weibull Documentation

Release 0.0

Jason R. Jones

Jan 11, 2018

Contents

1 Installation **1**
- 1.1 Using `pip` . . . 1
- 1.2 Using `setup.py` . . . 1
- 1.3 Using `conda` . . . 1
- 1.4 Dependencies . . . 1

2 Analysis **3**
- 2.1 Fitting . . . 3
- 2.2 Plotting . . . 4
- 2.3 Class Documentation . . . 6

3 Design **11**
- 3.1 Class Documentation . . . 11

4 Weibayes **13**
- 4.1 Class Documentation . . . 13

5 Examples **15**
- 5.1 Determining β and η Values . . . 15
- 5.2 Life Calculations . . . 16
- 5.3 Basic Life Statistics . . . 16
- 5.4 Plotting . . . 17
- 5.5 Test Design . . . 17
- 5.6 Weibayes Analysis . . . 18

6 Gallery **21**
- 6.1 Example 1: Multiple-Censored Data . . . 21

7 Introduction to Reliability Analysis **27**
- 7.1 Weibull Distribution . . . 27
- 7.2 Censoring Data . . . 31
- 7.3 Determining β and η . . . 31
- 7.4 B-Life . . . 32
- 7.5 Confidence Levels . . . 32

8 Introduction **35**

9 Installation **37**

9.1 Dependencies 37

10 Test Coverage **39**

11 Classes **41**

CHAPTER 1

Installation

1.1 Using `pip`

Ideally, you should be able to pip install weibull and simply be finished. This package is a pure-python package, so it should work on any os. Unfortunately, this package utilizes other packages which may be more difficult to install. Please consult package documentation for more details.

This user had the most issues installing statsmodels in a Windows 10 environment. Other package dependencies - numpy, pandas, and matplotlib - installed without issue.

1.2 Using `setup.py`

Additional dependencies are utilized by calling `python setup.py install` which ensure that the environment is appropriately set up for development. For instance, the additional dependencies are `flake8`, `pytest`, and `sphinx`.

Simply download a copy of the repository, `cd` into the directory, and `python setup.py install`.

1.3 Using `conda`

If you are having installation issues, perhaps try the Anaconda distribution! As I understand it, they have solved most of these installation problems for difficult packages!

1.4 Dependencies

For most installations, you must have `pandas`, `numpy`, `matplotlib`, and `scipy` properly installed into your environment.

CHAPTER 2

Analysis

The Analysis class is the primary class which will provide methods for analyzing your life data. This class is designed to take your data and calculate β and η values along with generating any appropriate plots for display of your data.

A typical use case of the `Analysis` case is as follows:

```
import weibull

# fail times include no censored data
fail_times = [
    9402.7,
    6082.4,
    13367.2,
    10644.6,
    8632.0,
    3043.4,
    12860.2,
    1034.5,
    2550.9,
    3637.1
]

# this is where the actual analysis and curve fitting occur
analysis = weibull.Analysis(fail_times, unit='hour')
```

2.1 Fitting

The `fit()` method is used to calculate appropriate β and η values, which are then stored into the class instance. When `fit()` is called with no parameters, then the linear regression method of calculation is assumed:

```
analysis.fit()
```

An alternative method is to use the Maximum Likelihood Estimation (MLE) method of fitting β and η to the data. This may be done by specifying that the `method='mle'`:

```
analysis.fit(method='mle')
```

In many cases, the `mle` and `lr` methods will yield very similar values for β and η, but there are some cases in which one is preferred over the other. It turns out that linear regression tends to work best for very low sample sizes, usually less than 15 while the maximum likelihood estimator works best for high sample sizes. In both cases, the `probplot()` method should be used to verify that the data is a good fit.

To retrieve the β and η values, simply use the instance variables `beta` and `eta`:

```
print(f'beta: {analysis.beta: .02f}')
print(f'eta: {analysis.eta: .02f}')
```

When using the `fit()` method, it is also possible to set the confidence levels

Use the `stats()` method to get a `pandas.Series` containing most internal estimates:

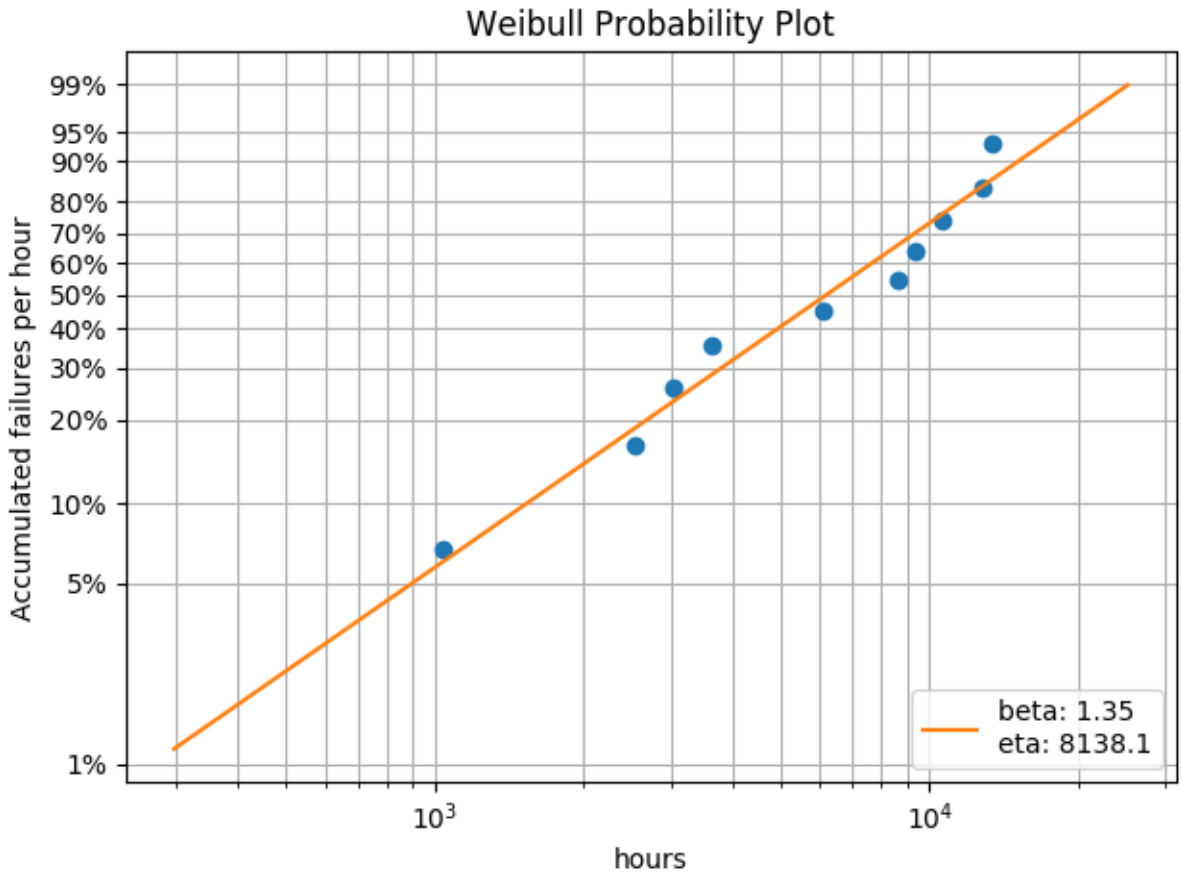
```
$> analysis.stats
fit method          maximum likelihood estimation
confidence                                    0.6
beta lower limit                          2.42828
beta nominal                              2.97444
beta upper limit                          3.64344
eta lower limit                           186.483
eta nominal                               203.295
eta upper limit                           221.622
mean life                                  181.47
median life                               179.727
b10 life                                   95.401
dtype: object
```

2.2 Plotting

One of the most often requested features of such a package is plotting the data, particularly in Jupyter Notebooks. The `weibull` package comes with built-in methods to easily display and save standard plots with one-line methods.

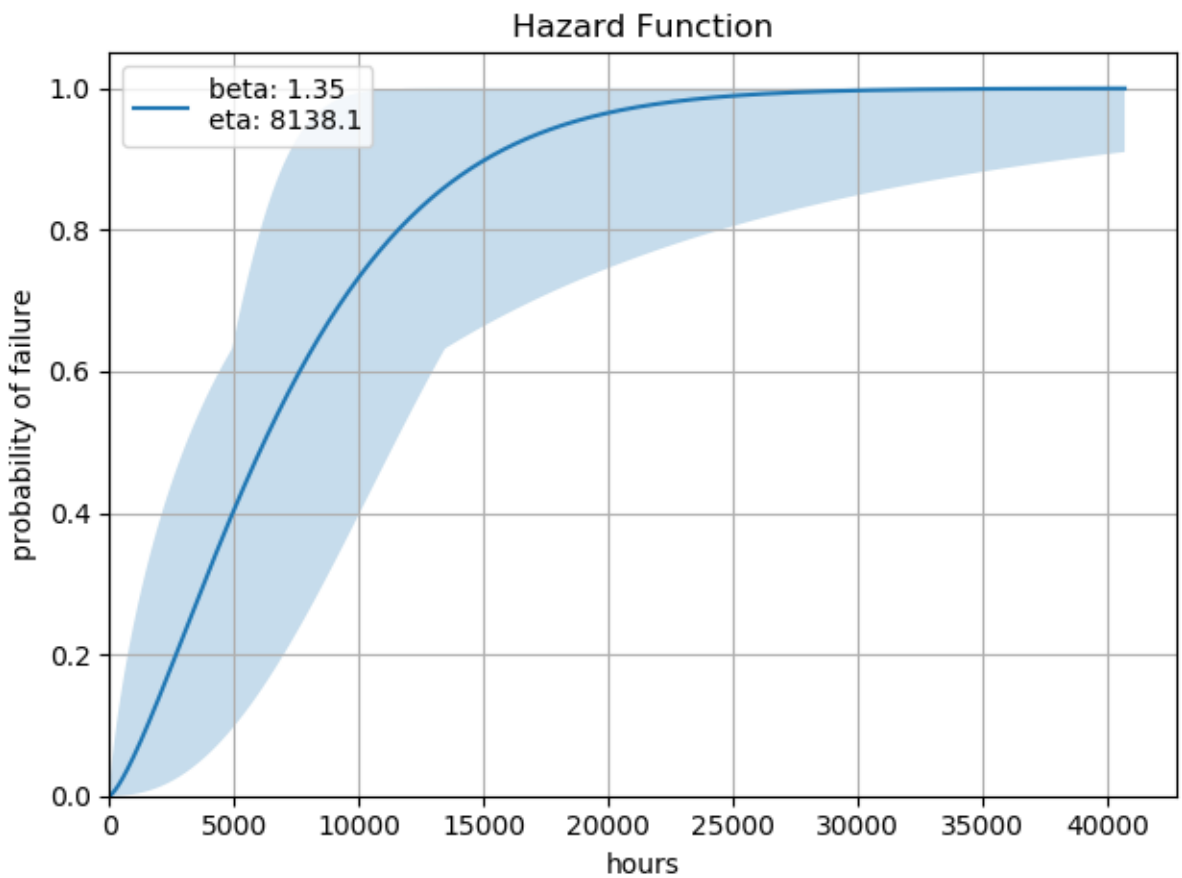
Building on the `analysis` instance above, we will examine the probability plot:

```
analysis.probplot()
```



We can also examine a number of other common function plots (only the hazard plot is shown, but the others are along the same line).:

```
analysis.pdf()
analysis.sf()
analysis.hazard()
analysis.cdf()
```



Each of these functions will generate a plot that is suitable for publication or insertion into a Jupyter Notebook. Again, note that some of these methods - such as `hazard()` and `cdf()` will produce the same plot with slightly different labeling.

2.2.1 Confidence Levels

Some plots will contain a shaded region which reflects the confidence_levels. The confidence levels are calculate for β and η and the min/max values for β and η are explored rather than all possible values. As a result, the visualizations shown are an approximation of the confidence limits.

2.3 Class Documentation

class `weibull.`**Analysis**(*data: list*, *suspended: list = None*, *unit: str = 'cycle'*)

Calculates and plots data points and curves for a standard 2-parameter Weibull for analyzing life data.

Parameters

- **data** – A list or numpy array of life data, i.e. `[127, 234, 329, 444]`
- **suspended** – A list or numpy array of suspensions as boolean values, i.e. `[False, False, True, True]`. At any point which indicates `True` means that the test was stopped - or that the item was removed from the test - before the item failed.

- **unit** – The unit ('hour', 'minute', 'cycle', etc.). This is used to add some useful information to the visualizations. For instance, if the unit is `hour`, then the x-axis will be labed in hours.

Variables

- **beta** – The current value of the shape parameter, β. This value is initially set to `None`. The proper value for `beta` will be calculated on call to the `fit()` method. The user may also set this value directly.
- **eta** – The current value of the scale parameter, η. This value is initially set to `None`. The proper value for `beta` will be calculated on call to the `fit()` method. The user may also set this value directly.
- **_fit_test** – Basic statistics regarding the results of `fit()`, such as R^2 and P-value.

b(*percent_failed: (<class 'float'>, <class 'str'>) = 10.0*)

Calculate the B-life value

Parameters **percent_failed** – the number of elements that have failed as a percent (i.e. 10)

Returns the life in cycles/hours/etc.

cdf(*show: bool = True*, *file_name: str = None*, *watermark_text=None*)

Plot the cumulative distribution function

Parameters

- **show** – True if the plot is to be shown, false if otherwise
- **file_name** – the file name to be passed to `matplotlib.pyplot.savefig`
- **watermark_text** – the text to include as a watermark

Returns None

characteristic_life

Returns the current characteristic life of the product, aka η

Returns the characteristic life of the product

fit(*method: str = 'lr'*, *confidence_level: float = 0.9*)

Calculate β and η using a linear regression or using the maximum likelihood method, depending on the 'method' value.

Parameters

- **method** – 'lr' for linear estimation or 'mle' for maximum likelihood estimation
- **confidence_level** – A number between 0.001 and 0.999 which expresses the confidence levels desired. This confidence level is reflected in all subsequent actions, especially in plots, and can also affect several internal variables which are shown in `stats`.

Returns None

fr(*show: bool = True*, *file_name: str = None*, *watermark_text=None*)

Plot failure rate as a function of cycles

Parameters

- **show** – True if the item is to be shown now, False if other elements to be added later
- **file_name** – if file_name is stated, then the probplot will be saved as a PNG
- **watermark_text** – the text to include as a watermark

Returns None

hazard(*show: bool = True*, *file_name: str = None*, *watermark_text=None*)
Plot the hazard (CDF) function

Parameters

- **show** – True if the plot is to be shown, false if otherwise
- **file_name** – the file name to be passed to `matplotlib.pyplot.savefig`
- **watermark_text** – the text to include as a watermark

Returns None

mean
Calculates and returns mean life (aka, the MTTF) is the integral of the reliability function between 0 and inf,

$$MTTF = \eta\Gamma(\frac{1}{\beta} + 1)$$

where gamma function, Γ, is evaluated at $\frac{1}{\beta+1}$

Returns the mean life of the product

median
Calculates and returns median life of the product

Returns The median life

mttf
Calculates and returns mean time between failures (MTTF)

Returns the mean time to failure

pdf(*show: bool = True*, *file_name: str = None*, *watermark_text=None*)
Plot the probability density function

Parameters

- **show** – True if the plot is to be shown, false if otherwise
- **file_name** – the file name to be passed to `matplotlib.pyplot.savefig`
- **watermark_text** – the text to include as a watermark

Returns None

probplot(*show: bool = True*, *file_name: str = None*, *watermark_text=None*, ***kwargs*)
Generate a probability plot. Use this to show the data points plotted with the beta and eta values.

Parameters

- **show** – True if the plot is to be shown, false if otherwise
- **file_name** – the file name to be passed to `matplotlib.pyplot.savefig`
- **watermark_text** – the text to include on the plot as a watermark
- **kwargs** – valid matplotlib options

Returns None

sf(*show: bool = True*, *file_name: str = None*, *watermark_text=None*)
Plot the survival function

Parameters

- **show** – True if the plot is to be shown, false if otherwise
- **file_name** – the file name to be passed to `matplotlib.pyplot.savefig`
- **watermark_text** – the text to include as a watermark

Returns None

stats

Returns the fit statistics, confidence limits, etc :return: a pandas series containing the fit statistics

CHAPTER 3

Design

The `Design` class is used to design a test based on certain assumptions.

A typical example of use:

```
import weibull

designer = weibull.Design(
    target_cycles=10000,
    reliability=0.9,
    confidence_level=0.90,
    expected_beta=1.5
)

# The 'test_cycles' parameter can be in any units.
# Days, weeks, hours, cycles, etc., so long
#   as the target unit is consistent
print(f'Minimum number of units for 10000 hour run: {designer.num_of_units(test_
↪cycles=10000)}')
print(f'Minimum hours for 20 units: {designer.num_of_cycles(number_of_units=20)}')
```

3.1 Class Documentation

class weibull.**Design**(*target_cycles: (<class 'int'>, <class 'float'>), reliability: float = 0.9, confidence_level: float = 0.9, expected_beta: float = 2.0*)

Will determine the required test time required given the number of units under test and the target cycles OR it will determine the number of units given the test time and the target cycles.

Parameters

- **target_cycles** – The target number of cycles/minutes/hours
- **reliability** – The fraction of units still running after target_cycles, 0.001 to 0.999
- **confidence_level** – The fractional level of confidence, 0.001 to 0.999

- **expected_beta** – The anticipated level of beta - often worse-case - based on historical data or other assumptions

num_of_cycles(*number_of_units: int*)

Design a test, calculating the test duration/cycles to prove the required reliability at target_cycles.

Returns the required duration or cycles

num_of_units(*test_cycles: (<class 'int'>, <class 'float'>)*)

Design a test, calculating the number of units required to run for the test duration / cycles in order to prove the reliability at target_cycles.

Returns The number of units required

CHAPTER 4

Weibayes

4.1 Class Documentation

class weibull.**Weibayes**(*data: list*, *confidence_level: float = None*, *beta: float = 2.0*)

Weibayes-style analysis of the data with a confidence level and beta.

Parameters

- **data** – The data for each unit
- **confidence_level** – The fractional level of confidence, 0.001 to 0.999
- **beta** – The shape parameter

b(*b_spec: int = 10*, *confidence_level: float = None*)

Calculates the B-life

Parameters

- **b_spec** – the B-specification (for instance, '10')
- **confidence_level** – the confidence level (usually between 0.01 and 0.99)

Returns the B life

plot(*confidence_level: float = None*, *show: bool = True*, *file_name: str = None*)

Plot the linear plot line.

Confidence_level the desired confidence level

Show True if the plot is to be shown

File_name Save the plot as "file_name"

CHAPTER 5

Examples

5.1 Determining β and η Values

Before any suppositions may be gathered, it is appropriate to calculate β and η values. Once we are satisfied that β and η match the raw data, we can move on to determining useful life characteristics for the product.

5.1.1 Example 1: Complete Test Data

In this example, we will take a complete set of failure data that has no censorship and apply basic weibull analysis tool suite in order to achieve a simple, accurate, and useful analysis.:

```
import weibull

# fail times include no censored data
fail_times = [
    9402.7,
    6082.4,
    13367.2,
    10644.6,
    8632.0,
    3043.4,
    12860.2,
    1034.5,
    2550.9,
    3637.1
]


# this is where the actual analysis and curve fitting occur
analysis = weibull.Analysis(fail_times,
                            unit='hour')
analysis.fit(method='mle')
```

```
analysis.probplot()
```

In this example, we chose to use the Maximum Likelihood Estimation method of estimating β and η, which is shown in the `analysis.fit(method='mle)` line. If the `fit()` method were called with no parameters, it would - by default - have used linear regression.

By examining the probability plot, we can visually determine if the β and η are appropriately calculated.

By specifying a file name, the probability plot can be saved to a file `analysis.probplot(file_name='prob.png')`. This is optional, of course, and not required.

5.1.2 Example 2: Right-Censored Data

Often, it is necessary to use only the smallest amount of data in order to calculate the values for β and η. For instance, a long-running test might have 10 units on the test bench, but only 3 of them have failed. When the data is so small, the default linear regression fit method is probably going to yield better results than the maximum-likelihood estimation:

```
current_run_time = 4200.0

fail_times = [current_run_time] * 10
fail_times[7] = 1034.5
fail_times[8] = 2550.9
fail_times[6] = 3043.4

suspended = [True, True, True, True, True,
             False, False, False, True, True]

analysis = weibull.Analysis(fail_times,
                            suspended=suspended,
                            unit='hour')
analysis.fit()

analysis.probplot()
```

Again, we plot the raw data points against the calculated β and η in order to ensure that the linear regression is an appropriate fit for the data. As more failures occur, more accurate curve fits may be run.

5.2 Life Calculations

Once β and η are determined, then they may be utilized to obtain the basic lifetime data that may be utilized for planning. One common reliability metric is the *B-Life*. Obtaining a B10 life using the `analysis` object is trivial:

```
print(f'B10 life: {analysis.b(10):.0f}')
```

As you can see, simply calling the `b()` function with the appropriate number as the parameter will return the B-life based on β and η.

5.3 Basic Life Statistics

For user convenience, the `mean`, `median`, `characteristic_life`, and `mttf` are defined as attributes of the class and may be called at any time after an initial curve fit. Note that there is some overlap with other class vari-

ables. For instance, the `characteristic_life` happens to be the same thing as `eta`, but if a customer asks for the characteristic life, then having this available makes the code more readable and correspond more closely to the specification.

5.4 Plotting

We can also plot various functions of interest, such as the survival function and hazard functions, amongst others.:

```
analysis.pdf()       # probability density function
analysis.sf()        # survival function
analysis.hazard()    # hazard function
analysis.cdf()       # cumulative distribution function
analysis.fr()        # failure rate
```

Each of these will generate a plot of the function. For all plotting methods, if `file_name` is specified as a parameter, then the method will save to a file rather than display. For instance:

```
analysis.sf(file_name='survival_function.png')
```

5.5 Test Design

The Design class is to be utilized for two scenarios:

- determine the required number of units to prove the target reliability given a test cycles/duration
- determine the required number of cycles/duration to prove the target reliability given a number of units

To begin, first import and instantiate the Designer, which is the utility for the test designer. There are several parameters to consider and all of them are requirements or assumptions that must be entered as parameters for the Designer class:

- target_cycles - the target to be proven in hours/days/weeks/cycles
- reliability - defaults to 0.9
- confidence_level - defaults to 0.95
- expected_beta - an initial assumption for beta (defaults to 2)

Shown are two example calculations for a target lifetime of 10000 hours with a reliability of 0.9 at a confidence level of 0.5 and beta assumption of 1.5:

```
import weibull

designer = weibull.Design(
    target_cycles=10000,
    reliability=0.9,
    confidence_level=0.90,
    expected_beta=1.5
)

# The 'test_cycles' parameter can be in any units.
# Days, weeks, hours, cycles, etc., so long
#   as the target unit is consistent
print(f'Minimum number of units for 10000 hour run:{designer.num_of_units(test_
↪cycles=10000)}')
print(f'Minimum hours for 20 units: {designer.num_of_cycles(num_of_units=20)}')
```

5.6 Weibayes Analysis

Use Weibayes analysis to assist with designing your test or evaluating reliability within a certain confidence interval based on historical data.

You have a product that needs to be tested to B2 life of 40 million time units with a confidence limit of 95%. The product had an expected beta of 2 (lots of historical data there). B2 life is the same as 98% survival.

Using the weibull test *Design* class, we need to run 62 units (the limit of our test rig) for 62 million time units with no failures:

```
import weibull

designer = weibull.Design(
    target_cycles=40e6,
    reliability=0.98,
    confidence_level=0.95,
    expected_beta=2
)

print(f'Minimum hours for 62 units: {designer.num_of_cycles(num_of_units=62)}')
```

Result:

```
61860134.45191945
```

Weibayes analysis on the data would arrive at the same result.:

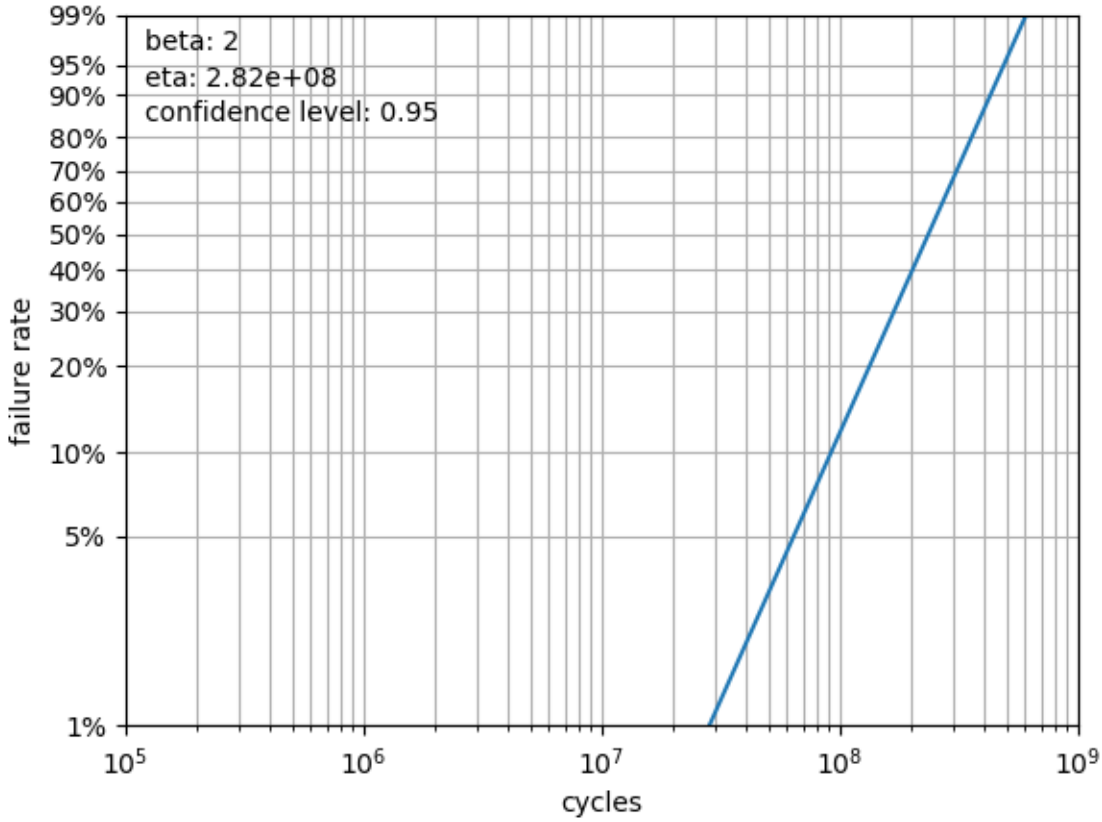
```
import weibull

# we want N units to run for H hours each
N = 62
H = 62.0e6

run_times_desired = [H] * N
weibayes = weibull.Weibayes(run_times_desired, confidence_level=0.95, beta=2)

print(f'B2 life: {weibayes.b(2)}')
```

Results:

```
B2 life: 40090439.86038491
```

Note that this *B2* matches very closely with *target_cycles* value found in the above iteration of the *Design* class.

We can further plot the data using *weibayes.plot()* resulting in:


beta: 2
eta: 2.82e+08
confidence level: 0.95
99%
95%
90%
80%
70%
60%
50%
40%
30%
20%
10%
5%
1%
failure rate
10^5
10^6
10^7
10^8
10^9
cycles

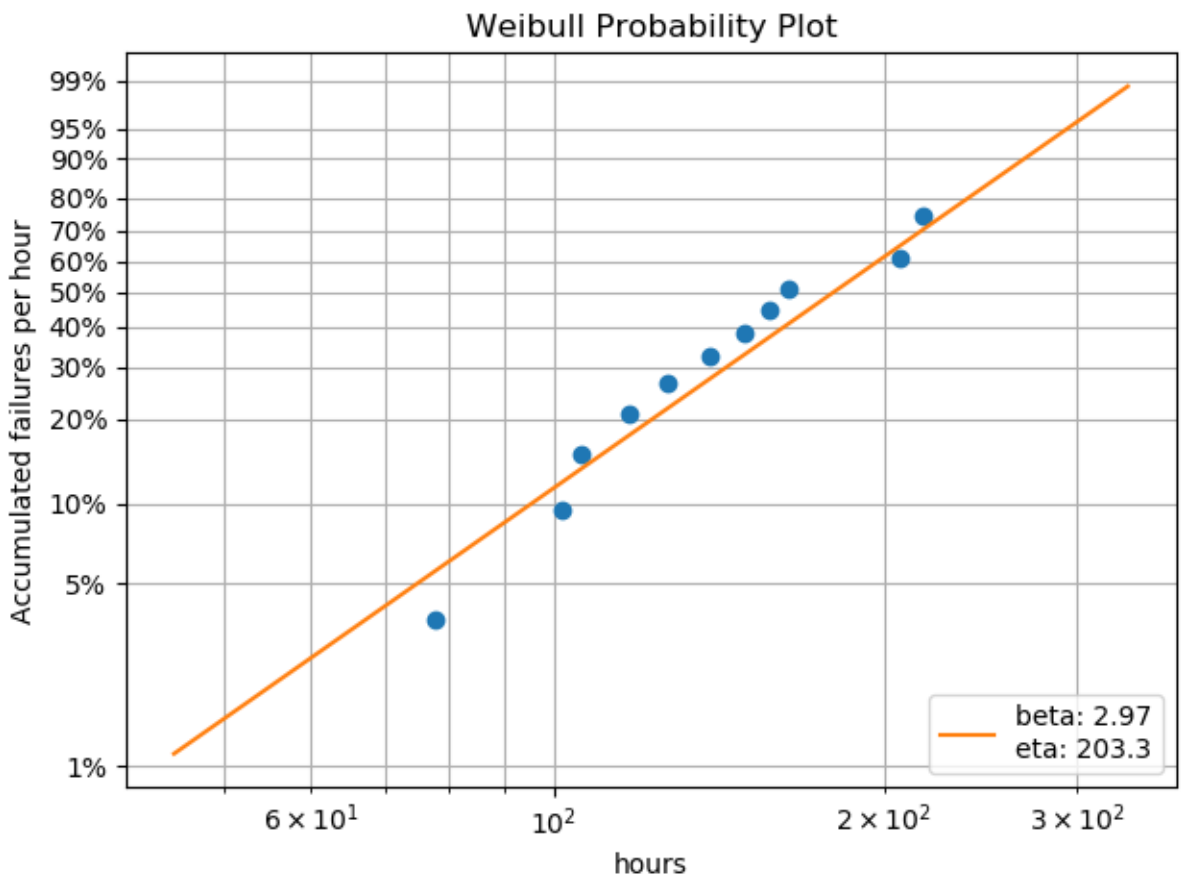
CHAPTER 6

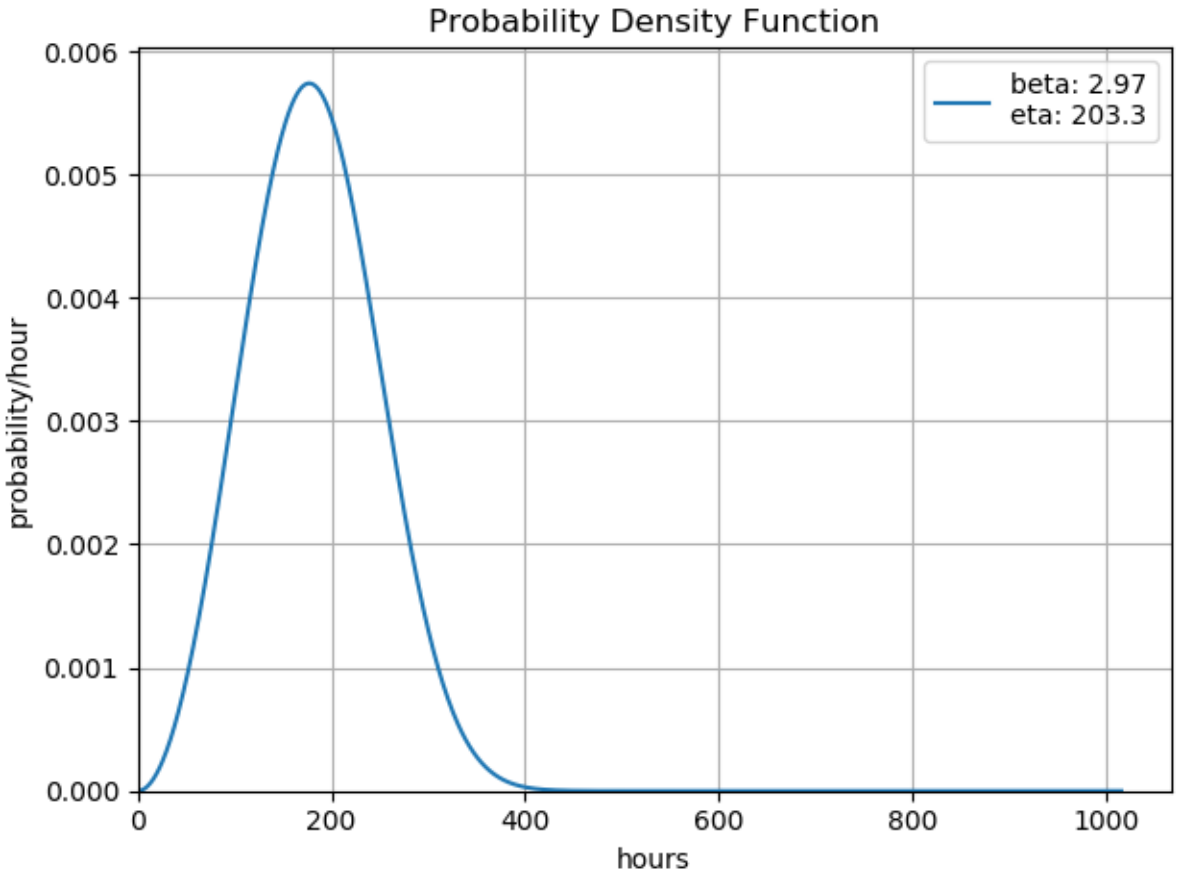
Gallery

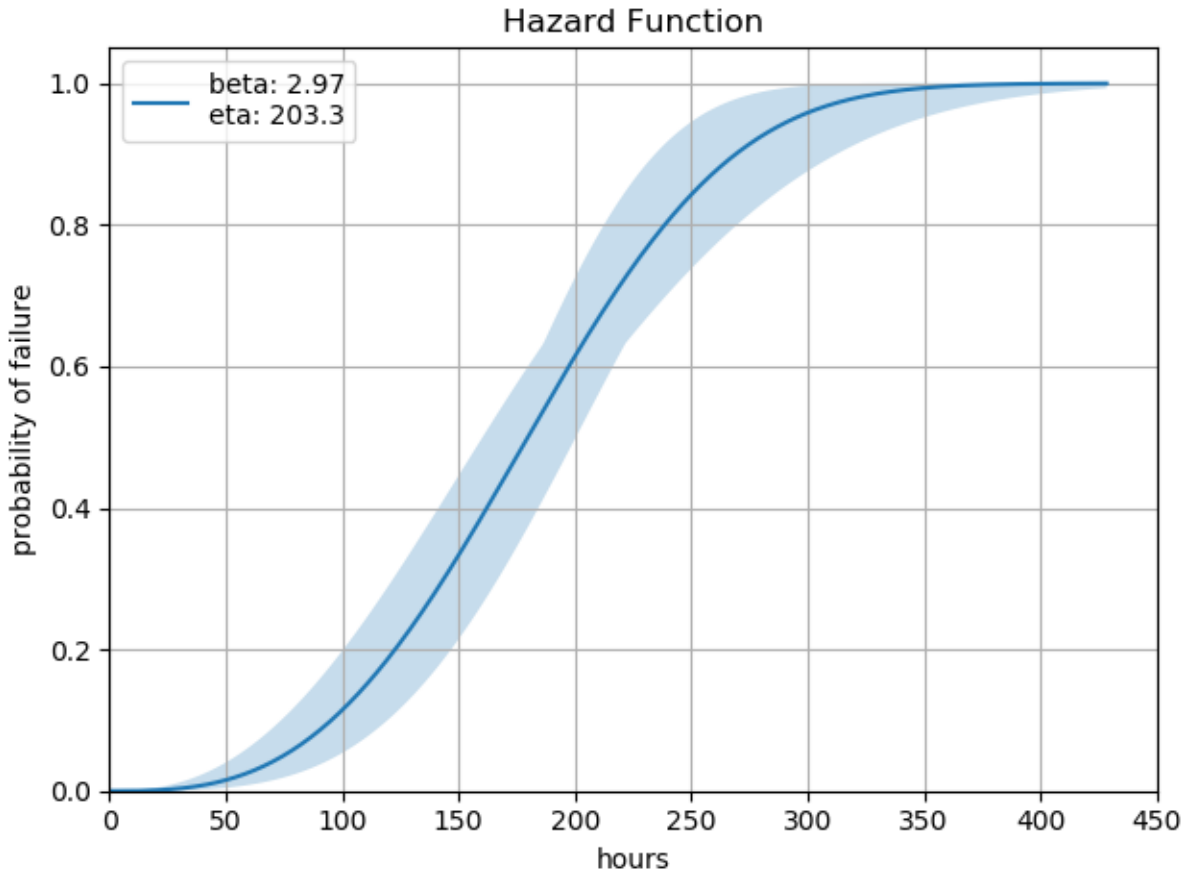
6.1 Example 1: Multiple-Censored Data

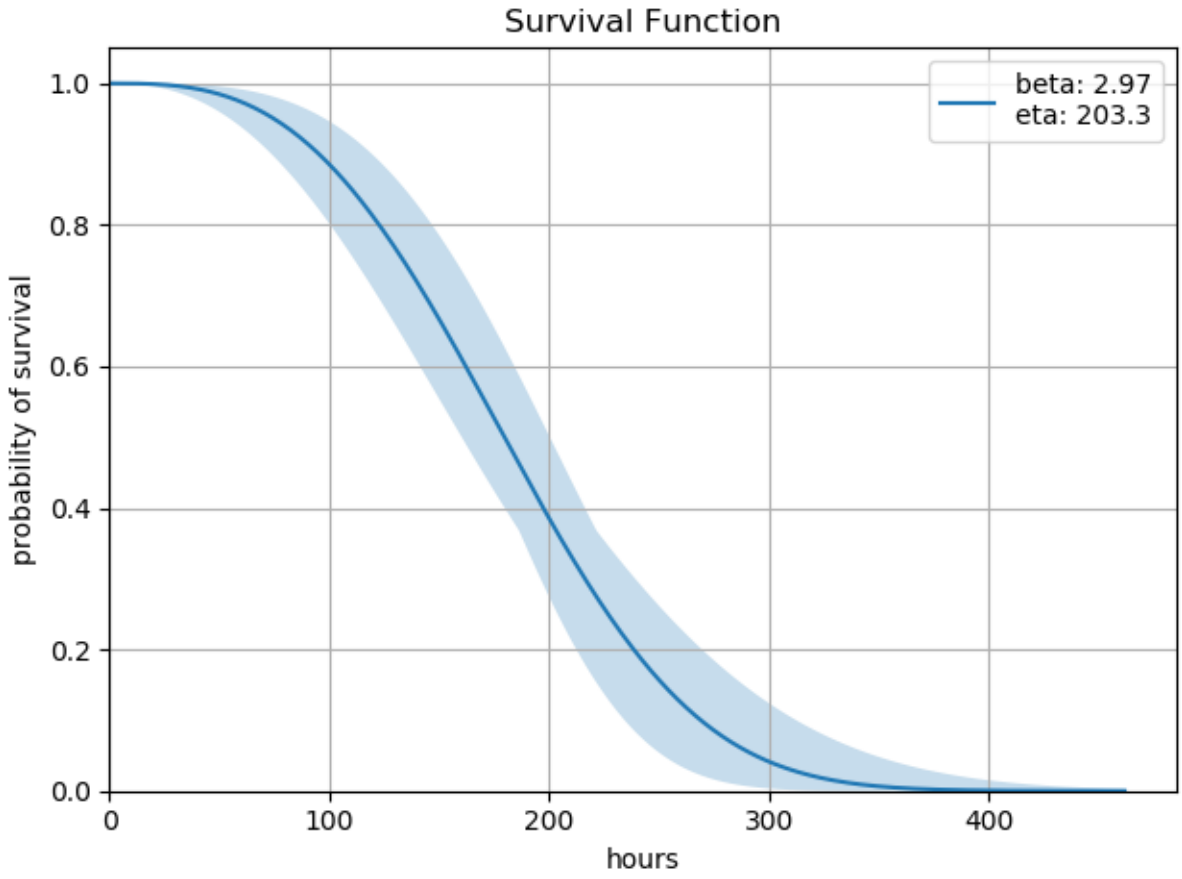
The code:

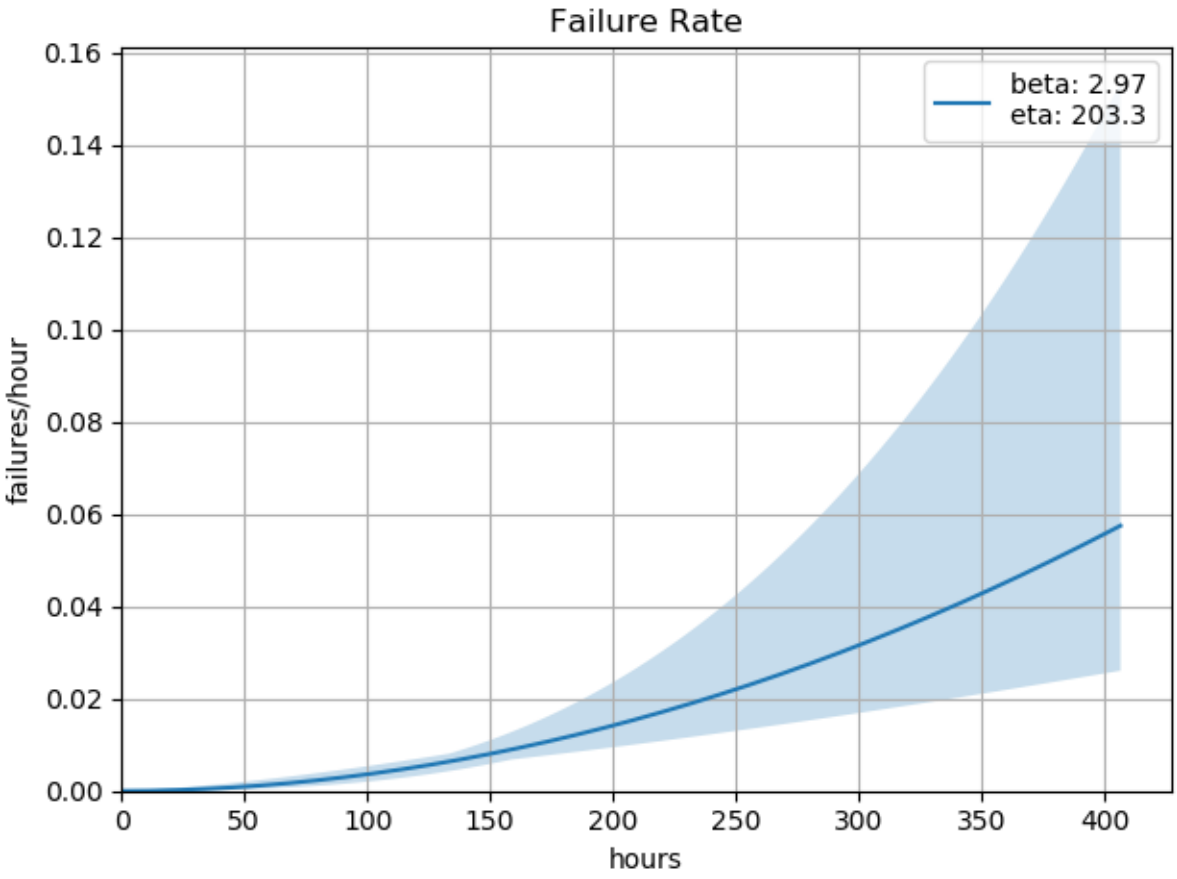
```
import weibull

fail_times = [
    42.1, 105.9, 151.3, 195.6,
    77.8, 117.0, 157.3, 207.0,
    83.3, 126.9, 163.8, 215.3,
    88.7, 138.7, 177.2, 217.4,
    101.8, 148.9, 194.3, 258.8
]

suspensions = [1, 0, 1, 1,
               0, 0, 0, 0,
               1, 0, 0, 1,
               1, 0, 1, 0,
               0, 0, 1, 1]

# this is where the actual analysis and curve fitting occur
analysis = weibull.Analysis(fail_times, suspensions, unit='hour')
analysis.fit(method='mle', confidence_level=0.6)

print(analysis.stats)

analysis.probplot(file_name='gallery-probplot.png')

analysis.pdf(file_name='gallery-pdf.png')
analysis.hazard(file_name='gallery-hazard.png')
analysis.sf(file_name='gallery-survival.png')
analysis.fr(file_name='gallery-fr.png')
```


Weibull Probability Plot
99%
95%
90%
80%
70%
60%
50%
40%
30%
20%
10%
5%
1%
Accumulated failures per hour
6×10^1
10^2
2×10^2
3×10^2
hours
beta: 2.97
eta: 203.3


Probability Density Function
beta: 2.97
eta: 203.3
probability/hour
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
200
400
600
800
1000
hours


Hazard Function
beta: 2.97
eta: 203.3
probability of failure
hours
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
250
300
350
400
450


Survival Function
beta: 2.97
eta: 203.3
probability of survival
hours
1.0
0.8
0.6
0.4
0.2
0.0
0
100
200
300
400


Failure Rate
beta: 2.97
eta: 203.3
failures/hour
hours
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16
0
50
100
150
200
250
300
350
400

CHAPTER 7

Introduction to Reliability Analysis

7.1 Weibull Distribution

In reliability analysis and, thus, in the `weibull` package, we are primarily concerned with the 2-parameter Weibull probability density function defined herein as:

$$F(x) = \frac{\beta}{\eta}\left(\frac{x}{\eta}\right)^{\beta-1} e^{-(x/\eta)^{\beta}}$$

where:

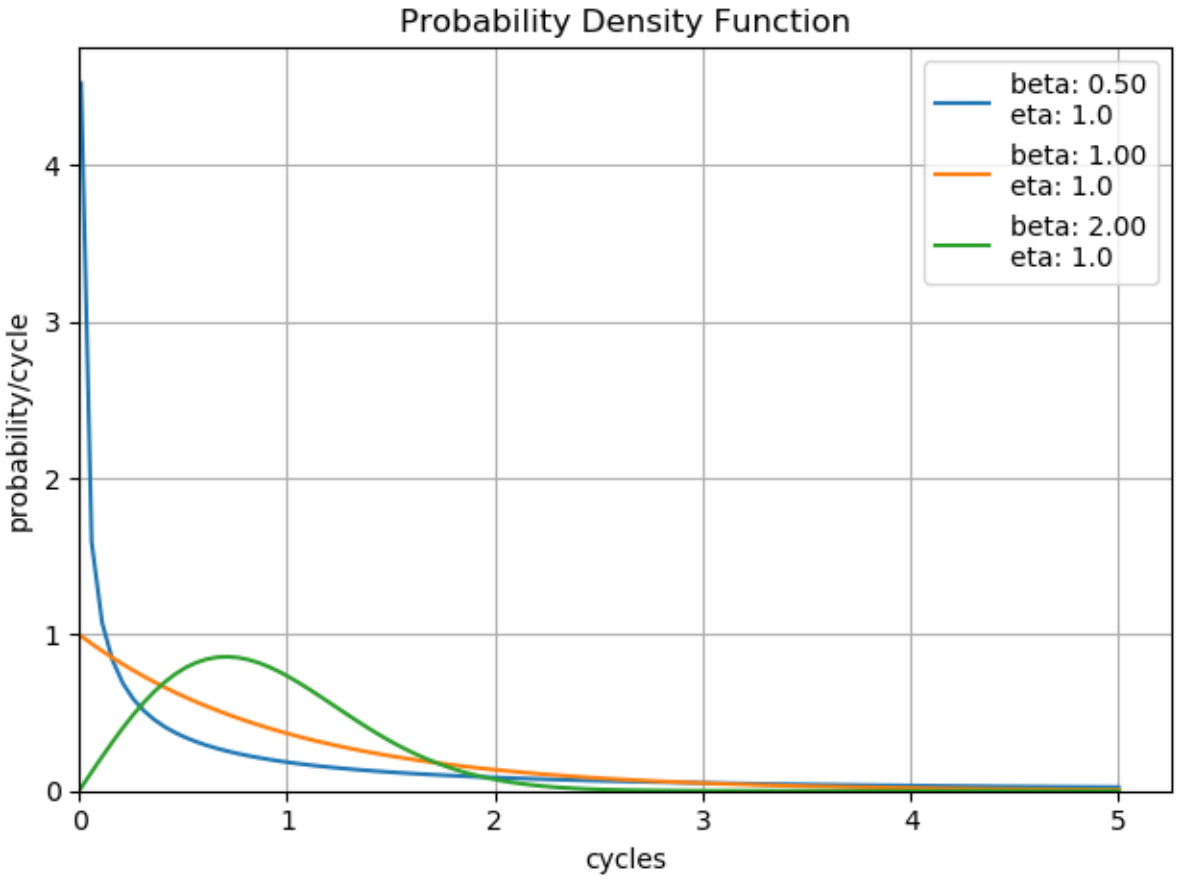
- β or *beta* represents the **shape** parameter
- η or *eta* represents the **scale** parameter
- x represents the value at which the function is to be evaluated

Were one to plot the above $F(x)$ with given β and η values, one would get the probability density function, commonly shortened to PDF. From the PDF alone, it is possible to derive the cumulative distribution function (a.k.a CDF and hazard functions), along wih the survival function which is very useful in reliability engineering.

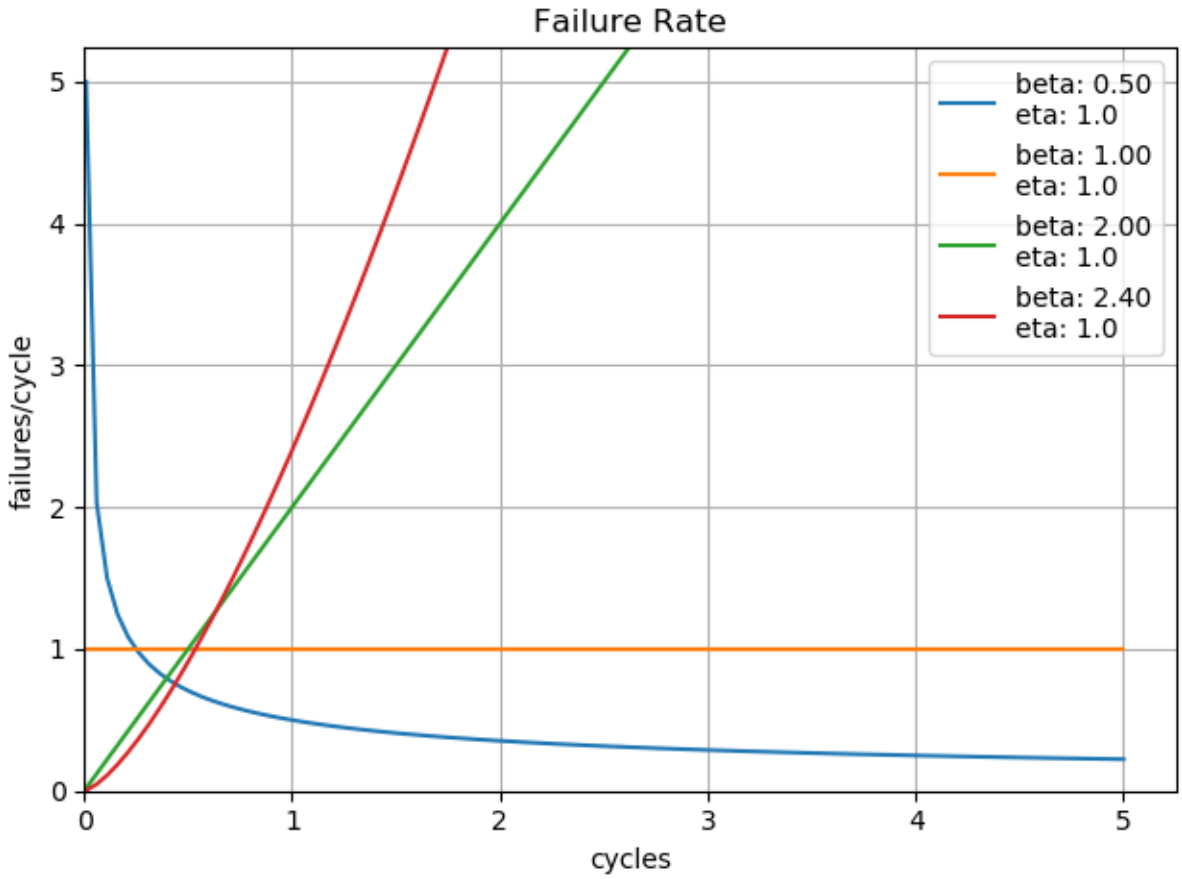
7.1.1 Distribution Shape

The **shape** parameter, β, determines the overall shape of the distribution. There are three primary regions in which β may fall:

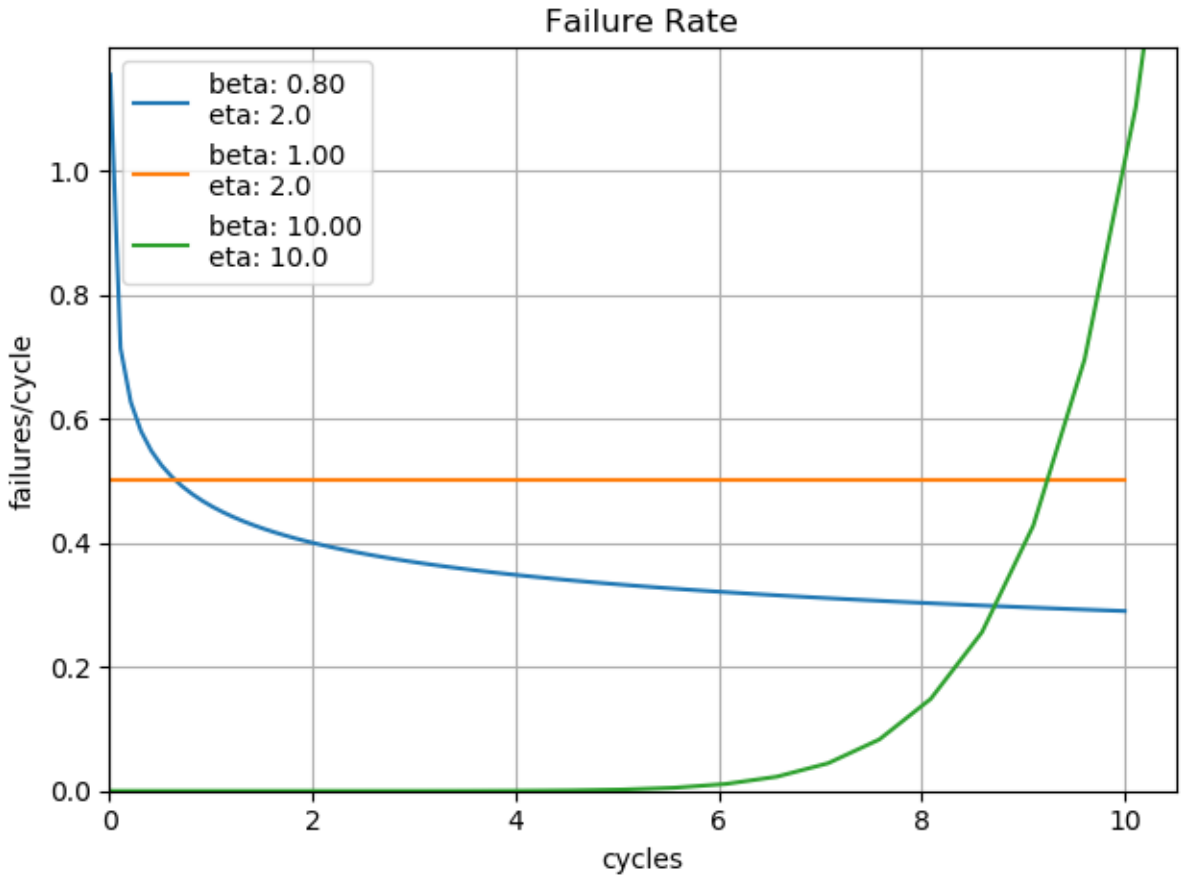
- $\beta < 1.0$ Indicates infant mortality, or decreasing failures as time increases. This is a distribution that may be observed when a phenomenon such as adhesive curing exists. As the adhesive cures, the product experiences fewer failures.
- $\beta = 1.0$ Indicates 'random' or 'constant' failures. This sort of distribution is most commonly applied to some electronic component categories, such as semiconductors.
- $\beta > 1.0$ Indicates a wearout style of distribution. This distribution is commonly observed on elements such as bearings which will increase their failure rate as wear increases.



It is possible for a product to exhibit all three of these characteristics on different components. Imagine a car which has adhesives, electronics, and bearings, each of which have their own failure distributions. With careful analysis of the failure modes, it is possible to determine the individual component failure distributions, which allows the designer to potentially identify and resolve the most problematic elements of the design first.

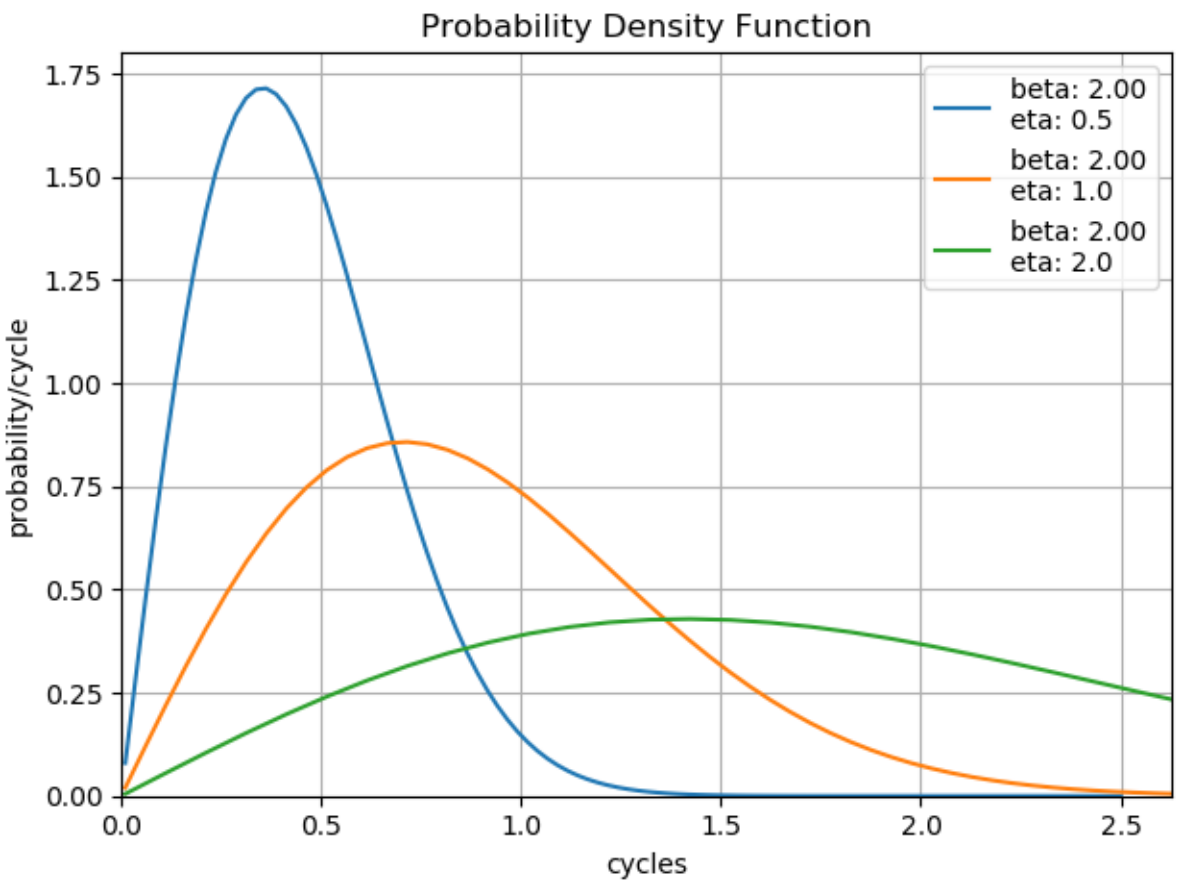


It is the combination of these three potentially different β characteristics that gives rise to the traditional bathtub curve commonly observed in reliability analysis. Looking closely at the plot of failure rates, one can see that a product composed of three different β characteristic components could begin to form an overall bathtub curve of the composite failure rates.



7.1.2 Distribution Scale

The **scale** parameter, η, determines the scale of the distribution. This parameter is also known as the *characteristic life* of the product and corresponds to the cycles at which 63% of the product has failed.



Note that changes in the scale factor keep the shape, but effectively change the length over which the distribution is 'stretched'. The hange in height is due to the property that the area under the curve must always be equal to 1.0.

7.2 Censoring Data

For the reliability engineer, there are three possible scenarios for a batch of units:

1. All units are running with no failures
2. Some units have failed, but not all OR test was stopped before all units failed (right-censored)
3. All units have failed (not censored, or 'complete' data)

The Weibull distribution can handle scenarios 2 and 3 well, but failures are inherently part of the equation. If there are no failures, the best that can be expected is a lower estimate for the life.

7.3 Determining β and η

The first task of the reliability engineer is to determine suitable values for β and η. The two most common options are:

- plot the points on Weibull paper, approximate a line, determine the slope and characteristic life
- maximum likelihood estimate (MLE)

In general, if only a small number of data points are available, then the approximation using Weibull paper or its equivalent will generate good results. If there is a relatively large body of data available, then MLE methods are preferred. Both methods will generally give acceptable results, assuming that the Weibull distribution adequately describes the process.

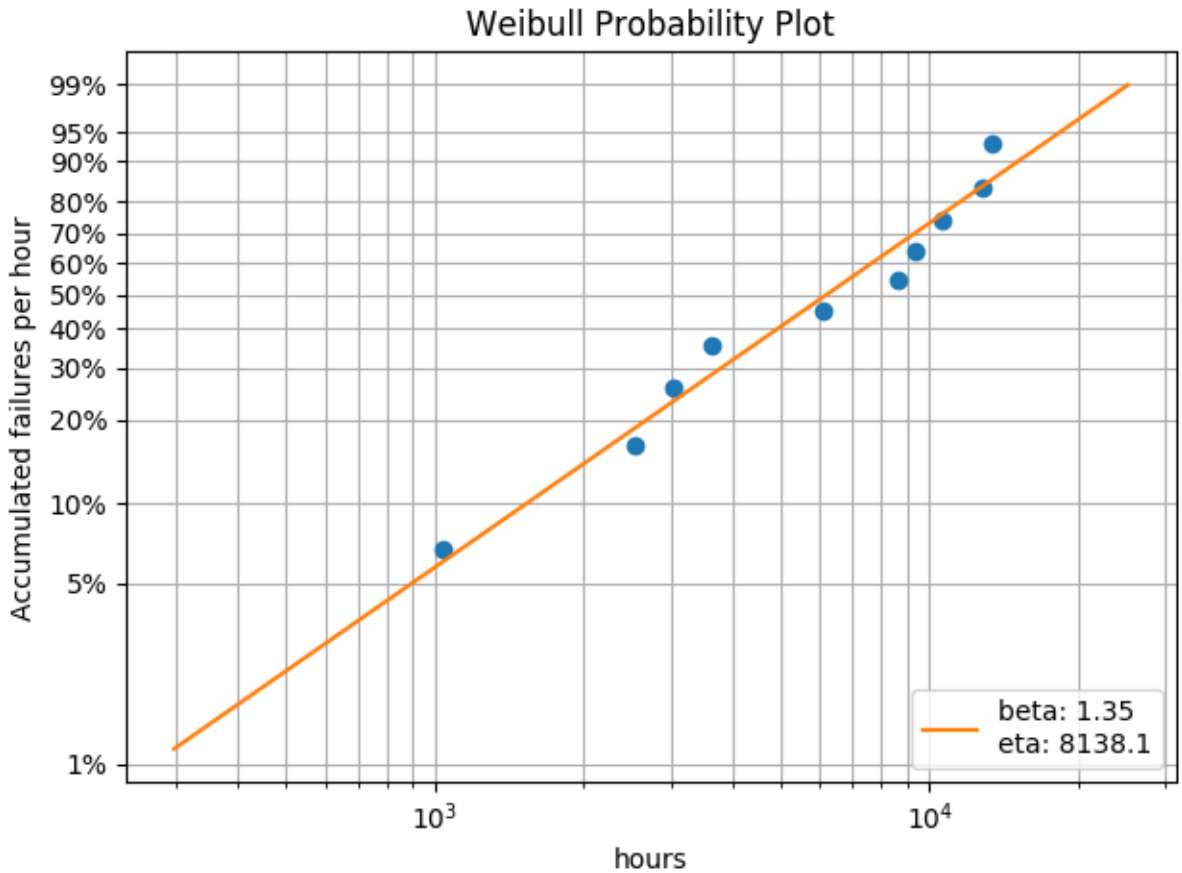
Currently, this package uses the linear regression (LR) and maximum likelihood estimation (MLE) techniques for estimating β and η. Generally, MLE is preferred for larger data sets while LR is preferred for smaller data sets (less than 15). Examination of the fit is still essential in both methods to determine if the model actually fits the data appropriately.

7.4 B-Life

A common and effective method for specifying the life of a product is to specify the time when a certain portion of the units are expected to experience failure. This is often called the B-life of a product where B is followed by the number which specifies the percent of units failed. For instance, if one were to say that the B10 life of a product were 100 hours, then it would be expected that 10% of all units will have failed at 100 hours.

7.5 Confidence Levels

Now that you have some β and η values, what does that mean? How certain are you that β and η are what you have calculated? After all, there is almost certainly some level of variation in the data. Enter `confidence limits`. In short, most confidence limits on statistical data will assume a normal distribution to the right or the left of the curve. So, for instance, if you are looking at a probability plot, you assume that most of the points will be close to the line while some smaller number of points will be further from the line. The distance from the line will fall in a normal distribution straddling the line.



This package integrates confidence limits on the calculation of β and η such that the confidence limits define the range of certainty for β and η independently. For instance, if the confidence limit is 0.95, then our calculation is 95% certain that the true β falls between β_{lower} and β_{upper}. In addition, our calculation is 95% certain that the true η falls between η_{lower} and η_{upper}. The confidence limits are generally calculated when the curve is fitted.

The best way to increase the confidence limits is to gather more data. As more data is gathered, then the distributions of β and η are better defined.

CHAPTER 8

Introduction

The `weibull` package is a package designed for easy reliability analysis using the weibull distribution. This documentation will not make a high effort to explain Weibull analysis but will, instead, focus on the use of the package. Despite this, there are still some obligatory convention to establish in order to make the documentation and packaging readable. Conventions are established in *Introduction to Reliability Analysis*.

CHAPTER 9

Installation

To install weibull into a Python 3 environment, simply `pip3 install weibull`. This package does *not* support Python 2.

9.1 Dependencies

The `weibull` package is built on `pandas`, `numpy`, `matplotlib`, and `scipy` libraries. If you are having trouble installing these libraries, particularly within windows, then you may wish to use the Anaconda distribution of Python.

CHAPTER 10

Test Coverage

The `weibull` package contains testing for the algorithmic elements of the analysis, but no testing for graphic generation.

CHAPTER 11

Classes

There are three primary classes available in `weibull`:

- `Analysis` - Used for analysis, you are probably wanting this
- `Design` - Used for reliability test design
- `Weibayes` - Used for analysis and design using some assumptions

genindex

Index

A

Analysis (class in weibull), 6

B

b() (weibull.Analysis method), 7
b() (weibull.Weibayes method), 13

C

cdf() (weibull.Analysis method), 7
characteristic_life (weibull.Analysis attribute), 7

D

Design (class in weibull), 11

F

fit() (weibull.Analysis method), 7
fr() (weibull.Analysis method), 7

H

hazard() (weibull.Analysis method), 8

M

mean (weibull.Analysis attribute), 8
median (weibull.Analysis attribute), 8
mttf (weibull.Analysis attribute), 8

N

num_of_cycles() (weibull.Design method), 12
num_of_units() (weibull.Design method), 12

P

pdf() (weibull.Analysis method), 8
plot() (weibull.Weibayes method), 13
probplot() (weibull.Analysis method), 8

S

sf() (weibull.Analysis method), 8
stats (weibull.Analysis attribute), 9

W

Weibayes (class in weibull), 13