Analysis of Repairable Systems with mcotear

Jake Warren

8/22/2019

Abstract

This document is intended to introduce the use of the `mcotear` package for analysis of data on repairable systems.

Getting started with `mcotear`

Assuming the `mcotear` package is installed on your computer, the first step would be to load the package. The package can be loaded using the `library` function as shown below. There are many ways you can learn more about the package, one is through listing all the functions in the package by using the command `ls("package:mcotear")`. For all the functions in the package you can visit the help package using the help. An example would be running the command `?ttt` (or equivalently `help(ttt)`) to see the help page/file for the `ttt` (Total Time on Test) function.

Additional, most, if not all of the functions in the package are documented with examples. If a function has an example you can see that example documented at the bottom of the function's help page. There you would see code that would execute the function. If you want to see what is actually produced by the example you could copy that code and run it in your R session, or you could run the example without having to copy/paste or retype the whole example by using the `example` function. For example, running the command `example(ttt)` in your R session would run the example documented for the `ttt` function.

Lastly, in the package I documented some of the references I used in order to create this package. You can see/access those references using the command `browseVignettes("mcotear")`.

During the loading of the `mcotear` package via the `library` function you should see via a message returned that the `ggplot2` package is loaded. In the last part of the following code block I set a `ggplot2` black and white color theme for use throughout this paper.

```
library(mcotear)
## Loading required package: ggplot2
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
## Loading required package: gridExtra
## Loading required package: goftest
# Loads the package

#ls("package:mcotear")
# Lists all of the functions/objects in the package.

#?ttt
#example(ttt)


#browseVignettes("mcotear")
# browseVignettes shows you some of the package
# documentation. I included some of the papers
# I used to create these functions within the
```

```
# documentation.

theme_set(theme_bw())
# Sets ggplot2 color theme
```

Objects in mcotear

In the mcotear package that are few objects that get loaded for use along with the functions. The objects that are loaded are amsaa, cbPalette, and cbbPalette. The first object amsaa is a data set that came from one of the references documented in package and accessible via browseVignettes("mcotear"). The data set contains simulated data consisting of failure times for three repairable systems (you can see this in the help documentation for amsaa using ?amsaa).

```
amsaa
##    Failure System  Time
## 1        1     S1   4.3
## 2        2     S1   4.4
## 3        3     S1  10.2
## 4        4     S1  23.5
## 5        5     S1  23.8
## 6        6     S1  26.4
## 7        7     S1  74.0
## 8        8     S1  77.1
## 9        9     S1  92.1
## 10      10     S1 197.2
## 11       1     S2   0.1
## 12       2     S2   5.6
## 13       3     S2  18.6
## 14       4     S2  19.5
## 15       5     S2  24.2
## 16       6     S2  26.7
## 17       7     S2  45.1
## 18       8     S2  45.8
## 19       9     S2  75.7
## 20      10     S2  79.7
## 21      11     S2  98.6
## 22      12     S2 120.1
## 23      13     S2 161.8
## 24      14     S2 180.6
## 25      15     S2 190.8
## 26       1     S3   8.4
## 27       2     S3  32.5
## 28       3     S3  44.7
## 29       4     S3  48.4
## 30       5     S3  50.6
## 31       6     S3  73.6
## 32       7     S3  98.7
## 33       8     S3 112.2
## 34       9     S3 129.8
## 35      10     S3 136.0
## 36      11     S3 195.8
```

The other objects cbPalette, and cbbPalette are hexadecimal color code values for color-blind friendly color palettes. These color-blind friendly color palettes come from the "Cookbook for R", at http://www.

cookbook-r.com/Graphs/Colors_(ggplot2)/. Again, this is documented in the `cbPalette`, and `cbbPalette` help pages/files.

```
cbPalette
## [1] "#999999" "#E69F00" "#56B4E9" "#009E73" "#F0E442" "#0072B2" "#D55E00"
## [8] "#CC79A7"
cbbPalette
## [1] "#000000" "#E69F00" "#56B4E9" "#009E73" "#F0E442" "#0072B2" "#D55E00"
## [8] "#CC79A7"
```

Repairable Systems Analysis Example

Now that the package has been introduced a little this section will provide an example of analysis of the repairable systems using the `amsaa` data set. The full `amsaa` data set was printed above. Below I print just a few rows of the data set, and the structure of the data set. We see that we have three columns giving the failure number ("Failure"), system identification ("System"), and time of the failure ("Time").

```
head(amsaa)
##   Failure System Time
## 1       1     S1  4.3
## 2       2     S1  4.4
## 3       3     S1 10.2
## 4       4     S1 23.5
## 5       5     S1 23.8
## 6       6     S1 26.4
str(amsaa)
## 'data.frame':    36 obs. of  3 variables:
##  $ Failure: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ System : Factor w/ 3 levels "S1","S2","S3": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Time   : num  4.3 4.4 10.2 23.5 23.8 ...
```

Several of the functions we will be using below accept arguments in the same manner; that is many of the functions have a common argument "t" that should be provided in the form of a list. The argument "t" is a list of vectors where each vector represents the failure times for one specific system. Since this same "t" argument will be used multiple times it can be constructed and saved as an object to eliminate having to retype the whole argument each time. The "t" object in the following code block will be used for the "t" argument in functions later on. The "t" object is a list of the failure times per system.

Many of the functions also have a common argument "T", also a list, that indicates the total cumulative time a system was on test. Reliability tests can be **time truncated** or **failure truncated**. Time truncated means that systems are pulled out of test at a given time, that time not being associated to a specific failure. Failure truncated mean that systems are pulled out of at a given failure, that is the time at which the last failure occurs is the same as the test end time. In the following code block two "T" objects are created, one of which corresponds to time truncation, the other corresponds to failure truncation. These "T" objects will be used for the "T" arguments in functions later on.

```
# create the object t which be input to several of the functions later on
# t is a list of failure times separated by system
(t <- split(amsaa$Time, amsaa$System))
## $S1
##  [1]   4.3   4.4  10.2  23.5  23.8  26.4  74.0  77.1  92.1 197.2
##
## $S2
##  [1]   0.1   5.6  18.6  19.5  24.2  26.7  45.1  45.8  75.7  79.7  98.6
## [12] 120.1 161.8 180.6 190.8
##
```

```
## $S3
##  [1]   8.4  32.5  44.7  48.4  50.6  73.6  98.7 112.2 129.8 136.0 195.8

# create the T objects which be input to several of the functions later on
# the T list objects that represent the total time a system was on test
# T_time_trunc indicates all systems were time truncted, on test for 200 hours
# (or whatever units)
(T_time_trunc <- list("S1" = 200, "S2" = 200, "S3" = 200))
## $S1
## [1] 200
##
## $S2
## [1] 200
##
## $S3
## [1] 200

# T_fail_trunc indicates all systems were failure truncted, on test until
# their last failure
(T_fail_trunc <- lapply(split(amsaa$Time, amsaa$System), max))
## $S1
## [1] 197.2
##
## $S2
## [1] 190.8
##
## $S3
## [1] 195.8
```

Plots to determine failure trends

The first thing we want to know when analyzing data from repairable systems is whether there is a **trend** in the times between failure. In other words, we want to know if the times between failure are getting longer, shorter, or staying constant. We can inspect this first using graphs. There are a few different types of plots that can help with this inspection such as the **Duane Plot**, **Total Time on Test** (TTT) plot, and a plot of the time between failures versus time or failure number.

Duane Plots

In the package I did not specifically include a function to generate a Duane Plot, but given the functions I created it is easy enough to create a Duane Plot. A Duane Plot is a plot of the **cumulative failure rate** $\left(\frac{N(t_i)}{t_i}\right)$ versus the time of failure (t_i), where t_i represents the time of the ith failure, and $N(t_i)$ represents the number of failures that have occurred up to time t_i. (Some implementations of the Duane Plot suggest inverting the value of the y-axis such that $\frac{t_i}{N(t_i)}$, the cumulative mean time between failure, is plotting instead of the cumulative failure rate.) The values plotted on a Duane Plot are plotted on a log-log scale. When points fall along a straight line this gives an informal graphical indication that a **power-law process** is a good model for the data. If the points follow an increasing or decreasing slope, or follow a constant zero-slope line this gives an indication of whether the failure rate is increasing or decreasing (this depends on whether you plot $\frac{N(t_i)}{t_i}$ or $\frac{t_i}{N(t_i)}$), or staying constant.

The `rocof` function, which stands for the **rate of occurrence of failures** (ROCOF), can be used to help generate a Duane Plot. The `rocof` function returns the cumulative **mean time between failure** (MTBF), and cumulative failure rate (aka ROCOF) as a data frame given the inputs of the failure times.

Assuming systems are identical

Below `rocof` function is used under the assumption that the systems are identical. If we do not want to make that assumption, or we want to visually inspect this assumption we need to compute the ROCOF per system, which will be done in the next section.

```
rocof_identical <- rocof(t = t, by = NULL)
head(rocof_identical)
##         t     mtbf      rocof
## S21   0.1 0.100000 10.0000000
## S11   4.3 2.150000  0.4651163
## S12   4.4 1.466667  0.6818182
## S22   5.6 1.400000  0.7142857
## S31   8.4 1.680000  0.5952381
## S13  10.2 1.700000  0.5882353
names(rocof_identical)[1] <- "Time"
head(rocof_identical)
##      Time     mtbf      rocof
## S21   0.1 0.100000 10.0000000
## S11   4.3 2.150000  0.4651163
## S12   4.4 1.466667  0.6818182
## S22   5.6 1.400000  0.7142857
## S31   8.4 1.680000  0.5952381
## S13  10.2 1.700000  0.5882353
# Get the nonparametric rocof and mtbf estimates.
# I show the first 6 rows so you can see
# what it looks like, and you can see the
# column names. Then I change the name
# of "t" to "Time" so it matches the name
# in the amsaa data set. The rocof and mtbf
# are inverses of one another. You can use
# either of these to create a Duane Plot.
```

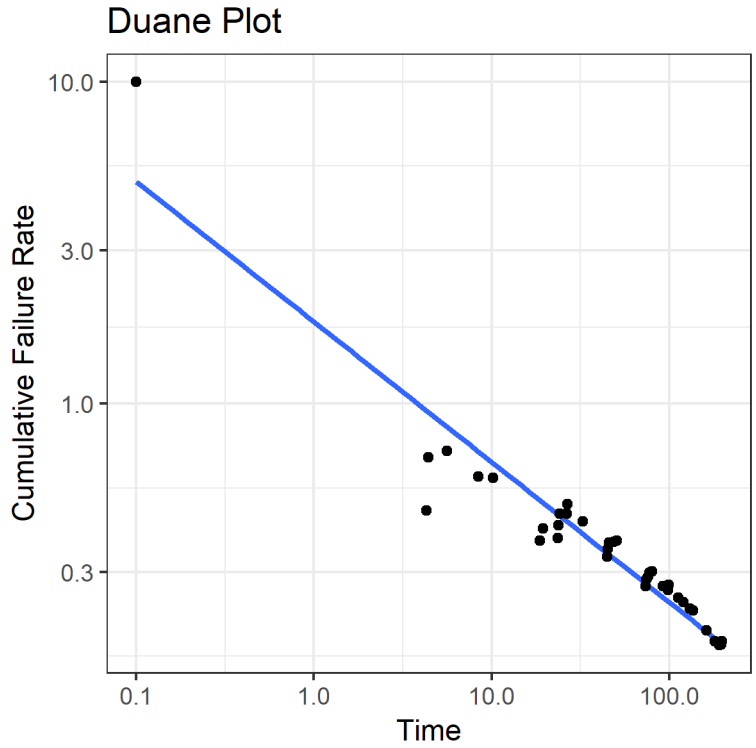
One I have the output from the `rocof` function I can associate that information to the original data set, merging the two together, using the `merge` function. When merging, you need to ensure there is a common link between the two data frames, which in this case is our "Time" variable.

```
amsaa_identical <- merge(amsaa, rocof_identical, by = "Time")
head(amsaa_identical)
##   Time Failure System     mtbf      rocof
## 1  0.1       1     S2 0.100000 10.0000000
## 2  4.3       1     S1 2.150000  0.4651163
## 3  4.4       2     S1 1.466667  0.6818182
## 4  5.6       2     S2 1.400000  0.7142857
## 5  8.4       1     S3 1.680000  0.5952381
## 6 10.2       3     S1 1.700000  0.5882353
# Merge our data set with the df_rocof by the Time variable
```

Now I can create the Duane Plot as follows. In the following plot uses the "rocof" ($\frac{N(t_i)}{t_i}$) on the y-axis. Since the points appear to follow along a straight line we get an indication that a power law process may be a good model. We also see that the ROCOF is decreasing indicating that the systems are improving over time.

```
ggplot(amsaa_identical,
  aes(
    x = Time,
    y = rocof)) +
```

```
  scale_x_log10() +
  scale_y_log10() +
  geom_smooth(method='lm', se = FALSE) +
  geom_point() +
  labs(y = "Cumulative Failure Rate") +
  scale_colour_manual(values = cbPalette) +
  ggtitle("Duane Plot")
```

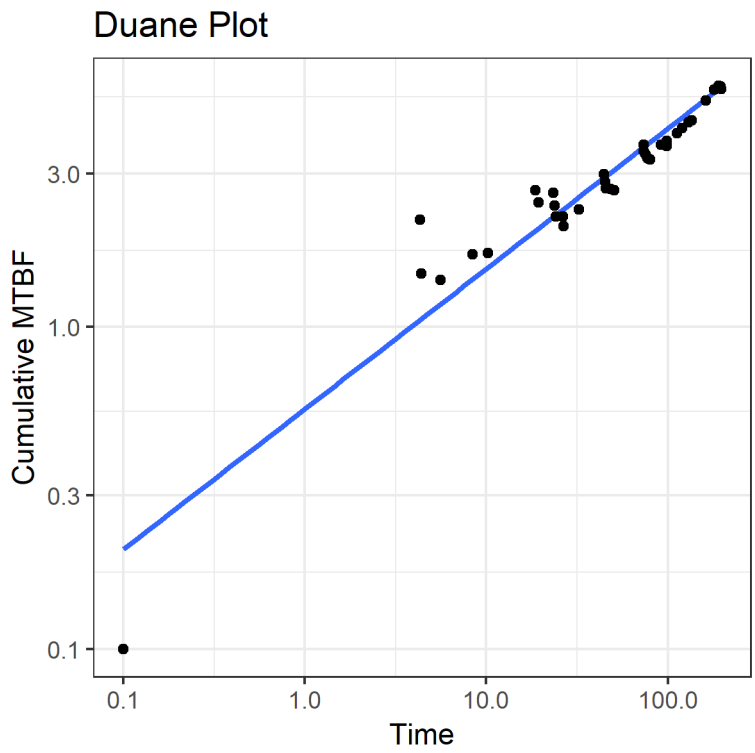



```
# Create a Duane Plot. A Duane Plot is a
# tool to visually assess if the data appears to follow
# a Power NHPP by assessing if the data appears to follow
# a straight line when plotted on a log-log scale.
# If the data followed a horizontal straight line
# that is an indication of a HPP rather than a NHPP.
# Here since I plotted the rocof vs time, you can
# see that it appears as though the rocof is decreasing,
# or in other words the failure rate is decreasing, or
# the time between failures is increasing.
```

Alternatively I could plot the inverse of "rocof", the "mtbf" on the y-axis. The following plot uses the "mtbf" $\left(\frac{t_i}{N(t_i)}\right)$ on the y-axis. Since the MTBF is increasing this indicates that the system(s) are improving over time.

```
ggplot(amsaa_identical,
  aes(
    x = Time,
    y = mtbf)) +
  scale_x_log10() +
  scale_y_log10() +
```

```
  geom_smooth(method='lm', se = FALSE) +
  geom_point() +
  labs(y = "Cumulative MTBF") +
  scale_colour_manual(values = cbPalette) +
  ggtitle("Duane Plot")
```




```
# This is the same Duane Plot, just now using the
# mtbf instead of the rocof. These are equivalent,
# and lead to the same conclusions.
```

Inspecting the assumption that systems are identical

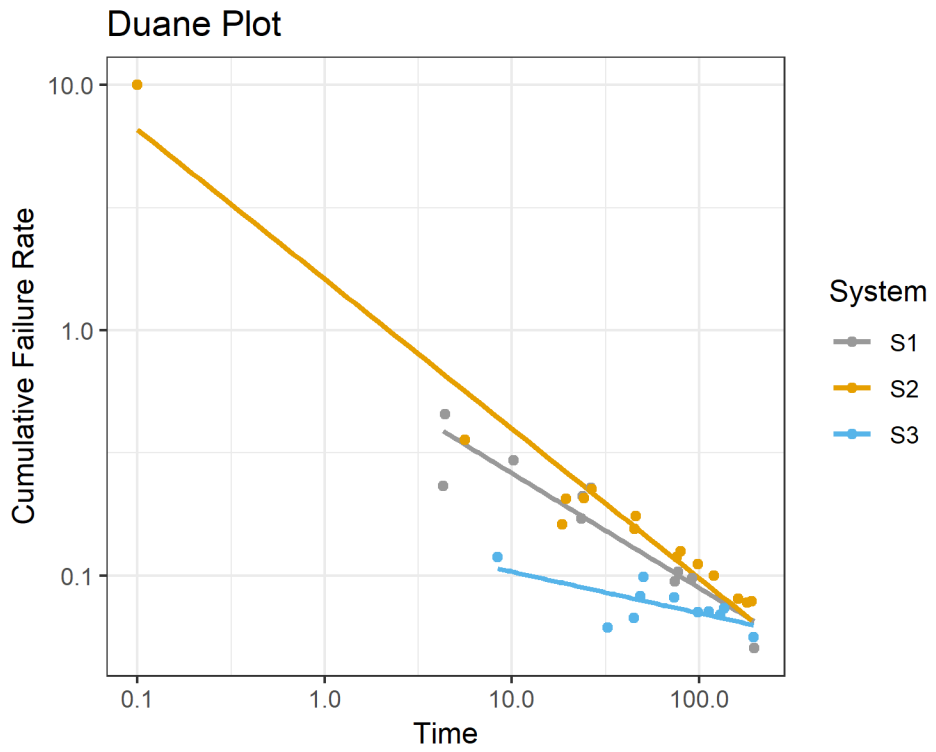
The following use of the `rocof` function differs from how it was used earlier only in that now the "by" argument is not `NULL`, but rather a vector of names taken from the elements of our object "t" (the list of failure times separated by system). Supplying the "by" argument like this allows for visual inspection of the trend along with the assumption that the systems are identical.

```
rocof_ni <- rocof(t = t, by = names(t))
head(rocof_ni)
##   by    t  mtbf     rocof
## 1 S1  4.3 4.300 0.2325581
## 2 S1  4.4 2.200 0.4545455
## 3 S1 10.2 3.400 0.2941176
## 4 S1 23.5 5.875 0.1702128
## 5 S1 23.8 4.760 0.2100840
## 6 S1 26.4 4.400 0.2272727
names(rocof_ni)[1:2] <- c("System", "Time")
amsaa_ni <- merge(amsaa, rocof_ni, by = c("Time", "System"))
head(amsaa_ni)
##    Time System Failure     mtbf       rocof
```

```
## 1   0.1     S2       1  0.10000 10.00000000
## 2  10.2     S1       3  3.40000  0.29411765
## 3 112.2     S3       8 14.02500  0.07130125
## 4 120.1     S2      12 10.00833  0.09991674
## 5 129.8     S3       9 14.42222  0.06933744
## 6 136.0     S3      10 13.60000  0.07352941
# The rocof and mtbf estimates we got before were
# based on grouping all systems together.
# Instead I could do it by System, and create
# a Duane plot with the Systems separated to
# see if they all seem to follow a similar pattern.
# This would be a visual assessment of a common beta.
```

Now, with the ROCOF calculated per system, rather than having all of the system grouped together, the Duane Plot can be created again, but this time the "colour" argument can be used to indicate the trend per system. From this it shows that all three systems are experiencing decreasing failure rates (i.e. the systems are improving), but the differences in slopes tells us it's possible that although they are all decreasing, they may not be close enough to assume they are identical. A statistical test can be performed to formally test this. That will be done later on. (Note that here only the Duane Plot with the ROCOF on the y-axis was plotted, but we could have plotted the MTBF on the y-axis instead.)

```
ggplot(amsaa_ni,
  aes(
    x = Time,
    y = rocof,
    colour = System)) +
  scale_x_log10() +
  scale_y_log10() +
  geom_smooth(method='lm', se = FALSE) +
  geom_point() +
  labs(y = "Cumulative Failure Rate") +
  scale_colour_manual(values = cbPalette) +
  ggtitle("Duane Plot")
```




```
# This is a Duane Plot by system. We see that
# the slopes are not exactly equal, but they
# are all indicating a decreasing failure rate.
# The common_beta test/function already told us
# that there was no statistical difference between
# the systems' failure rates.
```

Total Time on Test Plots

Similar to the Duane Plot the package does not include a function that exactly creates the TTT plot; however, the `ttt` function can help to create the TTT plot just as the `rocof` function helped to create the Duane Plot.

The TTT plot is a plot of the scaled TTT versus $\frac{i}{n}$, where i is the failure number and n is the total number of failures (i.e. $1, 2, 3, ..., n$). Details of the scaled TTT statistic can be found in the package documentation (see `browseVignettes("mcotear")` and the PDF named *Tests for trend in more than on repairable system*).

Assuming systems are identical

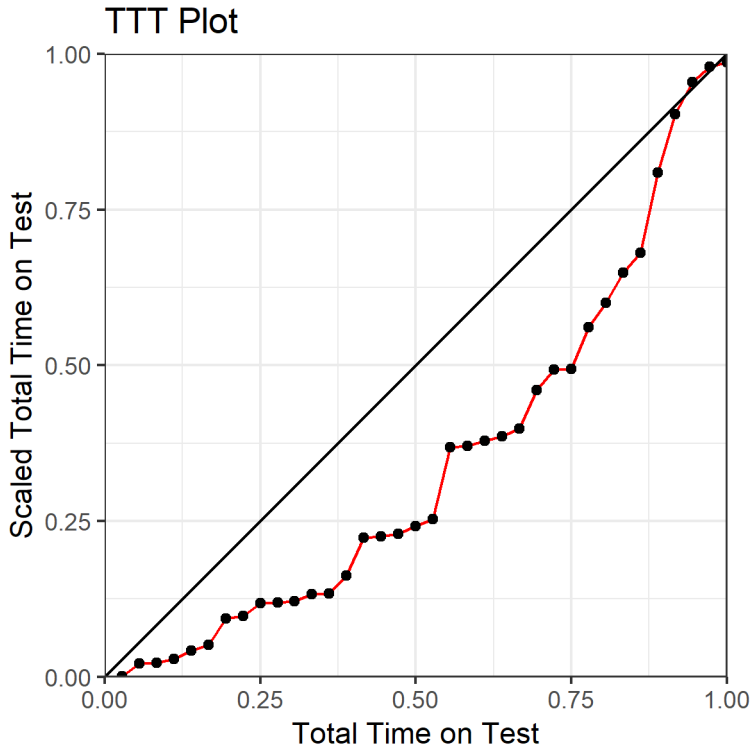
Just as with the Duane Plot and the `rocof` function we can assume that the systems are identical or account/inspect for them being different. First we start with the assumption that the systems are identical.

```
ttt_identical <- ttt(
  t = t,
  T = T_time_trunc)
head(ttt_identical)
##        t        ttt scaled_ttt
## S21  0.1 0.02777778     0.0005
## S11  4.3 0.05555556     0.0215
## S12  4.4 0.08333333     0.0220
## S22  5.6 0.11111111     0.0280
## S31  8.4 0.13888889     0.0420
```

```
## S13 10.2 0.16666667     0.0510
tail(ttt_identical)
##           t       ttt scaled_ttt
## S310 136.0 0.8611111       0.680
## S213 161.8 0.8888889       0.809
## S214 180.6 0.9166667       0.903
## S215 190.8 0.9444444       0.954
## S311 195.8 0.9722222       0.979
## S110 197.2 1.0000000       0.986
# This ttt function is for the total time on test (TTT).
# The TTT is used in certain trend tests (the
# Military Handbook Test, and Laplace Centroid Tests
# both have TTT versions, see the trend_test function).
# The TTT can also be used in plots to visually assess
# if the data follow a Power Law Process.
```

The following code block creates the TTT plot. When the points fall along the diagonal of the unit square this gives an informal graphical indication that the failure rate is constant. Points giving a concave up pattern (as we see below) indicate the failure rate is decreasing (i.e. improving), and points giving a concave down pattern indicate the failure rate is increasing (i.e. deteriorating).

```
ggplot(ttt_identical, aes(x = ttt, y = scaled_ttt)) +
   geom_line(colour = "red") + geom_point() +
   geom_abline(intercept = 0, slope = 1) +
   labs(
     x = "Total Time on Test",
     y = "Scaled Total Time on Test") +
   scale_x_continuous(limits = c(0, 1), expand = c(0, 0))  +
   scale_y_continuous(limits = c(0, 1), expand = c(0, 0)) +
   theme(plot.margin = margin(t = 10, r = 10, unit = "pt")) +
   coord_fixed() +
   ggtitle("TTT Plot")
```




```
# A power-law process is appropriate if the TTT plot lies
# close to the diagonal or is a curve that is either concave
# up or concave down. If there is no pattern, or a curve that
# shifts between being concave up and concave down, the
# power-law process is inadequate. We seem to have a pattern
# with a concave up curve. Concave up indicates decreasing
# failure rate. Concave down would indicate increasing
# failure rate, and no curve (i.e. following the straight line)
# would indicate a HPP.
```

Inspecting the assumption that systems are identical

The following differs from the previous section in that we now account for systems being different. In order to do that I use a few functions from other R packages to assist in passing each element of objects "t" and "T_time_trunc" to the `ttt` function separately, and then combine the results in a single data frame.

```
(ttt_ni <- purrr::map2(t, T_time_trunc, ttt))
## $S1
##        t ttt scaled_ttt
## 1    4.3 0.1     0.0215
## 2    4.4 0.2     0.0220
## 3   10.2 0.3     0.0510
## 4   23.5 0.4     0.1175
## 5   23.8 0.5     0.1190
## 6   26.4 0.6     0.1320
## 7   74.0 0.7     0.3700
## 8   77.1 0.8     0.3855
## 9   92.1 0.9     0.4605
## 10 197.2 1.0     0.9860
##
```

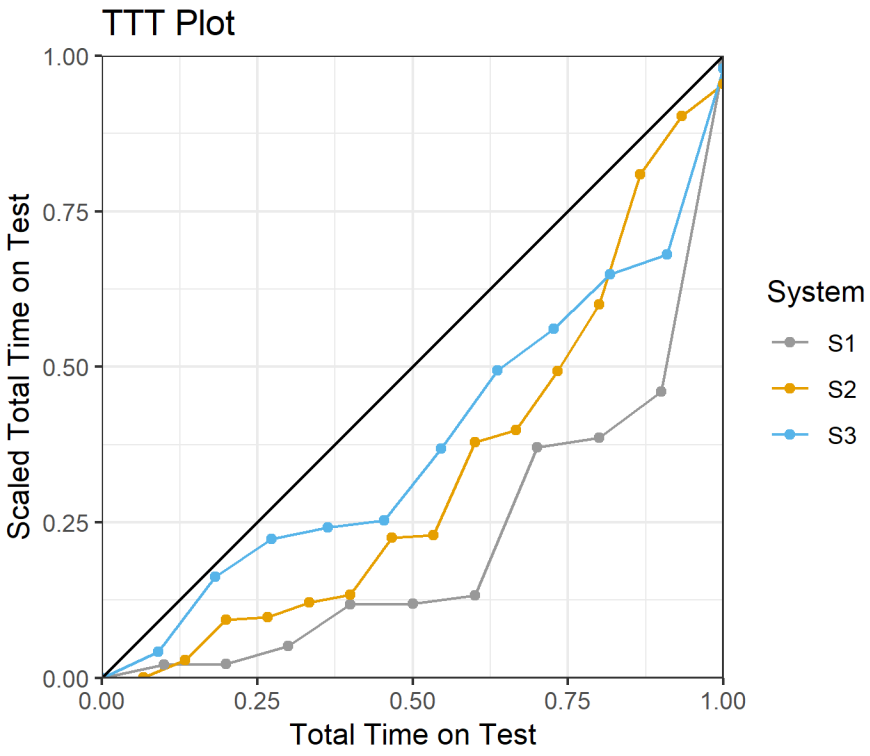
```
## $S2
##        t        ttt scaled_ttt
## 1    0.1 0.06666667     0.0005
## 2    5.6 0.13333333     0.0280
## 3   18.6 0.20000000     0.0930
## 4   19.5 0.26666667     0.0975
## 5   24.2 0.33333333     0.1210
## 6   26.7 0.40000000     0.1335
## 7   45.1 0.46666667     0.2255
## 8   45.8 0.53333333     0.2290
## 9   75.7 0.60000000     0.3785
## 10  79.7 0.66666667     0.3985
## 11  98.6 0.73333333     0.4930
## 12 120.1 0.80000000     0.6005
## 13 161.8 0.86666667     0.8090
## 14 180.6 0.93333333     0.9030
## 15 190.8 1.00000000     0.9540
##
## $S3
##        t        ttt scaled_ttt
## 1    8.4 0.09090909     0.0420
## 2   32.5 0.18181818     0.1625
## 3   44.7 0.27272727     0.2235
## 4   48.4 0.36363636     0.2420
## 5   50.6 0.45454545     0.2530
## 6   73.6 0.54545455     0.3680
## 7   98.7 0.63636364     0.4935
## 8  112.2 0.72727273     0.5610
## 9  129.8 0.81818182     0.6490
## 10 136.0 0.90909091     0.6800
## 11 195.8 1.00000000     0.9790
(ttt_ni <- Map(rbind, ttt_ni, data.frame(0,0,0)))
## $S1
##        t ttt scaled_ttt
## 1    4.3 0.1     0.0215
## 2    4.4 0.2     0.0220
## 3   10.2 0.3     0.0510
## 4   23.5 0.4     0.1175
## 5   23.8 0.5     0.1190
## 6   26.4 0.6     0.1320
## 7   74.0 0.7     0.3700
## 8   77.1 0.8     0.3855
## 9   92.1 0.9     0.4605
## 10 197.2 1.0     0.9860
## 11   0.0 0.0     0.0000
##
## $S2
##        t        ttt scaled_ttt
## 1    0.1 0.06666667     0.0005
## 2    5.6 0.13333333     0.0280
## 3   18.6 0.20000000     0.0930
## 4   19.5 0.26666667     0.0975
## 5   24.2 0.33333333     0.1210
```

```
## 6   26.7 0.40000000     0.1335
## 7   45.1 0.46666667     0.2255
## 8   45.8 0.53333333     0.2290
## 9   75.7 0.60000000     0.3785
## 10  79.7 0.66666667     0.3985
## 11  98.6 0.73333333     0.4930
## 12 120.1 0.80000000     0.6005
## 13 161.8 0.86666667     0.8090
## 14 180.6 0.93333333     0.9030
## 15 190.8 1.00000000     0.9540
## 16   0.0 0.00000000     0.0000
##
## $S3
##        t        ttt scaled_ttt
## 1    8.4 0.09090909     0.0420
## 2   32.5 0.18181818     0.1625
## 3   44.7 0.27272727     0.2235
## 4   48.4 0.36363636     0.2420
## 5   50.6 0.45454545     0.2530
## 6   73.6 0.54545455     0.3680
## 7   98.7 0.63636364     0.4935
## 8  112.2 0.72727273     0.5610
## 9  129.8 0.81818182     0.6490
## 10 136.0 0.90909091     0.6800
## 11 195.8 1.00000000     0.9790
## 12   0.0 0.00000000     0.0000
(df_ttt_ni <- dplyr::bind_rows(ttt_ni, .id = "System"))
##    System     t        ttt scaled_ttt
## 1      S1   4.3 0.10000000     0.0215
## 2      S1   4.4 0.20000000     0.0220
## 3      S1  10.2 0.30000000     0.0510
## 4      S1  23.5 0.40000000     0.1175
## 5      S1  23.8 0.50000000     0.1190
## 6      S1  26.4 0.60000000     0.1320
## 7      S1  74.0 0.70000000     0.3700
## 8      S1  77.1 0.80000000     0.3855
## 9      S1  92.1 0.90000000     0.4605
## 10     S1 197.2 1.00000000     0.9860
## 11     S1   0.0 0.00000000     0.0000
## 12     S2   0.1 0.06666667     0.0005
## 13     S2   5.6 0.13333333     0.0280
## 14     S2  18.6 0.20000000     0.0930
## 15     S2  19.5 0.26666667     0.0975
## 16     S2  24.2 0.33333333     0.1210
## 17     S2  26.7 0.40000000     0.1335
## 18     S2  45.1 0.46666667     0.2255
## 19     S2  45.8 0.53333333     0.2290
## 20     S2  75.7 0.60000000     0.3785
## 21     S2  79.7 0.66666667     0.3985
## 22     S2  98.6 0.73333333     0.4930
## 23     S2 120.1 0.80000000     0.6005
## 24     S2 161.8 0.86666667     0.8090
## 25     S2 180.6 0.93333333     0.9030
```

```
## 26     S2 190.8 1.00000000     0.9540
## 27     S2   0.0 0.00000000     0.0000
## 28     S3   8.4 0.09090909     0.0420
## 29     S3  32.5 0.18181818     0.1625
## 30     S3  44.7 0.27272727     0.2235
## 31     S3  48.4 0.36363636     0.2420
## 32     S3  50.6 0.45454545     0.2530
## 33     S3  73.6 0.54545455     0.3680
## 34     S3  98.7 0.63636364     0.4935
## 35     S3 112.2 0.72727273     0.5610
## 36     S3 129.8 0.81818182     0.6490
## 37     S3 136.0 0.90909091     0.6800
## 38     S3 195.8 1.00000000     0.9790
## 39     S3   0.0 0.00000000     0.0000
#head(df_ttt_ni)
names(df_ttt_ni)[2] <- "Time"
# In the ttt function I did not include a "by"
# arguement that lets you get the TTT per
# system, for example. But, if you install
# the purrr and dplyr packages you can easily get
# the TTT per system as I show above. You could
# also just run the ttt function 3 times by only
# passing in the t and T for each system separately,
# but if you had 10 systems, 100 systems, etc. you
# probably wouldn't want to do that. You could follow
# this appoach in the other functions too (mcf and rocof),
# but the "by" variable in those was included to
# make it a litter easier. (Note that the Map, step
# I included is really not necessary, but I included
# it so the ttt plot lines would start at the origin.)
```

Below is a TTT plot by system. All three systems show a concave up pattern indicating the failure rates are decreasing (i.e. improving), although we are left with a subjective determination as to whether the systems' failure rates are by chance, or different per system.

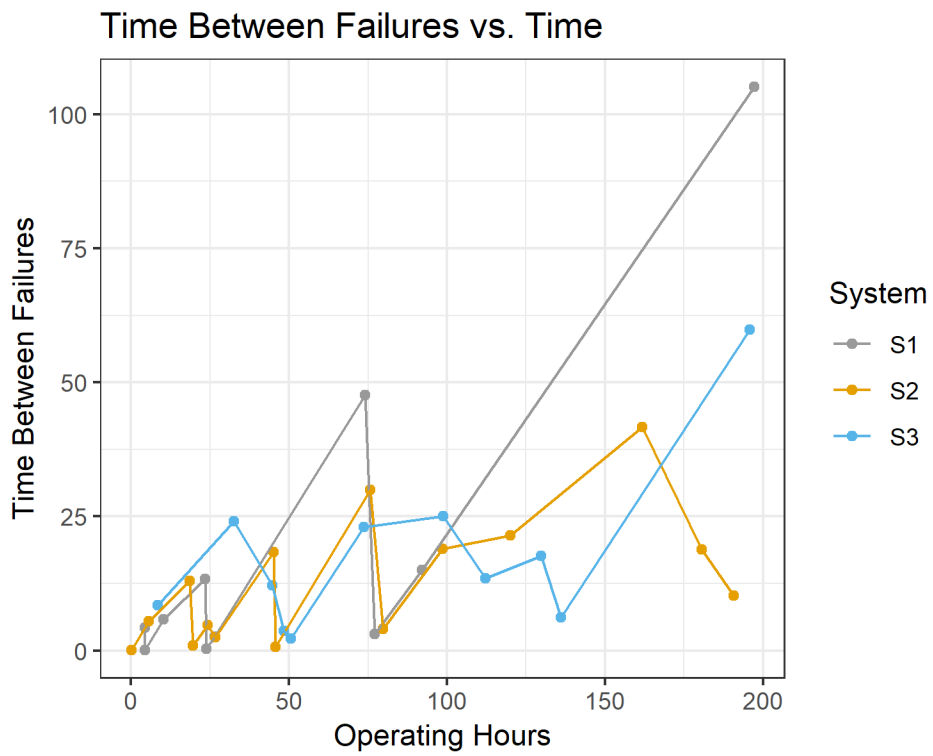
```
ggplot(df_ttt_ni, aes(x = ttt, y = scaled_ttt, colour = System)) +
   geom_line() + geom_point() +
   geom_abline(intercept = 0, slope = 1) +
   labs(
     x = "Total Time on Test",
     y = "Scaled Total Time on Test") +
   scale_x_continuous(limits = c(0, 1), expand = c(0, 0))  +
   scale_y_continuous(limits = c(0, 1), expand = c(0, 0)) +
   theme(plot.margin = margin(t = 10, r = 10, unit = "pt")) +
   coord_fixed() +
   scale_colour_manual(values = cbPalette) +
   ggtitle("TTT Plot")
```



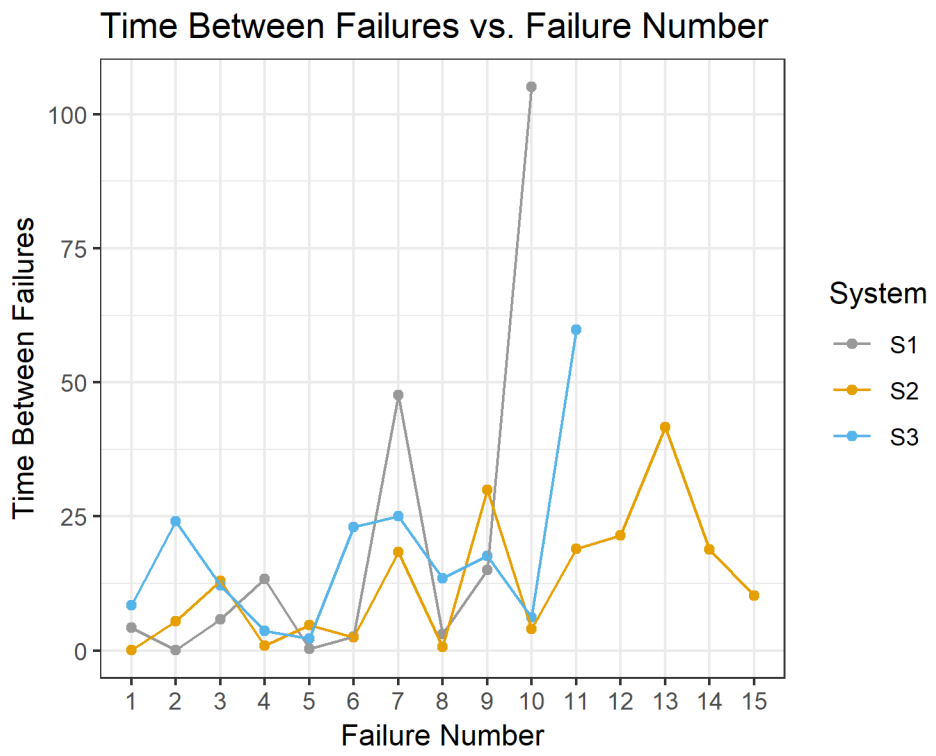
Time Between Failures versus Time of Failure Number

Plotting the time between failures versus time or failure number is another way to get a informal indication of whether the failure rate is increasing, decreasing, of remaining constant. There are no special functions required for this. Below are two plots for the: 1. the time between failures versus time, and 2. the time between failures versus failure number. Both figures appear to show that the time between failures appears to generally be increasing, indicating that the failure rate is decreasing.

```
# Plot Time Between Failures vs Time
ggplot(amsaa,
  aes(
    x = Time,
    y = ifelse(diff(c(0,amsaa$Time)) < 0, amsaa$Time, diff(c(0,amsaa$Time))),
    colour = System)) +
  geom_point() + geom_line() +
  labs(
    x = "Operating Hours",
    y = "Time Between Failures",
    main = "Time Between Failures vs Time"
  ) + scale_colour_manual(values = cbPalette) +
  ggtitle("Time Between Failures vs. Time")
```




```
# Plot Time Between Failures vs Failure Number
ggplot(amsaa,
  aes(
    x = Failure,
    y = ifelse(diff(c(0,amsaa$Time)) < 0, amsaa$Time, diff(c(0,amsaa$Time))),
    colour = System)) +
  geom_point() + geom_line() +
  labs(
    x = "Failure Number",
    y = "Time Between Failures",
    main = "Time Between Failures vs Time"
  ) + scale_colour_manual(values = cbPalette) +
  ggtitle("Time Between Failures vs. Failure Number") +
  scale_x_continuous(breaks = 1:15) +
  theme(panel.grid.minor.x = element_blank())
```



Statistical Modeling

Now that we have finished with some preliminary data visualization we may be interested in performing some specific statistical tests and fitting some models.

Test for Trend

The first test is for trend in the failure rate. This was visually inspected in the plotting above, but now we can formally test for trend to see if the failure rate is constant (indicating a Homogeneous Poisson Process that can be modeled by the Exponential distribution), or increasing or decreasing (indicating a Non-homogeneous Poisson Process). The results of the Military Handbook Test (called this due to its origin from the MIL-HDBK-189), the Laplace Test, and the Anderson-Darling Test for Trend are all printed. Details of these tests can be found in the package documentation (see `browseVignettes("mcotear")`, the PDF named *Tests for trend in more than on repairable system*, and `?trend_test`). From the results of the tests (low p-values) we would conclude that the failure rate is not constant, and that the systems' failures follow a Non-homogeneous Poisson Process (NHPP).

```
trend_test(
  t = t,
  T = T_time_trunc,
  fail.trunc = FALSE)
## $`Mil-HDBK`
##                test statistic df        pval
## 1 Mil-HDBK Combined  117.0092 72 0.001269251
## 2      Mil-HDBK TTT  117.0092 72 0.001269251
##
## $Laplace
##               test statistic        pval
## 1 Laplace Combined -3.012036 0.002595015
```

```
## 2      Laplace TTT -3.012036 0.002595015
##
## $`Anderson-Darling`
##               test statistic        pval
## 1 Anderson-Darling  5.498379 0.001663774
# Low p-values across all tests indicates we do
# not have a Homogeneous Poisson Process
# (failure rate is not constant).
```

```
# purrr::map2(t, T_time_trunc, trend_test, FALSE)
# purrr::map2(t, T_fail_trunc, trend_test, TRUE)
# trend_test(t, T, fail.trunc = FALSE)
# Low p-values across all tests indicates we do
# not have a Homogeneous Poisson Process
# (failure rate is not constant).
```

Fitting a Power Law Process

Since the data indicates that the times between failure are not constant (i.e. are not consistent with a Homogeneous Poisson Process (HPP)) a NHPP should be used to describe the times between failure rather than methods applicable to the exponential distribution. Below the `power_law_prcess` function is used to estimate the parameters of a NHPP with a power law intensity function. (Note that the "iter" argument is only applicable to failure truncated data.) The "beta" parameter (β) affects how the system deteriorates or improves over time. $\beta > 1$ indicates the intensity function is increasing, i.e. that failures are occurring more frequently, the system is deteriorating. $\beta < 1$ indicates the intensity function is decreasing, i.e. that failures are occurring less frequently, the system is improving. $\beta = 1$ indicates the intensity function is constant and the power law process reduces to the HPP.

```
(m <- power_law_process(
  t = t,
  T = T_time_trunc,                    # If any one was failure truncated
  alpha = 0.05,
  fail.trunc = FALSE,                 # then make this true
  iter = 10
))
## $estimates
##    lambda      beta
## 3.5255156 0.6153363
##
## $CIs
##            lcb      ucb
## beta 0.4309739 0.832012
##
## $beta.convergence
## [1] 0.6153363
##
## $lambda.convergence
## [1] 0.4605471
# Fit Power-Law NHPP (AMSAA-Crow Model)
# the .converge items can be ignored since
# we have time terminated data.
# If the data were failure terminated
# then iterative methods need to be used
# to solve for the parameters, so the
```

```
# .converge items are included so you can
# assess if they if the reached convergence.
# So, the iter parameter in the function is
# only applicable to failure terminated data,
# when we would make fail.trunc = TRUE.
# Also, note that the final lambda estimate
# is a transformation of the lambda.converge,
# so they will not match.
# Additionally, alpha is not implemented right
# now. I plan to include confidence intervals
# for the parameters in a future update, which
# will then depend upon alpha.
```

The estimated β value of 0.615 indicates that the failure rate is decreasing. The 95% confidence interval for β does not contain 1. This indicates, just as the trend tests from above did, that the failure rate is not constant and a HPP must be rejected in favor of a NHPP. The "beta.converge" and "lambda.converge" results can be ignored in this case as those are only applicable to failure truncated data when an iterative calculation procedure must be used. The purpose of those values to see that the β and λ estimates have stabilized, in other words, that they are not still changing when we truncate the iterative calculation procedure.

Test for Common Shape Parameter

The power law process fit above assumes that the three systems have identical failure rates. From the plots above we observed that all three systems had decreasing failure rates, but there were differences in them. Now we formally test if the differences observed are large enough to suggest they did not come from systems with identical failure rates. The `common_beta` function tests the null hypothesis that the systems have the same failure rate, versus the alternative hypothesis that they do not have the same failure rate. Small p-values reject the null hypothesis suggesting that the systems do not share a common failure rate.

```
common_beta(
  t = t,
  T = T_time_trunc,
  fail.trunc = FALSE)
## $`Test Statistic`
## [1] 1.762104
##
## $`P-Value`
## [1] 0.4143468
##
## $df
## [1] 2
# Large p-value, do not reject null hypothesis
# of a common shape parameter
```

Based on the p-value from the test of 0.414 we do not have enough evidence to reject the null hypothesis.

Mean Cumulative Function

The *mean cumulative function* (mcf) is a non-decreasing function that indicates the expected number of failures through time t. In the `mcotear` package I have implemented the `power_law_mcf` and `mcf` functions. The `power_law_mcf` provides an estimate of the expected number of failures at specified times based on λ and β values from a power law process. Below I pass the failure times for all three systems, along with the estimated λ and β values obtained earlier from the `power_law_process` function.

```
power_law_mcf(t = t, m$est[1], m$est[2])
##        t  power_mcf
## 1    0.1  0.1116703
## 2    4.3  1.1299785
## 3    4.4  1.1460771
## 4    5.6  1.3294172
## 5    8.4  1.7061477
## 6   10.2  1.9226592
## 7   18.6  2.7826040
## 8   19.5  2.8646999
## 9   23.5  3.2132301
## 10  23.8  3.2384096
## 11  24.2  3.2717931
## 12  26.4  3.4517434
## 13  26.7  3.4758271
## 14  32.5  3.9227507
## 15  44.7  4.7727438
## 16  45.1  4.7989792
## 17  45.8  4.8446768
## 18  48.4  5.0121089
## 19  50.6  5.1510962
## 20  73.6  6.4868174
## 21  74.0  6.5084881
## 22  75.7  6.6000905
## 23  77.1  6.6749350
## 24  79.7  6.8125595
## 25  92.1  7.4465349
## 26  98.6  7.7656680
## 27  98.7  7.7705134
## 28 112.2  8.4083154
## 29 120.1  8.7678331
## 30 129.8  9.1970511
## 31 136.0  9.4649409
## 32 161.8 10.5326738
## 33 180.6 11.2697508
## 34 190.8 11.6572637
## 35 195.8 11.8443028
## 36 197.2 11.8963433
# power_law_mcf(t = list(amsaa$Time), m$est[1], m$est[2])
# power_law_mcf(t = split(amsaa$Time, amsaa$System), m$est[1], m$est[2])
# Any one of these work. This function can
# useful in plotting the function versus the
# observed/emperical mcf estimates so you can
# visually assess how well the model fits the data.
```

The `mcf` function provides a non-parametric estimate of the mcf. For this function, all we need to provide is the failure times, and if applicable, a "by" variable which indicates what variable if we need to split the data (for example `mcf(t = t, by = names(t)` would split the data so we get a mcf per system, instead of a combined mcf that treats the systems as one).

```
df_mcf <- mcf(t = t, by = NULL)
#head(df_mcf)
names(df_mcf)[1] <- "Time"
head(df_mcf)
```

```
##   Time       mcf
## 1  0.1 0.3333333
## 2  4.3 0.6666667
## 3  4.4 1.0000000
## 4  5.6 1.3333333
## 5  8.4 1.6666667
## 6 10.2 2.0000000
# Get the nonparametric mcf estimates.
# I show the first 6 rows so you can see
# what it looks like, and you can see the
# column names. Then I change the name
# of "t" to "Time" so it matches the name
# in the amsaa data set.
```

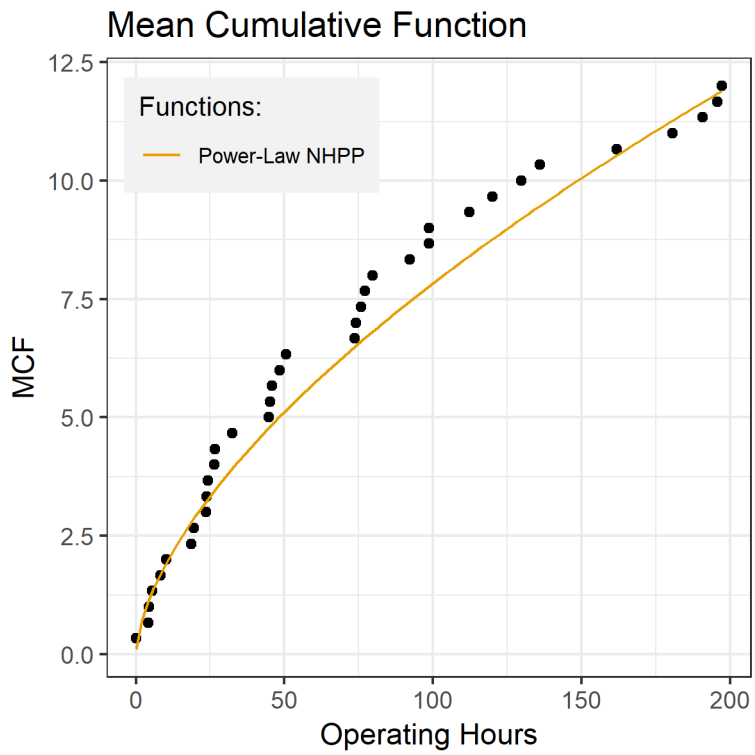
If desired we can merge the mcf estimates with our data. I do this below, merging the non-parametric mcf estimates with the larger data frame.

```
amsaa_identical <- merge(amsaa_identical, df_mcf, by = "Time")
head(amsaa_identical)
##   Time Failure System     mtbf      rocof       mcf
## 1  0.1       1     S2 0.100000 10.0000000 0.3333333
## 2  4.3       1     S1 2.150000  0.4651163 0.6666667
## 3  4.4       2     S1 1.466667  0.6818182 1.0000000
## 4  5.6       2     S2 1.400000  0.7142857 1.3333333
## 5  8.4       1     S3 1.680000  0.5952381 1.6666667
## 6 10.2       3     S1 1.700000  0.5882353 2.0000000
# Merge the nonparametric mcf estimates
# with amsaa into a new data.frame amsaa1
```

Using the merged data and passing the `power_law_mcf` function to ggplot's `stat_function` I am able to plot the non-parametric mcf along with the curve for the estimated mcf based on the power law process.

```
#dev.new(height = 4, width = 4)
ggplot(amsaa_identical, aes(x = Time, y = mcf)) +
  geom_point() +
  labs(
    x = "Operating Hours", y = "MCF",
    title = "Mean Cumulative Function") +
  stat_function(
    fun = function(x){
      power_law_mcf(t = x, lambda = m$e[[1]], beta = m$e[[2]])$power_mcf
    },
    mapping = aes(colour = "Power-Law NHPP")
  ) +
  scale_colour_manual("Functions:",
    breaks = c("Power-Law NHPP"),
    values = c("Power-Law NHPP" = cbPalette[2]),
    guide = guide_legend(
      override.aes = list(
        linetype = c("solid")
    ))
  ) +
  theme(legend.position = c(.225,.875),
    legend.background = element_rect(fill="grey95"),
    legend.key = element_rect(fill="grey95"),
```

```
    legend.text = element_text(size=8),
    legend.title=element_text(size=10)
  )
```




```
# setwd("C:/Users/MCOTEA_User_001/Desktop/Analysis of Repairable Systems")
# ggsave("AMSAA_MCF.png", dpi = 300, height = 4, width = 4)
# Mean Cumulative Function Plot
# Looks like the data fits pretty well.
# I show how you can set your working directory
# and save the plot to file.
```