Package: cmstatrExt (via r-universe)

October 12, 2024

Type Package

Title More Statistical Methods for Composite Material Data

Version 0.4.0

Date 2024-05-05

Description A companion package to 'cmstatr' <https://cran.r-project.org/package=cmstatr>. 'cmstatr' contains statistical methods that are published in the Composite Materials Handbook, Volume 1 (2012, ISBN: 978-0-7680-7811-4), while 'cmstatrExt' contains statistical methods that are not included in that handbook.

URL https://github.com/cmstatr/cmstatrExt, https://cmstatrExt.cmstatr.net

BugReports https://github.com/cmstatr/cmstatrExt/issues

License AGPL-3

Imports dplyr, generics, Rcpp, rlang (>= 0.4.0), stats

LinkingTo Rcpp, testthat

SystemRequirements C++17

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Suggests cmstatr, testthat (>= 3.0.0), tidyverse, lintr, xml2, rmarkdown, knitr

Config/testthat/parallel true

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 2.10)

LazyData true

Repository https://cmstatr.r-universe.dev

RemoteUrl https://github.com/cmstatr/cmstatrext

RemoteRef HEAD

RemoteSha 4e758f9bfa70e06b117147606c8e51e77cb4b6ec

Contents

`augment.average_curve_lm` *Augment a* `data.frame` *with the results from* `average_curve_lm`

Description

Augment a `data.frame` with the results from `average_curve_lm`

Usage

```
## S3 method for class 'average_curve_lm'
augment(x, newdata = NULL, extrapolate = FALSE, ...)
```

Arguments

`x`	an `average_curve_lm` object
`newdata`	(optional) a new `data.frame` to which to augment the object
`extrapolate`	whether to show the curve fit on all data or only the data within the original fitted range. Default: FALSE
`...`	ignored

Value

a `data.frame` with new columns `.fit`, `.extrapolate` and `.residual`

See Also

`average_curve_lm()`

Examples

```
curve_fit <- average_curve_lm(
  pa12_tension,
  Coupon,
  Stress ~ I(Strain) + I(Strain^2) + I(Strain^3) + 0,
  n_bins = 100
)
augment(curve_fit)
## # A tibble: 3,105 × 6
##    Coupon     Strain  Stress  .fit .extrapolate .residual
##    <chr>       <dbl>   <dbl> <dbl> <lgl>            <dbl>
##  1 Coupon 4 0        -0.353  0     FALSE           -0.353
##  2 Coupon 4 0.000200 -0.0604 0.235 FALSE           -0.295
##  3 Coupon 4 0.000400  0.283  0.469 FALSE           -0.185
##  4 Coupon 4 0.000601  0.475  0.702 FALSE           -0.228
##  5 Coupon 4 0.000801  0.737  0.935 FALSE           -0.198
##  6 Coupon 4 0.00100   0.803  1.17  FALSE           -0.364
##  7 Coupon 4 0.00120   1.25   1.40  FALSE           -0.151
##  8 Coupon 4 0.00140   1.32   1.63  FALSE           -0.305
##  9 Coupon 4 0.00160   1.53   1.86  FALSE           -0.325
## 10 Coupon 4 0.00180   2.01   2.09  FALSE           -0.0735
## # i 3,095 more row
## # i Use `print(n = ...)` to see more rows
```

`augment.average_curve_optim`

Augment a `data.frame` *with the results from* `average_curve_optim`

Description

Augment a `data.frame` with the results from `average_curve_optim`

Usage

```
## S3 method for class 'average_curve_optim'
augment(x, newdata = NULL, extrapolate = FALSE, ...)
```

Arguments

`x`	an `average_curve_optim` object
`newdata`	(optional) a new `data.frame` to which to augment the object
`extrapolate`	whether to show the curve fit on all data or only the data within the original fitted range. Default: FALSE
`...`	ignored

Value

a `data.frame` with new columns `.fit`, `.extrapolate` and `.residual`

See Also

`average_curve_lm()`

Examples

```
curve_fit <- average_curve_optim(
  pa12_tension,
  Coupon,
  Strain,
  Stress,
  function(strain, par) {
    sum(par * c(strain, strain^2, strain^3))
  },
  c(c1 = 1, c2 = 1, c3 = 1),
  n_bins = 100
)
augment(curve_fit)
## # A tibble: 3,105 × 6
## Coupon     Strain  Stress  .fit .extrapolate .residual
##    <chr>       <dbl>   <dbl> <dbl> <lgl>            <dbl>
##  1 Coupon 4 0        -0.353  0     FALSE          -0.353
##  2 Coupon 4 0.000200 -0.0604 0.235 FALSE          -0.295
##  3 Coupon 4 0.000400  0.283  0.469 FALSE          -0.185
##  4 Coupon 4 0.000601  0.475  0.702 FALSE          -0.228
##  5 Coupon 4 0.000801  0.737  0.935 FALSE          -0.198
##  6 Coupon 4 0.00100   0.803  1.17  FALSE          -0.364
##  7 Coupon 4 0.00120   1.25   1.40  FALSE          -0.151
##  8 Coupon 4 0.00140   1.32   1.63  FALSE          -0.305
##  9 Coupon 4 0.00160   1.53   1.86  FALSE          -0.325
## 10 Coupon 4 0.00180   2.01   2.09  FALSE          -0.0735
## # i 3,095 more rows
## # i Use `print(n = ...)` to see more rows
```

`average_curve_lm` *Generate an average curve using* `lm`

Description

The user must decide on a single dependent variable (`Y`) and a single independent variable (`X`). The user will specify a `formula` with the relationship between the dependent and independent variables. For a `data.frame` containing stress-strain (or load-deflection) data for more than one coupon, the maximum value of `X` for each coupon is found and the smallest maximum value determines the range over which the curve fit is performed: the range is from zero to this value. Only positive values of `X` are considered. For each coupon individually, the data is divided into a user-specified number of bins and averaged within each bin. The resulting binned/averaged data is then passed to `stats::lm()` to perform the curve fitting.

Usage

```
average_curve_lm(data, coupon_var, model, n_bins = 100)
```

Arguments

`data`	a `data.frame`
`coupon_var`	the variable for coupon identification
`model`	a `formula` for the curve to fit
`n_bins`	the number of bins to average the data inside into before fitting

Details

When specifying the formula (argument `model`), there are two things to keep in mind. First, based on physical behavior, it is normally desirable to set the intercept to zero (e.g. so that there is 0 stress at 0 strain). To do this, include a term `+0` in the formula. Second, when specifying a term for a power of the `X` variable (for example, $X^2$), this needs to be wrapped inside the "as-is" operator `I()`, otherwise, `R` will treat it as an interaction term, rather than an exponent. In other words, if you want to include a quadratic term, you need to write `I(X^2)` (replacing `X` with the appropriate variable from your `data.frame`).

Value

an object of class `average_curve_lm` with the following content:

- `data` the original data provided to the function
- `binned_data` the data after the binning/averaging operation
- `fit_lm` the results of the call to `lm`
- `n_bins` the number of bins specified by the user
- `max_x` the upper end of the range used for fitting
- `y_var` the independent (`Y`) variable
- `x_var` the dependent (`X`) variable

See Also

`~`, `I()`, `lm()`, `average_curve_optim()`, `print.average_curve_lm()`, `summary.average_curve_lm()`, `augment.average_curve_lm()`

Examples

```
# using the `pa12_tension` dataset and fitting a cubic polynomial with
# zero intercept:
curve_fit <- average_curve_lm(
  pa12_tension,
  Coupon,
  Stress ~ I(Strain) + I(Strain^2) + I(Strain^3) + 0,
  n_bins = 100
)
print(curve_fit)
## Range: ` Strain ` in  [ 0,  0.1409409 ]
##
## Call:
##   average_curve_lm(data = pa12_tension, coupon_var = Coupon,
##                    model = Stress ~ I(Strain) + I(Strain^2) + I(Strain^3)
##                    + 0, n_bins = 100)
##
## Coefficients:
##    I(Strain)   I(Strain^2)   I(Strain^3)
##        1174         -8783         20586
```

average_curve_optim *Generate an average curve using* `optim`

Description

The user must decide on a single dependent variable (`Y`) and a single independent variable (`X`). The user will specify a function defining the relationship between the dependent and independent variables. For a `data.frame` containing stress-strain (or load-deflection) data for more than one coupon, the maximum value of `X` for each coupon is found and the smallest maximum value determines the range over which the curve fit is performed: the range is from zero to this value. Only positive values of `X` are considered. For each coupon individually, the data is divided into a user-specified number of bins and averaged within each bin. The resulting binned/averaged data is then used for curve fitting. The mean squared error between the observed value of `Y` and the result of the user-specified function evaluated at each `X` is minimized by varying the parameters `par`.

Usage

```
average_curve_optim(
  data,
  coupon_var,
  x_var,
  y_var,
  fn,
  par,
  n_bins = 100,
  method = "L-BFGS-B",
  ...
)
```

Arguments

`data`	a `data.frame`

`coupon_var`	the variable for coupon identification
`x_var`	the independent variable
`y_var`	the dependent variable
`fn`	a function defining the relationship between `Y` and `X`. See Details for more information.
`par`	the initial guess for the parameters
`n_bins`	the number of bins to average the data inside into before fitting
`method`	The method to be used by `optim()`. Defaults to "L-BFGS-B"
`...`	extra parameters to be passed to `optim()`

Details

The function `fn` must have two arguments. The first argument must be the value of the independent variable (`X`): this must be a numeric value (of length one). The second argument must be a vector of the parameters of the model, which are to be varied in order to obtain the best fit. See below for an example.

Value

an object of class `average_curve_optim` with the following content:

- `data` the original data provided to the function
- `binned_data` the data after the binning/averaging operation
- `fn` the function supplied
- `fit_optim` the results of the call to `optim`
- `call` the call
- `n_bins` the number of bins specified by the user
- `max_x` the upper end of the range used for fitting
- `y_var` the independent (`Y`) variable
- `x_var` the dependent (`X`) variable

See Also

`optim()`, `average_curve_lm()`, `print.average_curve_optim()`, `augment.average_curve_optim()`

Examples

```
# using the `pa12_tension` dataset and fitting a cubic polynomial with
# zero intercept:
curve_fit <- average_curve_optim(
  pa12_tension,
  Coupon,
  Strain,
  Stress,
  function(strain, par) {
    sum(par * c(strain, strain^2, strain^3))
  },
  c(c1 = 1, c2 = 1, c3 = 1),
  n_bins = 100
)
## Range: ` Strain ` in  [ 0,  0.1409409 ]
##
## Call:
## average_curve_optim(data = pa12_tension, coupon_var = Coupon,
##                     x_var = Strain, y_var = Stress,
##                     fn = function(strain, par) {
##                       sum(par * c(strain, strain^2, strain^3))
##                     }, par = c(c1 = 1, c2 = 1, c3 = 1), n_bins = 100)
##
## Parameters:
##       c1        c2        c3
## 1174.372 -8783.106 20585.898
```

`fff_shear` *Example shear stress-shear strain data*

Description

Example shear stress-strain data. This data was collected using a novel shear test method. Coupons were made using a thermoset via Fused Filament Fabrication (FFF). This data requires some clean-up, including removal of the "toe", offsetting the strain, and removal of the post-failure data points. These procedures are demonstrated in the `stress-strain` vignette. See: `vignette("stress-strain", package = "cmstatrExt")`

Usage

```
fff_shear
```

Format

`fff_shear`**:**

A data frame with 2,316 rows and 3 columns:

Coupon the coupon ID

Stress the shear stress measurement `[psi]`

Strain the shear strain measurement `[in/in]`

`iso_equiv_two_sample` *Calculate t1 and t2 pairs that have the same p-Value*

Description

Calculates pairs of t1 and t2 values, which have the same p-value for the two-sample equivalency test. See `p_equiv_two_sample()`.

Usage

```
iso_equiv_two_sample(n, m, alpha, t1max, t2max, n_points)
```

Arguments

`n`	the size of the qualification sample
`m`	the size of the equivalency sample
`alpha`	the desired p-value
`t1max`	the maximum value of t1 (only approximate)
`t2max`	the maximum value of t2 (only approximate)
`n_points`	the number of returned points is twice n_points

Details

The values t1 and t2 are based on the transformation:

t1 = (X_mean - Y_min) / S

t2 = (X_mean - Y_mean) / S

Where:

- X_mean is the mean of the qualification sample
- S is the standard deviation of the qualification sample
- Y_min is the minimum from the acceptance sample
- Y_mean is the mean of the acceptance sample

Value

A `data.frame` with values of t1 and t2

References

Kloppenborg, S. (2023). Lot acceptance testing using sample mean and extremum with finite qualification samples. Journal of Quality Technology, https://doi.org/10.1080/00224065.2022.2147884

See Also

`p_equiv_two_sample()`, `k_equiv_two_sample()`

Examples

```
if(requireNamespace("tidyverse")){
  library(cmstatrExt)
  library(tidyverse)
  curve <- iso_equiv_two_sample(24, 8, 0.05, 4, 1.5, 10)
  curve

  curve %>%
    ggplot(aes(x = t1, y = t2)) +
      geom_path() +
      ggtitle("Acceptance criteria for alpha=0.05")
}
```

`k_equiv_two_sample` *Calculate the factors for a two-sample acceptance test*

Description

Calculates the factors k1 and k2, which are used for setting acceptance values for lot acceptance. These factors consider both the size of the qualification sample (`n`) and the size of acceptance sample (`m`). This test is detailed in a forthcoming paper.

Usage

```
k_equiv_two_sample(alpha, n, m)
```

Arguments

`alpha`	the desired probability of Type 1 error
`n`	the size of the qualification sample
`m`	the size of the acceptance sample

Value

A vector of length 2 with the contents `c(k1, k2)`

References

Kloppenborg, S. (2023). Lot acceptance testing using sample mean and extremum with finite qualification samples. Journal of Quality Technology, https://doi.org/10.1080/00224065.2022.2147884

`pa12_tension` *Example stress-strain data*

Description

Example tension stress-strain data. This data was generated by tracing the stress-strain graph for PA12 from the manuscript referenced below. The non-linearity seen at low strain in the original data set was removed, then the data was re-sampling to produce more tightly spaced strain values. Normally-distributed error was added to the stress. The code used to generate the data set can be found at `https://github.com/cmstatr/cmstatrExt/blob/master/data-raw/pa12-tension.R`

Usage

```
pa12_tension
```

Format

`pa12_tension`**:**

A data frame with 3,212 rows and 3 columns:

Coupon the coupon ID

Strain the strain measurement `[mm/mm]`

Stress the stress measurement `[MPa]`

Source

Alomarah, Amer & Ruan, Dong & Masood, S. & Gao, Zhanyuan. (2019). Compressive properties of a novel additively manufactured 3D auxetic structure. Smart Materials and Structures. 28. 10.1088/1361-665X/ab0dd6.

`power_sim_dual` *Rejection rate for dual acceptance criteria based via simulation*

Description

Performs Monte Carlo simulation to determine the rejection rate of a dual acceptance criteria (based on sample minimum and mean). By specifying several sets of parameters for the "equivalency" distribution, a power curve for the acceptance test can be determined.

Usage

```
power_sim_dual(
  n_qual,
  m_equiv,
  replicates,
  distribution = "rnorm",
  param_qual,
  param_equiv,
  k1,
  k2
)
```

Arguments

`n_qual`	the sample size of the qualification sample
`m_equiv`	the sample size of the equivalency/acceptance sample
`replicates`	the number of simulated qualification samples and equivalency samples. If a single value is given, the same numbers used for both, if a vector of length two is given, the first element is the number of qualification replicates and the second element is the number of equivalency replicates.
`distribution`	a function name for generating a random sample (defaults to "rnorm")
`param_qual`	a data.frame (must be single row) with columns matching the arguments of the distribution function
`param_equiv`	a data.frame with columns matching the arguments of the distribution function. The simulation is repeated with the parameters specified in each row of the data.frame.
`k1`	a factor for determining the acceptance criterion for sample minimum, which is calculated as `mean_qual - k1 * sd_qual`
`k2`	a factor for determining the acceptance criterion for sample average, which is calculated as `mean_qual - k2 * sd_qual`

Details

This function performs simulation to estimate the performance of the dual acceptance criteria commonly used for composite materials in aerospace applications. These criteria are based on setting lower limits on the minimum individual (lower extremum) and the mean of an "acceptance" sample. These limits are computed based on the sample mean and sample standard deviation of an initial "qualification" sample. The criteria are intended to be a test of non-inferiority to determine if the material lots from which the "acceptance" samples are drawn should be accepted for production. Another common use of these criteria are to determine if a process change, equipment change, or second manufacturing site is acceptable for production.

For each set of distribution parameters given by the rows of `param_equiv`, a number of samples of size `m_equiv` are generated using the function `distribution`. Next, a number of qualification samples of size `n_qual` are generated using the `distribution` function and the parameters given in `param_qual`. Limits for minimum and average are determined for each qualification sample. Each equivalency sample is compared with the limits determined from each qualification sample.

The number of replicate in this simulation is given by `replicates`: if this is a vector of length two, the first element is the number of qualification samples and the second is the number of equivalency samples; if `replicates` is a single value, the same number is used for the number of qualification samples and acceptance samples. Therefore, for each row of `param_equiv` a total of `replicates[1]` * `replicates[2]` criteria are evaluated.

The argument `distribution` must correspond with a function that generates (pseudo) random numbers. This function must have an argument `n` that specifies the sample size to be generated. When the argument `distribution` matches certain common distribution functions (such as `rnorm`), the C++ implementation of the random number generation function is used instead of calling R code, which results in a significant speedup.

Value

A `data.frame`. The first column(s) are duplicate of the `data.frame` passed in the argument `param_equiv`. The last column is named `Rejection Rate` and has a value equal to the number of samples rejected for each simulation run.

See Also

`k_equiv_two_sample`

Examples

```
# Compute a power curve for a dual acceptance criteria for a qualification
# sample size of 18 and an equivalency sample size of 6, using 2000
# replicates. A standard normal distribution is used and the power to
# detect a decrease in mean is determined.
set.seed(12345) # make it reproducible
power_sim_dual(
  18, 6,
  2000,
  "rnorm",
  data.frame(mean = 0, sd = 1),
  data.frame(mean = c(-2, -1.5, -1, 0.5, 0), sd = 1),
  2.959, 0.954
)
##   mean sd Rejection Rate
## 1 -2.0  1     0.98349975
## 2 -1.5  1     0.88186900
## 3 -1.0  1     0.56382425
## 4  0.5  1     0.00864025
## 5  0.0  1     0.04826250
```

`print.average_curve_lm`

Print an `average_curve_lm` *object*

Description

Print an `average_curve_lm` object

Usage

```
## S3 method for class 'average_curve_lm'
print(x, ...)
```

Arguments

`x`	an `average_curve_lm` object
`...`	additional arguments passed to `print.lm`

Value

The object passed to this method is invisibly returned (via `invisible(x)`).

See Also

`average_curve_lm()`

`print.average_curve_optim` *Print an* `average_curve_optim` *object*

Description

Print an `average_curve_optim` object

Usage

```
## S3 method for class 'average_curve_optim'
print(x, ...)
```

Arguments

`x`	an `average_curve_optim` object
`...`	not used

Value

The object passed to this method is invisibly returned (via `invisible(x)`).

See Also

`average_curve_optim()`

`p_equiv` *p-Value for one-sample equivalency*

Description

Calculates the p-Value for a one-sample acceptance test based on Vangel (2002). This test considers the sample size of the acceptance sample (`m`).

Two test statistics are required:

t1 = (mu - Y_min) / sigma

t2 = (mu - Y_mean) / sigma

Where:

- mu is the mean of the population
- sigma is the standard deviation of the population
- Y_min is the minimum from the acceptance sample
- Y_mean is the mean of the acceptance sample

Usage

```
p_equiv(m, t1, t2)
```

Arguments

`m`	the size of the acceptance sample
`t1`	the test statistic described above. May be a vector.
`t2`	the test statistic described above. May be a vector.

Value

a vector of p-Values of the same length as t1 and t2

`p_equiv_two_sample` *p-Value for two-sample equivalency*

Description

Calculates the p-Value for a two-sample acceptance test. This test considers the sample size of the qualification sample (`n`) and the acceptance sample (`m`).

Two test statistics are required:

t1 = (X_mean - Y_min) / S

t2 = (X_mean - Y_mean) / S

Where:

- X_mean is the mean of the qualification sample
- S is the standard deviation of the qualification sample
- Y_min is the minimum from the acceptance sample
- Y_mean is the mean of the acceptance sample

Usage

```
p_equiv_two_sample(n, m, t1, t2)
```

Arguments

`n`	the size of the qualification sample
`m`	the size of the acceptance sample
`t1`	the test statistic described above. May be a vector.
`t2`	the test statistic described above. May be a vector.

Value

a vector of p-Values of the same length as t1 and t2

`summary.average_curve_lm` *Summarize an* `average_curve_lm` *object*

Description

Summarize an `average_curve_lm` object

Usage

```
## S3 method for class 'average_curve_lm'
summary(object, ...)
```

Arguments

`object`	an `average_curve_lm` object
`...`	arguments passed to `summary.lm`

Value

No return value. This method only produces printed output.

See Also

`average_curve_lm()`

Index