Package ‘VFP’

October 14, 2021

Version 1.3.1
Date 2021-10-14
Title Variance Function Program
Author Andre Schuetzenmeister [cre, aut],
Florian Dufey [aut],
Andrea Geistanger [ctb]
Maintainer Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>
Depends R (>= 3.1.0), gnm
Imports VCA, MASS, utils, stats, graphics, grDevices
Suggests RUnit, knitr, rmarkdown, prettydoc
Description Variance function estimation for models proposed by W. Sadler in his variance function program ('VFP', <http://www.aacb.asn.au/resources/useful-tools/variance-function-program-v14>). Here, the idea is to fit multiple variance functions to a data set and consequently assess which function reflects the relationship 'Var ~ Mean' best. For 'in-vitro diagnostic' (IVD) assays modeling this relationship is of great importance when individual test-results are used for defining follow-up treatment of patients.
License GPL (>= 2)
VignetteBuilder knitr, rmarkdown
NeedsCompilation no
Repository CRAN
Date/Publication 2021-10-14 16:20:02 UTC

R topics documented:

VFP-package ................................................................. 2
addGrid ............................................................... 4
as.rgb ............................................................ 4
B2mIntra_98 ......................................................... 5
VFP-package

(V)ariance (F)unction (P)rogram

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>bt.coef</td>
<td>6</td>
</tr>
<tr>
<td>coef.VFP</td>
<td>6</td>
</tr>
<tr>
<td>conditionHandler</td>
<td>7</td>
</tr>
<tr>
<td>deriveCx</td>
<td>8</td>
</tr>
<tr>
<td>fit.EP17</td>
<td>9</td>
</tr>
<tr>
<td>fit.vfp</td>
<td>11</td>
</tr>
<tr>
<td>getMat.VCA</td>
<td>15</td>
</tr>
<tr>
<td>Glucose</td>
<td>16</td>
</tr>
<tr>
<td>legend.rm</td>
<td>17</td>
</tr>
<tr>
<td>MultiLotReproResults</td>
<td>18</td>
</tr>
<tr>
<td>plot.VFP</td>
<td>18</td>
</tr>
<tr>
<td>powfun2simple</td>
<td>22</td>
</tr>
<tr>
<td>powfun3</td>
<td>23</td>
</tr>
<tr>
<td>powfun3simple</td>
<td>23</td>
</tr>
<tr>
<td>powfun4</td>
<td>24</td>
</tr>
<tr>
<td>powfun4simple</td>
<td>24</td>
</tr>
<tr>
<td>powfun5</td>
<td>24</td>
</tr>
<tr>
<td>powfun5simple</td>
<td>25</td>
</tr>
<tr>
<td>powfun6</td>
<td>25</td>
</tr>
<tr>
<td>powfun6simple</td>
<td>25</td>
</tr>
<tr>
<td>powfun7</td>
<td>26</td>
</tr>
<tr>
<td>powfun7simple</td>
<td>26</td>
</tr>
<tr>
<td>powfun8</td>
<td>26</td>
</tr>
<tr>
<td>powfun8simple</td>
<td>27</td>
</tr>
<tr>
<td>powfun9simple</td>
<td>27</td>
</tr>
<tr>
<td>precisionPlot</td>
<td>27</td>
</tr>
<tr>
<td>predict.modelEP17</td>
<td>30</td>
</tr>
<tr>
<td>predict.VFP</td>
<td>31</td>
</tr>
<tr>
<td>predictMean</td>
<td>32</td>
</tr>
<tr>
<td>print.VFP</td>
<td>34</td>
</tr>
<tr>
<td>RealData1</td>
<td>35</td>
</tr>
<tr>
<td>ReproData</td>
<td>35</td>
</tr>
<tr>
<td>Signif</td>
<td>36</td>
</tr>
<tr>
<td>summary.VFP</td>
<td>36</td>
</tr>
<tr>
<td>t.coef</td>
<td>37</td>
</tr>
<tr>
<td>T4S9_99</td>
<td>38</td>
</tr>
</tbody>
</table>

Index

39
**Description**

The intended use of this package is to implement variance functions proposed in Sadler’s VFP stand-alone software (see reference below), from which the name was borrowed as well. Main function of this package is `fit.vfp` for fitting non-linear variance-function models. Usually, these models are fitted to analysis-results of precision performance data e.g. frequently generated for in-vitro diagnostics (IVD). R-package VFP is designed to work best on objects of class `VCA` as generated by R-package VCA but it is not restricted to these. There are several functions operating on S3-objects of class `VFP`, e.g. `plot.VFP`, `print.VFP`, `summary.VFP`, and `predict.VFP`. Function `predictMean` is of special interest when a functional relationship is used to derive limit of quantitation (LoQ) or functional sensitivity, as the concentration at which the IVD-imprecision expressed as coefficient of variation (CV) undercuts a specific threshold.

**Details**

<table>
<thead>
<tr>
<th>Package:</th>
<th>VFP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>1.3.1</td>
</tr>
<tr>
<td>Date:</td>
<td>2021-10-14</td>
</tr>
<tr>
<td>License:</td>
<td>GPL (&gt;=3)</td>
</tr>
<tr>
<td>LazyLoad:</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Author(s)**

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>, Florian Dufey <florian.dufey@roche.com>, Andrea Geistanger <andrea.geistanger@roche.com>

**References**


addGrid

Add a Grid to an Existing Plot.

Description

It is possible to use automatically determined grid lines (x=NULL, y=NULL) or specifying the number of cells x=3,y=4 as done by grid. Additionally, x- and y-locations of grid-lines can be specified, e.g. x=1:10, y=seq(0,10,2).

Usage

addGrid(x = NULL, y = NULL, col = "lightgray", lwd = 1L, lty = 3L)

Arguments

x (integer, numeric) single integer specifies number of cells, numeric vector specifies vertical grid-lines
y (integer, numeric) single integer specifies number of cells, numeric vector specifies horizontal grid-lines
col (character) color of grid-lines
lwd (integer) line width of grid-lines
lty (integer) line type of grid-lines

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

as.rgb

Convert Color-Name or RGB-Code to Possibly Semi-Transparent RGB-code.

Description

Function takes the name of a color and converts it into the rgb space. Parameter "alpha" allows to specify the transparency within [0,1], 0 meaning completely transparent and 1 meaning completely opaque. If an RGB-code is provided and alpha != 1, the RGB-code of the transparency adapted color will be returned.

Usage

as.rgb(col = "black", alpha = 1)
Arguments

col  (character) name of the color to be converted/transformed into RGB-space (code). Only those colors can be used which are part of the set returned by function colors(). Defaults to "black".

alpha  (numeric) value specifying the transparency to be used, 0 = completely transparent, 1 = opaque.

Value

RGB-code

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

# convert character string representing a color to RGB-code
# using alpha-channel of .25 (75\% transparent)
as.rgb("red", alpha=.25)

# same thing now using the RGB-code of red (alpha=1, i.e. as.rgb("red"))
as.rgb("#FF0000FF", alpha=.25)

Description

Results in this file are all serum/CSF clinical specimen duplicates during 1998 from a low throughput Beta-2-microglobulin RIA.

Usage

data(B2mIntra_98)

Format

data.frame with 554 observations and 2 variables.

References

VFP Program Version 12.0\h http://www.aacb.asn.au/professionaldevelopment/useful-tools
bt.coef  
*Back-Transformation of Estimated Coefficients.*

**Description**

This function performs back-transformation from re-parameterized forms in the 'VFP' package into the original form.

**Usage**

```r
bt.coef(object, K = NULL, signJ = NULL, model = NULL, ...)
```

**Arguments**

- `object`: (object) of class 'gnm' representing a fitted model in re-parameterized form
- `K`: (numeric) constant value 'K'
- `signJ`: (integer) either 1 or -1
- `model`: (integer) specifying which model shall be back-transformed
- `...`: additional parameters

**Details**

In the 'VFP' package models are re-parameterized to have better control over the constraint solution-space, i.e. only models may be fitted generating non-negative fitted values. This function is intended to be for internal use only.

**Value**

(numeric) vector of coefficients in original parameterized form

**Author(s)**

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com> Florian Dufey <florian.dufey@roche.com>

---

coef.VFP  
*Extract Model-Coefficients from VFP-Objects.*

**Description**

Extract Model-Coefficients from VFP-Objects.

**Usage**

```r
## S3 method for class 'VFP'
coef(object, model.no = NULL, ...)
```
**conditionHandler**  

**Arguments**

- **object** (object) of class "VFP"
- **model.no** (integer) specifying one of models 1:10, must be one of the fitted models
- ... additional parameters passed forward

**Value**

(numeric) model coefficients

**Author(s)**

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com> Florian Dufey <florian.dufey@roche.com>

**Examples**

```r
library(VCA)
data(VCAdata1)
lst <- anovaVCA(y~(device+lot)/day/run, VCAdata1, by="sample")
mat <- getMat.VCA(lst) # automatically selects "total"
res <- fit.vfp(model.no=1:10, Data=mat)
coef(res)
```

**Description**

Function is intended to wrap expressions provided and catching all potentially useful information generated by the wrapped expression, i.e. errors, warnings, and messages.

**Usage**

```r
conditionHandler(expr, file = NULL)
```

**Arguments**

- **expr** (expression) for which exception handling should be provided
- **file** (character) string specifying a file to which all captured output shall be written

**Value**

(list) with element "result", "status" (0 = no warnings, no errors), 1 = warnings were caught, 2 = errors were caught no result generated, "warnings", "errors", "messages"
Author(s)
Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```r
c conditionalHandler(warning("This is a warning!"))
f <- function(expr){warning("This a warning!"); eval(expr)}
conditionalHandler(f(1/2))
conditionalHandler(stop("This is an error!"))
conditionalHandler(1/"a")
```

```
deriveCx                  Determine C5 and C95 or any Concentration Cx.

Description
This function makes use of a precision profile, determining the concentration of that sample when measured a large number of times, $100 \times Cx$ exactly 5 be larger than cutoff. This follows the CLSI EP12 guideline whenever an internal continuous result (ICR) is available and measurement imprecision can be assumed to be normally distributed. The CLSI EP12 recommends to base derivation of C5 and C95 on the results of intermediate precision analyses using multiple samples. This includes between-day and between-run as additional variance components besides repeatability.

Usage
```r
deriveCx(vfp, model.no = NULL, start = NULL, cutoff = NULL, Cx = 0.05, tol = 1e-06, plot = FALSE)
```

Arguments
- `vfp` (VFP) object representing a precision profile to be used
- `model.no` (integer) specifying which model to use, if NULL the "best" model will be used automatically
- `start` (numeric) start concentration, e.g. for C5 smaller than \( r \), for C95 larger than \( R \)
- `cutoff` (numeric) the cutoff of to be used
- `Cx` (numeric) the probability, e.g. for C5 use 0.05 and for C95 use 0.95
- `tol` (numeric) tolerance value determining when the iterative bisections search can terminate, i.e. when the difference becomes small enough
- `plot` (logical) TRUE = plot the result
**Value**

(numeric) Mean and SD of concentration Cx

**Author(s)**

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

**Examples**

```r
## Not run:
# perform variance component analysis
library(VCA)
data(VCAdata1)
# perform VCA-anaylsis
lst <- anovaVCA(y~(device+lot)/day/run, VCAdata1, by="sample")
# transform list of VCA-objects into required matrix
mat <- getMat.VCA(lst) # automatically selects "total"
mat
# fit all models batch-wise
res <- fit.vfp(model.no=1:10, Data=mat)
# now search for the C5 concentration
deriveCx(res, start=15, cutoff=20, Cx=0.05, plot=TRUE)
deriveCx(res, start=25, cutoff=20, Cx=0.95, plot=TRUE)
deriveCx(res, start=25, cutoff=20, Cx=0.25, plot=TRUE)
deriveCx(res, start=25, cutoff=20, Cx=0.75, plot=TRUE)

#
p <- c(seq(.01, .12, .01), seq(.15, .85, .05), seq(.88, .99, .01))
system.time(x <- deriveCx(res, Cx=p, cutoff=20))
## End(Not run)
```

**Description**

This function fits the model proposed in CLSI EP17 by log-transforming CV (Y) as well as mean-values (X) and performing a linear regression of these. More specifically CV = A * Conc^B, where Conc = mean concentration of a sample and CV is on the percent-scale, is fitted by ordinary least squares (OLS) estimation of log(CV) = A + B * log(Conc). Fitted values are subsequently back-transformed using formula cv = exp(a) * C^b, where cv, a and b represent estimates of CV, A and B.

**Usage**

```r
fit.EP17(x, y, DF, typeY = c("vc", "sd", "cv"), k = 2, ...)
```
Arguments

- **x**: (numeric) mean concentrations of samples
- **y**: (numeric) variability at 'x' on VC-, SD-, or CV-scale
- **DF**: (numeric) vector of degrees of freedom linked to variabilities 'y' used in derivation of deviance and AIC
- **typeY**: (character) specifying the scale of 'y'-values
- **k**: (numeric) numeric specifying the 'weight' of the equivalent degrees of freedom (edf) part in the AIC formula.
- ... additional arguments

Details

The AIC is computed following the implementation of `extractAIC.lm` in the 'stats' package with the adaption of using 'n = sum(df)' instead of 'n' being the number of residuals. The 'df' come from a precision analysis, thus, there are far more observations used to fit this model than indicated by the number of residuals.

Value

(list) with items "x" and "y" as provided, and "x.out" and "y.out" representing X- and Y-coordinates of fitted values for plotting

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```r
# data from appendix D of CLSI EP17-A2 (pg. 54)
EP17.dat <- data.frame(
  Lot=c(rep("Lot1", 9), rep("Lot2", 9)),
  Mean=c( 0.04, 0.053, 0.08, 0.111, 0.137, 0.164, 0.19, 0.214, 0.245,
         0.041, 0.047, 0.077, 0.106, 0.136, 0.159, 0.182, 0.205, 0.234),
  CV=c(40.2, 29.6, 19.5, 15.1, 10.0, 7.4, 6.0, 7.5, 5.4,
       44.1, 28.8, 15.1, 17.8, 11.4, 9.2, 8.4, 7.8, 6.2),
  SD=c(0.016, 0.016, 0.016, 0.017, 0.014, 0.012, 0.011, 0.016, 0.013,
       0.018, 0.014, 0.012, 0.019, 0.016, 0.015, 0.015, 0.016, 0.014),
  DF=rep(1, 18)
)

EP17.dat$VC <- EP17.dat$SD^2

lot1 <- subset(EP17.dat, Lot=="Lot1")
lot2 <- subset(EP17.dat, Lot=="Lot2")

# function fit.EP17 is not exported, use package namespace in call
fit.lot1 <- VFP:::fit.EP17(x=lot1$Mean, y=lot1$CV, typeY="cv", DF=lot1$DF)
```
Description

This function fits one out of ten or any subset of ten non-linear functions at once. This is done on precision data consisting of information about the variance, concentration at which these variance was observed and the respective degrees of freedom. Provided data must contain at least three column with these information. There are following variance-functions covered:

- constant variance $\sigma^2$
- constant CV $\sigma^2 = \beta_1 u^2$
- mixed constant, proportional variance $\sigma^2 = \beta_1 + \beta_2 u^2$
- constrained power model, constant exponent $\sigma^2 = (\beta_1 + \beta_2 u)^K$
- alternative constrained power model $\sigma^2 = \beta_1 + \beta_2 u^K$
- alternative unconstrained power model for VF’s with a minimum $\sigma^2 = \beta_1 + \beta_2 u + \beta_3 u^J$
- alternative unconstrained power model $\sigma^2 = \beta_1 + \beta_2 u^J$
- unconstrained power model (default model of Sadler) $\sigma^2 = (\beta_1 + \beta_2 u)^J$
- CLSI EP17 similar model $\sigma^2 = \beta_1 u^J$
- Exact CLSI EP17 model (fit by linear regression on logarithmic scale) $cv = \beta_1 u^J$

Fitting all ten models is somehow redundant if constant 'C' is chosen to be equal to 2, since models 3 and 5 are equivalent and these are constrained versions of model 7 where the exponent is also estimated. The latter also applies to model 4 which is a constrained version of model 8. Nevertheless, since computation time is not critical here for typical precision-profiles (of in-vitro diagnostics precision experiments) we chose to offer batch-processing as well. During computation, all models are internally reparameterized so as to guarantee that the variance function is positive in the range 'u' from 0 to 'u_max'. In models 7 and 8, 'J' is restricted to $0.1<J<10$ to avoid the appearance of sharp hooks. Occasionally, these restrictions may lead to a failure of convergence. This is then a sign that the model parameters are on the boundary and that the model fits the data very badly. This should not be taken as reason for concern. It occurs frequently for model 6 when the variance function has no minimum, which is normally the case.

Usage

```r
fit.vfp(
  Data,
  model.no = 7,
  K = 2,
  startvals = NULL,
  quiet = T,
  col.mean = "Mean",
)```

Arguments

Data (data.frame, matrix) containing mean-values, estimated variances and degrees of freedom for each sample
model.no (integer) in 1:10, may be any combination of these integer-values specifying models 1 to 9 which are consequently fitted to the data
K (numeric) value defining the constant used in models 4 and 5
startvals (numeric) vector of start values for the optimization algorithm, only used if a single model is specified by the user, otherwise, start values will be sought automatically
quiet (logical) TRUE = all output messages (warnings, errors) and regular console output is suppressed and can be found in elements "stderr" and "stdout" of the returned object
col.mean (character) string specifying a column/variable in 'Data' containing mean-values
col.var (character) string specifying a column/variable in 'Data' containing the variances
col.df (character) string specifying a column/variable in 'Data' containing the degrees of freedom
col.sd (character) string (optional) specifying a column/variable in 'Data' containing the SD-values, used for model 10 to derive more precise CV-value compared to derivation from 'col.var' in case 'col.cv' is not specified directly
col.cv (character) string (optional) specifying a column/variable in 'Data' containing the CV-values, if missing, first 'col.sd' is checked, if missing 'col.var' used to derive per-sample CV-values
minVC (numeric) value assigned to variances being equal to zero, defaults to NA, which results in removing these observations; could also be set to the smallest possible positive double (.Machine$double.eps)
...
additional parameters passed forward, e.g. 'vc' of function getMat.VCA for selecting a specific variance component in case of 'Data' being a list of 'VCA'-objects (see examples)

Details

Functions predict.VFP and predictMean are useful to make inference based on a fitted model. It is possible to derive concentrations at which a predefined variability is reached, which is sometimes referred to as "functional sensitivity" and/or "limit of quantitation" (LoQ). Function predictMean returns the fitted value at which a user-defined variance ("vc"), SD or CV is reached with its corresponding 100(1-alpha)% CI derived from the CI of the fitted model. The plotting method for
objects of class 'VFP' can automatically add this information to a plot using arguments 'Prediction' and 'Pred.CI' (see plot.VFP for details. Function predict.VFP makes predictions for specified mean-values based on fitted models.

Value

(object) of class 'VFP' with elements:

Models objects of class 'gnm' representing the fitted model
RSS residual sum of squares for each fitted model
AIC the Akaike information criterion for each fitted model
Deviance the deviance for each fitted model
Formulas formula(s) as expression for each fitted model
Mu.Func function(s) to transform mean value to re-parameterized scale
Mu list of transformed mean values
Data the original data set provided to fit model(s)
K constant as specified by the user; must be >0
startvals start values as specified by the user
errors messages of all errors caught
output if `quiet=TRUE` all output that was redirected to a file
warning messages of all warnings caught
notes all other notes caught during execution

Author(s)

Florian Dufey <florian.dufey@roche.com>, Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

plot.VFP, predict.VFP, predictMean

Examples

# load VCA-package and data
library(VCA)
data(VCAdata1)
# perform VCA-analysis
lst <- anovaVCA(y~(device+lot)/day/run, VCAdata1, by="sample")
# transform list of VCA-objects into required matrix
mat <- getMat.VCA(lst) # automatically selects "total"
mat

# fit all 9 models batch-wise
res <- fit.vfp(model.no=1:10, Data=mat)
# if 'mat' is not required for later usage, following works
# equally well
res2 <- fit.vfp(lst, 1:10)

# plot best-fitting model
plot(res)
plot(res, type="cv")
plot(res, type="cv", ci.type="lines", ci.col="red",
Grid=list(col="wheat"), Points=list(pch=2, lwd=2, col="black"))

# now derive concentration at which a specific reproducibility-
# imprecision of 10\% is reached and add this to the plot
pred <- plot(res, type="cv", ci.type="band",
   ci.col=as.rgb("red", .25), Grid=list(col="orange"),
   Points=list(pch=2, lwd=2, col="black"),
   Prediction=list(y=10, col="red"), Pred.CI=TRUE)

# (invisibly) returned object contains all relevant information
pred

# same for repeatability
mat.err <- getMat.VCA(lst, "error")
res.err <- fit.vfp(1:10, Data=mat.err)

# without extracting 'mat.err'
res.err2 <- fit.vfp(lst, 1:10, vc="error")
plot(res.err)

# another example using CA19_9 data from CLSI EP05-A3

data(CA19_9)

# fit reproducibility model to data
fits.CA19_9 <- anovaVCA(result~site/day, CA19_9, by="sample")

# fit within-laboratory-model treating site as fixed effect
fits.ip.CA19_9 <- anovaMM(result~site/(day), CA19_9, by="sample")

# the variability "between-site" is not part of "total"
fits.ip.CA19_9[[1]]
fits.CA19_9[[1]]

# extract repeatability
rep.CA19_9 <- getMat.VCA(fits.CA19_9, "error")

# extract reproducibility
repro.CA19_9 <- getMat.VCA(fits.CA19_9, "total")

# extract intermediate-precision (within-lab)
ip.CA19_9 <- getMat.VCA(fits.ip.CA19_9, "total")
# fit model \((a+bX)^c\) (model 8) to all three matrices

```r
test <- getMat.VCA()
```

## Description

Transform list of VCA-object into VFP-matrix required for fitting.

## Usage

```r
getMat.VCA(obj, vc = 1)
```

## Arguments

- **obj** (list) of VCA-objects
vc

(integer, character) either an integer specifying a variance component or the name of a variance component; can also be a vector of integers specifying a continuous sequence of variance components always including 'error' (repeatability)

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```r
library(VCA)
data(VCAdata1)
lst <- anovaVCA(y=(device+lot)/day/run, VCAdata1, by="sample")
getMat.VCA(lst) # automatically selects 'total'
# pooled version of intermediate precision (error+run+day)
getMat.VCA(lst, 4:6)
# only repeatability ('error')
getMat.VCA(lst, "error")
```

Description

Intra-day precision of arterial glucose measurement on the Precision PCx instrument (Abbott Laboratories, Abbott Park, Illinois, USA). The data were kindly provided by Peter Watkinson, The John Radcliffe Hospital, Oxford, UK. They provide an excellent example of a situation where a simple variance function should be imposed in preference to accepting the fit of one of the flexible functions.

Usage

data(Glucose)

Format

data.frame with 206 observations and 2 variables.

References

Add Legend to Right Margin.

Description

This function accepts all parameters applicable in and forwards them to function `legend`. There will be only made some modifications to the X-coordinate ensuring that the legend is plotted in the right margin of the graphic device. Make sure that you have reserved sufficient space in the right margin, e.g. `plot.VFP(....., mar=c(4,5,4,10))`.

Usage

```r
legend.rm(
  x = c("center", "bottomright", "bottom", "bottomleft", "left", "topleft", "top", 
         "topright", "right"),
  y = NULL,
  offset = 0.05,
  ...
)
```

Arguments

- `x` (character, numeric) either one of the character strings "center", "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" or a numeric values specifying the X-coordinate in user coordinates
- `y` (numeric) value specifying the Y-coordinate in user coordinates, only used in case `x` is numeric
- `offset` (numeric) value in [0, 0.5] specifying the offset as fraction in regard to width of the right margin
- `...` all parameters applicable in function `legend`

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```r
library(VCA)
data(VCAdata1)
# perform VCA-analysis
lst <- anovaVCA(y=(device+lot)/day/run, VCAdata1, by="sample")
# transform list of VCA-objects into required matrix
mat <- getMat.VCA(lst) # automatically selects "total"
mat

# fit all 9 models batch-wise
```
res <- fit.vfp(model.no=1:10, Data=mat)

plot(res, mar=c(5.1, 4.1, 4.1, 15), Crit=NULL)

legend.rm(cex=1.25, text.font=10,
legend=c(
  paste0("AIC: ", signif(as.numeric(res$AIC["Model_6"])), 4)),
  paste0("Dev: ", signif(as.numeric(res$Deviance["Model_6"])), 4)),
  paste0("RSS: ", signif(as.numeric(res$RSS["Model_6"])), 4)))

MultiLotReproResults  Result of a Real-World Precision Experiment on 10 Samples

**Description**

(data.frame) representing the results of an imprecision experiment. There are 10 observations and four variables which can be used directly as input for function 'fit.vfp'. Different samples are indicated by rownames.

**Usage**

data(MultiLotReproResults)

**Format**

data.frame with 10 observations and 4 variables.

---

**plot.VFP**  Plot VFP-Objects.

**Description**

Function takes an object of class 'VFP' and plots a fitted variance-function either on the original variance-scale ('type="vc"') or on the CV-scale ('cv'). The corresponding 100x(1-alpha)% confidence interval around the variance-function can be plotted either as lines ('ci.type="lines"') or as per default as CI-band.

**Usage**

```r
## S3 method for class 'VFP'
plot(
  x,
  model.no = NULL,
  type = c("vc", "sd", "cv"),
  add = FALSE,
  ```
alpha = 0.05,
ci.col = "gray90",
ci.type = c("band", "lines", "none"),
dispersion = NULL,
browse = FALSE,
BG = "white",
Title = list(),
Xlabel = list(),
Ylabel = list(),
Line = list(),
Points = list(),
Grid = list(),
Crit = list(),
ylim = NULL,
Prediction = NULL,
Pred.CI = NULL,
Model = TRUE,
CI.method = c("chisq", "t", "normal"),
use.log = FALSE,
Npred = 200,
...
)

Arguments

x (VFP) object as returned by function 'fit.vfp'
model.no (integer) specifying which model to plot, must be one of the fitted models
type (character) either "vc" to generate a plot on the original variance-scale or "cv" to plot it on the coefficient of variation scale, i.e. CV = 100*sqrt(VC)/Mean or "log" to plot after a variance stabilizing transformation. The latter will not work if 'Prediction' is specified!!.
add (logical) TRUE = the current (im)precision profile is added to an existing plot, FALSE = a new plot will be generated
alpha (numeric) value specifying the 100x(1-alpha)% confidence interval to be plotted around the function
ci.col (character) string specifying a color used for the CI-region
ci.type (character) either "band" to plot the CI as polygon, or "lines" to plot the CI-bounds as two separate line using color 'ci.col'
dispersion (numeric) NULL = the dispersion parameter will be estimated, numeric value = the dispersion parameter will be used as specified
browse (logical) TRUE = if multiple models were fitted, all will be displayed one after the other in increasing order of their respective AIC, mouse-klick on the plot triggers switch to the next model
BG (character) string specifying a background color
Title (list) passed to function mtext controlling content and style of the main title
Xlabel (list) passed to function mtext controlling the labeling of the X-axis
Ylabel (list) passed to function `mtext` controlling the labeling of the Y-axis

Line (list) passed to function `lines` controlling the visual appearance of the line representing the fitted variance-function

Points (list) passed to function `points` controlling how data used to fit a variance function shall be plotted

Grid (list) passed to function `addGrid` controlling the appearance of a grid, set to `NULL` to omit

Crit (list) passed to function `legend` per default used to present the optimality criterion for choosing the best fitting model, per default this is AIC and additionally residual sum of squares (RSS) is shown for the plotted model in the upper-right corner

ylim (numeric) vector of length two specifying plot-limits in Y-direction, if `NULL` these will be automatically determined

Prediction (list) with elements 'y' specifying values on VC-, SD- or CV-scale depending on 'type' for which predictions on the X-axis are requested or 'x' specifying mean-values on the X-axis for which predictions on the Y-axis are requested; furthermore, all graphical parameters accepted by function `lines` indicating predictions; `NULL` to omit (default); additionally to arguments accepted by function 'lines', one can specify parameters 'x.line' and 'y.line' used to indicated values at the respective axis in margin-lines see `mtext` for details; 'cex' and 'font' for specifying how these values are plotted. The same color as lines is used per default but can be changed setting 'text.col'. Can also be (numeric) vector (not a list) which results in using default graphical settings. See also parameter Pred.CI.

Pred.CI (list) with all parameters accepted by function `rect`: if not `NULL`, the 100x(1-alpha)% CI of predicted values will be added as a semi-transparent rectangle per default (see examples).

Model (logical) TRUE = plots the fitted model as subtitle below the main title, FALSE = omits this

CI.method (character) one of "t", "normal", "chisq" specifying which CI-method to use for deriving confidence intervals

use.log (logical) TRUE = X- and Y-axis will be log-transformed

Npred (integer) specifying the number of data points used to plot the fitted model, the larger the smoother (maybe slower if too large)

... additional parameters passed forward

Value (matrix) of predictions at user-specified X- or Y-coordinates is invisibly return in case Prediction is not `NULL`

Author(s)

Andre Schuetzemeister <andre.schuetzemeister@roche.com>
See Also

fit.vfp, predict.VFP, predictMean

Examples

library(VCA)
data(VCAdata1)
lst <- anovaVCA(y~(device+lot)/day/run, VCAdata1, by="sample")
mat <- getMat.VCA(lst) # automatically selects "total"
mat

res <- fit.vfp(model.no=1:10, Data=mat)
plot(res)
plot(res, type="cv")
plot(res, type="cv", ci.type="lines", ci.col="red",
     Grid=list(col="wheat"), Points=list(pch=2, lwd=2, col="black"))

# same for repeatability
mat.err <- getMat.VCA(lst, "error")
res.err <- fit.vfp(1:10, Data=mat.err)
plot(res.err)

# add predictions to plot, e.g. functional sensitivity
plot(res.err, type="cv", xlim=c(0, 4), Prediction=10)

# variability at X-values are of interest
plot(res.err, type="cv", xlim=c(0, 4), Prediction=list(x=0.5))

# one can specify X- and Y-values in the "Prediction" list-argument
plot(res.err, type="cv", xlim=c(0, 4),
     Prediction=list(x=c(0.25, 0.5), y=15))

#######################################################################
# another example using CA19_9 data from CLSI EP05-A3

data(CA19_9)

# fit reproducibility model to data
fits.CA19_9 <- anovaVCA(result~site/day, CA19_9, by="sample")

# fit within-laboratory-model treating site as fixed effect
fits.ip.CA19_9 <- anovaMM(result~site/(day), CA19_9, by="sample")

# the variability "between-site" is not part of "total"
fits.ip.CA19_9[[1]]
fits.CA19_9[[1]]

# extract repeatability
rep.CA19_9 <- getMat.VCA(fits.CA19_9, "error")

# extract reproducibility
repro.CA19_9 <- getMat.VCA(fits.CA19_9, "total")

# extract intermediate-precision (within-lab)
ip.CA19_9 <- getMat.VCA(fits.ip.CA19_9, "total")

# fit model \((a+bX)^c\) (model 8) to all three matrices
mod8.repro <- fit.vfp(repro.CA19_9, 8)
mod8.ip <- fit.vfp(ip.CA19_9, 8)
mod8.rep <- fit.vfp(rep.CA19_9, 8)

# plot reproducibility precision profile first
# leave enough space in right margin for a legend
plot(mod8.repro, mar=c(5.1, 7, 4.1, 15),
     type="cv", ci.type="none", Model=FALSE,
     Line=list(col="blue", lwd=3),
     Points=list(pch=15, col="blue", cex=1.5),
     xlim=c(10, 450), ylim=c(0,10),
     Xlabel=list(text="CA19-9, kU/L (LogScale) - 3 Patient Pools, 3 QC Materials", cex=1.5),
     Title=NULL,
     Ylabel=list(text="% CV", cex=1.5),
     Grid=NULL, Crit=NULL, log="x")

# add intermediate precision profile
plot (mod8.ip, type="cv", add=TRUE, ci.type="none",
     Points=list(pch=16, col="deepskyblue", cex=1.5),
     Line=list(col="deepskyblue", lwd=3), log="x")

# add repeatability precision profile
plot(mod8.rep, type="cv", add=TRUE, ci.type="none",
     Points=list(pch=17, col="darkorchid3", cex=1.5),
     Line=list(col="darkorchid3", lwd=3), log="x")

# add legend to right margin
legend.rm(x="center", pch=15:17, col=c("blue", "deepskyblue", "darkorchid3"),
         cex=1.5, legend=c("Reproducibility", "Within-Lab Precision", "Repeatability"), box.lty=0)

# repeatability precision profile with some beautifications
plot(mod8.rep, BG="darkgray",
     Points=list(pch=17, cex=1.5, col="blue"), Line=list(col="blue"),
     Grid=list(x=seq(0, 400, 50), y=seq(0, 100, 10), col="white"),
     Xlabel=list(cex=1.5, text="CA19-9 [U/mL]", col="blue"),
     Ylabel=list(cex=1.5, text="Repeatability on Variance-Scale", col="blue"),
     Crit=list(text.col="white", text.font=2, cex=1.25))
**powfun3**

**Description**
Internal Function Model 2.

**Usage**
powfun2simple(x)

**Arguments**

\[
x\quad \text{(numeric) parameter}
\]

---

**powfun3**  
*Internal Function Model 3.*

**Description**
Internal Function Model 3.

**Usage**
powfun3(x)

**Arguments**

\[
x\quad \text{(numeric) parameter}
\]

---

**powfun3simple**  
*Internal Function Model 3.*

**Description**
Internal Function Model 3.

**Usage**
powfun3simple(x)

**Arguments**

\[
x\quad \text{(numeric) parameter}
\]
**powfun4**  
*Internal Function Model 4.*

**Description**  
Internal Function Model 4.

**Usage**  
`powfun4(x, potenz)`

**Arguments**

- **x**  
  (numeric) parameter 1
- **potenz**  
  (numeric) parameter 2

---

**powfun4simple**  
*Internal Function Model 4.*

**Description**  
Internal Function Model 4.

**Usage**  
`powfun4simple(x, potenz)`

**Arguments**

- **x**  
  (numeric) parameter 1
- **potenz**  
  (numeric) parameter 2

---

**powfun5**  
*Internal Function Model 5.*

**Description**  
Internal Function Model 5.

**Usage**  
`powfun5(x)`

**Arguments**

- **x**  
  (numeric) parameter 1
**powfun5simple**  
*Internal Function Model 5.*

**Description**  
Internal Function Model 5.

**Usage**  
`powfun5simple(x, K)`  

**Arguments**  
- `x` (numeric) parameter 1  
- `K` (numeric) parameter 2

---

**powfun6**  
*Internal Function Model 6.*

**Description**  
Internal Function Model 6.

**Usage**  
`powfun6(x)`  

**Arguments**  
- `x` (numeric) parameter 1

---

**powfun6simple**  
*Internal Function Model 6.*

**Description**  
Internal Function Model 6.

**Usage**  
`powfun6simple(x)`  

**Arguments**  
- `x` (numeric) parameter 1
Internal Function Model 7.

**Description**

Internal Function Model 7.

**Usage**

powfun7(x)

**Arguments**

x (numeric) parameter 1

Internal Function Model 7.

**Description**

Internal Function Model 7.

**Usage**

powfun7simple(x)

**Arguments**

x (numeric) parameter 1

Internal Function Model 8.

**Description**

Internal Function Model 8.

**Usage**

powfun8(x, C, signJ)

**Arguments**

x (numeric) parameter 1
C (numeric) parameter 2
signJ (numeric) parameter 3
powfun8simple

**Internal Function Model 8.**

**Description**
Internal Function Model 8.

**Usage**
powfun8simple(x)

**Arguments**
x (numeric) parameter 1

---

powfun9simple

**Internal Function Model 9.**

**Description**
Internal Function Model 9.

**Usage**
powfun9simple(x)

**Arguments**
x (numeric) parameter 1

---

precisionPlot

**Precision Performance Plot of Qualitative Tests.**

**Description**
This function visualizes what is described in the CLSI EP12 guideline for qualitative test with internal continuous response (ICR). The hit rate, i.e. the number of measurements deemed to have a certain condition. The C5 and C95 concentrations will be derived per default by this function but it can be set to any set of hit rates. The histograms representing normal distribution of imprecisions at specific concentrations will be scaled to nicely fit into the plot, i.e. the area under the plot will not be equal to 1.
Usage

```r
precisionPlot(
  vfp,
  model.no = NULL,
  cutoff,
  prob = c(0.05, 0.5, 0.95),
  col = c("blue", "black", "red"),
  Cutoff = list(),
  Title = list(),
  Xlabel = list(),
  Ylabel = list(),
  HRLine = list(),
  Legend = FALSE,
  nclass = -1,
  BG = "gray90",
  digits = 3,
  alpha = 0.15,
  alpha2 = 0,
  xlim = NULL,
  col.grid = "white",
  Nrand = 1e+06
)
```

Arguments

- **vfp** (VFP) object modeling imprecision over the measuring range
- **model.no** (integer) specifying the VFP-model to used
- **cutoff** (numeric) specifying one or two cutoff(s), the latter will implicitly define an equivical zone with implications on how 'prob' will be interpreted (see 'prob' for details)
- **prob** (numeric) values 0 < x < 1 specifying coverage probability of an respective normal distribution at cutoff, in case of two cutoffs all elements of 'prob' < 0.5 will be evaluated in regard to cutoff 1, and all 'prob' > 0.5 in regard to cutoff 2
- **col** (character) strings specifying colors of the different distributions, which will be plotted semi-transparent using 'alpha1' for specifying the level of transparency (1=opaque, 0=fully transparent)
- **Cutoff** (list) specifying all parameters of the abline function. Vertical lines representing one or two cutoffs can be specified, the color will be re-used for a label in the upper margin. Set to NULL to omit.
- **Title** (list) specifying all parameters applicable in function mtext for specifying a main title of the plot
- **Xlabel** (list) specifying all parameters applicable in function mtext for specifying the X-axis label of the plot
- **Ylabel** (list) specifying all parameters applicable in function mtext for specifying the Y-axis label of the plot
**Examples**

```r
## Not run:
# perform variance component analysis
library(VCA)
data(VCAdata1)
# perform VCA-analysis
lst <- anovaVCA(y~(device+lot)/day/run, VCAdata1, by="sample")
# transform list of VCA-objects into required matrix
mat <- getMat.VCA(lst) # automatically selects "total"
mat
# fit all models batch-wise, the best fitting will be used automatically
res <- fit.vfp(model.no=1:10, Data=mat)
# plot hit and visualize imprecision using default settings
precisionPlot(res, cutoff=20)
# without normal distribution at cutoff do
precisionPlot(res, cutoff=20, prob=c(.05, .95), col=c("blue", "red"))
# highlight the proportion > cutoff (hit rate) more
precisionPlot(res, cutoff=20, prob=c(.05, .95), col=c("blue", "red"), alpha2=.5)
# plot with legend
precisionPlot(res, cutoff=20, prob=c(.05, .95), col=c("blue", "red"), alpha2=.5, Legend=TRUE)
# use different probabilities and colors
precisionPlot(res, cutoff=20, prob=c(.05, .95), col="black", alpha2=.3)

# now using two cutoffs, i.e. with equivocal zone
precisionPlot( res, cutoff=c(17, 19), prob=c(.05, .95), col=c("mediumblue", "red3"),
alpha2=.5, HRLine=list(col=c("mediumblue", "red3")))
```

---

**Author(s)**

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>
predict.modelEP17  
*Predict Method for Objects of Class* `modelEP17`.

**Description**

This is a helper function not intended to be used directly.

**Usage**

```r
## S3 method for class 'modelEP17'
predict(
  object,
  newdata = NULL,
  alpha = 0.05,
  ci.type = c("vc", "sd", "cv"),
  CI.method = c("chisq", "t", "normal"),
  use.log = FALSE,
  ...
)
```

**Arguments**

- `object` (object) of class `modelEP17`
- `newdata` (numeric) vector of data points at which prediction shall be made
- `alpha` (numeric) value defining the 100(1-alpha)% confidence interval for predictions
- `ci.type` (character) string specifying on which scale prediction shall be made
- `CI.method` (character) string specifying which type of CI shall be used
- `use.log` (logical) TRUE X- and Y-values will be returned on log-scale
- `...` additional arguments

**Author(s)**

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>
predict.VFP

**Predict Method for Objects of Class 'VFP'.**

## Description

Predict Method for Objects of Class 'VFP'.

## Usage

```r
## S3 method for class 'VFP'
predict(
  object,
  model.no = NULL,
  newdata = NULL,
  alpha = 0.05,
  dispersion = NULL,
  type = c("vc", "sd", "cv"),
  CI.method = c("chisq", "t", "normal"),
  use.log = FALSE,
  ...
)
```

## Arguments

- **object** (object) of class "VFP"
- **model.no** (integer) specifying a fitted model stored in 'object'
- **newdata** (numeric) optionally, a vector specifying mean-values for which predictions on the user-defined scale ('type') are requested. If omitted, fitted values will be returned.
- **alpha** (numeric) value specifying the 100 x (1-alpha)% confidence interval of predicted values
- **dispersion** (numeric) NULL = the dispersion parameter will be estimated, numeric value = the dispersion parameter will be used as specified
- **type** (character) specifying on which scale the predicted values shall be returned, possible are "vc" = variance, "sd"=standard deviation, "cv"=coefficient of variation
- **CI.method** (character) one of "t", "normal", "chisq" specifying which CI-method to use
- **use.log** (logical) TRUE = X- and Y-axis will be log-transformed
- **...** additional parameters passed forward to function `predict.gnm`

## Value

(data.frame) with numeric variables:

- Mean value at which predictions were requested
predictMean

Fitted prediction at 'Mean'
SE standard error of prediction
Scale residual scale
LCL lower confidence limit of the 100x(1-'alpha')% CI
UCL upper confidence limit of the 100x(1-'alpha')% CI

Examples

library(VCA)
data(VCAdata1)
lst <- anovaVCA(y=(device+lot)/day/run, VCAdata1, by="sample")
mat <- getMat.VCA(lst) # automatically selects "total"
res <- fit.vfp(model.no=1:10, Data=mat)
predict(res)
predict(res, dispersion=0.95)

---

predictMean

Finding X-Value for Given Y-Value Using a Bisection-Approach.

Description

For given variability-values (Y-axis) on one of three scales (see 'type'), those values on the X-axis are determined which give fitted values equal to the specification.

Usage

predictMean(
  obj,
  type = c("vc", "sd", "cv"),
  model.no = NULL,
  alpha = 0.05,
  newdata = NULL,
  tol = 1e-04,
  ci = TRUE,
  ...
)

Arguments

obj (object) of class 'VFP'
type (character) "vc" = variance, "sd" = standard deviation = sqrt(variance), "cv" = coefficient of variation
model.no (integer) specifying which model to use in case 'obj' represents multiple fitted models
**predictMean**

- **alpha**  
  (numeric) value specifying the 100x(1-alpha)% confidence interval for the predicted value(s)

- **newdata**  
  (numeric) values representing variability-values on a specific scale ('type')

- **tol**  
  (numeric) tolerance value relative to 'newdata' specifying the stopping criterion for the bisection algorithm, also used to evaluate equality of lower and upper bounds in a bisection step for checking whether a boundary can be determined or not

- **ci**  
  (logical) indicates whether confidence intervals for predicted concentrations are required (TRUE) or not (FALSE), if 'newdata' contains many values the overall computation time can be minimized to 1/3 leaving out runs of the bisection-algorithm for LCL and UCL

- **...**  
  additional parameter passed forward or used internally

**Details**

This is achieved using a bisection algorithm which converges according to the specified tolerance 'tol'. In case of 'type="cv"', i.e. if specified Y-values are coefficients of variation, these are interpreted as percentages (15 = 15%).

**Value**

(data.frame) with variables "Mean" (X-value), "VC", "SD" or "CV" depending on 'type', "Diff" the difference to the specified Y-value, "LCL" and "UCL" as limits of the 100x(1-alpha)% CI.

**Author(s)**

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

**See Also**

fit.vfp, predict.VFP, plot.VFP

**Examples**

```r
# perform variance component analyses first
library(VCA)
data(CA19_9)
fits.CA19_9 <- anovaVCA(result~site/day, CA19_9, by="sample")

# extract repeatability
mat.CA19_9 <- getMat.VCA(fits.CA19_9, "error")
res.CA19_9 <- fit.vfp(mat.CA19_9, 1:10)
summary(res.CA19_9)
print(res.CA19_9)

# predict CA19_9-concentration with 5\% CV
predictMean(res.CA19_9, newdata=5)
```
# this is used in function plot.VFP as well
plot(res.CA19_9, Prediction=list(y=5), type="cv")
plot(res.CA19_9, Prediction=list(y=5), type="cv",
xlim=c(0, 80), ylim=c(0, 10))

print.VFP

## S3 method for class \'VFP\'
print(x, model.no = NULL, digits = 4, ...)

Arguments

x (object) of class 'VFP'
model.no (integer) specifying a fitted model in 'x', if NULL the best fitting model will be printed, i.e. the one with min(AIC)
digits (integer) number of significant digits
...

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

library(VCA)
data(CA19_9)
fits.CA19_9 <- anovaVCA(result~site/day, CA19_9, by="sample")
# extract repeatability
mat.CA19_9 <- getMat.VCA(fits.CA19_9, "error")
res.CA19_9 <- fit.vfp(mat.CA19_9, 1:10)
res.CA19_9
RealData1

**Result of a Real-World Precision Experiment on 10 Samples**

**Description**

(data.frame) representing the results of an imprecision experiment. There are 10 observations and six variables which can be used directly as input for function `fit.vfp`. Different samples are indicated by variable "No".

**Usage**

```r
data(RealData1)
```

**Format**

data.frame with 10 observations and 6 variables.

---

ReproData

**Serum Work Area Example Data of a Reproducibility Experiment**

**Description**

Multi-site precision experiment using 6 human serum-samples and 2 precision control samples, which were all measured at each site on 12 days, with 2 runs per day and 2 replicates per run, i.e. there are 8 (sample) x 3 (sites/labs) x 12 (days) x 2 (runs) x 2 (replicates) = 1152 observations overall.

**Usage**

```r
data(ReproData)
```

**Format**

data.frame with 1152 observations and 5 variables.
Signif  

Adapted Version of Function 'signif'

Description

This function adapts base-function `signif` by always returning integer values in case the number of requested significant digits is less than the the number of digits in front of the decimal separator.

Usage

`Signif(x, digits = 4, force = TRUE, ...)`

Arguments

- `x` (numeric) value to be rounded to the desired number of significant digits
- `digits` (integer) number of significant digits
- `force` (logical) TRUE = force the return value to have at least 4 significant digits, i.e. to integers with less digits zeros will be appended after the decimal separator, otherwise the return value will be casted from character to numeric
- `...` additional parameters

Value

number with 'digits' significant digits, if 'force=TRUE' "character" objects will be returned otherwise objects of mode "numeric"

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

summary.VFP  

Summary Objects of Class 'VFP'

Description

Summary Objects of Class 'VFP'

Usage

## S3 method for class 'VFP'
summary(  
  object,  
  model.no = NULL,  
  digits = 4,  
  type = c("simple", "complex"),  
  ...  
)
t.coef

Transformation of Coefficients.

Description

This function performs transformation from the original parameterization into the 'VFP'-package internal re-parameterized form.

Usage

```r
## S3 method for class 't'

fcoefs0,
K = NULL,
Maxi = NULL,
model = NULL,
signJ = NULL,
eps = sqrt(.Machine$double.eps),
...
```
Arguments

- `coeffs0` (numeric) vector of function coefficients to be transformed into the re-parameterized form
- `K` (numeric) constant value 'K'
- `Maxi` (numeric) max. value
- `model` (integer) specifying which model shall be back-transformed
- `signJ` (integer) either 1 or -1
- `eps` (numeric) constant used instead of zero in case of log-transformation
- `...` additional parameters

Details

In the 'VFP' package models are re-parameterized to have better control over the constrained solution-space, i.e. only models may be fitted generating non-negative fitted values. This function is intended to be for internal use only.

Value

(numeric) vector of coefficients in re-parameterized form

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com> Florian Dufey <florian.dufey@roche.com>

Example Data T4S9_99.VFP (T4 RIA) from the Variance Function Program 12.0 from Sadler

Description

Clinical specimen T4 RIA duplicates produced during 1999 by a staff member.

Usage

data(T4S9_99)

Format

data.frame with 8553 observations and 2 variables.

References

VFP Program Version 12.0\ http://www.aacb.asn.au/professionaldevelopment/useful-tools
Index

* datasets
  B2mIntra_98, 5
  Glucose, 16
  MultiLotReproResults, 18
  RealData1, 35
  ReproData, 35
  T4S9_99, 38

* package
  VFP-package, 2

abline, 28
addGrid, 4, 20
as.rgb, 4
B2mIntra_98, 5
bt.coef, 6
coeff.VFP, 6
conditionHandler, 7
deriveCx, 8
fit.EP17, 9
fit.vfp, 3, 11, 21, 33
getMat.VCA, 12, 15
Glucose, 16
legend, 17, 20
legend.rm, 17
lines, 20, 29
mtext, 19, 20, 28
MultiLotReproResults, 18
plot.VFP, 3, 13, 18, 33
points, 20
powfun2simple, 22
powfun3, 23
powfun3simple, 23
powfun4, 24
powfun4simple, 24
powfun5, 24
powfun5simple, 25
powfun6, 25
powfun6simple, 25
powfun7, 26
powfun7simple, 26
powfun8, 26
powfun8simple, 27
powfun9simple, 27
precisionPlot, 27
predict.gnm, 31
predict.modelEP17, 30
predict.VFP, 3, 12, 13, 21, 31, 33
predictMean, 3, 12, 13, 21, 32
print.VFP, 3, 34
RealData1, 35
rect, 20
ReproData, 35
Signif, 36
signif, 36
summary.VFP, 3, 36
t.coef, 37
T4S9_99, 38
VFP (VFP-package), 2
VFP-package, 2