced in a new figure. If an axes subclass is passed then the plot be generated in that axes. If False then no plot will be generated. Default is True.
- **print_results** (*bool, optional*) – If True the results will be printed to console. Default = True.
- **kwargs** – Plotting keywords that are passed directly to matplotlib (e.g. color, label, linestyle).

Returns

- **ORT** (*float*) – The optimal replacement time
- **min_cost** (*float*) – The minimum cost per unit time



71.11.5 reliability_growth

```
class reliability.Repairable_systems.reliability_growth(times=None, target_MTBF=None,
                                                       show_plot=True, print_results=True,
                                                       log_scale=False, model='Duane',
                                                       **kwargs)
```

Fits a reliability growth model to failure data using either the Duane model or the Crow-AMSAA model.

Parameters

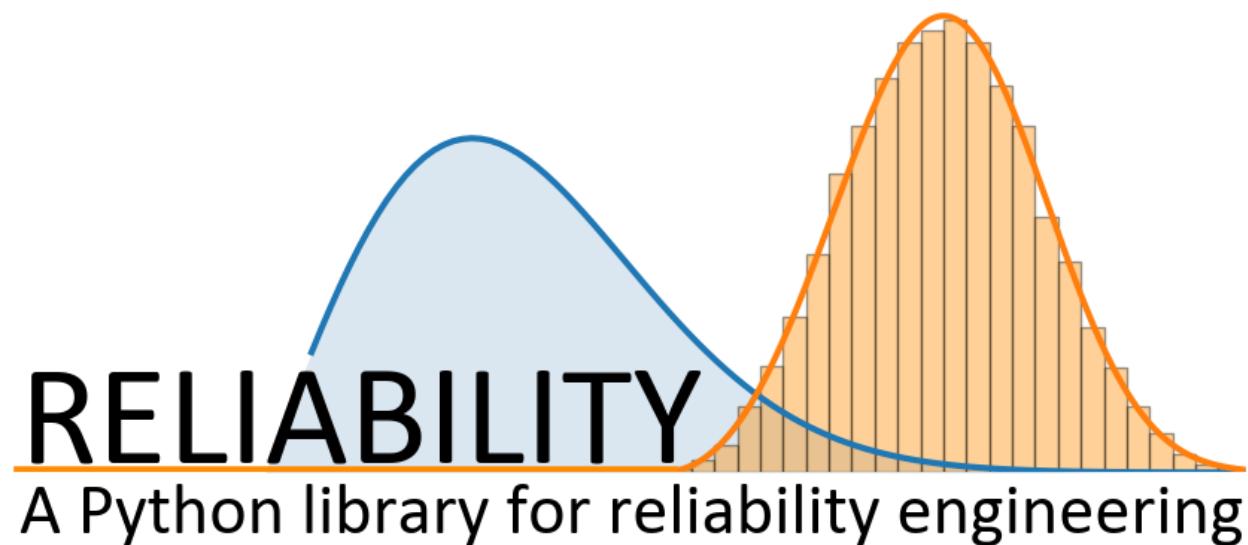
- **times** (*list, array*) – The failure times relative to an initial start time. These are actual failure times measured from the start of the test NOT failure interarrival times.
- **target_MTBF** (*float, int, optional*) – The target MTBF for the reliability growth curve. Default is None.
- **log_scale** (*bool, optional*) – Sets the x and y scales to log scales. Only used if show_plot is True.
- **show_plot** (*bool, optional*) – Default is True. If True the plot will be generated. Use plt.show() to show it.
- **model** (*str, optional*) – The model to use. Must be ‘Duane’ or ‘Crow-AMSAA’. Default is ‘Duane’.
- **print_results** (*bool, optional*) – Default is True. If True the results will be printed to the console.
- **kwargs** – Other keyword arguments passed to matplotlib.

Returns

- **Lambda** (*float*) – The Lambda parameter from the Crow-AMSAA model. Only returned if model=’Crow-AMSAA’.
- **Beta** (*float*) – The Beta parameter from the Crow-AMSAA model. Only returned if model=’Crow-AMSAA’.
- **growth_rate** (*float*) – The growth rate of the Crow-AMSAA model. Growth rate = 1 - Beta. Only returned if model=’Crow-AMSAA’.
- **A** (*float*) – The A parameter from the Duane model. Only returned if model=’Duane’.
- **Alpha** (*float*) – The Alpha parameter from the Duane model. Only returned if model=’Duane’.
- **DMTBF_C** (*float*) – The Demonstrated cumulative MTBF. The is the cumulative MTBF at the final failure time.
- **DMTBF_I** (*float*) – The Demonstrated instantaneous MTBF. The is the instantaneous MTBF at the final failure time.
- **DFI_C** (*float*) – The demonstrated cumulative failure intensity. This is 1/DMTBF_C.
- **DFI_I** (*float*) – The demonstrated instantaneous failure intensity. This is 1/DMTBF_I.
- **time_to_target** (*float, str*) – The time to reach target_MTBF. If target_MTBF is None then time_to_target will be a str asking for the target_MTBF to be specified. This uses the model for cumulative MTBF.

Notes

For more information see the documentation.



71.12 Utils

Utils (utilities)

This is a collection of utilities that are used throughout the python reliability library. Functions have been placed here as to declutter the dropdown lists of your IDE and to provide a common resource across multiple modules. It is not expected that users will be using any utils directly.

Included functions are:

- ALT_MLE_optimization - performs optimization for the ALT_Fitters
- ALT_fitters_input_checking - performs input checking for the ALT_Fitters
- ALT_least_squares - least squares estimation for ALT_Fitters
- ALT_prob_plot - probability plotting for ALT_Fitters
- LS_optimization - least squares optimization for Fitters
- MLE_optimization - maximum likelihood estimation optimization for Fitters
- anderson_darling - calculated the anderson darling (AD) goodness of fit statistic
- axes_transforms - Custom scale functions used in Probability_plotting
- clean_CI_arrays - cleans the CI arrays of nan and illegal values
- colorprint - prints to the console in color, bold, italic, and underline
- distribution_confidence_intervals - calculates and plots the confidence intervals for the distributions
- fill_no_autoscale - creates a shaded region without adding it to the global list of objects to consider when autoscale is calculated
- fitters_input_checking - error checking and default values for all the fitters
- generate_X_array - generates the X values for all distributions

- `get_axes_limits` - gets the current axes limits
- `least_squares` - provides parameter estimates for distributions using the method of least squares. Used extensively by Fitters.
- `life_stress_plot` - generates the life stress plot for ALT_Fitters
- `line_no_autoscale` - creates a line without adding it to the global list of objects to consider when autoscale is calculated
- `linear_regression` - given x and y data it will return slope and intercept of line of best fit. Includes options to specify slope or intercept.
- `make_fitted_dist_params_for_ALT_probplots` - creates a class structure for the ALT probability plots to give to Probability_plotting
- `no_reverse` - corrects for reversals in confidence intervals
- `probability_plot_xylims` - sets the x and y limits on probability plots
- `probability_plot_xyticks` - sets the x and y ticks on probability plots
- `removeNaNs` - removes nan
- `restore_axes_limits` - restores the axes limits based on values from `get_axes_limits()`
- `round_and_string` - applies different rounding rules and converts to string
- `show_figure_from_object` - Re-shows a figure from an axes or figure handle even after the figure has been closed.
- `transform_spaced` - Creates linearly spaced array (in transform space) based on a specified transform. This is like `np.logspace` but it can make an array that is weibull spaced, normal spaced, etc.
- `validate_CI_params` - checks that the confidence intervals have all the right parameters to be generated
- `write_df_to_xlsx` - converts a dataframe to an xlsx file
- `xy_transform` - provides conversions between spatial (-inf,inf) and axes coordinates (0,1).
- `zeroise_below_gamma` - sets all y values to zero when x < gamma. Used when the HF and CHF equations are specified

```
class reliability.Utils.**ALT_MLE_optimization**(**model, dist, LL_func, initial_guess, optimizer, failures, failure_stress_1, failure_stress_2=None, right_censored=None, right_censored_stress_1=None, right_censored_stress_2=None**)
```

This performs the MLE method to find the parameters. If the optimizer is `None` then all bounded optimizers will be tried and the best result (lowest log-likelihood) will be returned. If the optimizer is specified then it will be used. If it fails then the initial guess will be returned with a warning.

Parameters

- **`model` (str)** – Must be either “Exponential”, “Eyring”, “Power”, “Dual_Exponential”, “Power_Exponential”, or “Dual_Power”.
- **`dist` (str)** – Must be either “Weibull”, “Exponential”, “Lognormal”, or “Normal”.
- **`LL_func` (function)** – The log-likelihood function from the fitter
- **`initial_guess` (list, array)** – The initial guess of the model parameters that is used by the optimizer.
- **`optimizer` (str, None)** – This must be either “TNC”, “L-BFGS-B”, “nelder-mead”, “powell”, “best”, “all” or `None`. For detail on how these optimizers are used, please see the documentation.

- **failures** (*list, array*) – The failure data
- **right_censored** (*list, array*) – The right censored data. If there is no right censored data then this should be an empty array.
- **failure_stress_1** (*array, list*) – The failure stresses.
- **failure_stress_2** (*array, list*) – The failure second stresses. This is only used for dual stress models.
- **right_censored_stress_1** (*array, list*) – The right censored stresses. If there is no right censored data then this should be an empty array.
- **right_censored_stress_2** (*array, list*) – The right censored second stresses. If there is no right censored data then this should be an empty array. This is only used for dual stress models.

Returns

- **a** (*float*) – Only returned for Exponential, Eyring, Power, Dual_exponential, and Power_Exponential
- **b** (*float*) – Only returned for Exponential and Dual_Exponential
- **c** (*float*) – Only returned for Eyring, Dual_Exponential, Power_Exponential and Dual_Power
- **n** (*float*) – Only returned for Power, Power_Exponential, and Dual_Power
- **m** (*float*) – Only returned for Dual_Power
- **beta** (*float*) – Only returned for Weibull models
- **sigma** (*float*) – Only returned for Normal and Lognormal models
- **success** (*bool*) – Whether at least one optimizer succeeded. If False then the least squares result will be returned in place of the MLE result.
- **optimizer** (*str, None*) – The optimizer used. If MLE failed then None is returned as the optimizer.

Notes

Not all of the above returns are always returned. It depends on which model is being used.

If the MLE method fails then the initial guess (from least squares) will be returned with a printed warning.

```
class reliability.Utils. ALT_fitters_input_checking(dist, life_stress_model, failures, failure_stress_1, failure_stress_2=None, right_censored=None, right_censored_stress_1=None, right_censored_stress_2=None, CI=0.95, use_level_stress=None, optimizer=None)
```

This function performs error checking and some basic default operations for all the inputs given to each of the ALT_fitters.

Parameters

- **dist** (*str*) – Must be one of “Exponential”, “Weibull”, “Lognormal”, “Normal”, “Everything”.
- **life_stress_model** (*str*) – Must be one of “Exponential”, “Eyring”, “Power”, “Dual_Exponential”, “Power_Exponential”, “Dual_Power”, “Everything”.
- **failures** (*array, list*) – The failure data
- **failure_stress_1** (*array, list*) – The stresses corresponding to the failure data

- **failure_stress_2** (*array, list, optional*) – The second stresses corresponding to the failure data. Only required for dual stress models. Default is None.
- **right_censored** (*array, list, optional*) – The right censored data. Default is None.
- **right_censored_stress_1** (*array, list, optional*) – The stresses corresponding to the right censored data. Default is None.
- **right_censored_stress_2** (*array, list, optional*) – The second stresses corresponding to the right censored data. Only required for dual stress models. Default is None.
- **CI** (*float, optional*) – The confidence interval (between 0 and 1). Default is 0.95 for 95% confidence interval (two sided).
- **optimizer** (*str, None*) – This will return “TNC”, “L-BFGS-B”, “nelder-mead”, “powell”, “best”, or None. Default is None.
- **use_level_stress** (*float, int, list, array, optional*) – The use level stress. Must be float or int for single stress models. Must be array or list [stress_1, stress_2] for dual stress models. Default is None.

Returns

- **failures** (*array*) – The failure times
- **failure_stress_1** (*array*) – The failure stresses
- **failure_stress_2** (*array*) – The second failure stresses. This will be an empty array if the input was None.
- **right_censored** (*array*) – The right censored times. This will be an empty array if the input was None.
- **right_censored_stress_1** (*array*) – The right censored failure stresses. This will be an empty array if the input was None.
- **right_censored_stress_2** (*array*) – The right censored second failure stresses. This will be an empty array if the input was None.
- **CI** (*float*) – The confidence interval (between 0 and 1)
- **optimizer** (*str, None*) – This will return “TNC”, “L-BFGS-B”, “nelder-mead”, “powell”, “best”, or None.
- **use_level_stress** (*float, array, None*) – The use level stress. This will be a float for single stress models, or an array for dual stress models. This will be None if the input was None.
- **failure_groups** (*array*) – An array of arrays. This is the failure data grouped by failure stresses.
- **right_censored_groups** (*array*) – An array of arrays. This is the right censored data grouped by right censored stresses.
- **stresses_for_groups** (*array*) – An array of arrays. These are the stresses for each of the groups.

Notes

For full detail on what is checked and the errors produced, you should read the source code.

Some returns are None if the input is None. How None affects the behavior is governed by other functions such as the individual ALT fitters and other Utils.

`reliability.Utils.ALT_least_squares(model, failures, stress_1_array, stress_2_array=None)`

Uses least squares estimation to fit the parameters of the ALT stress-life model to the time to failure data.

Unlike least_squares for probability distributions, this function does not use the plotting positions because it is working on the life-stress model $L(S)$ and not the life distribution $F(t)$, so it uses the failure data directly.

This function therefore only fits the parameters of the model to the failure data and does not take into account the right censored data. Right censored data is only used when fitting the life-stress distribution (eg. “Weibull Eyring”) which is done using MLE.

The output of this function is used as the initial guess for the MLE method for the life-stress distribution.

Parameters

- **model** (*str*) – Must be either “Exponential”, “Eyring”, “Power”, “Dual_Exponential”, “Power_Exponential”, or “Dual_Power”
- **failures** (*array, list*) – The failure data
- **stress_1_array** (*list, array*) – The stresses corresponding to the failure data.
- **stress_2_array** (*list, array, optional*) – The second stresses corresponding to the failure data. Used only for dual-stress models. Default is None.

Returns

model_parameters (*list*) – The model’s parameters in a list. This depends on the model. Exponential - [a,b], Eyring - [a,c], Power - [a,n], Dual_Exponential - [a,b,c], Power_Exponential - [a,c,n], Dual_Power - [c,m,n]

Notes

For more information on least squares estimation, see the [documentation](#). For more information on fitting ALT models, see the [documentation](#).

For models with more than two parameters, linear algebra is equally valid, but in these cases it is not finding the line of best fit, it is instead finding the plane of best fit.

`reliability.Utils.ALT_prob_plot(dist, model, stresses_for_groups, failure_groups, right_censored_groups, life_func, shape, scale_for_change_df, shape_for_change_df, use_level_stress=None, ax=True)`

Generates an ALT probability plot using the inputs provided.

Parameters

- **dist** (*str*) – Must be either “Weibull”, “Exponential”, “Lognormal”, or “Normal”
- **model** (*str*) – Must be either “Exponential”, “Eyring”, “Power”, “Dual_Exponential”, “Power_Exponential”, or “Dual_Power”.
- **stresses_for_groups** (*list*) – The stresses for the failure groups
- **failure_groups** (*list*) – The failure groups. This is a list of lists.
- **right_censored_groups** – The failure groups. This is a list of lists.
- **life_func** (*function*) – The life function for the ALT life model.
- **shape** (*float, int*) – The shape parameter of the model.
- **scale_for_change_df** (*array, list*) – The list of scale parameters for the lines.
- **shape_for_change_df** – The list of shape parameters for the lines.
- **use_level_stress** (*float, int, array, list, None*) – The use level stress. This must be an array or list for dual stress models. Default is None.

- **ax** (*axis, bool, optional*) – The axis handle to use. Default is True which will create a new plot. If False then no plot will be generated.

Returns

current_axis (*axis*) – The axis handle of the plot. If ax is specified in the inputs then this will be the same handle.

```
class reliability.Utils.LS_optimization(func_name, LL_func, failures, right_censored, method='LS',
                                         force_shape=None, LL_func_force=None)
```

This function is a control function for least squares regression and it is used by each of the Fitters. There is no actual “optimization” done here, with the exception of checking which method (RRX or RRY) gives the better solution.

Parameters

- **func_name** (*str*) – The function to be fitted. Eg. “Weibull_2P”.
- **LL_func** (*function*) – The log-likelihood function from the fitter
- **failures** (*list, array*) – The failure data
- **right_censored** (*list, array*) – The right censored data. If there is no right censored data then this should be an empty array.
- **method** (*str, optional*) – Must be either “RRX”, “RRY”, “LS”, or “NLLS”. Default is “LS”.
- **force_shape** (*float, int, optional*) – The shape parameter to be forced. Default is None which results in no forcing of the shape parameter.
- **LL_func_force** (*function*) – The log-likelihood function for when the shape parameter is forced. Only required if force_shape is not None.

Returns

- **guess** (*list*) – The guess of the models parameters. The length of this list depends on the number of parameters in the model. The guess is obtained using `Utils.least_squares`
- **method** (*str*) – The method used. This will be either “RRX”, “RRY” or “NLLS”.

Notes

If method=“LS” then both “RRX” and “RRY” will be tried and the best one will be returned.

```
class reliability.Utils.MLE_optimization(func_name, LL_func, initial_guess, failures, right_censored,
                                         optimizer, force_shape=None, LL_func_force=None)
```

This function performs Maximum Likelihood Estimation (MLE) to find the optimal parameters of the probability distribution. This functions is used by each of the fitters.

Parameters

- **func_name** (*str*) – The function to be fitted. Eg. “Weibull_2P”.
- **LL_func** (*function*) – The log-likelihood function from the fitter
- **initial_guess** (*list, array*) – The initial guess of the model parameters that is used by the optimizer.
- **failures** (*list, array*) – The failure data
- **right_censored** (*list, array*) – The right censored data. If there is no right censored data then this should be an empty array.

- **optimizer** (*str; None*) – This must be either “TNC”, “L-BFGS-B”, “nelder-mead”, “powell”, “best”, “all” or None. For detail on how these optimizers are used, please see the [documentation](#).
- **force_shape** (*float, int, optional*) – The shape parameter to be forced. Default is None which results in no forcing of the shape parameter.
- **LL_func_force** (*function*) – The log-likelihood function for when the shape parameter is forced. Only required if force_shape is not None.

Returns

- **scale** (*float*) – Only returned for Weibull_2P, Weibull_3P, Lognormal_2P, Lognormal_3P, Gamma_2P, Gamma_3P, Loglogistic_2P, Loglogistic_3P, Exponential_1P, Exponential_2P, Normal_2P, Beta_2P, and Gumbel_2P
- **shape** (*float*) – Only returned for Weibull_2P, Weibull_3P, Lognormal_2P, Lognormal_3P, Gamma_2P, Gamma_3P, Loglogistic_2P, Loglogistic_3P, Normal_2P, Beta_2P, and Gumbel_2P
- **alpha** (*float*) – Only returned for Weibull_DS, Weibull_ZI and Weibull_DSZI
- **beta** (*float*) – Only returned for Weibull_DS, Weibull_ZI and Weibull_DSZI
- **gamma** (*float*) – Only returned for Weibull_3P, Exponential_2P, Gamma_3P, Lognormal_3P, and Loglogistic_3P.
- **DS** (*float*) – Only returned for Weibull_DS and Weibull_DSZI
- **ZI** (*float*) – Only returned for Weibull_ZI and Weibull_DSZI
- **alpha_1** (*float*) – Only returned for Weibull_mixture and Weibull_CR
- **beta_1** (*float*) – Only returned for Weibull_mixture and Weibull_CR
- **alpha_2** (*float*) – Only returned for Weibull_mixture and Weibull_CR
- **beta_2** (*float*) – Only returned for Weibull_mixture and Weibull_CR
- **proportion_1** (*float*) – Only returned for Weibull_mixture
- **proportion_2** (*float*) – Only returned for Weibull_mixture
- **success** (*bool*) – Whether at least one optimizer succeeded. If False then the least squares result will be returned in place of the MLE result.
- **optimizer** (*str; None*) – The optimizer used. If MLE failed then None is returned as the optimizer.

Notes

Not all of the above returns are always returned. It depends on which model is being used.

If the MLE method fails then the initial guess (from least squares) will be returned with a printed warning.

`reliability.Utils.anderson_darling(fitted_cdf, empirical_cdf)`

Calculates the Anderson-Darling goodness of fit statistic. These formulas are based on the method used in MINITAB which gives an adjusted form of the original AD statistic described on Wikipedia.

Parameters

- **fitted_cdf** (*list, array*) – The fitted CDF values at the data points
- **empirical_cdf** (*list, array*) – The empirical (rank adjustment) CDF values at the data points

Returns

AD (*float*) – The anderson darling (adjusted) test statistic.

class reliability.Utils.axes_transforms

Custom scale functions used in Probability_plotting Each of these functions is either a forward or inverse transform.

There are no parameters for this class, only a collection of subfunctions which can be called individually to perform the transforms.

```
static beta_forward(F, alpha, beta)
static beta_inverse(R, alpha, beta)
static exponential_forward(F)
static exponential_inverse(R)
static gamma_forward(F, beta)
static gamma_inverse(R, beta)
static gumbel_forward(F)
static gumbel_inverse(R)
static loglogistic_forward(F)
static loglogistic_inverse(R)
static normal_forward(F)
static normal_inverse(R)
static weibull_forward(F)
static weibull_inverse(R)
```

reliability.Utils.clean_CI_arrays(*xlower*, *xupper*, *ylower*, *yupper*, *plot_type*='CDF', *x*=None, *q*=None)

This function cleans the CI arrays of nans and numbers ≤ 0 and also removes numbers ≥ 1 if *plot_type* is CDF or SF.

Parameters

- **xlower** (*list, array*) – The lower x array for the confidence interval
- **xupper** (*list, array*) – The upper x array for the confidence interval
- **ylower** (*list, array*) – The lower y array for the confidence interval
- **yupper** (*list, array*) – The upper y array for the confidence interval
- **plot_type** (*str, optional*) – Must be “CDF”, “SF”, “CHF”. Default is “CDF”
- **x** (*array, optional*) – The x array for CI extraction
- **q** (*array, optional*) – The q array for CI extraction

Returns

- **xlower** (*array*) – The “cleaned” lower x array for the confidence interval
- **xupper** (*array*) – The “cleaned” upper x array for the confidence interval
- **ylower** (*array*) – The “cleaned” lower y array for the confidence interval

- **ylower** (*array*) – The “cleaned” upper y array for the confidence interval

Notes

The returned arrays will all be the same length

The cleaning is done by deleting values. If the cleaned arrays are < 2 items in length then an error will be triggered.

```
reliability.Utils.colorprint(string, text_color=None, background_color=None, bold=False,  
                             underline=False, italic=False)
```

Provides easy access to color printing in the console.

This function is used to print warnings in red text, but it can also do a lot more.

Parameters

- **string**
- **text_color** (*str, None, optional*) – Must be either grey, red, green, yellow, blue, pink, or turquoise. Use None to leave the color as white. Default is None.
- **background_color** (*str, None, optional*) – Must be either grey, red, green, yellow, blue, pink, or turquoise. Use None to leave the color as the transparent. Default is None.
- **bold** (*bool, optional*) – Default is False.
- **underline** (*bool, optional*) – Default is False.
- **italic** (*bool, optional*) – Default is False.

Returns

None – The output is printed to the console.

Notes

Some flexibility in color names is allowed. eg. red and r will both give red.

As there is only one string argument, if you have multiple strings to print, you must first combine them using str(string_1, string_2, ...).

```
class reliability.Utils.distribution_confidence_intervals
```

This class contains several subfunctions that provide all the confidence intervals for CDF, SF, CHF for each distribution for which it is implemented.

The class has no parameters or returns as it is used primarily to create the confidence interval object which is used by the subfunctions.

Parameters

None

Returns

None

```
static exponential_CI(self, func='CDF', plot_CI=None, CI=None, text_title='', color=None, q=None,  
                      x=None)
```

Generates the confidence intervals for CDF, SF, and CHF of the Exponential distribution.

Parameters

- **self** (*object*) – The distribution object
- **func** (*str*) – Must be either “CDF”, “SF” or “CHF”. Default is “CDF”

- **plot_CI** (*bool, None*) – The confidence intervals will only be plotted if plot_CI is True.
- **CI** (*float*) – The confidence interval. Must be between 0 and 1
- **text_title** (*str*) – The existing CDF/SF/CHF text title to which the confidence interval string will be added.
- **color** (*str*) – The color to be used to fill the confidence intervals.
- **q** (*array, list, optional*) – The quantiles to be calculated. Default is None.
- **x** (*array, list, optional*) – The x-values to be calculated. Default is None.

Returns

- **t_lower** (*array*) – The lower bounds on time. Only returned if q is not None.
- **t_upper** (*array*) – The upper bounds on time. Only returned if q is not None.
- **R_lower** (*array*) – The lower bounds on reliability. Only returned if x is not None.
- **R_upper** (*array*) – The upper bounds on reliability. Only returned if x is not None.

Notes

self must contain particular values for this function to work. These include self.Lambda_SE and self.Z.

As a Utils function, there is very limited error checking done, as this function is not intended for users to access directly.

For the Exponential distribution, the bounds on time and reliability are the same.

For an explanation of how the confidence intervals are calculated, please see the [documentation](#).

```
static gamma_CI(self, func='CDF', plot_CI=None, CI_type=None, CI=None, text_title='', color=None, q=None, x=None)
```

Generates the confidence intervals for CDF, SF, and CHF of the Gamma distribution.

Parameters

- **self** (*object*) – The distribution object
- **func** (*str*) – Must be either “CDF”, “SF” or “CHF”. Default is “CDF”.
- **plot_CI** (*bool, None*) – The confidence intervals will only be plotted if plot_CI is True.
- **CI_type** (*str*) – Must be either “time” or “reliability”
- **CI** (*float*) – The confidence interval. Must be between 0 and 1
- **text_title** (*str*) – The existing CDF/SF/CHF text title to which the confidence interval string will be added.
- **color** (*str*) – The color to be used to fill the confidence intervals.
- **q** (*array, list, optional*) – The quantiles to be calculated. Default is None. Only used if CI_type='time'.
- **x** (*array, list, optional*) – The x-values to be calculated. Default is None. Only used if CI_type='reliability'.

Returns

- **t_lower** (*array*) – The lower bounds on time. Only returned if CI_type is “time” and q is not None.

- **t_upper** (array) – The upper bounds on time. Only returned if CI_type is “time” and q is not None.
- **R_lower** (array) – The lower bounds on reliability. Only returned if CI_type is “reliability” and x is not None.
- **R_upper** (array) – The upper bounds on reliability. Only returned if CI_type is “reliability” and x is not None.

Notes

self must contain particular values for this function to work. These include self.mu_SE, self.beta_SE, self.Cov_mu_beta, self.Z.

As a Utils function, there is very limited error checking done, as this function is not intended for users to access directly.

For an explanation of how the confidence intervals are calculated, please see the [documentation](#).

```
static gumbel_CI(self, func='CDF', plot_CI=None, CI_type=None, CI=None, text_title='', color=None, q=None, x=None)
```

Generates the confidence intervals for CDF, SF, and CHF of the Gumbel distribution.

Parameters

- **self** (object) – The distribution object
- **func** (str) – Must be either “CDF”, “SF” or “CHF”. Default is “CDF”.
- **plot_CI** (bool, None) – The confidence intervals will only be plotted if plot_CI is True.
- **CI_type** (str) – Must be either “time” or “reliability”
- **CI** (float) – The confidence interval. Must be between 0 and 1
- **text_title** (str) – The existing CDF/SF/CHF text title to which the confidence interval string will be added.
- **color** (str) – The color to be used to fill the confidence intervals.
- **q** (array, list, optional) – The quantiles to be calculated. Default is None. Only used if CI_type='time'.
- **x** (array, list, optional) – The x-values to be calculated. Default is None. Only used if CI_type='reliability'.

Returns

- **t_lower** (array) – The lower bounds on time. Only returned if CI_type is “time” and q is not None.
- **t_upper** (array) – The upper bounds on time. Only returned if CI_type is “time” and q is not None.
- **R_lower** (array) – The lower bounds on reliability. Only returned if CI_type is “reliability” and x is not None.
- **R_upper** (array) – The upper bounds on reliability. Only returned if CI_type is “reliability” and x is not None.

Notes

self must contain particular values for this function to work. These include self.mu_SE, self.sigma_SE, self.Cov_mu_sigma, self.Z.

As a Utils function, there is very limited error checking done, as this function is not intended for users to access directly.

For an explanation of how the confidence intervals are calculated, please see the [documentation](#).

```
static loglogistic_CI(self, func='CDF', plot_CI=None, CI_type=None, CI=None, text_title="",
                      color=None, q=None, x=None)
```

Generates the confidence intervals for CDF, SF, and CHF of the Loglogistic distribution.

Parameters

- **self** (*object*) – The distribution object
- **func** (*str*) – Must be either “CDF”, “SF” or “CHF”. Default is “CDF”.
- **plot_CI** (*bool, None*) – The confidence intervals will only be plotted if plot_CI is True.
- **CI_type** (*str*) – Must be either “time” or “reliability”
- **CI** (*float*) – The confidence interval. Must be between 0 and 1
- **text_title** (*str*) – The existing CDF/SF/CHF text title to which the confidence interval string will be added.
- **color** (*str*) – The color to be used to fill the confidence intervals.
- **q** (*array, list, optional*) – The quantiles to be calculated. Default is None. Only used if CI_type=’time’.
- **x** (*array, list, optional*) – The x-values to be calculated. Default is None. Only used if CI_type=’reliability’.

Returns

- **t_lower** (*array*) – The lower bounds on time. Only returned if CI_type is “time” and q is not None.
- **t_upper** (*array*) – The upper bounds on time. Only returned if CI_type is “time” and q is not None.
- **R_lower** (*array*) – The lower bounds on reliability. Only returned if CI_type is “reliability” and x is not None.
- **R_upper** (*array*) – The upper bounds on reliability. Only returned if CI_type is “reliability” and x is not None.

Notes

self must contain particular values for this function to work. These include self.alpha_SE, self.beta_SE, self.Cov_alpha_beta, self.Z.

As a Utils function, there is very limited error checking done, as this function is not intended for users to access directly.

For an explanation of how the confidence intervals are calculated, please see the [documentation](#).

```
static lognormal_CI(self, func='CDF', plot_CI=None, CI_type=None, CI=None, text_title="",
                      color=None, q=None, x=None)
```

Generates the confidence intervals for CDF, SF, and CHF of the Lognormal distribution.

Parameters

- **self** (*object*) – The distribution object
- **func** (*str*) – Must be either “CDF”, “SF” or “CHF”. Default is “CDF”.
- **plot_CI** (*bool, None*) – The confidence intervals will only be plotted if plot_CI is True.
- **CI_type** (*str*) – Must be either “time” or “reliability”
- **CI** (*float*) – The confidence interval. Must be between 0 and 1
- **text_title** (*str*) – The existing CDF/SF/CHF text title to which the confidence interval string will be added.
- **color** (*str*) – The color to be used to fill the confidence intervals.
- **q** (*array, list, optional*) – The quantiles to be calculated. Default is None. Only used if CI_type=’time’.
- **x** (*array, list, optional*) – The x-values to be calculated. Default is None. Only used if CI_type=’reliability’.

Returns

- **t_lower** (*array*) – The lower bounds on time. Only returned if CI_type is “time” and q is not None.
- **t_upper** (*array*) – The upper bounds on time. Only returned if CI_type is “time” and q is not None.
- **R_lower** (*array*) – The lower bounds on reliability. Only returned if CI_type is “reliability” and x is not None.
- **R_upper** (*array*) – The upper bounds on reliability. Only returned if CI_type is “reliability” and x is not None.

Notes

self must contain particular values for this function to work. These include self.mu_SE, self.sigma_SE, self.Cov_mu_sigma, self.Z.

As a Utils function, there is very limited error checking done, as this function is not intended for users to access directly.

For an explanation of how the confidence intervals are calculated, please see the [documentation](#).

```
static normal_CI(self, func='CDF', plot_CI=None, CI_type=None, CI=None, text_title='', color=None, q=None, x=None)
```

Generates the confidence intervals for CDF, SF, and CHF of the Normal distribution.

Parameters

- **self** (*object*) – The distribution object
- **func** (*str*) – Must be either “CDF”, “SF” or “CHF”. Default is “CDF”.
- **plot_CI** (*bool, None*) – The confidence intervals will only be plotted if plot_CI is True.
- **CI_type** (*str*) – Must be either “time” or “reliability”
- **CI** (*float*) – The confidence interval. Must be between 0 and 1
- **text_title** (*str*) – The existing CDF/SF/CHF text title to which the confidence interval string will be added.

- **color** (*str*) – The color to be used to fill the confidence intervals.
- **q** (*array, list, optional*) – The quantiles to be calculated. Default is None. Only used if *CI_type*='time'.
- **x** (*array, list, optional*) – The x-values to be calculated. Default is None. Only used if *CI_type*='reliability'.

Returns

- **t_lower** (*array*) – The lower bounds on time. Only returned if *CI_type* is “time” and *q* is not None.
- **t_upper** (*array*) – The upper bounds on time. Only returned if *CI_type* is “time” and *q* is not None.
- **R_lower** (*array*) – The lower bounds on reliability. Only returned if *CI_type* is “reliability” and *x* is not None.
- **R_upper** (*array*) – The upper bounds on reliability. Only returned if *CI_type* is “reliability” and *x* is not None.

Notes

self must contain particular values for this function to work. These include *self.mu_SE*, *self.sigma_SE*, *self.Cov_mu_sigma*, *self.Z*.

As a Utils function, there is very limited error checking done, as this function is not intended for users to access directly.

For an explanation of how the confidence intervals are calculated, please see the [documentation](#).

```
static weibull_CI(self, func='CDF', plot_CI=None, CI_type=None, CI=None, text_title='', color=None, q=None, x=None)
```

Generates the confidence intervals for CDF, SF, and CHF of the Weibull distribution.

Parameters

- **self** (*object*) – The distribution object
- **func** (*str*) – Must be either “CDF”, “SF” or “CHF”. Default is “CDF”
- **plot_CI** (*bool, None*) – The confidence intervals will only be plotted if *plot_CI* is True.
- **CI_type** (*str*) – Must be either “time” or “reliability”
- **CI** (*float*) – The confidence interval. Must be between 0 and 1
- **text_title** (*str*) – The existing CDF/SF/CHF text title to which the confidence interval string will be added.
- **color** (*str*) – The color to be used to fill the confidence intervals.
- **q** (*array, list, optional*) – The quantiles to be calculated. Default is None. Only used if *CI_type*='time'.
- **x** (*array, list, optional*) – The x-values to be calculated. Default is None. Only used if *CI_type*='reliability'.

Returns

- **t_lower** (*array*) – The lower bounds on time. Only returned if *CI_type* is “time” and *q* is not None.

- **t_upper** (array) – The upper bounds on time. Only returned if CI_type is “time” and q is not None.
- **R_lower** (array) – The lower bounds on reliability. Only returned if CI_type is “reliability” and x is not None.
- **R_upper** (array) – The upper bounds on reliability. Only returned if CI_type is “reliability” and x is not None.

Notes

self must contain particular values for this function to work. These include self.alpha_SE, self.beta_SE, self.Cov_alpha_beta, self.Z.

As a Utils function, there is very limited error checking done, as this function is not intended for users to access directly.

For an explanation of how the confidence intervals are calculated, please see the [documentation](#).

```
reliability.Utils.distributions_input_checking(self, func, xvals, xmin, xmax, show_plot=None,  
                                              plot_CI=None, CI_type=None, CI=None, CI_y=None,  
                                              CI_x=None)
```

Performs checks and sets default values for the inputs to distributions sub function (PDF, CDF, SF, HF, CHF)

Parameters

- **self** (object) – Distribution object created by reliability.Distributions
- **func** (str) – Must be either ‘PDF’, ‘CDF’, ‘SF’, ‘HF’, ‘CHF’
- **xvals** (array, list) – x-values for plotting.
- **xmin** (int, float) – minimum x-value for plotting.
- **xmax** (int, float) – maximum x-value for plotting.
- **show_plot** (bool) – Whether the plot is to be shown.
- **plot_CI** (bool, optional) – Whether the confidence intervals are to be shown on the plot. Default is None.
- **CI_type** (str, optional) – If specified, it must be “time” or “reliability”. Default is None
- **CI** (float, optional) – The confidence intervals. If specified, it must be between 0 and 1. Default is None.
- **CI_y** (list, array, optional) – The confidence interval y-values to trace. Default is None.
- **CI_x** (list, array, optional) – The confidence interval x-values to trace. Default is None.

Returns

- **X** (array) – An array of the x-values for the plot. Created using generate_X_array
- **xvals** (array, list) – x-values for plotting.
- **xmin** (int, float) – minimum x-value for plotting.
- **xmax** (int, float) – maximum x-value for plotting.
- **show_plot** (bool) – Whether the plot is to be shown. Default is True. Only returned if func is ‘PDF’, ‘CDF’, ‘SF’, ‘HF’, or ‘CHF’
- **plot_CI** (bool) – Whether the confidence intervals are to be shown on the plot. Default is True. Only returned if func is ‘CDF’, ‘SF’, or ‘CHF’ and self.name != ‘Beta’.

- **CI_type** (str) – The type of confidence interval. Will be either “time”, “reliability”, or “none”. Default is “time”. Only returned if func is ‘CDF’, ‘SF’, or ‘CHF’. If self.name ==’Beta’ or self.name==’Exponential’ it will return ‘none’. If self.CI_type is specified and CI_type is not specified then self.CI_type will be used for CI_type.
- **CI** (float) – The confidence intervals between 0 and 1. Default is 0.95. Only returned if func is ‘CDF’, ‘SF’, or ‘CHF’ and self.name !=’Beta’. If self.CI is specified and CI is None then self.CI will be used for CI.
- **CI_y** (list, array, float, int) – The confidence interval y-values to trace. Default is None. Only returned if func is ‘CDF’, ‘SF’, or ‘CHF’ and self.name !=’Beta’.
- **CI_x** (list, array, float, int) – The confidence interval x-values to trace. Default is None. Only returned if func is ‘CDF’, ‘SF’, or ‘CHF’ and self.name !=’Beta’.

`reliability.Utils.extract_CI(dist, func='CDF', CI_type='time', CI=0.95, CI_y=None, CI_x=None)`

Extracts the confidence bounds at CI_x or CI_y.

Parameters

- **dist** (object) – Distribution object from reliability.Distributions
- **func** (str) – Must be either ‘CDF’, ‘SF’, ‘CHF’
- **CI_type** (str) – Must be either ‘time’ or ‘reliability’
- **CI** (float) – The confidence interval. Must be between 0 and 1.
- **CI_y** (list, array) – The y-values from which to extract the confidence interval (x-values) for bounds on time.
- **CI_x** (list, array) – The x-values from which to extract the confidence interval (y-values) for bounds on reliability.

Returns

- **lower** (array) – An array of the lower confidence bounds at CI_x or CI_y
- **upper** (array) – An array of the upper confidence bounds at CI_x or CI_y

Notes

If CI_type=“time” then CI_y must be specified in order to extract the confidence bounds on time.

If CI_type=“reliability” then CI_x must be specified in order to extract the confidence bounds on reliability.

`reliability.Utils.fill_no_autoscale(xlower, xupper, ylower, yupper, **kwargs)`

Creates a filled region (polygon) without adding it to the global list of autoscale objects. Use this function when you want to plot something but not have it considered when autoscale sets the range.

Parameters

- **xlower** (list, array) – The lower x array for the polygon.
- **xupper** (list, array) – The upper x array for the polygon.
- **ylower** (list, array) – The lower y array for the polygon.
- **yupper** (list, array) – The upper y array for the polygon.
- **kwargs** – keyword arguments passed to the matplotlib PolyCollection

Returns

None – The filled polygon will be added to the plot.

```
class reliability.Utils.fitters_input_checking(dist,failures,right_censored=None,method=None,  
optimizer=None,CI=0.95,quantiles=False,  
force_beta=None,force_sigma=None,CI_type=None)
```

This function performs error checking and some basic default operations for all the inputs given to each of the fitters.

Parameters

- **dist** (*str*) – Must be one of “Everything”, “Weibull_2P”, “Weibull_3P”, “Gamma_2P”, “Gamma_3P”, “Exponential_1P”, “Exponential_2P”, “Gumbel_2P”, “Normal_2P”, “Lognormal_2P”, “Lognormal_3P”, “Loglogistic_2P”, “Loglogistic_3P”, “Beta_2P”, “Weibull_Mixture”, “Weibull_CR”, “Weibull_DSZI”, “Weibull_DS”, “Weibull_ZI”.
- **failures** (*array, list*) – The failure data
- **right_censored** (*array, list, optional*) – The right censored data
- **method** (*str, optional*) – Must be either “MLE”, “LS”, “RRX”, or “RRY”. Some flexibility in input is tolerated. eg “LS”, “LEAST SQUARES”, “LSQ”, “NLRR”, “NLLS” will all be recognised as “LS”. Default is MLE
- **optimizer** (*str, optional*) – Must be one of “TNC”, “L-BFGS-B”, “nelder-mead”, “powell”, “best”. Default is None which will result in each being tried until one succeeds. For more detail see the [documentation](#).
- **CI** (*float, optional*) – Confidence interval. Must be between 0 and 1. Default is 0.95 for 95% confidence interval (2 sided).
- **quantiles** (*array, list, bool, optional*) – An array or list of the quantiles to calculate. If True then the default array will be used. Default array is [0.01, 0.05, 0.1, 0.2, 0.25, 0.5, 0.75, 0.8, 0.9, 0.95, 0.99]. If False then no quantiles will be calculated. Default is False.
- **force_beta** (*float, int, optional*) – Used to force beta for the Weibull_2P distribution. Default is None which will not force beta.
- **force_sigma** (*float, int, optional*) – Used to force sigma for the Normal_2P and Lognormal_2P distributions. Default is None which will not force sigma.
- **CI_type** (*str, optional*) – Must be either “time” or “reliability”. Default is None which results in “time” being used (controlled in Fitters). Some flexibility in strings is allowed. eg. “r”, “R”, “rel”, “REL”, “reliability”, “RELIABILITY” will all be recognized as “reliability”.

Returns

- **failures** (*array*) – The failure times
- **right_censored** (*array*) – The right censored times. This will be an empty array if the input was None.
- **CI** (*float*) – The confidence interval (between 0 and 1)
- **method** (*str, None*) – This will return “MLE”, “LS”, “RRX”, “RRY” or None.
- **optimizer** (*str, None*) – This will return “TNC”, “L-BFGS-B”, “nelder-mead”, “powell”, “best”, or None.
- **quantiles** (*array, None*) – The quantiles or None.
- **force_beta** (*float, None*) – The beta parameter to be forced in Weibull_2P
- **force_sigma** (*float, None*) – The sigma parameter to be forced in Normal_2P, or Lognormal_2P
- **CI_type** (*str, None*) – “time”, “reliability”, ‘None’ or None

Notes

For full detail on what is checked and the errors produced, you should read the source code.

Some returns are None if the input is None. How None affects the behavior is governed by other functions such as the individual fitters and other Utils.

`reliability.Utils.generate_X_array(dist, xvals=None, xmin=None, xmax=None)`

Generates the array of X values for each of the PDF, CDF, SF, HF, CHF functions within `reliability.Distributions`

This is done with a variety of cases in order to ensure that for regions of high gradient (particularly asymptotes to inf) the points are more concentrated. This ensures that the line always looks as smooth as possible using only 200 data points.

Parameters

- **dist** (*object*) – The distribution object
- **xvals** (*array, list, optional*) – The xvals for the plot if specified
- **xmin** (*array, list, optional*) – The xmin for the plot if specified
- **xmax** (*array, list, optional*) – The xmax for the plot if specified

Returns

X (*array*) – The X array that was generated.

`reliability.Utils.get_axes_limits()`

This function works in a pair with `restore_axes_limits`. This function gets the previous `xlim` and `ylim` and also checks whether there was a previous plot (based on whether the default 0,1 axes had been changed).

It returns a list of items that are used by `restore_axes_limits` after the plot has been performed.

Parameters

None – The plot properties are extracted automatically for analysis

Returns

output (*list*) – A list of [`xlims`, `ylims`, `use_prev_lims`]. These values are used by `restore_axes_limits` to determine how the axes limits need to be changed after plotting.

`reliability.Utils.least_squares(dist, failures, right_censored, method='RRX', force_shape=None)`

Uses least squares or non-linear least squares estimation to fit the parameters of the distribution to the plotting positions.

Plotting positions are based on failures and `right_censored` so while least squares estimation does not consider the `right_censored` data in the same way as MLE, the plotting positions do. This means that right censored data are not ignored by least squares estimation, but the way the values are treated differs between least squares and MLE.

The output of this method may be used as the initial guess for the MLE method.

Parameters

- **dist** (*object*) – The distribution object
- **failures** (*array, list*) – The failure data
- **right_censored** (*array, list*) – The right censored data. If there is no data then this should be an empty list.
- **method** (*str, optional*) – Must be “RRX” or “RRY”. Default is RRX.
- **force_shape** (*float, int, optional*) – Used to force the shape (beta or sigma) parameter. Default is None which will not force the slope.

Returns

model_parameters (*list*) – The model's parameters in a list. eg. for "Weibull_2P" it will return [alpha,beta]. For "Weibull_3P" it will return [alpha,beta,gamma].

Notes

For more information on least squares estimation, see the [documentation](#).

For cases where the CDF is not linearizable (eg. Weibull_3P), this function uses non-linear least squares (NLLS) which uses scipy's curve_fit to find the parameters. This may sometimes fail as curve_fit is an optimization routine that needs an initial guess provided by this function.

```
reliability.Utils.life_stress_plot(model, dist, life_func, failure_groups, stresses_for_groups,  
use_level_stress=None, ax=True)
```

Generates a life stress plot using the inputs provided. The life stress plot is an output from each of the ALT_fitters.

Parameters

- **model** (*str*) – Must be either "Exponential", "Eyring", "Power", "Dual_Exponential", "Power_Exponential", or "Dual_Power".
- **dist** (*str*) – Must be either "Weibull", "Exponential", "Lognormal", or "Normal"
- **life_func** (*function*) – The life function for the ALT life model.
- **failure_groups** (*list*) – The failure groups. This is a list of lists.
- **stresses_for_groups** (*list*) – The stresses for the failure groups
- **use_level_stress** (*float, int, array, list, None*) – The use level stress. This must be an array or list for dual stress models. Default is None.
- **ax** (*axes, bool, optional*) – The axes handle to use. Default is True which will create a new plot. If False then no plot will be generated. This is also used to flip the x and y axes to make a stress-life plot. Use ax='swap' to make the life-stress plot change its axes to be a stress-life plot (similar to SN diagram).

Returns

current_axes (*axes*) – The axes handle of the plot. If ax is specified in the inputs then this will be the same handle.

```
reliability.Utils.line_no_autoscale(x, y, **kwargs)
```

Creates a line without adding it to the global list of autoscale objects. Use this when you want to plot something but not have it considered when autoscale sets the range.

Parameters

- **x** (*array, list*) – The x values for the line
- **y** (*array, list*) – The y values for the line
- **kwargs** – keyword arguments passed to the matplotlib LineCollection

Returns

None – The line will be added to the plot.

```
reliability.Utils.linear_regression(x, y, slope=None, x_intercept=None, y_intercept=None,  
RRX_or_RRY='RRX', show_plot=False, **kwargs)
```

This function provides the linear algebra solution to find line of best fit passing through points (x,y). Options to specify slope or intercept enable these parameters to be forced.

Rank regression can be on X (RRX) or Y (RRY). Default is RRX. Note that slope depends on RRX_or_RRY. If you use RRY then slope is dy/dx but if you use RRX then slope is dx/dy.

Parameters

- **x** (*array, list*) – The x values
- **y** (*array, list*) – The y values
- **slope** (*float, int, optional*) – Used to force the slope. Default is None.
- **x_intercept** (*float, int, optional*) – Used to force the x-intercept. Default is None. Only used for RRY.
- **y_intercept** (*float, int, optional*) – Used to force the y-intercept. Default is None. Only used for RRX.
- **RRX_or_RRY** (*str, optional*) – Must be “RRY” or “RRX”. Default is “RRY”.
- **show_plot** (*bool, optional*) – If True, a plot of the line and points will be generated. Use `plt.show()` to show it.
- **kwargs** – Keyword arguments for the plot that are passed to matplotlib for the line.

Returns

- **slope** (*float*) – The slope of the line.
- **intercept** (*float*) – The intercept (x or y depending on RRX_or_RRY) of the line.

Notes

The equation of a line used here is $Y = \text{slope} * X + \text{intercept}$. This is the RRY form. For RRX it can be rearranged to $X = (Y - \text{intercept})/\text{slope}$

For more information on linear regression, see the [documentation](#).

`class reliability.Utils.make_fitted_dist_params_for_ALT_probplots(dist, params)`

This function creates a class structure for the ALT probability plots to give to `Probability_plotting`.

Parameters

- **dist** (*str*) – The distribution. Must be either “Weibull”, “Lognormal”, “Normal”, or “Exponential”.
- **params** (*list, array*) – The parameters of the model. Must be 2 elements for Weibull, Lognormal, and Normal, and must be 1 element for Exponential.

Returns

- **alpha** (*float*) – Only returned for Weibull
- **beta** (*float*) – Only returned for Weibull
- **gamma** (*int*) – This will always be 0. Only returned for Weibull, Lognormal, and Exponential.
- **alpha_SE** (*None*) – Only returned for Weibull
- **beta_SE** (*None*) – Only returned for Weibull
- **Cov_alpha_beta** (*None*) – Only returned for Weibull
- **mu** (*float*) – Only returned for Normal and Lognormal
- **sigma** (*float*) – Only returned for Normal and Lognormal
- **Cov_mu_sigma** (*None*) – Only returned for Normal and Lognormal
- **Lambda** (*float*) – Only returned for Exponential

- **Lambda_SE** (*None*) – Only returned for Exponential

Notes

This function only exists to convert a list or array of parameters into a class with the correct parameters for the probability plots to use.

`reliability.Utils.no_reverse(x, CI_type, plot_type)`

This is used to convert an array that decreases and then increases into an array that decreases then is constant at its minimum.

The always decreasing rule will apply unless `CI_type` = ‘time’ and `plot_type` = ‘CHF’

This function is used to provide a correction to the confidence intervals which mathematically are correct but practically should never decrease.

Parameters

- **x** (*array, list*) – The array or list to which the `no_reverse` correction is applied
- **CI_type** (*str*) – Must be either ‘time’ or ‘reliability’
- **plot_type** (*str*) – Must be either ‘CDF’, ‘SF’, or ‘CHF’

Returns

x (*array*) – A corrected form of the input `x` that obeys the always decreasing rule (or the always increasing rule in the case of `CI_type` = ‘time’ and `plot_type` = ‘CHF’).

`reliability.Utils.probability_plot_xylims(x, y, dist, spacing=0.1, gamma_beta=None, beta_alpha=None, beta_beta=None)`

This function finds what the `x` and `y` limits of probability plots should be and sets these limits. This is similar to autoscaling, but the rules here are different to the matplotlib defaults. It is used extensively by the functions within the `probability_plotting` module to achieve the plotting style used within this library.

Parameters

- **x** (*list, array*) – The `x`-values from the plot
- **y** (*list, array*) – The `y`-values from the plot
- **dist** (*str*) – Must be either “weibull”, “lognormal”, “loglogistic”, “normal”, “gamma”, “exponential”, “beta”, or “gumbel”.
- **spacing** (*float*) – The spacing between the points and the edge of the plot. Default is 0.1 for 10% spacing.
- **gamma_beta** (*float, int, optional*) – The beta parameter from the gamma distribution. Only required if `dist` = “gamma”.
- **beta_alpha** (*float, int, optional*) – The alpha parameter from the beta distribution. Only required if `dist` = “beta”.
- **beta_beta** (*float, int, optional*) – The beta parameter from the beta distribution. Only required if `dist` = “beta”.

Returns

None – There are no outputs from this function. It will set the `xlim()` and `ylim()` of the probability plot automatically.

`reliability.Utils.probability_plot_xyticks(yticks=None)`

This function sets the `x` and `y` ticks for probability plots.

`X` ticks are selected using either `MaxNLocator` or `LogLocator`. `X` ticks are formatted using a custom formatter.

Y ticks are specified with FixedLocator due to their irregular spacing. Minor y ticks use MaxNLocator. Y ticks are formatted using a custom Percent Formatter that handles decimals better than the default.

This function is used by all the probability plots.

Within this function are several sub functions that are called internally.

Parameters

yticks (*list, array*) – The yticks to use. If unspecified, the default yticks are [0.0001, 0.001, 0.002, 0.005, 0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99, 0.999, 0.9999, 0.99999].

Returns

None – This function will set the ticks but it does not return anything.

reliability.Utils.removeNaNs(*X*)

Removes NaNs from a list or array.

Parameters

X (*array, list*) – The array or list to be processed.

Returns

output (*list, array*) – A list or array of the same type as the input with the NaNs removed.

Notes

This is better than simply using “`x = x[numpy.logical_not(numpy.isnan(x))]`” as numpy crashes for str and bool.

reliability.Utils.reshape_figure(*handle*)

Shows a figure from an axes handle or figure handle. This is useful if the handle is saved to a variable but the figure has been closed. Note that the Navigation Toolbar (for pan, zoom, and save) is still connected to the old figure. There is no known work around for this issue.

Parameters

handle (*object*) – The axes handle (type(`_axes.Axes`)) or figure handle (type(`Figure`))

Returns

None – The figure is automatically shown using `plt.show()`.

reliability.Utils.restore_axes_limits(*limits, dist, func, X, Y, xvals=None, xmin=None, xmax=None*)

This function works in a pair with `get_axes_limits`. Using the values produced by `get_axes_limits` which are `[xlims, ylims, use_prev_lims]`, this function will determine how to change the axes limits to meet the style requirements of the library.

Parameters

- **limits** (*list*) – A list of `[xlims, ylims, use_prev_lims]` created by `get_axes_limits`
- **dist** (*object*) – The distribution object which the axes limits are influenced by.
- **X** (*array, list*) – The x-values of the plot
- **Y** (*array, list*) – The y-values of the plot
- **xvals** (*array, list, optional*) – The plot `xvals` if specified. May be `None` if not specified.
- **xmin** (*int, float, optional*) – The plot `xmin` if specified. May be `None` if not specified.
- **xmax** (*int, float, optional*) – The plot `xmax` if specified. May be `None` if not specified.

Returns

None – This function will scale the plot but it does not return anything

Notes

No scaling will be done if the axes are not linear due to errors that result from log and function scaled axes when a limit of 0 is used. This means that this function is not able to be applied to the probability plots are they have non-linear scaled axes.

```
reliability.Utils.round_and_string(number, decimals=5, integer_floats_to_ints=True,  
large_scientific_limit=1000000000.0, small_scientific_limit=0.0001)
```

This function is used to round a number to a specified number of decimals and convert to string. It is used heavily in the formatting of the parameter titles within reliability.Distributions

The rules applied are slightly different than rounding to a number of significant figures as it keeps more preceding zeros. Special formatting rules are applied when:

```
abs(number) <= small_scientific_limit abs(number) >= large_scientific_limit number = 0
```

Parameters

- **number** (*float*) – The number to be rounded
- **decimals** (*int*) – The number of decimals (not including preceding zeros) that are to be in the output
- **integer_floats_to_ints** (*bool, optional*) – Default is True. Removes trailing zeros from floats if there are no significant decimals (eg. 12.0 becomes 12).
- **large_scientific_limit** (*int, float, optional*) – The limit above which to keep numbers formatted in scientific notation. Default is 1e9.
- **small_scientific_limit** (*int, float, optional*) – The limit below which to keep numbers formatted in scientific notation. Default is 1e-4.

Returns

out (*string*) – The number formatted and converted to string.

Notes

Examples (with decimals = 5):

```
original: -1e+20 // new: -1e+20 original: -100000000.0 // new: -100000000 original: -10.0 // new: -10  
original: -5.4857485e-05 // new: -5.48575e-05 original: -2.54875415e-16 // new: -2.54875e-16 original:  
0.0 // new: 0 original: 0 // new: 0 original: 2.54875415e-16 // new: 2.54875e-16 original: 5.4857485e-05  
// new: 5.48575e-05 original: 10.0 // new: 10 original: 100000000.0 // new: 100000000 original: 1e+20 //  
new: 1e+20
```

```
reliability.Utils.transform_spaced(transform, y_lower=1e-08, y_upper=0.99999999, num=1000,  
alpha=None, beta=None)
```

Creates linearly spaced array based on a specified transform.

This is similar to np.linspace or np.logspace but is designed for weibull space, exponential space, normal space, gamma space, loglogistic space, gumbel space and beta space.

It is useful if the points generated are going to be plotted on axes that are scaled using the same transform and need to look equally spaced in the transform space.

Parameters

- **transform** (*str*) – The transform name. Must be either weibull, exponential, normal, gamma, gumbel, loglogistic, or beta.
- **y_upper** (*float, optional*) – The lower bound (must be within the bounds 0 to 1). Default is 1e-8

- **y_lower** (*float, optional*) – The upper bound (must be within the bounds 0 to 1). Default is 1-1e-8
- **num** (*int, optional*) – The number of values in the array. Default is 1000.
- **alpha** (*int, float, optional*) – The alpha value of the beta distribution. Only used if the transform is beta
- **beta** (*int, float, optional*) – The beta value of the beta or gamma distribution. Only used if the transform is beta or gamma

Returns

transformed_array (*array*) – transform spaced array. This appears linearly spaced when plotted in transform space.

Notes

Note that lognormal is the same as normal, since the x-axis is what is transformed in lognormal, not the y-axis.

reliability.Utils.unpack_single_arrays(*array*)

Unpacks arrays with a single element to return just that element

Parameters

array (*float, int, list, array*) – The value for unpacking

Returns

output (*float, list, int, array*) – If the input was a single length numpy array then the output will be the item from the array. If the input was anything else then the output will match the input

reliability.Utils.validate_CI_params(*args)

Returns False if any of the args is None or Nan, else returns True. This function is different to using all() because it performs the checks using np.isfinite(arg).

Parameters

***args** (*bool*) – Any number of boolean arguments

Returns

is_valid (*bool*) – False if any of the args is None or Nan else returns True.

reliability.Utils.write_df_to_xlsx(*df, path, **kwargs*)

Writes a dataframe to an xlsx file For use exclusively by the Convert_data module

Parameters

- **df** (*dataframe*) – The dataframe to be written
- **path** (*str*) – The file path to the xlsx file.

Returns

None – Writing the dataframe is the only action from this function.

Notes

The path must include the full file path including the extension. It is also necessary to use r at the start to specify raw text. See the [documentation](#) for an example.

reliability.Utils.xy_downsample(*x, y, downsample_factor=None, default_max_values=1000*)

This function downsamples the x and y arrays. This exists to make plotting much faster, particularly when matplotlib becomes very slow for tens of thousands of datapoints.

Parameters

- **x** (*array, list*) – The x values

- **y** (*array, list*) – The y values
- **downsample_factor** (*int, optional*) – How must downsampling to do. See Notes for more detail.
- **default_max_values** (*int, optional*) – The maximum number of values to be returned if downsample_factor is None. See Notes for more detail.

Returns

- **x** (*array*) – The downsampled x values
- **y** (*array*) – The downsampled y values

Notes

Downsampling is done using the downsample_factor. If the down_sample factor is 2 then every second value will be returned, if 3 then every third value will be returned. The first and last items will always be included in the downsampled dataset.

If downsample_factor is not specified, downsampling will only occur if there are more than default_max_values. The downsample factor will aim for the number of values to be returned to be between default_max_values/2 and default_max_values. By default this is between 500 and 1000.

`reliability.Utils.xy_transform(value, direction='forward', axis='x')`

This function converts between data values and axes coordinates (based on xlim() or ylim()).

If direction is forward the returned value will always be between 0 and 1 provided the value is on the plot.

If direction is reverse the input should be between 0 and 1 and the returned value will be the data value based on the current plot lims

Parameters

- **value** (*int, float, list, array*) – The value/s to be transformed
- **direction** (*str, optional*) – Must be “forward” or “inverse”. Default is “forward”
- **axis** (*str, optional*) – Must be “x” or “y”. Default is “x”.

Returns

transformed_values (*float, array*) – The transformed values. This will be a float if the input was int or float, or an array if the input was list or array.

`reliability.Utils.zeroise_below_gamma(X, Y, gamma)`

This will make all Y values 0 for the corresponding X values being below gamma (the threshold parameter for Weibull, Exponential, Gamma, Loglogistic, and Lognormal).

Used by HF and CHF which need to be zeroized if the gamma shifted form of the equation is used.

Parameters

- **X** (*array, list*) – The x values of the distribution. These are used to determine which Y values to zeroize.
- **Y** (*array, list*) – The y-values of the distribution
- **gamma** (*float, int*) – The gamma parameter. This is the point at which Y values corresponding to X values below gamma will be zeroized.

Returns

Y (*array*) – The zeroized Y array

PYTHON MODULE INDEX

r

reliability.Datasets, ??
reliability.Utils, ??