

# Package ‘Mifuns’

February 14, 2012

**Type** Package

**Title** Pharmacometric tools for data preparation, modeling, simulation, and reporting

**Version** 5.1

**Date** 2011-09-07

**Author** Metrum Institute (<http://metruminstitute.org>): Bill Knebel, Leonid Gibianski, Tim Bergsma

**Maintainer** Tim Bergsma <[timb@metrumrg.com](mailto:timb@metrumrg.com)>

**Depends** reshape, methods, lattice, grid, XML, MASS

**Suggests** chron, fork

**Description** Pharmacometric tools for common data preparation tasks, stratified bootstrap resampling of data sets, NONMEM control stream creation/editing, NONMEM model execution, creation of standard and user-defined diagnostic plots, execution and summary of bootstrap and predictive check results, implementation of simulations from posterior parameter distributions, reporting of output tables and creation of detailed analysis logs. **Deprecated.** Please see the replacement package: `metrumrg`.

**License** GPL

**Repository** CRAN

**Date/Publication** 2011-09-08 04:09:59

## R topics documented:

Mifuns-package . . . . .	5
accept . . . . .	6
acceptance . . . . .	7
align.decimal . . . . .	8
as.comment . . . . .	9
as.data.frame.block . . . . .	10
as.flag . . . . .	11

as.keyed . . . . .	13
as.nm . . . . .	15
as.nmctl . . . . .	18
as.pxml.ext . . . . .	20
as.unilog.run . . . . .	21
as.xml . . . . .	22
attribute . . . . .	24
AUC . . . . .	25
aug . . . . .	26
backtrans . . . . .	27
bin . . . . .	27
bracket . . . . .	29
breaks . . . . .	30
c.miTemporal . . . . .	31
check.subjects . . . . .	33
CLNR . . . . .	34
closers . . . . .	35
colname<- . . . . .	36
command . . . . .	37
compileflag . . . . .	38
compiler . . . . .	39
compute.cwres . . . . .	40
config . . . . .	47
constant . . . . .	48
contains . . . . .	49
css . . . . .	50
ctl . . . . .	51
dataFormat . . . . .	52
dataSynthesis . . . . .	53
deranged . . . . .	55
diagnosticPlots . . . . .	57
electronicAppendix . . . . .	59
episcript . . . . .	60
filename . . . . .	61
first . . . . .	62
fixedwidth . . . . .	63
fixProblem . . . . .	65
fTable2data.frame . . . . .	67
getCovs . . . . .	68
getCwres . . . . .	69
getdname . . . . .	70
getPars . . . . .	70
getTabs . . . . .	71
glue . . . . .	72
groupnames . . . . .	73
half.matrix . . . . .	74
hash . . . . .	75
helpAdminister . . . . .	76

helpClasses	76
helpDataFrame	77
helpList	77
helpMatrix	88
helpModel	88
helpPharmacometric	89
helpPrepare	90
helpQuantify	90
helpReport	91
helpSimulate	91
helpStrategic	92
helpVector	92
helpVisualize	93
ibw	93
inner.data.frame	94
is.alpha	96
is.latex.token	97
is.one.nonalpha	98
is.square.matrix	99
isSubversioned	100
isSubversionedFile	101
iterations	101
justUnits	103
latest	105
latex.args	106
latex.options	107
locf	108
lookup	109
ltable	110
ltable.data.frame	111
map	112
metaSub.character	113
miTemporal	117
naInContext	123
nest	124
nix	125
nm.pl	126
nmPlots	128
nmVersion	128
NONR	129
omegacor	134
Ops.keyed	136
ord.matrix	137
packageCheck	138
panel.densitystrip	139
panel.hist	144
panel.stratify	145
params	147

parens . . . . .	148
partab . . . . .	149
plot.nm . . . . .	151
plotfilename . . . . .	152
PLOTR . . . . .	153
posmat . . . . .	156
prev . . . . .	157
purge.dir . . . . .	158
purge.files . . . . .	159
qsub . . . . .	159
reapply . . . . .	160
resample.data.frame . . . . .	161
rinvchisq . . . . .	164
riwish . . . . .	164
rlog . . . . .	165
row2tabular . . . . .	167
runCommand . . . . .	168
runlog . . . . .	170
runNonmem . . . . .	173
runstate . . . . .	175
safe.call . . . . .	177
safeQuote . . . . .	178
setCwres . . . . .	179
shuffle . . . . .	179
simblock . . . . .	180
simpar . . . . .	181
snap . . . . .	183
spaces . . . . .	185
sqrtn . . . . .	186
stableMerge . . . . .	186
star . . . . .	187
strain . . . . .	188
summary.nm . . . . .	188
svnIsText . . . . .	191
svnMarkAsNonText . . . . .	192
svnMarkAsText . . . . .	193
svnMimeType . . . . .	194
svnPropGet . . . . .	195
svnPropGetFile . . . . .	196
svnPropSet . . . . .	197
svnPropSetFile . . . . .	198
svnSetMimeType . . . . .	199
synthesis . . . . .	200
tabular . . . . .	201
tabular.data.frame . . . . .	202
tabularformat . . . . .	204
tagvalue . . . . .	205
text2decimal . . . . .	206

Tmax . . . . .	207
Tmin . . . . .	208
variants . . . . .	209
wikiparse . . . . .	210
win . . . . .	211
wrap . . . . .	212
xyplot.ext . . . . .	213

<b>Index</b>	<b>216</b>
--------------	------------

---

MIfuns-package	<i>Pharmacometric Tools for Data Preparation, Modeling, Simulation, and Reporting</i>
----------------	---------------------------------------------------------------------------------------

---

## Description

This package includes pharmacometric tools for common data preparation tasks, stratified bootstrap resampling of data sets, NONMEM control stream creation/editing, NONMEM model execution, creation of standard and user-defined diagnostic plots, execution and summary of bootstrap and predictive check results, implementation of simulations from posterior parameter distributions, reporting of output tables, and creation of detailed analysis logs.

Deprecated. Please see the replacement package **metrumrg**.

## Details

I want to ....

- [prepare](#) or manipulate data
- [model](#) data in NONMEM
- [simulate](#) from existing models
- [plot](#) model inputs or outputs
- [calculate](#) general statistics
- [report](#) modeling and simulation results systematically
- [administer](#) related files and file systems
- [list](#) all functions

Package: MIfuns  
 Type: Package  
 Version: 5.1  
 Date: 2011-09-07  
 License: GPL

**Author(s)**

Developed by Metrum Institute (Bill Knebel, Tim Bergsma, Leonid Gibianski, others). Maintainer: Tim Bergsma <timb@metrumrg.com>

**References**

<http://mifuns.googlecode.com>

---

accept

*Document Acceptance of an R Installation*

---

**Description**

Execution of `accept` is a procedural act pertinent to installation qualification. The function logs an act of acceptance of the current installation, noting the conditions of acceptance. Currently, the conditions include the login of the acceptor, the time of acceptance (GMT) and any contingent packages that survive `packageCheck`.

**Usage**

```
accept(  
  contingencies = c(  
    'akima',  
    'boot',  
    'coda',  
    'chron',  
    'foreign',  
    'fork',  
    'lattice',  
    'locfit',  
    'MASS',  
    'nlme',  
    'plyr',  
    'R2WinBUGS',  
    'reshape',  
    'SASxport',  
    'survival'  
  ),  
  installMissing=TRUE  
)
```

**Arguments**

`contingencies` a character vector of packages that must survive `packageCheck` in order for acceptance to succeed. Can be NULL. Defaults are arbitrarily chosen to suit the needs of Metrum Research Group, LLC.

`installMissing` scalar logical: should `install.packages` be attempted for missing contingencies?

**Details**

The file 'accept.xml' in .Library is created if it does not exist, and appended if it does. In addition to the acceptor's login and the time, names and versions of any contingent packages are stored.

If installMissing is TRUE, an attempt will be made to install packages not in the default library (.Library), before loading is attempted.

**Value**

an invisible named vector of version identifiers, where the names are package names.

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [packageCheck](#)
- [acceptance](#)

---

acceptance

*List the History of Acceptance.*

---

**Description**

This function reads 'accept.xml' in .Library, if it exists, and prints it with the default method.

**Usage**

```
acceptance()
```

**Details**

Data is displayed as XML markup.

**Value**

an object of class XMLNode

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [accept](#)
- [packageCheck](#)

---

align.decimal

*Format a Vector Aligning on Decimal*

---

**Description**

Formats a numeric vector so that decimal mark is a constant distance from the end of each element.

**Usage**

```
align.decimal(x, decimal.mark = ".", ...)
```

**Arguments**

x	numeric
decimal.mark	character indicating decimal
...	ignored

**Details**

Each element is formatted separately using `prettyNum`, then the character results are padded with spaces on the right, so that the decimals align. Whole numbers without the decimal mark get an extra space in its place.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [tabular.data.frame](#)

**Examples**

```
align.decimal(c(.12, 1.2, 12.0))
```

---

as.comment

*Create and Manipulate Comment Objects*


---

**Description**

Some conventions include the use of ‘C’ or ‘.’ as the first element in a row. ‘C’ is typically used to identify a row to be ignored “commented-out”. Here, a comment is really a logical that should never be NA, and prints as ‘C’ or ‘.’ (TRUE,FALSE).

**Usage**

```
## S3 method for class 'comment'
x[... , drop = TRUE]
## S3 method for class 'comment'
x[[... , drop = TRUE]]
## Default S3 method:
as.comment(x, ...)
## S3 method for class 'comment'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
## S3 method for class 'comment'
c(... , recursive = FALSE)
## S3 method for class 'comment'
format(x, ...)
## S3 method for class 'data.frame'
hide(x, where, why, ...)
## S3 method for class 'comment'
print(x, ...)
## S3 method for class 'comment'
rep(x, ...)
## S3 method for class 'data.frame'
hidden(x, ...)
## S3 method for class 'hidden.data.frame'
summary(object, ...)
```

**Arguments**

x	the comment object, or something coercible to logical (data.frame for hidden.data.frame)
...	extra arguments, generally ignored
drop	coerce to lowest possible dimension
row.names	row names for the result
optional	Optional to use the object name as the column name?
recursive	unused, present for historical reasons

where	logical with length nrow(x), indicating rows to hide
why	a column name suggesting the reason for hiding
object	hidden.data.frame

### Details

Objects of class `comment` can be used exactly like logicals, but look like conventional comments. `hide.data.frame` implements consistent commenting of rows, including a reason for the comment. `hidden` and `summary` methods analyze the result. `hidden.data.frame` returns just the hidden rows, classified as `'hidden.data.frame'`. `summary.hidden.data.frame` indicates, for each flag, the number of records commented for that reason (total) and the number of records commented for only that reason (unique), if any. If `hidden(x)` has no hide flags, the result is a data frame with one cell: `ncol=0`. If `hidden(x)` has no rows, the result is a data frame with one cell: `nrow=0`.

### Value

comment, or data.frame for `hide.data.frame`, etc.

### Examples

```
Theoph <- hide(Theoph, where=Theoph$conc < 1,why='blq')
Theoph <- hide(Theoph, where=Theoph$Subject==1,why='suspect')
hidden(Theoph)
summary(hidden(Theoph))
```

---

as.data.frame.block    *Convert a Block of Lines to Data Frame*

---

### Description

A block is a character vector where each element represents a line of text from a table. This function converts the character vector to a data.frame. The first line must be a header, and names will not be checked.

### Usage

```
## S3 method for class 'block'
as.data.frame(x, ...)
```

### Arguments

x	character
...	extra arguments passed to <code>read.table</code>

### Details

The function calls `read.table` with `header=TRUE`, `as.is=TRUE`, and `check.names=FALSE`. It is an error to specify these arguments.

**Value**

data.frame

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [read.table](#)

**Examples**

```
as.data.frame.block(  
  c(  
    'just an example',  
    'run param estimate',  
    '1 CL 45',  
    '1 V2 70',  
    '1 Ka 3.14'  
  )  
)
```

---

as.flag

*Create and Manipulate Flags*

---

**Description**

A flag is is an integer that may be imputed as zero where missing. These functions implement the class. Other functions may do the imputation as necessary.

**Usage**

```
## S3 method for class 'flag'  
x[... , drop = TRUE]  
## S3 method for class 'flag'  
x[[... , drop = TRUE]]  
f(x, ...)  
## S3 method for class 'flag'  
as.character(x, ...)  
## S3 method for class 'flag'  
as.data.frame(x, row.names = NULL, optional = FALSE, ...)  
## Default S3 method:  
as.flag(x, ...)
```

```
## S3 method for class 'flag'
as.flag(x, ...)
## S3 method for class 'flag'
c(..., recursive = FALSE)
## S3 method for class 'flag'
format(x, ...)
## S3 method for class 'flag'
print(x, ...)
## S3 method for class 'flag'
rep(x, ...)
```

### Arguments

x	the flag object, or something coercible to integer
...	extra arguments, ususally ignored
drop	coerce to lowest possible dimension
row.names	row names for the result
optional	Optional to use the object name as the column name?
recursive	unused, present for historical reasons

### Details

Typically a flag takes on the values of zero and 1, and is used to indicate the presence of a particular condition. Other functions are welcome to impute NAs as 0. Methods are defined for common classes. `f` is an alias to `as.flag`.

### Value

flag

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### Examples

```
hide(Theoph,where=Theoph$conc < 1,why='BLQ')
```

**Description**

The class *keyed* is a subclass of *data.frame* with a *key* attribute. The key is a vector of column names which, taken together, should provide enough information to uniquely distinguish each row. Specific functions and methods take advantage of this information.

**Usage**

```
## S3 method for class 'keyed'
x[...]
## S3 method for class 'keyed'
aggregate(x, by=x[,setdiff(key(x),across),drop=FALSE], FUN, across=character(0), ...)
## S3 method for class 'data.frame'
as.keyed(x, key, ...)
dupKeys(x, ...)
key(x, ...)
key(x) <- value
## S3 method for class 'keyed'
merge(x, y, ...)
naKeys(x, ...)
## S3 method for class 'keyed.summary'
print(x, ...)
## S3 method for class 'keyed'
sort(x, decreasing = FALSE, ...)
## S3 method for class 'keyed'
summary(object, ...)
## S3 method for class 'keyed'
transform('_data', ...)
## S3 method for class 'keyed'
uniKey(x,key=NULL,...)
```

**Arguments**

<code>x</code>	a (keyed) <i>data.frame</i>
<code>by</code>	a list of indices as long as <code>nrow(x)</code> , whose interaction gives the aggregates (groups)
<code>FUN</code>	an aggregating function
<code>across</code>	column names in <code>key(x)</code> across which to aggregate; see details
<code>...</code>	extra arguments, usually ignored, but passed to <code>FUN</code> in <code>aggregate</code>
<code>key</code>	a character vector of column names in <code>x</code> that should uniquely distinguish each row
<code>value</code>	a key (character vector of column names)

<code>y</code>	the right argument in the merge
<code>decreasing</code>	(coercible to) logical; length 1 or length <code>key(x)</code>
<code>object</code>	a keyed data.frame
<code>_data</code>	a keyed data.frame

## Details

The generic `as.keyed` is the usual way of creating a keyed object. The method `as.keyed.data.frame` calls `key<-`. The function `key` allows checking an object's key. A data.frame can be re-keyed by a subsequent call. Generally, a data.frame should be keyed on columns that actually exist, but this is not enforced.

In `aggregate.keyed`, the default behavior is to aggregate by the key columns, i.e., to eliminate duplicate keys by aggregation. `by` can be specified arbitrarily, but must be a named list (e.g., a data.frame) with each element as long as `nrow(x)`. Each element in `by` will displace any like-named element in `x`, and `names(by)` will serve as the key of the result. If `by` has length zero, (as it does by default when `across` is `key(x)`) the entire data set is aggregated into a one row data.frame.

`across` is a convenience argument to `aggregate.keyed`. If specified, it must be a subset of (or all of) `key(x)`. Columns indicated by `across` are dropped from `x` and from the default `by` value, and aggregation proceeds irrespective of those columns.

The function `naKeys` detects rows for which one or more key fields is NA.

The function `dupKeys` detects all rows for which there is another row (earlier or later) with identical key. That means it can never identify a single row, as `duplicated` can: it identifies the duplicates as well as those rows of which they are duplicates. It is recommended to test for NAs before testing for duplicates.

`order` treats `decreasing` as an all-or-nothing argument, whereas `sort.keyed` allows independent specification for each column.

Methods for `merge` and `transform` are key-friendly. The method for `summary` is key-centric. `uniKey` creates a single character vector from all the key columns.

## Value

Most functions and methods documented here return objects with class `c('keyed', 'data.frame')`.

`key<-` and methods for `summary` and `print` are used for side effects.

`unikey.keyed` returns a character vector as long as `nrow(x)`.

`naKeys` and `dupKeys` return logical vectors as long as `nrow(x)`.

## Author(s)

Tim Bergsma

## References

<http://mifuns.googlecode.com>

**See Also**

- [Ops.keyed](#)

**Examples**

```
a <- sort(as.keyed(Theoph,key=c('Subject','Time')))
summary(a)
aggregate(a, across='Time',FUN=mean)
```

as.nm

*Create and Manipulate nm Objects***Description**

Objects of class *nm* are intended to support analysis using the software NONMEM ((c), Icon Development Solutions). *nm* gives a zero-row data.frame with suitable columns and column classes (essentially, a template for dataset construction). *as.nm* and *as.nm.data.frame* construct an *nm* object from an existing object. The read and write methods are wrappers for '.csv' equivalents. *read.nm* reconstitutes classes for flags, DATETIME, and C. *merge.nm* coerces its result using *as.nm*, guaranteeing a consistent left hand object when using *Ops.keyed*.

**Usage**

```
nm()
## S3 method for class 'data.frame'
as.nm(x, ...)
## S3 method for class 'nm'
merge(x, y, ...)
read.nm(x,na.strings='.',as.is=TRUE,key=c('SUBJ','TIME','SEQ'),flags=character(0),...)
write.nm(x, file, na = '.', row.names = FALSE, quote = FALSE, ...)
```

**Arguments**

x	data.frame or nm
y	right merge argument
na.strings	passed to read.csv
as.is	passed to read.csv
key	passed to as.keyed
flags	character vector naming columns to convert using as.flag
...	extra arguments, ignored or passed to write.csv
file	passed to write.csv
na	passed to write.csv
row.names	passed to write.csv
quote	passed to write.csv

## Details

`as.nm.data.frame` is the principal method that creates an `nm` classification. It alone enforces all qualities of class `nm`. `read.nm` is the only other function that creates an `nm` classification; use with caution, as it does not enforce all qualities.

- `SUBJ` must be present and defined, even for commented records. `ID` is (re)calculated as `as.numeric(factor(SUBJ))`.
- `C` (class: comment) will be created if not present.
- `NA C` will be imputed `FALSE`.
- Every active (non-commented) source record should define exactly one of `HOUR` or `DATETIME`.
- `HOUR` is taken to represent relative accumulation of hours from arbitrary origin.
- `DATETIME` is understood as seconds, coercible to `miDateTime`.
- `TIME` is calculated from either `HOUR` or `DATETIME`.
- Definition (or not) of `HOUR` vs. `DATETIME` should be constant within subject (for active records).
- `SEQ` (class `flag`) will be created if not present.
- `nm` will be keyed on `SUBJ`, `TIME`, and `SEQ`. `SEQ` determines sort order for rows with matching `TIME`.
- Result will be sorted.
- `TIME` will be relativized to earliest extant value, incl. those in comments.
- `TAFD` (time after first dose), `TAD` (time since most recent dose), and `LDOS` (most recent dose) will be calculated if `AMT` is present.
- `TAD` will consider `ADDL` and `II` if present.
- `NA` flags will be imputed as zero.
- `MDV` (missing dependent value) will be calculated if `DV` is present, preserving non-`NA` `MDV`, if present.
- resulting column order will lead with `C` followed by key columns.

Column summary:

- required inputs: `SUBJ`; `HOUR` or `DATETIME`
- optional inputs: `AMT`, `ADDL`, `II`, `DV`
- enforced outputs: `SUBJ`, `ID`, `C`, `TIME`, `SEQ`
- conditional outputs: `TAFD`, `TAD`, `LDOS`, `MDV`

## Value

`write.nm` is used for side effects. Others return an object with class `c('nm', 'keyed', 'data.frame')`.

## Note

In the examples below, note that assembly chains beginning with `nm` will have class `nm` for every sub-result. This requires more evaluations of `as.nm` but is useful if later operations depend on imputed or calculated variables. Note, for example, that `-.moot` only makes sense if the left operand already resolves to class `nm`, since currently `moot.nm` is the only method defined for generic `moot`.

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [summary.nm](#)
- [Ops.keyed](#)

**Examples**

```
dose <- data.frame(
  SUBJ = rep(letters[1:3], each = 2),
  HOUR = rep(c(0,20),3),
  AMT = rep(c(40,60,80), each = 2)
)
dose <- as.keyed(dose,key=c('SUBJ','HOUR'))
samp <- data.frame(
  SUBJ = rep(letters[1:3], each = 4),
  HOUR = rep(c(0,10,20,30),3),
  DV = signif(rnorm(12),2) + 2
)
samp <- as.keyed(samp,key=c('SUBJ','HOUR'))
demo <- data.frame(
  SUBJ = letters[2:5],
  RACE = c('asian','white','black','other'),
  SEX = c('female','male','female','male'),
  WT = c(75, 70, 73, 68)
)
demo <- as.keyed(demo,key=c('SUBJ'))
meds <- as.keyed(
  data.frame(
    SUBJ=c('a','c'),
    HOUR=c(0,15),
    STOP=c(10,25),
    C3A4=as.flog(c(1,1))
  ),
  key=c('SUBJ','HOUR')
)

dose + samp
dose & samp
samp - dose
dose | demo
demo | dose

nm()
nm() + dose
```

```

as.nm(dose)
as.nm(dose + samp)
as.nm(dose + samp | demo) #as.nm executes once
nm() + dose + samp | demo #same result, but as.nm executes 4 times

meds
long <- deranged(meds,start='HOUR',stop='STOP')
long$EVID <- 2
nm() + dose + samp + long
nm() + dose + samp + long - as.moot()
nm() + dose + samp + as.rigged(n=10)
data <- nm() + transform(dose,EVID=1) + transform(samp,EVID=0) | demo
summary(data,by=c('EVID','SEQ'))

```

---

as.nmctl

---

*Create, Manipulate, Read, and Write NONMEM Control Streams*


---

## Description

This family of functions implements the class `nmctl`: an object model of the NONMEM control stream. `nmctl` models a control stream as a list of records; each record is a character vector. The read and write functions (not generic) convert `nmctl` to and from file format. The print, format, and `as.character` methods display `nmctl` as it normally looks in a text editor. `as.list.nmctl` simply unclasses its argument. `as.nmctl.character` does the heavy work, breaking up a character vector into records and storing as a list.

## Usage

```

## S3 method for class 'nmctl'
as.character(x, ...)
## S3 method for class 'nmctl'
as.list(x, ...)
## S3 method for class 'character'
as.nmctl(
  x,
  pattern='^ *\[^\ ]+(.*)?$',
  head='\1',
  tail='\2',
  ...
)
## S3 method for class 'nmctl'
format(x, ...)
## S3 method for class 'nmctl'
print(x, ...)
read.nmctl(con, ...)
write.nmctl(
  x,
  file = "data",

```

```

ncolumns = 1,
append = FALSE,
sep = " ",
...
)
## S3 method for class 'nmctl'
x[... , drop = TRUE]
## S3 method for class 'nmctl'
x[[... , drop = TRUE]]

```

### Arguments

x	an nmctl object (or analogous character vector)
...	extra arguments passed to other functions
pattern	regular expression for first line of a control record
head	regular expression (relative to pattern) giving the name of the control record
tail	regular expression (relative to pattern) giving the balance of the control record
con	a connection or the name of a file to open
file	passed to write
ncolumns	passed to write
append	passed to write
sep	passed to write
drop	coerce to lowest possible dimension

### Details

Serendipitously, the record indicator in NONMEM control stream syntax is the same as the element selector in R list syntax: `\$`. The convention is that names of elements in nmctl (lower case) are converted to record types (upper case) in the control stream. The user is free to add, delete, rearrange, and edit records using standard list manipulation techniques. When printed, records appear in list order. The write function warns if the 80 character limit is exceeded (not including comments).

### Value

as.character.nmctl	a character vector representing a control stream
as.list	a list representing a control stream
as.nmctl.character	a control stream object
format.nmctl	character
print.nmctl	character
read.nmctl	a control stream object
write.nmctl	used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [write](#)

---

as.pxml.ext

*Convert a Param File to XML*

---

**Description**

Convert the contents of a file to XML, encoding the leading lines as a ‘note’ element and the following lines as a table.

**Usage**

```
as.pxml.ext(file, lead = 1, tag = 'param', ...)
```

**Arguments**

file	path to a parameter file
lead	number of leading ‘note’ lines
tag	name for the enclosing element
...	ignored

**Details**

The ‘note’ and ‘body’ are identified, and coerced using `as.xml`. The result is nested in an element with the name specified by `tag`. The first column of ‘body’ is used as the key.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [as.xml](#)
- [xyplot.ext](#)

**Examples**

```

ext <- c(
'TABLE NO. 1: First Order: Goal Function=MINIMUM VALUE OF OBJECTIVE FUNCTION',
' ITERATION   THETA1      THETA2      THETA3      SIGMA(1,1)  OMEGA(1,1)  OBJ',
'      0 1.70000E+00 1.02000E-01 2.90000E+01 0.00000E+00 1.17001E+00 11.570086639848398',
'      2 1.78158E+00 1.06239E-01 3.05314E+01 0.00000E+00 1.08862E+00 9.377909428896904',
'      4 1.91182E+00 1.05179E-01 3.14289E+01 0.00000E+00 8.96680E-01 8.983605357031118',
'      6 1.94836E+00 1.01426E-01 3.20728E+01 0.00000E+00 9.06374E-01 8.940731060922468',
'      8 1.93983E+00 1.01742E-01 3.20128E+01 0.00000E+00 8.99988E-01 8.940110966224346',
'     10 1.94057E+00 1.01681E-01 3.20217E+01 0.00000E+00 8.99322E-01 8.940101673144566',
'     11 1.94057E+00 1.01681E-01 3.20217E+01 0.00000E+00 8.99322E-01 8.940101673144566',
' -1000000000 1.94057E+00 1.01681E-01 3.20217E+01 0.00000E+00 8.99322E-01 8.940101673144566',
' -1000000001 6.28499E-01 7.36368E-03 1.25313E+00 0.00000E+00 5.44628E-01 0.'
)
file <- textConnection(ext)
pxml <- as.pxml.ext(file)
close(file)
pxml

```

as.unilog.run

*Create a Run Log for NONMEM7 using NONMEM6 Format***Description**

MIfuncs includes an INFN routine for NONMEM6 and earlier that creates a comma-separated listing of parameter estimates and their percent relative standard errors. This function emulates that data, using the '.ext' file produced under NONMEM7. as.runlog.unilog converts the output to the format used for NONMEM6.

**Usage**

```

as.unilog.run(
run,
logfile='NonmemRunLog.csv',
outfile=paste(run,'lst',sep='.'),
extfile=file.path(
dirname(outfile),
paste(run,'ext',sep='.'))
),
tool='nm6',
...
)

```

**Arguments**

run	name of the NONMEM run, typically integer
logfile	path for the NONMEM run log file
outfile	path for the NONMEM output file, typically 'run.lst'
extfile	path for the NONMEM estimates, by default 'run.ext'
tool	currently 'nm7'
...	passed to other functions

**Details**

The 'ext' file is digested. In particular the rows labeled -1000000000 and -1000000001 are taken as the parameter estimates and standard errors, respectively. The minimization status (min) is either 0 (minimized) or 1 for NONMEM7, which is less informative than for NONMEM6. The covariance status (cov) is normally 0. It will be 1 if covariance was requested but no standard errors appear in extfile (as for NONMEM6).

**Value**

data.frame with columns *tool*, *run*, *parameter*, *moment*, *value*.

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [as.unilog.lst](#)
- [as.unilog.pxml](#)
- [as.runlog.unilog](#)

---

as.xml

*Represent an Object as XML*

---

**Description**

as.xml is generic, with a default method and methods for character and data.frame. The intent is to convert text-like objects to informal XML that can be navigated with XPath expressions, e.g. using package XML.

**Usage**

```
## Default S3 method:
as.xml(x, tag, ...)
## S3 method for class 'character'
as.xml(x, tag, ...)
## S3 method for class 'data.frame'
as.xml(x, keyname='row', key=rownames(x), ...)
```

**Arguments**

x	an object to convert
tag	an XML element name for open and close tags in which to enclose members of x; can be a vector
keyname	an XML element name to hold each value of each column in x
key	a vector of unique identifiers to distinguish each value of each column in x
...	name/value pairs to construct attributes in the 'open' tags (ignored for the data.frame method)

**Details**

The default method simply coerces its argument to character.

The character method pastes tag as an XML element name (in brackets) before and after x. Dots are used to construct attributes. Like tag itself, the attributes can be vectors; in both cases the usual recycling rules apply, as for paste.

The data.frame method calls the character method on each of its columns, passing keyname and keys as tag and key, respectively. keys should generally be unique. Note that the columns have no parent: supply one manually with nest, if necessary.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [as.pxml.ext](#)

**Examples**

```
nest(as.xml(head(Theoph)), tag='frame')
```

---

attribute	<i>Encode an Attribute for an XML Open Tag</i>
-----------	------------------------------------------------

---

**Description**

Encode an attribute for an XML open tag, in the form `tag='x'`.

**Usage**

```
attribute(x, tag, ...)
```

**Arguments**

x	character vector: attribute value(s)
tag	character (vector): attribute name
...	ignored

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [bracket](#)

**Examples**

```
attribute(letters, 'letter')
```

---

AUC

*Calculate AUC Using the Trapezoidal Method*

---

### **Description**

Calculate the area under the curve (AUC) for each subject over the time interval for dv using the trapezoidal rule.

### **Usage**

```
AUC(data, time = 'TIME', id = 'ID', dv = 'DV')
```

### **Arguments**

data	data.frame containing the data to use for the AUC calculation
time	chronologically ordered time variable present in data
id	variable in data defining subject level data
dv	dependent variable used to calculate AUC present in data

### **Details**

The AUC function performs the calculation based on the variables id, time, and dv present in the R data object.

### **Value**

One area under the dv-time curve is returned for each subject.

### **Author(s)**

Leonid Gibianski

### **References**

<http://mifuns.googlecode.com>

---

`aug`*Augment a List-like Object*

---

**Description**

Add named elements to an object.

**Usage**

```
aug(x, ...)
```

**Arguments**

<code>x</code>	dispatch argument
<code>...</code>	name-value pairs

**Details**

For each extra argument, an attempt is made to assign its value to a like-named element of `x` (typically a list or `data.frame`). The augmented object is returned. Similar to `transform`, but simpler.

**Value**

like `x`

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [transform](#)

**Examples**

```
head(aug(Theoph, study='b'))
```

---

backtrans	<i>Back-transform Columns in a Data Frame</i>
-----------	-----------------------------------------------

---

**Description**

Exponentiate presumably log-transformed columns in a data.frame.

**Usage**

```
backtrans(x, cols)
```

**Arguments**

x	data.frame
cols	vector of column names

**Value**

data.frame

**Author(s)**

Tim Bergsma

**References**

[mifuns.google.code.com](http://mifuns.google.code.com)

**See Also**

- [PLOTTR](#)

---

bin	<i>Calculate Bin Sizes and Limits for a Vector</i>
-----	----------------------------------------------------

---

**Description**

Given a numeric vector, calculate bin limits, place each value in a bin, and return the number of values in each bin.

**Usage**

```
bin(  
  x,  
  population=x,  
  breaks=quantile(population,probs=probs,...),  
  probs=c(0,0.25,0.5,0.75,1),  
  include.lowest=TRUE,  
  ...  
)
```

**Arguments**

x	a vector of numeric values, to be placed in bins
population	a vector of numeric values serving as the reference population for constructing bins
breaks	bin limits (boundaries) to pass to cut
include.lowest	limit qualifier to pass to cut
probs	default probabilities for calculating breaks
...	other arguments to pass to quantile and cut

**Details**

By default, the population used to calculate bin limits is the same as the group of values being binned. By default, inner bin limits are the quartiles of the population.

**Value**

A table with bin limits encoded as column names.

**Author(s)**

Tim Bergsma

**See Also**

- [quantile](#)
- [table](#)
- [cut](#)

**Examples**

```
bin(1:100)  
bin(1:50,population=1:100)  
plot(  
  bin(  
    rnorm(1000),  
    breaks=seq(  
      from=-3,
```

```
    to=3,  
    by=0.5  
  )  
  )  
)
```

---

bracket

*Create an XML Tag*

---

### Description

Create an open or close XML tag of the form <x> or </x>.

### Usage

```
bracket(x, close = FALSE, ...)
```

### Arguments

x	an element name
close	whether the tag is a close tag; FALSE by default
...	name-value pairs of attributes for an open tag

### Details

close can be a vector. An attempt will be made to suppress attributes for 'close' tags.

### Value

character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [attribute](#)
- [as.xml.character](#)

### Examples

```
bracket(rep('name', 2), close=FALSE:TRUE, row=1, col=2)
```

---

`breaks`*Identify Boundaries Between Sets of Elements*

---

**Description**

Identify boundaries between sets elements.

**Usage**

```
breaks(x, ...)
```

**Arguments**

<code>x</code>	vector
<code>...</code>	ignored

**Details**

A vector of length  $n$  has at most  $n-1$  divisions between elements. If the elements are taken in runs of repeated elements, a set of divisions  $\leq n-1$  may be identified. This function returns a zero for each between-element position if the bounding elements are identical, and a one for each between-element position if the bounding elements differ. Used for placing lines between sets of rows or columns in a table.

**Value**

integer

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [tabular.data.frame](#)

**Examples**

```
breaks(c(1,1,1,2,2,2,3,3,3))
```

**Description**

These functions support classes miTemporal, miTime, miDate, and miDateTime (and related functions). They are mostly S3 methods for base R generics.

**Usage**

```
## S3 method for class 'miTemporal'
c(..., recursive = FALSE)
## S3 method for class 'miTemporal'
x[... , drop = TRUE]
## S3 replacement method for class 'miTemporal'
x[... ] <- value
## S3 method for class 'miTemporal'
x[[... , drop = TRUE]]
## S3 method for class 'miTemporal'
as.character(x, ...)
as.chartime(x, ...)
## S3 method for class 'numeric'
as.chartime(x, format, mark=TRUE,...)
## S3 method for class 'chartime'
as.numeric(x, format,...)
## S3 method for class 'miTemporal'
print(x, ...)
## S3 method for class 'miTemporal'
rep(x, ...)
## S3 method for class 'miTemporal'
seq(from, to, by = NULL, length.out = NULL, along.with = NULL, ...)
```

**Arguments**

...	arguments to c, or passed to other functions
recursive	same meaning as for c
x	object of class miTemporal
drop	same meaning as for '[' and '['[
value	value to be assigned, as for '[<-'
format	input or output format describing character time (see <a href="#">strftime</a> )
mark	boolean: mark times with dangling seconds using '+'
from	as for <a href="#">seq.default</a>
to	as for <a href="#">seq.default</a>
by	as for <a href="#">seq.default</a>
length.out	as for <a href="#">seq.default</a>
along.with	as for <a href="#">seq.default</a>

**Details**

Normally you shouldn't need to worry about these functions. `c` and the `'[]'` variants exist just so that class information is not lost on invocation of the generic. `as.character.miTemporal` and `print.miTemporal` just call `format`. `chartime` variants are used internally by other functions. `seq.miTemporal` requires `from` and `to`, sets smart defaults for `by` (if null), and preserves class information, relative to `seq.default`.

**Value**

<code>print</code>	an invisible object with same class as <code>x</code>
<code>as.chartime</code>	generic: does not return
<code>as.chartime.numeric</code>	character (time)
<code>as.numeric.chartime</code>	numeric (seconds)
<code>as.character.miTemporal</code>	character (time)
others	object with same class as <code>x</code>

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [miTemporal](#)
- [seq.default](#)
- [strftime](#)

**Examples**

```
#as.data.frame
data.frame(
  dt=as.miDateTime(seq(from=0,by=86500,length.out=3)),
  d=as.miDate(seq(from=0,by=86400,length.out=3)),
  t=as.miTime(c(60,120,180))
)
#           dt           d           t
# 1 1970-01-01 00:00 1970-01-01 00:01
# 2 1970-01-02 00:01+ 1970-01-02 00:02
# 3 1970-01-03 00:03+ 1970-01-03 00:03

#combine
```

```

c(as.miTime(0),as.miTime(60))
# 00:00 00:01
c(as.miDate(0),as.miDate(86400))
# 1970-01-01 1970-01-02
c(as.miDateTime(0),as.miDateTime(86500))
# 1970-01-01 00:00 1970-01-02 00:01+

#subset
as.miTime(c('08:00','09:00'))[2]
# 09:00
as.miDate(c('2008-01-01','2008-01-04'))[2]
# 2008-01-04
as.miDateTime(c('2008-01-01 12:00','2008-01-04 12:30'))[2]
# 2008-01-04 12:30

#element selection
as.miTime(c('08:00','09:00'))[[2]]
# 09:00
as.miDate(c('2008-01-01','2008-01-04'))[[2]]
# 2008-01-04
as.miDateTime(c('2008-01-01 12:00','2008-01-04 12:30'))[[2]]
# 2008-01-04 12:30

#assignment
a <- as.miTime(seq(60,300, by=60))
a#00:01 00:02 00:03 00:04 00:05
a[5] <- 60
a#00:01 00:02 00:03 00:04 00:01
a[3] <- NA
a#00:01 00:02 <NA> 00:04 00:01

#identity
as.miTime(as.miTime(0))
# 00:00
as.miDate(as.miDate(0))
# 1970-01-01
as.miDateTime(as.miDateTime(0))
# 1970-01-01 00:00

#repetition
rep(as.miTime(86340),2)
# 23:59 23:59

```

---

check.subjects

*Summarize Columns of Subject Data*


---

### Description

This function checks for missing data, numeric data, ranges, etc. It is a tool developed to review data that has been read into R.

**Usage**

```
check.subjects(x, subject)
```

**Arguments**

x	data.frame
subject	subject identifier in the data (column name)

**Value**

summary of each column in x by subject

**Author(s)**

Leonid Gibianski, modified by Tim Bergsma

**References**

<http://mifuns.googlecode.com>

CLNR

*Delete Files and Directories*

**Description**

CLNR deletes files and directories and logs what was deleted to a text file in 'project'.

**Usage**

```
CLNR(Dir, project, note = 'Files removed', test = TRUE)
```

**Arguments**

Dir	Complete path of directory to be deleted. This directory and all files and sub-directories will be deleted. Example: Dir='/Dir/to/be/Removed'
project	System path to the directory containing the NONMEM control (*.ctl) streams.
note	character vector (defaults to 'Files removed') for user text to describe why something was deleted. Example: note=c('Files removed')
test	logical to determine if you want to actually delete the files or just see what files would be deleted. To delete directory, files, and sub-directories use test=F.

**Author(s)**

Bill Knebel

**References**

<http://mifuns.googlecode.com>

---

`closers`*Calculate Closing Text for Nested Script Levels*

---

**Description**

Detect target locations for subscript and superscript end-tags, and place the specified substitutes.

**Usage**

```
closers(x, sub, sup, ...)
```

**Arguments**

<code>x</code>	character
<code>sub</code>	character
<code>sup</code>	character
<code>...</code>	ignored

**Details**

Used by `wikiparse`.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [wikiparse](#)

---

colname<-                      *Change Selected Names*~~

---

### Description

Change selected element names.~~

### Usage

```
colname(x) <- value
```

### Arguments

x	an object with named elements
value	named character vector

### Details

Changing column names is a common task. One can change them all at once, e.g.

```
colnames(x) <- c(...)
```

```
names(x) <- c(...)
```

or one at a time, e.g.

```
names(x)[names(x)==old] <- new
```

**reshape** supplies `rename`, a function that changes a subset of identified names, returning the object itself.

```
library(reshape)
```

```
x <- rename(x, c(label=value, ...))
```

In contrast, `name` and its data frame equivalent `colname` change a subset of identified names using assignment syntax.

```
name(x) <- c(label=value, ...)
```

### Value

used for side effects

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

**See Also**

- [names](#)
- [colnames](#)

**Examples**

```
names(Theoph)
name(Theoph) <- c(Subject='SUBJECT')
names(Theoph)
colname(Theoph) <- c(SUBJECT='subject',conc='concentration')
names(Theoph)
names(letters) <- LETTERS
letters
name(letters) <- c(A='a',B='b',C='c')
letters
```

---

command

*Format a Latex Command*

---

**Description**

Format a latex command.

**Usage**

```
command(x, options = NULL, args = NULL, depth = 0)
```

**Arguments**

x	length one character
options	vector or list
args	vector or list
depth	integer

**Details**

x is formatted as a latex command, with the options (possibly named) inserted in square brackets, and the arguments each enclosed in curly braces. depth spaces are added to the left end of the string.

**Value**

character

**Author(s)**

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [wrap](#)
- [ltable.data.frame](#)
- [spaces](#)

## Examples

```
command('caption', options='short', args='long')
```

---

compileflag

*Develop a Value for a Compiler Resource on the Grid*

---

## Description

License management under Sun Grid Engine may require an argument to qsub of, e.g., '-l compile=1' (in the case where the compiler is 'ifort'). This function calculates that value flexibly.

## Usage

```
compileflag(compiler, mappings = list(ifort = 1), ...)
```

## Arguments

compiler	name of the compiler in use
mappings	values for the flag, per compiler
...	ignored

## Value

character

## Author(s)

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [runCommand](#)

---

compiler	<i>Extract the Compiler Name from an NMQual Configuration File</i>
----------	--------------------------------------------------------------------

---

### Description

This function reads the configuration file and extracts the last path element of the first space-delimited component of the 'nmtran' instruction, typically a compiler name.

### Usage

```
compiler(config, pathsep = '/', ...)
```

### Arguments

config	path for a configuration file
pathsep	file system path separator
...	ignored

### Value

character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [runCommand](#)

### Examples

```
## Not run: compiler('../NONMEM/nm6osx1/test/config.xml')
```

---

 compute.cwres

---

*Compute the Conditional Weighted Residuals*


---

## Description

This function computes the conditional weighted residuals (CWRES) from a NONMEM run. CWRES are an extension of the weighted residuals (WRES), but are calculated based on the first-order with conditional estimation (FOCE) method of linearizing a pharmacometric model (WRES are calculated based on the first-order (FO) method). The function requires a NONMEM table file and an extra output file that must be explicitly asked for when running NONMEM. See details below.

## Usage

```
compute.cwres(run.number,
              tab.prefix="cwtab",
              sim.suffix="",
              est.tab.suffix=".est",
              deriv.tab.suffix=".deriv",
              old.file.convention=FALSE,
              id="ALL",
              printToOutfile=TRUE,
              onlyNonZero=TRUE,
              ...)
```

## Arguments

run.number	The run number of the NONMEM from which the CWRES are to be calculated.
tab.prefix	The prefix to two NONMEM file containing the needed values for the computation of the CWRES, described in the details section.
sim.suffix	The suffix ,before the ".", of the NONMEM file containing the needed values for the computation of the CWRES, described in the details section. For example, the table files might be named cwtab1sim.est and cwtab1sim.deriv, in which case sim.suffix="sim".
est.tab.suffix	The suffix, after the ".", of the NONMEM file containing the estimated parameter values needed for the CWRES calculation.
deriv.tab.suffix	The suffix, after the ".", of the NONMEM file containing the derivatives of the model with respect to the random parameters needed for the CWRES calculation.
old.file.convention	For backwards compatibility. Use this if you are using the previous file convention for CWRES (table files named cwtab1, cwtab1.50, cwtab1.51, ... , cwtab.58 for example).
id	Can be either "ALL" or a number matching an ID label in the datasetname.

printToOutfile	Logical (TRUE/FALSE) indicating whether the CWRES values calculated should be appended to a copy of the datasetname. Only works if id="ALL". If chosen the resulting output file will be datasetname.cwres.
onlyNonZero	Logical (TRUE/FALSE) indicating if the return value (the CWRES values) of compute.cwres should include the zero values associated with non-measurement lines in a NONMEM data file.
tab.suffix	The suffix to the NONMEM table file containing the derivative of the model with respect to the etas and epsilons, described in the details section.
...	Other arguments passed to basic functions in code.

### Details

compute.cwres This function is the computational 'brains' of the CWRES computation. The function simply reads in the following two files:

```
paste(tab.prefix,run.number,sim.suffix,est.tab.suffix,sep="")
paste(tab.prefix,run.number,sim.suffix,deriv.tab.suffix,sep="")
```

Which might be for example:

```
cwtab1.est
cwtab1.deriv
```

and (depending on the input values to the function) returns the CWRES in vector form as well as creating a new table file named:

```
paste(tab.prefix,run.number,sim.suffix,sep="")
```

Which might be for example:

```
cwtab1
```

### Value

compute.cwres Returns a vector containing the values of the CWRES.

### Setting up the NONMEM model file

In order for this function to calculate the CWRES, NONMEM must be run while requesting certain tables and files to be created. How these files are created differs depending on if you are using `\$PRED` or `ADVAN` as well as the version of NONMEM you are using. These procedures are known to work for NONMEM VI but may be different for NONMEM V. We have attempted to indicate where NONMEM V may be different, but this has not been extensively tested!

There are five main insertions needed in your NONMEM control file:

## 1. 1. \ \$ABB COMRES=X.

Insert this line directly after your \ \$DATA line. The value of X is the number of ETA() terms plus the number of EPS() terms in your model. For example for a model with three ETA() terms and two EPS() terms the code would look like this:

```
$DATA temp.csv IGNORE=@
$ABB COMRES=5
$INPUT ID TIME DV MDV AMT EVID
$SUB ADVAN2 TRANS2
```

## 2. 2. Verbatim code.

## • A. Using ADVAN.

If you are using ADVAN routines in your model, then Verbatim code should be inserted directly after the \ \$ERROR section of your model file. The length of the code depends again on the number of ETA() terms and EPS() terms in your model. For each ETA(y) in your model there is a corresponding term G(y,1) that you must assign to a COM() variable. For each EPS(y) in your model, there is a corresponding HH(y,1) term that you must assign to a COM() variable.

For example for a model using ADVAN routines with three ETA() terms and two EPS() terms the code would look like this:

```
"LAST
" COM(1)=G(1,1)
" COM(2)=G(2,1)
" COM(3)=G(3,1)
" COM(4)=HH(1,1)
" COM(5)=HH(2,1)
```

## • B. Using PRED.

If you are using \ \$PRED, the verbatim code should be inserted directly after the \ \$PRED section of your model file. For each ETA(y) in your model there is a corresponding term G(y,1) that you must assign to a COM() variable. For each EPS(y) in your model, there is a corresponding H(y,1) term that you must assign to a COM() variable. The code would look like this for three ETA() terms and two EPS() terms:

```
"LAST
" COM(1)=G(1,1)
" COM(2)=G(2,1)
" COM(3)=G(3,1)
" COM(4)=H(1,1)
" COM(5)=H(2,1)
```

## 3. 3. INFN routine.

## • A. Using ADVAN with NONMEM VIb.

If you are using ADVAN routines in your model, then an \ \$INFN section should be placed directly after the \ \$PK section using the following code. In this example we are assuming that the model file is named something like '1.ct1'. NOTE: Files 51, 53, 55 and 57 are not used in the CWRES calculation and can be removed from the NONMEM model file if problems arise due to failure of the \ \$COV step in a NONMEM run.

```

$INFN
IF (ICALL.EQ.3) THEN
  OPEN(50,FILE='cwtab1.est')
  WRITE(50,*) 'ETAS'
  DO WHILE(DATA)
    IF (NEWIND.LE.1) WRITE (50,*) ETA
  ENDDO
  WRITE(50,*) 'THETAS'
  WRITE(50,*) THETA
  WRITE(50,*) 'OMEGAS'
  WRITE(50,*) OMEGA(BLOCK)
  WRITE(50,*) 'SIGMAS'
  WRITE(50,*) SIGMA(BLOCK)
ENDIF

```

- B. Using ADVAN with NONMEM V.

If you are using ADVAN routines in your model, then you need to use an INFN subroutine. If we call the INFN subroutine 'myinfn.for' then the \\$\$SUBS line of your model file should include the INFN option. That is, if we are using ADVAN2 and TRANS2 in our model file then the \\$\$SUBS line would look like:

```
$SUB ADVAN2 TRANS2 INFN=myinfn.for
```

The 'myinfn.for' routine for 4 thetas, 3 etas and 1 epsilon is shown below. If your model has different numbers of thetas, etas and epsilons then the values of NTH, NETA, and NEPS, should be changed respectively. These values are found in the DATA statement of the subroutine. Please note that the 4th and 5th lines of code should be one line with the '...' removed from each line, reading: COMMON /ROCM6/ THETAF(40), OMEGAF(30,30), SIGMAF(30,30).

NOTE: Files 51, 53, 55 and 57 are not used in the CWRES calculation and can be removed from the subroutine if problems arise due to failure of the \\$\$COV step in a NONMEM run.

```

SUBROUTINE INFN(ICALL, THETA, DATREC, INDXS, NEWIND)
DIMENSION THETA(*), DATREC(*), INDXS(*)
DOUBLE PRECISION THETA
COMMON /ROCM6/ ...
... THETAF(40), OMEGAF(30,30), SIGMAF(30,30)
COMMON /ROCM7/ SETH(40), SEOM(30,30), SESIG(30,30)
COMMON /ROCM8/ OBJECT
COMMON /ROCM9/ IERE, IERC
DOUBLE PRECISION THETAF, OMEGAF, SIGMAF
DOUBLE PRECISION OBJECT
REAL SETH, SEOM, SESIG
DOUBLE PRECISION ETA(10)
INTEGER J, I
INTEGER IERE, IERC
INTEGER MODE
INTEGER NTH, NETA, NEPS
DATA NTH, NETA, NEPS/4, 3, 1/
IF (ICALL.EQ.0) THEN
C   open files here, if necessary
  OPEN(50,FILE='cwtab1.est')

```

```

ENDIF
IF (ICALL.EQ.3) THEN
  MODE=0
  CALL PASS(MODE)
  MODE=1
WRITE(50,*) 'ETAS'
20  CALL PASS(MODE)
   IF (MODE.EQ.0) GO TO 30
   IF (NEWIND.NE.2) THEN
     CALL GETETA(ETA)
     WRITE (50,97) (ETA(I),I=1,NETA)
   ENDIF
   GO TO 20
30  CONTINUE
   WRITE (50,*) 'THETAS'
   WRITE (50,99) (THETA(J),J=1,NTH)
WRITE(50,*) 'OMEGAS'
   DO 7000 I=1,NETA
7000  WRITE (50,99) (OMEGAF(I,J),J=1,NETA)
WRITE(50,*) 'SIGMAS'
   DO 7999 I=1,NEPS
7999  WRITE (50,99) (SIGMAF(I,J),J=1,NEPS)
ENDIF
99  FORMAT (20E15.7)
98  FORMAT (2I8)
97  FORMAT (10E15.7)
RETURN
END

```

- C. Using \SPRED with NONMEM VIb.

If you are using \SPRED, then the following code should be placed at the end of the \SPRED section of the model file (together with the verbatim code). NOTE: Files 51, 53, 55 and 57 are not used in the CWRES calculation and can be removed from the NONMEM model file if problems arise due to failure of the \SCOV step in a NONMEM run.

```

IF (ICALL.EQ.3) THEN
  OPEN(50,FILE='cwtab1.est')
  WRITE(50,*) 'ETAS'
  DO WHILE(DATA)
    IF (NEWIND.LE.1) WRITE (50,*) ETA
  ENDDO
  WRITE(50,*) 'THETAS'
  WRITE(50,*) THETA
  WRITE(50,*) 'OMEGAS'
  WRITE(50,*) OMEGA(BLOCK)
  WRITE(50,*) 'SIGMAS'
  WRITE(50,*) SIGMA(BLOCK)

```

```
ENDIF
```

- D. Using \SPRED with NONMEM V.

If you are using \SPRED with NONMEM V, then you need to add verbatim code immediately after the \SPRED command. In this example we assume 4 thetas, 3 etas and 1 epsilon. If your model has different numbers of thetas, etas and epsilons then the values of NTH, NETA, and NEPS, should be changed respectively. These vales are found in the DATA statement below. Please note that the 3rd and 4th lines of code should be one line with the '...' removed from each line. NOTE: Files 51, 53, 55 and 57 are not used in the CWRES calculation and can be removed from the NONMEM model file if problems arise due to failure of the \SCOV step in a NONMEM run.

```
$PRED
"FIRST
"      COMMON /ROCM6/ ...
...THETA(4), OMEGAF(30,30), SIGMAF(30,30)
"      COMMON /ROCM7/ SETH(40), SEOM(30,30), SESIG(30,30)
"      COMMON /ROCM8/ OBJECT
"      DOUBLE PRECISION THETA(4), OMEGAF, SIGMAF
"      DOUBLE PRECISION OBJECT
"      REAL SETH, SEOM, SESIG
"      INTEGER J, I
"      INTEGER MODE
"      INTEGER NTH, NETA, NEPS
"      DATA NTH, NETA, NEPS/4, 3, 1/
```

After this verbatim code you add all of the abbreviated code needed for the \SPRED routine in your model file. After the abbreviated code more verbatim code is needed. This verbatim code should be added before the verbatim code discussed above under point 2.

```
"      IF (ICALL.EQ.0) THEN
"C      open files here, if necessary
"      OPEN(50, FILE='cwtab1.est')
"      ENDIF
"      IF (ICALL.EQ.3) THEN
"      MODE=0
"      CALL PASS(MODE)
"      MODE=1
"      WRITE(50,*) 'ETAS'
" 20      CALL PASS(MODE)
"      IF (MODE.EQ.0) GO TO 30
"      IF (NEWIND.NE.2) THEN
"      CALL GETETA(ETA)
"      WRITE (50,97) (ETA(I), I=1, NETA)
"      ENDIF
"      GO TO 20
" 30      CONTINUE
"      WRITE (50,*) 'THETAS'
"      WRITE (50,99) (THETAF(J), J=1, NTH)
"      WRITE (50,*) 'OMEGAS'
"      DO 7000 I=1, NETA
```

```

" 7000      WRITE (50,99) (OMEGAF(I,J),J=1,NETA)
"          WRITE (50,*) 'SIGMAS'
"          DO 7999 I=1,NEPS
" 7999      WRITE (50,99) (SIGMAF(I,J),J=1,NEPS)
"          ENDIF
" 99       FORMAT (20E15.7)
" 98       FORMAT (2I8)
" 97       FORMAT (10E15.7)

```

#### 4. 4. cwtab\*.deriv table file.

A special table file needs to be created to print out the values contained in the COMRES variables. In addition the ID, IPRED, MDV, DV, PRED and RES data items are needed for the computation of the CWRES. The following code should be added to the NONMEM model file. In this example we continue to assume that we are using a model with three ETA() terms and two EPS() terms, extra terms should be added for new ETA() and EPS() terms in the model file.

```

$TABLE ID COM(1)=G11 COM(2)=G21 COM(3)=G31
COM(4)=H11 COM(5)=H21
IPRED MDV NOPRINT ONEHEADER FILE=cwtab1.deriv

```

#### 5. 5. \ESTIMATION.

To compute the CWRES, the NONMEM model file must use (at least) the FO method with the POSTHOC step. If the FO method is used and the POSTHOC step is not included then the CWRES values will be equivalent to the WRES. The CWRES calculations are based on the FOCE approximation, and consequently give an idea of the ability of the FOCE method to fit the model to the data. If you are using another method of parameter estimation (e.g. FOCE with interaction), the CWRES will not be calculated based on the same model linearization procedure.

#### Author(s)

Andrew Hooker

#### References

Hooker A, Staats CE, Karlsson MO. *Conditional weighted residuals, an improved model diagnostic for the FO/FOCE methods*. PAGE 15 (2006) Abstr 1001 [<http://www.page-meeting.org/?abstract=1001>].

#### See Also

- [compute.cwres](#)

---

config	<i>Identify the Configuration File in an NMQual-mediated NONMEM installation.</i>
--------	-----------------------------------------------------------------------------------

---

### Description

Gives the path for the configuration file in a NONMEM installation directory.

### Usage

```
config(dir, ...)
```

### Arguments

dir	directory in which to find 'config.xml'
...	ignored

### Value

character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [runNonmem](#)

### Examples

```
config('/common/nonmem/nm6')
```

---

 constant

*Test for Nested Factors*


---

**Description**

Various Tests for Nested Factors.

**Usage**

```
a %nests% b
a %nested.in% b
crosses(a,b)
## Default S3 method:
constant(x,within,...)
```

**Arguments**

a	a list of equal length factors, or one factor
b	a list of factors, or one factor, same length as a
x	an object like a, or a dataframe
within	An object like b, or just column names in x for the data.frame version
...	extra arguments passed to other methods

**Details**

`nests` tests whether the factor(s) on the right are nested within the factor(s) on the left: i.e. b implies a. `nested.in` tests the opposite: i.e., for unique interaction-wise levels of a, is there exactly one interaction-wise corresponding level in b (a implies b)? `constant`, by default, tests whether `within` implies x.

`crosses` is the basis for all other functions. For each level in b, it checks for corresponding levels of a, returning FALSE for the first and TRUE for any others, in which cases a crosses b, or b is crossed on (not nested in) a.

**Value**

Logical vector for `crosses`; scalar logical for all others.

**Note**

All these functions are NA-safe; i.e., they treat NA as a distinct level.

**Author(s)**

Tim Bergsma

**Examples**

```

C02[,c('Type','Treatment')] %nests% C02$Plant #TRUE
C02$Plant %nested.in% C02$Type #TRUE
with(C02,constant(list(Type,Treatment),within=Plant)) #TRUE
with(ChickWeight,constant(Diet,within=Chick)) #TRUE
with(ChickWeight,constant(weight,within=list(Chick,Time))) #TRUE
crosses(c(1,1,1,NA,NA,4),c(2,3,3,NA,5,5)) #FALSE FALSE FALSE FALSE TRUE
constant(Theoph$Wt,within=Theoph$Subject)#TRUE
constant(Theoph, within='Subject')

```

contains

*Test a Character Vector for Occurences of a Pattern***Description**

Test a character vector for occurrences of a pattern.

**Usage**

```
contains(pattern, text, ...)
```

**Arguments**

pattern	regular expression
text	character vector
...	extra arguments to regexpr

**Details**

Each element of text is tested for pattern, and returns TRUE if pattern matches.

**Value**

logical vector as long as text

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [regexpr](#)

## Examples

```
contains(  
  'glu',  
  c(  
    'glucose',  
    'glucuronidase',  
    'glucogen',  
    'fibrogen'  
  )  
)
```

---

css

*Calculate One-Compartment Model Properties*

---

## Description

These functions calculate steady-state properties from the parameters of a one-compartment pharmacokinetic model, assuming all doses occur in the post-distribution phase ( $\tau \gg t_{max}$ ).

## Usage

```
acr(cl, v, tau, ...)  
cavg(cl, tau, dose, ...)  
cmax(cl, v, ka, tau, dose, ...)  
cmin(cl, v, ka, tau, dose, ...)  
css(cl, v, ka, tau, dose, time, ...)  
ke(cl, v, ...)  
tmax(cl, v, ka, tau, ...)  
auc(cl, dose, ...)
```

## Arguments

cl	apparent clearance (L/h)
v	apparent volume (L)
ka	absorption constant ( $h^{-1}$ )
tau	steady-state trough-to-trough interval (h)
dose	dose (arbitrary units)
time	arbitrary time point, $\leq \tau$
...	other arguments, ignored

**Details**

acr calculates accumulation ratio (unitless, called by *css*). ke calculates the elimination constant ( $h^{-1}$ , called by *css*). *css* calculates steady-state compartment concentration (dose units per L) at a given time. *cavg* calculates average concentration over dosing interval. *cmin* and *cmax* calculate minimum and maximum steady-state concentrations; they call *css*, passing either *tau* itself or *tmax* for the time argument. *tmax* calculates the time of the maximum concentration. *auc* calculates area under the curve.

Note that for the post-distributive assumption to hold, *tau* must be very large relative to the time needed for drug disposition. To the extent that it does not hold, these functions give biased results.

**Value**

numeric

**Note**

Multiple properties can be calculated within the same function or loop, since all arguments can be passed even if not needed. For example, the same five arguments can be passed, even un-named, to *cmax* and *tmax* (*tmax* will ignore *dose*). But be sure to pass by name the same arguments to *cmax* and *cavg*: the latter will ignore unused arguments but has the common arguments in a different order.

**Author(s)**

Tim Bergsma

**References**

Gibaldi M, Perrier D. Multiple dosing. Pharmacokinetics. New York: Marcel Dekker, Inc., 1982: p. 113- 144.

**Examples**

```
cavg(cl=0.05, tau=300, dose=100)
cmax(cl=0.05, v=10, ka=0.015, tau=300, dose=100)
cmin(cl=0.05, v=10, ka=0.015, tau=300, dose=100)
tmax(cl=0.05, v=10, ka=0.015, tau=300)
```

---

ctl

*A Sample NONMEM(r) Control Stream*

---

**Description**

The control stream for running control test 8 of NMQual from Metrum Institute.

**Usage**

```
data(ctl)
```

**Format**

The format is: chr [1:89] "PROBLEM 8, Parent-Metabolite-Urine Model" ...

**Details**

This control stream is stored here as a practice target for `resample.character`.

**Source**

The file was borrowed from 8.ct1 as distributed with NMQual from <http://nmqual.googlecode.com>.

**Examples**

```
data(ct1)
```

---

dataFormat

*Integrate NONMEM Data for Plotting*

---

**Description**

Integrates `data.frames` representing standard NONMEM output and the source data.

**Usage**

```
dataFormat(
  tabdata,
  covdata,
  pardata,
  logtrans = FALSE,
  grp = NULL,
  grpnames = NULL,
  cont.cov = NULL,
  cat.cov = NULL,
  par.list = NULL,
  eta.list = NULL,
  missing = -99,
  run,
  ...
)
```

**Arguments**

tabdata	data.frame for ‘*.TAB’
covdata	data.frame for underlying dataset, one record per ID
pardata	data.frame for ‘*par.TAB’
logtrans	see PLOTR

grp	see PLOTR
grpnames	see PLOTR
cont.cov	see PLOTR
cat.cov	see PLOTR
par.list	see PLOTR
eta.list	see PLOTR
missing	see PLOTR
run	see PLOTR
...	passed to synthesis

**Details**

Called by dataSynthesis, and in turn calls synthesis.

**Value**

data.frame

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [dataSynthesis](#)
- [synthesis](#)

---

dataSynthesis

*Build a Standard Plotting Data Set for a Given NONR Run*

---

**Description**

dataSynthesis scavenges columns from the ‘TAB’ file, the ‘par.TAB’ file, and the underlying dataset, returning just observation records (EVID==0) that are not commented. The second two files are limited to the first record per ID. Columns to scavenge are supplied by grp, cont.cov, cat.cov, par.list, and eta.list. All ‘TAB’ file columns are returned, plus first instance of ‘scavenge’ columns from either the ‘par’ file or the parent data set (in that order) unless already present. Exception: par.list and eta.list items are not sought in the underlying dataset.

**Usage**

```

dataSynthesis(
  run,
  project=getwd(),
  logtrans = FALSE,
  grp = NULL,
  grpnames = NULL,
  cont.cov = NULL,
  cat.cov = NULL,
  par.list = NULL,
  eta.list = NULL,
  missing = -99,
  rundir = filename(project, run),
  ctlfile = filename(rundir,run,'.ctl'),
  outfile = filename(rundir,run,'.lst'),
  datfile = getdname(ctlfile),
  ...
)

```

**Arguments**

run	a control stream name, typically integer
project	the directory containing the NONMEM run directories
logtrans	whether to transform the NONMEM output variables DV, PRED, and IPRED. Default: FALSE.
grp	item in NONMEM datafile or output table file that can be used to condition plots generated by PLOTR. Default value is NULL. Example: grp=c('SEX'). Can be more than one, e.g., grp=c('SEX', 'TRT').
grpnames	optional vector of names for grp item. Vector length must equal number of conditions in grp and must have an order corresponding to an increasing sort of grp. Default value is NULL. Example: grpnames=c('Male', 'Female')
cont.cov	vector of continuous covariate names. Names must match those used as column headers in datfile. Values are retrieved from datfile so they do not need to be part of the NONMEM \TABLE step. Default value is NULL. Example: cont.cov=c('AGE', 'WT', 'CLCR')
cat.cov	vector of categorical covariate names. Names must match those used as column headers in datfile. Values are retrieved from datfile so they do not need to be part of the NONMEM \TABLE step. Default value is NULL. Example: cat.cov=c('SEX', 'FOOD')
par.list	vector of NONMEM model parameter names. Values are retrieved from '*par.TAB' created in NONMEM. Default value is NULL. This can be a superset of parameters but only those present in NONMEM output table will be used. Example: par.list=c('CL', 'V', 'V2', 'Q')
eta.list	vector of NONMEM model random effect names. Values are retrieved from '*par.TAB' created in NONMEM. Default value is NULL. This can be a superset of random parameters but only those present in NONMEM output table will be used. Example: eta.list=c('ETA1', 'ETA2', 'ETA3', 'ETA4')

missing	numeric item that defines value used to represent missing items in the NONMEM data file. Default value is '-99'.
ctlfile	Path and filename for the NONMEM control stream (*.ctl').
outfile	Path and filename for the NONMEM output file (*.lst').
datfile	Path and filename for the source dataset, as in DATA record in *.ctl'.
rundir	Path for the NONMEM run directory ('project/*').
...	extra arguments

**Value**

data.frame

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [diagnosticPlots](#)
- [dataFormat](#)

---

deranged

*Modify Assembly of nm Objects*

---

**Description**

These are special preparations of nm objects or their sources. Proxy methods for Ops.keyed are techniques for dynamic preparation.

**Usage**

```
## S3 method for class 'nm'
moot(x, scope = x$EVID == 2, disregard = c('TIME', 'DATETIME', 'DATE', 'TAFD', 'TAD', 'HOUR'), ...)
## S3 method for class 'nm'
rig(x, n = 50, dateTime = FALSE, noDups = FALSE, ...)
## S3 method for class 'keyed'
deranged(x, ...)
## S3 method for class 'data.frame'
deranged(x, start, stop, result = start, dropStop = TRUE, ...)

as.moot(...)
as.rigged(...)
```

```
## S3 method for class 'moot'
minus(x, y, ...)
## S3 method for class 'rigged'
plus(x, y)
```

## Arguments

<code>x</code>	dispatch argument
<code>y</code>	right operand to proxy method
<code>start</code>	name of column with start values
<code>stop</code>	name of column with stop values
<code>result</code>	name of column in which to place the result
<code>dropStop</code>	whether to drop the stop column before returning
<code>...</code>	extra arguments, usually ignored; see details
<code>scope</code>	an expression to evaluate within <code>x</code> , giving moot candidate rows
<code>disregard</code>	columns to disregard when testing for between-row differences
<code>n</code>	number of extra records to generate per subject
<code>dateTime</code>	whether to attempt derivation of timecourse from DATETIME instead of HOUR
<code>noDups</code>	whether to prohibit introduction of rows with times that already exist

## Details

`deranged.data.frame`. When a dataset attribute (column) is constant over some range of another discreet attribute, information may be stored more compactly by specifying the constant attribute once, together with the start and stop of the range. This function takes such a dataset and instantiates all implied records. Warning: the function calls `seq(start, stop)` for each row, and makes no assumptions about whether the data is ascending, descending, stationary (`start==stop`), or undefined (one or more of `start` and `stop` is NA). `deranged.keyed` is just a wrapper that preserves attributes.

`moot.nm` will identify rows that make no difference to NONMEM. If an EVID:2 row differs from the row just before it only with respect to temporal designations, the NONMEM result should not be affected. NONMEM should use the changed values from the first row in which non-temporal items change to estimate the midpoint of the prior interval.

`rig.nm` generates extra rows, spaced across each subject's timecourse, for obtaining model predictions in NONMEM (EVID: 2). It cannot work with HOUR and DATETIME simultaneously, so in mixed cases, `plus.rigged` tries both sequentially and combines the result.

`as.moot` and `as.rigged` return their arguments as a list classified as `moot` or `rigged`, respectively. In operator context, `- as.moot` or `+ as.rigged` cause `minus.moot` or `plus.rigged` to be called. These call `moot` and `rig` with the left operand as `x` and the right operand as a list of extra arguments; then they do appropriate processing. `minus.moot` drops moot rows; `plus.rigged` merges the rigged data.frame.

deranged and moot can be used together with effect. deranged creates explicit rows that can be merged by TIME to existing data. After the merge, it will be apparent that many such rows are moot. Functions here support dynamic removal. See example for as.nm.

### Value

moot.nm returns logical, with length nrow(x). as.moot returns a list of class moot. minus.moot returns a subset of its first argument, typically nm.

rig.nm returns a data.frame of rows evenly spaced across each subject's timecourse (i.e., 'rigging'). as.rigged returns a list of class rigged. plus.rigged returns a merge on its first argument, typically nm.

deranged returns keyed data.frame.

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [as.nm](#)
- [Ops.keyed](#)

---

diagnosticPlots

*Create Diagnostic Plots*

---

### Description

Create diagnostic plots from a data set.

### Usage

```
diagnosticPlots(  
  data,  
  dvname='DV',  
  group=NULL,  
  model=NULL,  
  ...  
)
```

```
covariatePlots(  
  data,  
  cont.cov=NULL,  
  cat.cov=NULL,
```

```
par.list=NULL,  
eta.list=NULL,  
...  
)  
  
cwresPlots(  
data,  
cont.cov=NULL,  
cat.cov=NULL,  
...  
)
```

### Arguments

<code>data</code>	a data.frame, typically created by <code>dataSynthesis</code>
<code>dvname</code>	name of the dependent variable to use as a label for the diagnostic plots
<code>group</code>	optionally, a column name in <code>data</code> used to condition the output of <code>diagnosticPlots</code>
<code>model</code>	If supplied, this text argument will be prepended to the figure titles.
<code>cont.cov</code>	as defined for <code>PLOTR</code>
<code>cat.cov</code>	as defined for <code>PLOTR</code>
<code>par.list</code>	as defined for <code>PLOTR</code>
<code>eta.list</code>	as defined for <code>PLOTR</code>
<code>...</code>	ignored arguments

### Details

`PLOTR` passes the result of `dataSynthesis` to these functions and prints the resulting lists (of trellis objects) on the open device.

### Value

a list of trellis objects

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [PLOTR](#)
- [dataSynthesis](#)

---

electronicAppendix	<i>Make an FDA-style Electronic Appendix from a Subversioned Directory.</i>
--------------------	-----------------------------------------------------------------------------

---

**Description**

Make an FDA-style Electronic Appendix from a Subversioned Directory.

**Usage**

```
electronicAppendix(x, as, pattern = NULL, recursive = TRUE, ignore.case = TRUE, zip = FALSE, ...)
```

**Arguments**

x	a Subversioned directory, e.g. a checkout of a repository or subdirectory thereof.
as	a file path for a directory to be created
pattern	passed to dir
recursive	passed to dir
ignore.case	passed to dir
zip	logical indicating whether to compress as to a zipped file.
...	ignored

**Details**

FDA submissions may require electronic copies of file trees, where text files must have the extension 'txt'. This function creates a copy of x using `svn export` (a system call). It fixes the names of those files that Subversion considers text, and optionally compresses the resulting directory to a zipped file (also a system call). Not tested on all platforms, but should work fine on most Unix-alikes.

For finer control over what is considered text, see `svnMarkAsText` and `svnMarkAsNonText`.

Note that non-subversioned files will not be included in the electronic appendix. An error results if any file is subversioned but missing.

as must be specified as a directory name. If zip is TRUE, .zip will be added.

as cannot pre-exist unless zip is true, in which case as will be renamed temporarily while its namesake is zipped and unlinked.

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [dir](#)
- [svnIsText](#)
- [svnMarkAsText](#)
- [svnMarkAsNonText](#)

---

episcript

*Run an Epilog Script in a Pre-populated Environment*

---

**Description**

Sources the named script in an environment where all other passed arguments (...) have been defined by assignment.

**Usage**

```
episcript(script, ...)
```

**Arguments**

script	path for an R script
...	variables with which to populate the script's environment

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [runNonmem](#)

---

filename	<i>Construct a Canonical File Name</i>
----------	----------------------------------------

---

**Description**

Helper function to build file and directory names from components.

**Usage**

```
filename(dir, run = NULL, ext = NULL)
```

**Arguments**

dir	directory
run	run name
ext	file extension

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [dataSynthesis](#)

**Examples**

```
filename('dir','3','.ctl')
```

---

 first
 

---



---

*Identify Elements Within Cells Meeting Some Instance of a Condition*


---

**Description**

Within each level of within, find the  $n^{\text{th}}$  element for which where is TRUE, and repeat it at all positions in the level.

**Usage**

```

nth (x,where,within,n=1,...)
first(x,where,within,...)
last (x,where,within,...)
only (x,where,within,...)
distance(where,within,n=1,...)
before( where,within,n=1,...)
at(     where,within,n=1,...)
after(  where,within,n=1,...)

```

**Arguments**

x	vector (possibly factor)
where	vector, coerced to logical
within	a vector or list of vectors serving as an index
n	an integer giving the instance of interest
...	ignored arguments.

**Details**

nth returns, for each position in x, the  $n^{\text{th}}$  element of x, optionally limiting candidate elements by where, and optionally breaking the evaluation across subsets as specified by within. n can be 0, returning all NA, or negative, which counts instances from the end of the vector or subsets. If n is NA, all elements are returned.

All of x, where, and within are optional. The ‘scale’ of the function is set to the longest of the three (member-wise for within if it is a list). Missing x is replaced with the subscripts implied by ‘scale’. Missing where and within are replaced with TRUE, repeated to length ‘scale’. No warnings are given if within has NA, or list members with different lengths.

first and last are convenience wrappers for nth that set n to 1 and -1, respectively. only is like first, but returns NA if a second instance exists.

distance returns subscripts less the nth subscripts, optionally considering where and within. before, at, and after test whether distance is less than, equal to, or greater than zero.

**Value**

vector of same class as x, if supplied, else of class integer. Logical for before, at, and after.

**Author(s)**

Tim Bergsma

**References**<http://mifuns.googlecode.com>**See Also**

- [match](#)

**Examples**

```

first(1:10)
first(1:10,within=rep(1:2,each=5))
last( 1:10,within=rep(1:2,each=5))
first(1:10,where=0:1,within=rep(1:2,each=5))
test <- data.frame(
  id=c(1,1,1,1,1,2,2,2,2,2),
  time=c(1,2,3,4,5,1,2,3,4,5),
  evid=c(0,1,0,1,0,1,0,1,0,1)
)
test$secondDose <- with(test,nth(where=evid==1, n=2,within=id))
test$lastDose <- with(test,nth(where=evid==1,n=-1,within=id))
test$beforeDose2 <- with(test,before(where=evid==1, n=2,within=id))
test$tafd <- with(test,time - first(time,where=evid==1,within=id))#time after first dose
test
#   id time evid secondDose lastDose beforeDose2 tafd
# 1  1   1   0         4         4         TRUE   -1
# 2  1   2   1         4         4         TRUE    0
# 3  1   3   0         4         4         TRUE    1
# 4  1   4   1         4         4        FALSE    2
# 5  1   5   0         4         4        FALSE    3
# 6  2   1   1         8        10         TRUE    0
# 7  2   2   0         8        10         TRUE    1
# 8  2   3   1         8        10        FALSE    2
# 9  2   4   0         8        10        FALSE    3
#10  2   5   1         8        10        FALSE    4
only(
  1:9,
  where= c(0,1,0, 1,0,1, 0,0,0),
  within=c(1,1,1, 2,2,2, 3,3,3)
)
#[1] 2 2 2 NA NA NA NA NA NA

```

**Description**

Generic, with a method for `data.frame` that right-justifies headers and columns, padding with spaces.

**Usage**

```
## S3 method for class 'data.frame'  
fixedwidth(x, ...)
```

**Arguments**

<code>x</code>	a <code>data.frame</code>
<code>...</code>	extra arguments, ignored

**Details**

There does not seem to be a way to get `write.table` to produce fixed-width output. `fixedwidth.data.frame` gives essentially the same result as `print.data.frame`. Columns and headers are padded, suitable for output to a file using the defaults for `write.table` (`sep=' '`).

**Value**

`data.frame`; all columns are character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [write.table](#)

**Examples**

```
fixedwidth(head(Theoph))
```

---

fixProblem

*Clean Up a Control Stream*


---

## Description

These functions are used to re-write a control stream for NONR(checkrunno=TRUE, ...).

## Usage

```
fixProblem(x,run)
fixFile(x,run)
explicitPath(x)
extractPath(x)
resolve(file,dir)
scavenge(expr,lines)
extfile(ctlfile,dir,extreg,...)
tabfile(ctlfile,dir,tabreg='(?<!par)\\.tab',...)
parfile(ctlfile,dir,parreg='par\\.tab',...)
msffile(ctlfile,dir,msfreg='^(?!\\$MSFI).*\\.msf',...)
```

## Arguments

x	character vector, i.e. read from control stream file
run	run designator to substitute at predefined locations
file	an absolute or relative filepath
dir	a directory
expr	a regular expression to locate relevant lines
lines	lines in which to search for expr
ctlfile	character vector representing the control stream
extreg	a regular expression to locate an arbitrary file
tabreg	a regular expression to locate the '.TAB' file
parreg	a regular expression to locate the 'par.TAB' file
msfreg	a regular expression to locate the '.MSF' file
...	ignored

## Details

fixProblem tries to substitute the run name in the NONMEM \$PROBLEM statement with the current run name. It looks for any number of leading spaces, and then RUN or RUN\# (optional). The next space delimited word is replaced.

fixFile tries to replace the run name in filenames ending in '.msf', 'par.tab', or '.tab'. Currently it relies on dirname and basename, which use the same file separator on 'Nix and Windows. basename fails if there is no path separator, e.g., FILE=40.tab. fixFile pre-processes its argument

with `explicitPath`, as a work-around. Lines containing 'MSFI' formerly were coerced using explicit paths, but are now ignored.

`explicitPath` finds lines that contain '.tab' or '.msf' but not '/'. It tries to place './' before 'run.msf' or 'run.tab'.

'extractPath' isolates the file portion of a line of text. The file portion consists of a sequence of nonspace characters following 'MSF=', 'MSFO=', or 'FILE='. Spaces may occur before and after the equality sign. This function is currently used by `runNonmem` to locate the tabfile, parfile, and msffile.

`resolve` does nothing to absolute filepaths in `file`, but expresses others (those beginning with '.') relative to `dir`.

`tabfile`, `parfile`, and `msffile` calculate corresponding filepaths from their arguments, relying on `extfile` as the common engine.

### Value

character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [runNonmem](#)
- [dataSynthesis](#)

### Examples

```

prob <- c(
  '$ 1000 desc',
  '$1000 desc',
  '$RUN1000 desc',
  '$ RUN#1000 desc',
  '$ RUN# 1000 desc'
)
fixProblem(prob,2000)
msf <- c(
  '$EST MAXEVAL=9999 MSF=../1000.MSF',#standard
  '$EST MAXEVAL=9999 msf=../1000.MSF',#case change
  '$EST MAXEVAL=9999 MSF = ../1000.MSF',#one or two spaces
  '$EST MAXEVAL=9999 MSFO=../1000.MSF',#optional 0
  '$EST MAXEVAL=9999 MSF=../1000.msf',#case change
  '$EST MAXEVAL=9999 MSF=../1000.msf INTER',#trailing text
  '$EST MAXEVAL=9999 MSF=../1000/1000.msf',#non-target path elements
  '$EST MAXEVAL=9999 MSF=1000.msf',#no path

```

```

'$MSFI 1000.MSF',#non-target file name
'$INPUT etc'#non-target
)
fixFile(msf,2000)
tab <- c(
'ONEHEADER NOPRINT FILE=../1000.TAB',
'ONEHEADER NOPRINT FILE = ../1000.TAB',
'ONEHEADER NOPRINT FILE=../1000par.TAB',
'ONEHEADER NOPRINT file=../1000par.tab',
'ONEHEADER NOPRINT file=../1000par.tab',
'ONEHEADER NOPRINT file= 1000par.tab',
'ONEHEADER NOPRINT FILE=../1000.TAB',
'ONEHEADER NOPRINT FILE=~ /1000.TAB'
)
fixFile(tab,2000)
files <- c(
'1000.msf',
'1000.tab',
'../1000.tab',
'~/1000.tab',
'$TABLE EVID FILE=1000.tab',
'$TABLE EVID FILE=1000par.tab',
'$MSFI 1000.msf'
)
explicitPath(files)
extractPath(msf)
extractPath(tab)
resolve('../file.ext',dir='project/pk')
resolve('../file.ext',dir='project/pk/')
scavenge('.tab',lines=c('text file.TAB','text file.csv'))
ctlfile <- c('text file=../100.tab etc','text file=../100par.tab etc','text file=../100.msf etc')
tabfile(ctlfile,dir='projectdir/100')
parfile(ctlfile,dir='projectdir/100')
msffile(ctlfile,dir='projectdir/100')

```

---

fhtable2data.frame	<i>Convert ftable to data.frame</i>
--------------------	-------------------------------------

---

## Description

Convert ftable to data.frame As Displayed

## Usage

```
fhtable2data.frame(x,...)
```

## Arguments

x	ftable
...	ignored

**Details**

`as.data.frame.ftable` does indeed convert an `ftable` to `data.frame`, but it gives a stacked result. Use this function to give something more like what is displayed at the prompt (suitable for passing to latex).

**Value**

`data.frame`

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**Examples**

```
ftable(mtcars[c("cyl", "vs", "am", "gear")])
as.data.frame(ftable(mtcars[c("cyl", "vs", "am", "gear")]))
ftable2data.frame(ftable(mtcars[c("cyl", "vs", "am", "gear")]))
```

---

<code>getCovs</code>	<i>Return Non-commented Non-duplicated Rows of a NONMEM Data Set</i>
----------------------	----------------------------------------------------------------------

---

**Description**

Given a file and directory, retrieve the NONMEM source data set and limit to one row per subject.

**Usage**

```
getCovs(file, dir)
```

**Arguments**

<code>file</code>	relative path to a data file
<code>dir</code>	directory to which that path is relative

**Details**

The data file in a control stream is typically expressed relative to the control stream's directory. This tool resolves that reference and returns the file, modified to represent constant covariates (comments stripped, first row per ID).

**Value**

`data.frame`

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTTR](#)

---

getCwres

*Return the Conditional Weighted Residuals Pertinent to a Given Run*

---

**Description**

Locates 'cwtab1.deriv' and calls compute.cwres. Then returns the CWRES column from 'cwtab1'.

**Usage**

getCwres(directory)

**Arguments**

directory      NONMEM run directory

**Value**

numeric

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTTR](#)

---

getdname	<i>Extract the Path and File from the DATA Block in the Control Stream Associated with a Filename</i>
----------	-------------------------------------------------------------------------------------------------------

---

**Description**

Extracts the data path from the control stream, e.g., for use with getCovs.

**Usage**

```
getdname(filename)
```

**Arguments**

filename	path for the control stream
----------	-----------------------------

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTR](#)

---

getPars	<i>Return the Non-duplicated (ID) Rows Corresponding to the Specified Filename</i>
---------	------------------------------------------------------------------------------------

---

**Description**

Useful for retrieving parameters that are constant within subject.

**Usage**

```
getPars(file)
```

**Arguments**

file	path for a file, typically '*par.TAB'
------	---------------------------------------

**Value**

data.frame

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTTR](#)

---

getTabs	<i>Return the Observation Rows (EVID==0) Corresponding to the Specified Filename</i>
---------	--------------------------------------------------------------------------------------

---

**Description**

Useful for returning the PK portion of NONMEM run output.

**Usage**

```
getTabs(file)
```

**Arguments**

file                    path for a file, typically ‘\*.TAB’

**Value**

data.frame

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTTR](#)

---

`glue`*Paste Items with No Space*

---

**Description**

The default separator for `paste` is a space. Pasting with an empty string as separator seems so ubiquitous as to merit its own function. `glue` is just a wrapper for `paste` with an empty string separator.

**Usage**

```
glue(..., sep='', collapse=NULL)
```

**Arguments**

<code>...</code>	one or more R objects, to be converted to character vectors.
<code>sep</code>	a character string to separate the terms.
<code>collapse</code>	an optional character string to separate the results.

**Details**

All arguments are passed unmodified to `paste`. See the related help for details.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [paste](#)

**Examples**

```
glue('cross', 'road')
```

---

groupnames	<i>Construct a Single Vector of Group Names from Possibly Multiple Indices to the Plotting Data Set</i>
------------	---------------------------------------------------------------------------------------------------------

---

## Description

Callers of PLOTR may wish to group on more than one index. This function combines those indices into text for a single index vector.

## Usage

```
groupnames(data, grp, grpnames = NULL, run)
```

## Arguments

data	data.frame in which to find the groups
grp	columns by which to group
grpnames	optional substitute names
run	run name

## Value

factor

## Author(s)

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [PLOTR](#)

half.matrix

*Interconvert Between a Symmetric Matrix and a Triangular Matrix***Description**

Given a symmetric matrix, `half.matrix` converts the corresponding lower (upper) triangular matrix to a vector in row- (column-) major order. The vector is named using the convention `row.column` (for the lower interpretation) and is classed as `halfmatrix`. The function `as.matrix.halfmatrix` converts a `halfmatrix`-like vector to a symmetric matrix. The `halfmatrix` method for generic `offdiag` selects just the off-diagonal elements from the (named) vector of the corresponding class.

**Usage**

```
## S3 method for class 'matrix'
half(x,...)
## S3 method for class 'halfmatrix'
as.matrix(x,...)
## S3 method for class 'halfmatrix'
as.halfmatrix(x,...)
## Default S3 method:
as.halfmatrix(x,...)
## S3 method for class 'halfmatrix'
offdiag(x,...)
```

**Arguments**

`x` symmetric matrix (half) or `halfmatrix` equivalent (`as.matrix`)  
`...` extra arguments, ignored

**Details**

It is an error if `x` is a matrix but not symmetric. If `x` is a vector of appropriate length it can be converted to a symmetric matrix by specifying the method explicitly, even if `x` is not classed as `halfmatrix`: `as.matrix.halfmatrix(x)`. `x` can also be converted to `halfmatrix` explicitly, in which case `as.matrix(x)` suffices.

**Value**

vector with as many elements as a triangular matrix corresponding to `x` (except `as.matrix` returns matrix)

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [ord.matrix](#)
- [isSymmetric.matrix](#)

**Examples**

```
half(diag(3))
as.matrix(half(diag(3)))
as.matrix.halfmatrix(1:15)
as.matrix(as.halfmatrix(1:15))
as.halfmatrix(1:10)
offdiag(as.halfmatrix(1:10))
```

---

hash

*Supply Comment Characters with Output*

---

**Description**

Any expression that generates output can be wrapped in a call to hash. The usual output will be returned, with a hash character at the start of each line.

**Usage**

```
hash(x, char='#')
```

**Arguments**

x	an R expression
char	text to prepend to each line

**Details**

This is useful for embedding the result of an expression in a script.

**Value**

used for side effects

**Warning**

Don't "hash" an expression that directly or indirectly calls a function being debugged. The debug output will be 'sunk' to a connection. If you get in this situation, recover by typing `sink(NULL)` until output is restored.

**Author(s)**

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [file](#)
- [sink](#)

---

helpAdminister      *Mifuns: Administrative*

---

## Description

Tools for administrative tasks.

## Details

### [Mifuns](#)

I want to ....

- [delete](#) files and directories; log what was deleted to a text file
- [embed](#) R output in a script
- [document](#) acceptance of an R installation
- [check](#) whether your files are subversioned
- [test](#) whether Subversion thinks your files are text
- [tell](#) Subversion to treat your files as text
- [create](#) an FDA-style electronic appendix from a Subversioned directory
- [locate](#) path-variant versions of a file

---

helpClasses      *Mifuns: Data Classes*

---

## Description

Classes for building feature-rich data.

**Details**[Mifuns > Data](#)

I want to ....

- [create](#) and manipulate comment objects
- [create](#) and manipulate flags
- [create](#) and manipulate keyed data.frames
- [join](#) keyed data.frames
- [create](#) and manipulate nm objects
- [modify](#) the assembly of nm objects

---

`helpDataFrame`*Mifuns: Working with Data Frames*

---

**Description**

Tools for processing data.frames.

**Details**[Mifuns > Data](#)

I want to ....

- [move](#) columns in a data frame
- [rename](#) selected columns
- [format](#) a data.frame as fixed-width
- [add](#) named elements to a data.frame or list

---

`helpList`*Mifuns: Defined Functions*

---

**Description**Exhaustive list of functions defined by **Mifuns**.

## Details

### Mifuns

- [\[.comment](#) subset a comment
- [\[.flag](#) subset a flag
- [\[.keyed](#) subset a keyed data frame
- [\[.miTemporal](#) subset a miTemporal object
- [\[.nmctl](#) subset a control stream object
- [\[\[.comment](#) select element from a comment
- [\[\[.flag](#) select element from a flag
- [\[\[.miTemporal](#) select element from a miTemporal object
- [\[\[.nmctl](#) select element from a control stream object
- [\[<-.miTemporal](#) assign to a miTemporal object
- [%nested.in%](#) test whether a is nested in b
- [%nests%](#) test whether a nests b
- [accept](#) document the acceptance of an R installation
- [acceptance](#) display the acceptance documentation for an R installation
- [acr](#) calculate accumulation ratio
- [after](#) test whether elements occur after some condition is TRUE
- [align.decimal](#) align numeric column in latex table on the decimal
- [aggregate.keyed](#) aggregate a keyed data frame
- [and.keyed](#) produce inner join of two keyed data frames
- [as.character.comment](#) convert comment to character
- [as.character.flag](#) convert flag to character
- [as.character.miTemporal](#) convert miTemporal to character
- [as.character.nmctl](#) convert nmctl to character
- [as.chartime](#) convert to time (generic, internal)
- [as.chartime.numeric](#) convert numeric to time
- [as.comment](#) convert to comment
- [as.comment.comment](#) convert comment to comment
- [as.comment.default](#) convert default to comment
- [as.csv.filename](#) convert to csv.filename
- [as.csv.filename.character](#) convert character to csv.filename
- [as.data.frame.block](#) convert a block of lines to data frame
- [as.data.frame.comment](#) convert comment to data frame
- [as.data.frame.flag](#) convert flag to data frame
- [as.file.runlog](#) convert to convert runlog to file
- [as.filename](#) convert to filename

- [as.filename.character](#) convert character to filename
- [as.flag](#) convert to flag
- [as.flag.default](#) convert default to flag
- [as.flag.flag](#) convert flag to flag
- [as.halfmatrix](#) generic
- [as.halfmatrix.default](#) convert a vector to halfmatrix
- [as.halfmatrix.halfmatrix](#) preserve halfmatrix
- [as.keyed](#) convert to keyed
- [as.keyed.data.frame](#) convert data frame to keyed
- [as.list.nmctl](#) convert nmctl to list
- [as.matrix.halfmatrix](#) convert a vector to a symmetric matrix
- [as.miDate](#) convert to miDate
- [as.miDate.character](#) convert character to miDate
- [as.miDate.Date](#) convert Date to miDate
- [as.miDate.dates](#) convert dates (chron) to miDate
- [as.miDate.miDate](#) convert miDate to miDate
- [as.miDate.numeric](#) convert numeric to miDate
- [as.miDateTime](#) convert to miDateTime
- [as.miDateTime.character](#) convert character to miDateTime
- [as.miDateTime.chron](#) convert chron to miDateTime
- [as.miDateTime.miDate](#) convert miDate to miDateTime
- [as.miDateTime.miDateTime](#) convert miDateTime to miDateTime
- [as.miDateTime.numeric](#) convert numeric to miDateTime
- [as.miDateTime.POSIXct](#) convert POSIXct to miDateTime
- [as.miDateTime.POSIXlt](#) convert POSIXlt to miDateTime
- [as.miTime](#) convert to miTime
- [as.miTime.character](#) convert character to miTime
- [as.miTime.miTime](#) convert miTime to miTime
- [as.miTime.numeric](#) convert numeric to miTime
- [as.miTime.times](#) convert times (chron) to miTime
- [as.moot](#) convert to moot
- [as.nm](#) convert to nm (NMTRAN-ready NONMEM data set)
- [as.nm.data.frame](#) convert data frame to nm
- [as.nmctl](#) convert to nmctl
- [as.nmctl.character](#) convert character to nmctl
- [as.numeric.chartime](#) convert chartime to numeric
- [as.pxml.ext](#) convert a param file to XML

- [as.rigged](#) convert to rigged
- [as.runlog.file](#) convert file to runlog
- [as.runlog.unilog](#) convert unilog to runlog
- [as.unilog.lst](#) convert nonmem output (.lst) to unilog
- [as.unilog.pxml](#) convert nonmem output (.ext) to unilog
- [as.unilog.run](#) convert run to unilog
- [as.unilog.runlog](#) convert runlog to unilog
- [as.xml](#) convert object to XML
- [as.xml.character](#) convert character to XML
- [as.xml.data.frame](#) convert data.frame to XML
- [as.xml.default](#) convert arbitrary object to XML
- [at](#) test whether elements occur where some condition is TRUE
- [attribute](#) encode an attribute for an XML open tag
- [auc](#) calculate area under the curve
- [AUC](#) calculate the area under the curve for each subject over the time interval for dv using the trapezoidal rule
- [aug](#) add named elements to an object
- [autolog.pl](#) prepare command lines for autolog.pl.
- [backtrans](#) backtransform cols in x, returning x
- [badAmt](#) generic
- [badAmt.nm](#) AMT is NA where EVID is 1
- [badDv](#) generic
- [badDv.nm](#) DV is NA where EVID is 0
- [badII](#) generic
- [badII.nm](#) II is greater than 0 where ADDL is NA or 0 (unless SS > 0)
- [bakfor](#) impute NAs using last observation carried forward after next observation carried backward
- [before](#) test whether elements occur before some condition is TRUE
- [bin](#) calculate bin limits for x and place each value in a bin, returning the number of values in each bin
- [bmi](#) calculate body mass index
- [bracket](#) create an XML tag
- [breaks](#) calculate breaks between grouped elements
- [bsa](#) calculate body surface area
- [c.comment](#) combine comment
- [c.flag](#) combine flag
- [c.miTemporal](#) combine miTemporal
- [cavg](#) calculate average concentration

- [check.subjects](#) check for missing data, numeric data, ranges, etc.
- [clear](#) drop regular patterns from a character vector
- [closers](#) set closing positions for wikmath nests
- [CLNR](#) delete files and directories and log what was deleted to a text file in the project
- [cmax](#) calculate maximum steady-state concentration
- [cmin](#) calculate minimum steady-state concentration
- [colname<-](#) change identified column names
- [command](#) generate a latex command
- [compileflag](#) calculate compiler flag for qsub
- [compiler](#) extract compiler specification from configuration file
- [compute.cwres](#) compute conditional weighted residuals
- [config](#) calculate path to configuration file
- [constant](#) generic
- [constant.default](#) test whether values of x are constant within an index
- [contains](#) test a character vector for occurrences of a pattern
- [covariatePlots](#) create diagnostic plots for covariates
- [crcl](#) calculate creatinine clearance
- [crosses](#) test whether a crosses b
- [css](#) calculate steady-state concentration
- [ctl2xml](#) isolate xml comments from a control stream
- [cwresPlots](#) create conditional weighted residual diagnostic plots
- [dataFormat](#) combine NONMEM predictions, parameter estimates, and covariates into a standard format
- [dataSynthesis](#) build a standard plotting data set for a given NONR run
- [deranged](#) generic
- [deranged.data.frame](#) instantiate records implied by range data
- [deranged.keyed](#) instantiate records implied by range data
- [diagnosticPlots](#) create standard diagnostic plots
- [distance](#) calculate element positions relative to the  $n^{th}$  TRUE element positions (usually, within levels of an index)
- [dupKeys](#) generic
- [dupKeys.default](#) give a logical index to duplicated rows of a keyed data frame, AND the rows of which they are duplicates
- [dupKeys.nm](#) default behavior, after stripping comments
- [electronicAppendix](#) create an FDA-style electronic appendix from a subversioned directory
- [episcrpt](#) run a script in an environment that defines the passed arguments
- [explicitPath](#) add separators to control stream paths that have none
- [extfile](#) extract arbitrary file specification from a control stream

- `extractPath` isolate file path from surrounding text (MSF, TAB files).
- `f` alias for `as.flag`
- `falseAmt` generic
- `falseAmt.nm` AMT defined where EVID is not 1
- `falseDv` generic
- `falseDv.nm` DV is defined where EVID is not 0
- `filename` generate standard file name (internal)
- `first` repeat, for each element, the first element where a condition is TRUE (usually, within levels of an index)
- `fixedwidth` convert to fixed-width format
- `fixedwidth.data.frame` convert a `data.frame` to fixed-width format
- `fixFile` rewrite run name in control stream file specifications
- `fixProblem` rewrite run name in control stream PROBLEM statement
- `forbak` impute NAs using last observation carried forward and then next observation carried backward
- `format.comment` format comment
- `format.flag` format flag
- `format.miDate` format `miDate`
- `format.miDateTime` format `miDateTime`
- `format.miTime` format `miTime`
- `format.nmctl` format `nmctl`
- `fable2data.frame` convert `fable` to `data.frame` as displayed
- `getCovs` return non-commented non-duplicated rows of a NONMEM data set (internal)
- `getCwres` return the conditional weighted residuals pertinent to a given run (internal, expects `cwtab1.deriv`)
- `getdname` extract the path and file from the DATA block in the control stream associated with `filename` (internal)
- `getPars` return the non-duplicated (ID) rows corresponding to the specified `filename` (internal)
- `getTabs` return the observation rows (`EVID==0`) corresponding to the specified `filename` (internal)
- `glue` paste with no separator
- `groupnames` construct a single vector of group names from possibly multiple indices to the plotting data set (internal)
- `half` generic
- `half.matrix` vectorize the lower triangular portion of a symmetric matrix
- `hash` prepend a character to each line of output
- `hide` generic
- `hide.data.frame` comment out the specified rows of a data frame, appending a flag that suggests the reason

- [hidden](#) generic
- [hidden.data.frame](#) show commented rows of a data frame
- [ibw](#) calculate ideal body weight
- [iterations](#) retrieve iteration statistics from NONMEM output file
- [ind.cwres](#) compute conditional weighted residuals (internal)
- [inner](#) generic
- [inner.data.frame](#) reduce columns to inner quantiles by imputing NA
- [is.alpha](#) test whether text is letters-only
- [is.cwres.readable.file](#) compute conditional weighted residuals (internal)
- [is.latex.token](#) test whether text is valid latex command or environment
- [is.one.nonalpha](#) test whether text is singular non-letter
- [is.square](#) generic
- [is.square.matrix](#) test whether a matrix is square
- [isSubversioned](#) check if files are subversioned
- [isSubversionedFile](#) check if one file is subversioned
- [justUnits](#) extract units from wikimath
- [ke](#) calculate elimination constant
- [key](#) return the key of a keyed data frame
- [key<-](#) assign a key
- [last](#) repeat, for each element, the last element where a condition is TRUE (usually, within levels of an index)
- [latest](#) identify the latest of each set of variants
- [latex.args](#) format latex arguments
- [latex.options](#) format latex options
- [lbm](#) calculate lean body mass
- [left.keyed](#) produce left join of two keyed data frames
- [lhs](#) extract left-hand side of wikimath
- [locf](#) impute NA using last observation carried forward
- [lookup](#) translate among parameter attributes (vector)
- [lookup.one](#) translate among parameter attributes (scalar)
- [ltable](#) convert to a latex table
- [ltable.data.frame](#) convert a data.frame to a latex table
- [map](#) map one set of values to another
- [maxChar](#) return the number of printed characters for the widest element of x
- [merge.keyed](#) merge keyed data frames, preserving attributes
- [merge.nm](#) merge nm objects
- [metaSub](#) generic

- [metaSub.character](#) systematically substitute elements in a character vector
- [metaSub.filename](#) systematically substitute elements in a text file
- [minus.keyed](#) drop rows in x that have matching rows in y
- [minus.moot](#) drop moot rows
- [moot](#) generic
- [moot.nm](#) identify moot rows (rows not influencing NONMEM estimates)
- [msffile](#) extract msffile specification from a control stream
- [naInContext](#) display rows of data with missing values, as well as other rows with the same key
- [naKeys](#) generic
- [naKeys.default](#) detect rows for which one or more key fields is NA
- [naKeys.nm](#) default behavior, after stripping comments
- [name<-](#) change identified object names
- [nest](#) nest an XML fragment in a parent element
- [nix](#) identify Unix-like platforms (internal)
- [nm](#) give a zero-row data frame with suitable columns and column classes; i.e. a template for NMTRAN data sets
- [nm.pl](#) prepare command lines for nm.pl
- [nmPlots](#) a list of functions that plot nm objects
- [nmVersion](#) extract NONMEM version specification from configuration file
- [nocb](#) impute NA with next observation carried backward
- [NONR](#) run NONMEM and create diagnostic plots
- [NONR72](#) run NONMEM 7.2.0 or later
- [nospace](#) drop spaces from text
- [noPk](#) generic
- [noPk.nm](#) rows where EVID is never 0 within SUBJ
- [noUnits](#) drop units from wikimath
- [nth](#) repeat, for each element, the  $n^{th}$  element where a condition is TRUE (usually, within levels of an index)
- [nxt](#) calculate the next element for each element of x
- [offdiag](#) generic
- [offdiag.halfmatrix](#) select just the off-diagonal elements of a halfmatrix
- [omegacor](#) convert omega covariance in NONMEM output to correlation matrix
- [only](#) repeat, for each element, the only element where a condition is TRUE (usually, within levels of an index)
- [Ops.keyed](#) use concise syntax to join data frames
- [ord](#) generic
- [ord.halfmatrix](#) compute the order of a half matrix
- [ord.matrix](#) compute the order of a symmetric matrix

- [packageCheck](#) load a package and run package-level examples
- [panel.bar](#) for each level, plot vertical (or horizontal) bars corresponding to x (y).
- [panel.covplot](#) plot distributions with respect to reference values; i.e. combine [panel.cuts](#), [panel.densitystrip](#), and [panel.ref](#)
- [panel.cuts](#) for each level, plot percent observations in each vertical category defined by cuts
- [panel.densitystrip](#) for each level, plot a filled polygon representing smoothed density of the distribution
- [panel.hist](#) for each level, plot a histogram
- [panel.ref](#) plot a reference region in a stripplot
- [panel.stratify](#) handle strips (levels) of data one at a time
- [params](#) list documented model parameters
- [parens](#) wrap text in parentheses
- [parfile](#) extract parfile specification from a control stream
- [partab](#) construct a model parameter table
- [plot.nm](#) generate standard plots for an nm object using [nmPlots](#)
- [plotfilename](#) make a name for a diagnostic plot file
- [PLOTTR](#) create diagnostic plots for NONMEM runs
- [plus.keyed](#) produce and outer join of keyed data frames
- [plus.rigged](#) merge an nm object with a rigged data frame
- [posmat](#) coerce a matrix to be positive definite
- [predoseDv](#) generic
- [predoseDv.nm](#) DV is defined before the first record within SUBJ where EVID is 1
- [prev](#) calculate the previous element for each element of x
- [print.comment](#) print comment
- [print.flag](#) print flag
- [print.halfmatrix](#) print halfmatrix
- [print.keyed.summary](#) print keyed summary
- [print.miTemporal](#) print miTemporal
- [print.nm.summary](#) print nm summary
- [print.nmctl](#) print nmctl
- [purge.dir](#) purge a directory (internal)
- [purge.files](#) purge files (internal)
- [qsub](#) prepare commandlines for qsub
- [read.cwres.data](#) compute conditional weighted residuals (internal)
- [read.nm](#) read a csv file and try to transform to nm
- [read.nmctl](#) read a NONMEM control stream, converting to modifiable object
- [reapply](#) apply a function across cells of an indexed vector, giving an identically-indexed result
- [rep.comment](#) repeat comment

- [rep.flag](#) repeat flag
- [rep.miTemporal](#) repeat miTemporal
- [resample](#) generic
- [resample.csv.filename](#) create replicate data sets by stratified sampling with replacement, after reading csv file
- [resample.data.frame](#) create replicate data sets by stratified sampling with replacement
- [resample.filename](#) create replicate data sets by stratified sampling with replacement, after reading file
- [resolve](#) reinterpret relative file paths with respect to specified directory.
- [rhs](#) extract right-hand side of wikimath
- [rig](#) generic
- [rig.nm](#) generate extra rows, spaced across each subject's timecourse, for obtaining model predictions in NONMEM (EVID==2)
- [rinvchisq](#) generate inverse chi-square distribution
- [riwish](#) generate inverse Wishart distribution
- [rlog](#) generate a combined run log for multiple NONMEM runs
- [row2tabular](#) convert a vector to a latex tabular row
- [runCommand](#) issue the system call that invokes NONMEM (internal)
- [runhead](#) determine whether each element is the start of a run
- [runlog](#) create a zero-row runlog
- [runNonmem](#) process a request for NONMEM invocation
- [runstate](#) determine the status of a run
- [safe.call](#) call a function, passing only those extra arguments that the function accepts
- [safeQuote](#) single-quote unquoted text that contains space
- [scavenge](#) find an expression in lines of text
- [seq.miTemporal](#) generate miTemporal sequence
- [setCwres](#) append conditional weighted residuals to an appropriate file
- [shuffle](#) move columns in a data frame
- [sigmacor](#) convert sigma covariance in NONMEM output to correlation matrix
- [simpar](#) generate deviates of a set of model parameters for simulation
- [simblock](#) generate deviates of the elements of a variance-covariance matrix
- [snap](#) coerce values to nearest of candidates
- [sort.keyed](#) sort a keyed data frame
- [spaces](#) create a string of spaces
- [sqrtn](#) support compute.cwres
- [stableMerge](#) produce a left join, with strict error checking
- [star](#) replace asterisk in x with y (internal)
- [strain](#) reduce x to those elements that occur in options (internal)

- [summary.keyed](#) summarize a keyed data frame
- [summary.hidden.data.frame](#) summarize hidden rows of a data frame
- [summary.nm](#) summarize an nm object
- [svnIsText](#) check if subversioned files are text rather than binary
- [svnMarkAsText](#) tell Subversion to treat files as text
- [svnMarkAsNonText](#) tell Subversion to treat files as binary
- [svnMimeType](#) get the Subversion mime-type for files
- [svnPropGet](#) get a Subversion property for files
- [svnPropGetFile](#) get a Subversion property for one file
- [svnPropSet](#) set a subversion property on files
- [svnPropSetFile](#) set a Subversion property on a file
- [synthesis](#) sequentially left-join an arbitrary number of data frames, picking up novel columns (internal)
- [tabfile](#) extract tabfile specification from a control stream
- [tabular](#) convert to a latex tabular environment
- [tabular.data.frame](#) convert a data.frame to a latex tabular environment
- [tabularformat](#) generate a format string for a latex tabular environment
- [tagvalue](#) convert a list to a string of delimited tag-value pairs
- [text2decimal](#) convert mixed text to decimal
- [tmax](#) calculate the time of the maximum concentration
- [Tmax](#) calculate the time associated with the maximum concentration for each subject
- [Tmin](#) calculate the time associated with the minimum concentration for each subject
- [tos](#) extract thetas, omegas, and sigmas from wikimath
- [transform.keyed](#) transform a keyed data frame, preserving class
- [uniKey](#) generic
- [uniKey.keyed](#) create a single character vector from all the key columns
- [unilog](#) create a zero-row unilog
- [unilogcor](#) convert omega or sigma covariance in NONMEM output to correlation matrix
- [unique.miTemporal](#) find unique elements of a temporal vector
- [unitDensity](#) calculate univariate density with maximum equal to 1
- [unitHist](#) calculate univariate histogram architecture with maximum height equal to 1
- [nmVersion](#) extract NONMEM version specification from configuration file
- [variants](#) locate variants of a file in distinctive subdirectories
- [wiki2label](#) convert wikimath to a label, e.g. CL/F
- [wiki2latex](#) convert wikimath to latex
- [wiki2parameter](#) convert wikimath to a parameter, e.g. THETA1
- [wiki2plotmath](#) convert wikimath to plotmath

- [wikiparse](#) parse wikimath
- [wikitab](#) extract wikimath specification from a control stream
- [win](#) identify Windows platforms (internal)
- [wrap](#) wrap text in a latex environment
- [write.nm](#) write an nm object to file
- [write.nmctl](#) write an nmctl object to file
- [xtfrm.comment](#) produce a numeric vector that sorts in the same order as comment
- [xtfrm.flag](#) produce a numeric vector that sorts in the same order as flag
- [xtfrm.miTemporal](#) produce a numeric vector that sorts in the same order as miTemporal
- [xyplot.ext](#) plot the parameter search history for a NONMEM7 run
- [zeroAmt](#) generic
- [zeroAmt.nm](#) AMT is zero where EVID is 1
- [zeroDv](#) generic
- [zeroDv.nm](#) DV is zero where EVID is zero

---

helpMatrix

*Mifuns: Working with Matrices*

---

### Description

Tools for processing matrices.

### Details

[Mifuns > Data](#)

I want to ....

- [interconvert](#) between square matrices and triangular matrices

---

helpModel

*Mifuns: Modeling*

---

### Description

Tools for preparing, executing, and analyzing models.

**Details**[Mifuns](#)

I want to ....

- [assemble](#) an analysis data set
- [make](#) multiple, systematic substitutions in text files, e.g., control streams
- [convert](#) NONMEM control streams to modifiable objects
- [run](#) NONMEM and create diagnostic plots
- [determine](#) the status of a run
- [compute](#) conditional weighted residuals in NONMEM 6, etc.
- [visualize](#) model results

---

helpPharmacometric      *Mifuns: Pharmacometrics*

---

**Description**

Tools for investigating pharmacometric properties.

**Details**[Mifuns > Data](#)

I want to ....

- [calculate](#) various clinical measurements, e.g., body size
- [calculate](#) one-compartment model properties
- [summarize](#) a table of subject data
- [calculate](#) Tmax
- [calculate](#) Tmin
- [calculate](#) AUC

---

helpPrepare

*Mlfuns: Data Preparation*

---

### Description

Tools for preparing pharmacometric analysis data sets.

### Details

#### [Mlfuns](#)

I want to ....

- work with [vectors](#)
- work with [data.frames](#)
- work with [matrices](#)
- [use](#) classes specialized for data assembly
- [investigate](#) pharmacometric properties
- [perform](#) strategic data manipulations

---

helpQuantify

*Mlfuns: Statistics*

---

### Description

Tools for calculating statistics.

### Details

#### [Mlfuns](#)

I want to ....

- [assign](#) values in a vector to quantile bins
- [coerce](#) values to nearest of candidates

---

helpReport

*Mifuns: Reporting*

---

## Description

Tools for summarizing and reporting pharmacometric analyses.

## Details

### Mifuns

I want to ....

- [identify](#) documented parameter names
- [lookup](#) some other form of parameter names
- [generate](#) a combined run log from multiple NONMEM runs
- [create](#) a model parameter table
- [convert](#) wikimath to latex or plotmath
- [manipulate](#) wikimath elements
- [list](#) the iteration statistics in the NONMEM output
- [convert](#) the NONMEM output omega or sigma covariance matrix to correlations
- [reformat](#) an ftable for printing with latex
- [convert](#) a data.frame for printing with latex

---

helpSimulate

*Mifuns: Simulation*

---

## Description

Tools for conducting simulations.

## Details

### Mifuns

I want to ....

- [create](#) replicate datasets by stratified sampling with replacement
- [convert](#) a vectorized triangular matrix of random effects to a full matrix
- [create](#) parameters for simulation with uncertainty
- [make](#) multiple, systematic substitutions in text files, e.g., control streams
- [run](#) NONMEM and create diagnostic plots
- [visualize](#) simulation results

---

helpStrategic

*Mifuns: Strategic Data Manipulation*

---

### Description

Tools for strategic data manipulation.

### Details

[Mifuns > Data](#)

I want to ....

- [apply](#) a function across cells of an indexed vector, giving an identically-indexed result
- [identify](#) elements within cells that meet some instance of a condition
- [add](#) keyed data to a data.frame safely

---

helpVector

*Mifuns: Working with Vectors*

---

### Description

Tools for processing vectors.

### Details

[Mifuns > Data](#)

I want to ....

- [convert](#) mixed text to decimal
- [test](#) a character vector for occurrences of a pattern
- [impute](#) missing vector values
- [check](#) some properties of vectors
- [create](#) date, time, and date-time objects
- [map](#) one set of values to another

---

helpVisualize                      *Mlfuns: Plotting*

---

### Description

Tools for creating plots.

### Details

#### Mlfuns

I want to ....

- [create](#) diagnostic plots for finished model runs
- [make](#) ‘forest plots’ of covariate effects
- [handle](#) each level of a stripplot independently
- [plot](#) nm objects
- [plot](#) the parameter search history for a NONMEM7 run

---

ibw                                      *Calculate Various Clinical Indicators*

---

### Description

Calculate body mass index, body surface area, ideal body weight, lean body mass, or creatinine clearance using common equations.

### Usage

```
bmi(wt, ht)
bsa(wt, ht)
lbm(wt, ht, male)
ibw(ht, male, floor = FALSE)
crcl(age, wt, male, scr)
```

### Arguments

wt	weight (actually, mass) in kilograms
ht	height in centimeters
age	age in years
male	logical indicating sex: TRUE indicates male
floor	if TRUE, “inches over 5 feet” is not allowed to be negative.
scr	serum creatinine in mg/dL

**Details**

Units of return values:

**bmi**  $kg/m^2$

**bsa**  $m^2$

**lbm** kg

**ibw** kg

**crcl**  $mL/min$

**Value**

Numeric.

**Author(s)**

Tim Bergsma

**References**

**bmi** [<http://www.halls.md/body-mass-index/bmirefs.htm>]

**bsa** (BSA: Gehan & George variation) <http://www.halls.md/body-surface-area/refs.htm>

**lbm** <http://www.halls.md/body-mass-index/leanbody.htm>

**ibw** <http://www.halls.md/ideal-weight/devine.htm>

**crcl** [http://en.wikipedia.org/wiki/Renal\\_function](http://en.wikipedia.org/wiki/Renal_function)

**Examples**

```
bmi(70,160)
```

```
bsa(70,160)
```

```
lbm(70,160,TRUE)
```

```
ibw(160,TRUE)
```

```
crcl(35,70,TRUE,1.0)
```

---

inner.data.frame

*Limit Data to Inner Quantiles by Imputing NA.*

---

**Description**

inner is generic. inner.data.frame imputes NA for cells within columns that represent extreme quantiles. By default, the 'inner' 95 percent of each column is preserved.

**Usage**

```
## S3 method for class 'data.frame'
inner(
  x,
  prob=0.95,
  tail=0.5*(1-prob),
  lo=tail,
  hi=prob+tail,
  include.lowest=TRUE,
  include.highest=TRUE,
  preserve=character(0),
  id.var=character(0),
  measure.var=setdiff(names(x),c(preserve,id.var)),
  na.rm=FALSE,
  ...
)
```

**Arguments**

<code>x</code>	data.frame
<code>prob</code>	the fraction of data to preserve
<code>tail</code>	the fraction of data to ignore at each extreme
<code>lo</code>	the probability below which data will be ignored
<code>hi</code>	the probability above which data will be ignored
<code>include.lowest</code>	whether to preserve values at probability equal to <code>lo</code>
<code>include.highest</code>	whether to preserve values at probability equal to <code>hi</code>
<code>preserve</code>	vector of names for columns to preserve but ignore
<code>id.var</code>	vector of names for columns that indicate data subsets
<code>measure.var</code>	vector of names for columns to limit
<code>na.rm</code>	passed to <code>quantile</code>
<code>...</code>	passed to <code>fun.aggregate</code>

**Details**

`prob` and `tail` are not actually used internally, but serve only to calculate symmetric defaults for `lo` and `hi`. If the latter are supplied, the former are ignored. Tails need not be symmetric.

By default, all columns are classified as `measure.var`: an attempt will be made to limit such. Columns classified as `preserve` will simply be passed through to the result. Columns classified as `id.var` specify row subsets that are limited independently of each other. See examples.

**Value**

a data frame with the same rows, columns, row order, and column order as `x` (except for dropped columns)

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [helpPrepare](#)

**Examples**

```
x <- airquality[c(1:10,32:41),]
x
inner(x,id.var=c('Month','Day'),na.rm=TRUE)#identity
inner(x,id.var='Month',preserve='Day',na.rm=TRUE)#quantiles within Month
inner(x,preserve=c('Month','Day'),na.rm=TRUE)#quantiles across all rows
inner(x,measure.var=c('Ozone','Solar.R','Wind','Temp'),na.rm=TRUE)#like previous, but dropping Month, Day
inner(rock,prob=0.5)
```

---

is.alpha

*Test For Alphanumeric Content*

---

**Description**

Verify that each element contains only letters.

**Usage**

```
is.alpha(x, ...)
```

**Arguments**

x	character
...	ignored

**Details**

Verifies that each element in x contains only upper or lower case letters.

**Value**

logical

**Author(s)**

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [is.latex.token](#)

## Examples

```
is.alpha(c('aB', 'a2'))
```

---

is.latex.token	<i>Determine if String is Valid Latex Command or Environment</i>
----------------	------------------------------------------------------------------

---

## Description

Determine if string is valid latex command or environment name.

## Usage

```
is.latex.token(x, ...)
```

## Arguments

x	character
...	ignored

## Details

Latex commands (and presumably, environment names) must consist entirely of upper or lower case letters, or must be a single non-letter.

## Value

logical

## Author(s)

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [is.alpha](#)
- [is.one.nonalpha](#)

**Examples**

```
is.latex.token('2', 'word', 'word2')
```

---

is.one.nonalpha

*Check That Each Element is a Single Non-letter*

---

**Description**

Verify that each element is just one non-letter.

**Usage**

```
is.one.nonalpha(x, ...)
```

**Arguments**

x	character
...	ignored

**Details**

Used to help identify proper latex command words.

**Value**

logical

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [is.latex.token](#)

**Examples**

```
is.one.nonalpha(c(2, 'b', 23))
```

---

is.square.matrix	<i>Test Whether a Matrix is Square</i>
------------------	----------------------------------------

---

**Description**

A matrix is square if its dimensions are equal.

**Usage**

```
## S3 method for class 'matrix'  
is.square(x, ...)
```

**Arguments**

x	matrix
...	extra arguments, ignored

**Value**

logical

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

[ord.matrix](#), [isSymmetric.matrix](#)

**Examples**

```
is.square(diag(3))
```

---

isSubversioned	<i>Check Whether Files Are Subversioned</i>
----------------	---------------------------------------------

---

**Description**

Check whether multiple files under Subversion control.

**Usage**

```
isSubversioned(x, ...)
```

**Arguments**

x	character vector of file names
...	ignored arguments.

**Details**

This is a vectorized version of isSubversionedFile.

**Value**

logical

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [isSubversionedFile](#)

---

isSubversionedFile	<i>Check If Single file Is Subversioned</i>
--------------------	---------------------------------------------

---

**Description**

Check if a single file is under Subversion control.

**Usage**

```
isSubversionedFile(file)
```

**Arguments**

file	scalar character file name
------	----------------------------

**Details**

May return TRUE even if file does not exist.

**Value**

logical

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [isSubversioned](#)

---

iterations	<i>Harvest Iteration Statistics from NONMEM Output and Convert from Covariance to Correlation</i>
------------	---------------------------------------------------------------------------------------------------

---

**Description**

iterations excises the relevant portion of the output file, converting the text to parameter and gradient estimates.

**Usage**

```
iterations(x, ...)
```



```

'      -8.1633E+01',
' OITERATION NO.: 15 OBJECTIVE VALUE: 2487.99745856896 NO. OF FUNC. EVALS.:11',
' CUMULATIVE NO. OF FUNC. EVALS.: 178',
' PARAMETER: -9.0277E-02 9.8751E-01 -1.0251E+00 -1.0561E+00 1.8848E-01 1.0188E-01 1.3933E+00 4.9185E-01 2.8957E-01
'      -4.7147E-02',
' GRADIENT: -5.0246E-01 2.2589E-01 -1.7718E+00 7.8352E-01 6.4402E-01 -1.3915E-01 -1.1238E-01 1.7533E-01 1.7063E-01
'      2.8887E-01',
' OITERATION NO.: 20 OBJECTIVE VALUE: 2487.98367110010 NO. OF FUNC. EVALS.:21',
' CUMULATIVE NO. OF FUNC. EVALS.: 246',
' PARAMETER: -8.8682E-02 9.8198E-01 -1.0240E+00 -1.0659E+00 1.7898E-01 1.0123E-01 1.3935E+00 4.9043E-01 2.8974E-01
'      -4.7448E-02',
' GRADIENT: -5.8862E-01 -1.1992E+00 -4.6879E+00 -4.7829E-01 -2.2729E-01 -1.2550E-01 -2.9251E-01 4.8799E-02 1.5000E-01
'      -1.3573E-01',
' OITERATION NO.: 25 OBJECTIVE VALUE: 2487.88855559751 NO. OF FUNC. EVALS.:16',
' CUMULATIVE NO. OF FUNC. EVALS.: 340',
' PARAMETER: -8.8040E-02 1.0105E+00 -9.9958E-01 -1.0162E+00 1.7939E-01 9.7973E-02 1.4278E+00 4.8858E-01 2.7833E-01
'      -4.6731E-02',
' GRADIENT: -9.4356E-03 8.1782E-03 4.5771E-02 -1.9044E-02 -7.8125E-04 -1.0624E-03 -7.2147E-04 3.5290E-03 -2.6680E-03
'      9.0593E-03',
' Elapsed estimation time in seconds: 3.11',
' ',
' #TERM:',
' MINIMIZATION SUCCESSFUL',
' NO. OF FUNCTION EVALUATIONS USED: 340',
' NO. OF SIG. DIGITS IN FINAL EST.: 3.2',
' ',
' ETABAR IS THE ARITHMETIC MEAN OF THE ETA-ESTIMATES',
' AND THE P-VALUE IS GIVEN FOR THE NULL HYPOTHESIS THAT THE TRUE MEAN IS 0.',
' ',
' ETABAR: 2.3774E-03 8.5597E-04 6.8554E-04',
' SE: 6.8571E-02 4.6635E-02 4.7602E-02',
' ',
' P VAL.: 9.7234E-01 9.8536E-01 9.8851E-01',
' ',
' ETAshrink(%): 7.3693E-01 1.6697E+01 6.5847E+00',
' EPSshrink(%): 8.8722E+00',
' ',
' #TERE:',
' Elapsed covariance time in seconds: 4.27',
'1'
)
iterations <- iterations(lst)
iterations
it.dat <- melt(iterations,measure.var=names(iterations)[contains('X',names(iterations))])
xyplot(value~iteration|variable,it.dat[it.dat$course=='parameter',],type='l',ylab='scaled parameter',as.table=TRUE)
xyplot(value~iteration|variable,it.dat[it.dat$course=='gradient',],type='l',ylab='gradient',as.table=TRUE,scal

```

**Description**

Extract logical portions of text formatted with wikimath conventions.

**Usage**

```
justUnits(x, ...)
noUnits(x, ...)
lhs(x, ...)
rhs(x, ...)
nospace(x, ...)
tos(x, ...)
```

**Arguments**

x	character; typically wikimath
...	ignored, or passed to sub for lhs and rhs

**Details**

justUnits extracts the contents of the first parenthetical inclusion. noUnits drops the first parenthetical inclusion. nospace removes all spaces. tos extracts the first occurrence of x\_num, where x is 'theta', 'omega', or 'sigma' (case insensitive) and num is an unsigned decimal (1, 2.1, etc.).

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [wikiparse](#)
- [wiki2plotmath](#)
- [wiki2latex](#)

**Examples**

```
wiki <- "CL/F (L) ~ theta_1 * (WT/70)^theta_2"
justUnits(wiki)
noUnits(wiki)
lhs(wiki)
rhs(wiki)
nospace(noUnits(lhs(wiki)))
tos(wiki)
```

---

latest	<i>Identify the Latest Variants of Each File as Distinguished by Enclosing Subdirectories</i>
--------	-----------------------------------------------------------------------------------------------

---

**Description**

For each of a set of file variants as returned by `variants`, identify that file whose enclosing directory sorts last (i.e., latest if a timestamp).

**Usage**

```
latest(x)
```

**Arguments**

`x` character vector of file paths of the form 'path/file/var1/file'

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [variants](#)

**Examples**

```
latest(  
  c(  
    'path/file/var2/file',  
    'path/file/var1/file'  
  )  
)
```

---

`latex.args`*Format Latex Command Arguments*

---

**Description**

Format latex command arguments.

**Usage**

```
latex.args(x, ...)
```

**Arguments**

<code>x</code>	list or vector
<code>...</code>	ignored

**Details**

Each element wrapped in curly braces; elements are strung together.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [wrap](#)
- [command](#)

**Examples**

```
latex.args(c('arg1', 'arg2'))
```

---

`latex.options`*Format Latex Command Options*

---

**Description**

Format latex command options.

**Usage**

```
latex.options(x, ...)
```

**Arguments**

<code>x</code>	list or vector
<code>...</code>	ignored

**Details**

Elements are strung in a comma separated list, optionally with name= syntax. List is enclosed in square brackets.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [wrap](#)
- [command](#)
- [tagvalue](#)

**Examples**

```
latex.options(c(ht=2,width=3))
```

---

`locf`*Impute Missing Vector Values*

---

**Description**

Impute NA values using other values in the vector.

**Usage**

```
locf(x)
forbak(x)
bakfor(x)
nocb(x)
```

**Arguments**

`x` A vector with some missing values and some non-missing values.

**Details**

`locf` implements “last observation carried forward”: NA’s are imputed with the most recent non-NA value. `nocb` is the complement: “next observation carried backward”: NA’s are imputed with the next non-NA value. `forbak` first executes `locf`, then `nocb`, so that even leading NAs are imputed. If even one non-NA value is present, `forbak` should not return any NAs. `bakfor` does the reverse.

**Value**

A vector of the same class as `x`.

**Author(s)**

Tim Bergsma

**Examples**

```
locf(c(NA, 1, 2, NA, NA, 3, NA, 4, NA))
nocb(c(NA, 1, 2, NA, NA, 3, NA, 4, NA))
forbak(c(NA, 1, 2, NA, NA, 3, NA, 4, NA))
```

**Description**

Parameter names have differing canonical forms depending on whether they are being used in a NONMEM control stream, an R plot, a latex table, etc. The functions described here form a generalized interface for interconverting these forms, given some xml-encoded data in a file, typically a control stream.

**Usage**

```
lookup( x, within, by = 'name', as = NULL, type = 'parameter', ...)
lookup.one(x, within, by = 'name', as = NULL, type = 'parameter', ...)
ct12xml( x, ...)
clear( x, drop = NULL, ...)
```

**Arguments**

<code>x</code>	a vector of parameter names (character)
<code>within</code>	an XML 'document' containing parameter elements (character)
<code>by</code>	the parameter attribute by which to interpret <code>x</code>
<code>as</code>	the parameter attribute as which to re-present <code>x</code> . If null, the text value of the element (if any) is returned.
<code>type</code>	the element type to seek in 'within'
<code>...</code>	passed to other functions
<code>drop</code>	for <code>clear</code> : a vector of patterns to be replaced sequentially with "" (nothing)

**Details**

`clear` helps to isolate XML fragments from a text file. `ct12xml` puts it to use, assuming the fragments occur after the control stream comment character: it drops material up to and including the comment character, and then drops everything up to the first '<'. It is expected that a <parameter> declaration follows.

`lookup.one` is the engine that researches elements of `x`, one at a time.

`lookup` is the typical user interface to this system. Try this.

- In your control stream, create parameter comments for each element, e.g. `TVCL = THETA(1) ; <parameter name='THETA1' lattice='TH1' latex='\theta_1'>typical value of clearance</parameter>`
- In your script, use `readLines` to acquire your control stream as a character vector.
- Use `ct12xml` to convert that vector to XML.
- Pass the converted vector to `lookup` as the argument `within`.

**Value**

All these functions return character. `lookup` returns a vector of names corresponding to `x`, but having some other form (representing some other attribute).

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**Examples**

```
codes <- c(
  '; etc <parameter name="THETA1" lattice="TH1" latex="\\Theta_1">effect of weight on clearance</parameter>',
  'and some other stuff',
  ';<parameter name="OMEGA1" lattice="OM1" latex="\\Omega_1">interindividual variance on volume</parameter>',
  'with maybe a < sign',
  ';<parameter name="SIGMA1" lattice="SG1" latex="\\Sigma_1">additive error</parameter>'
)
codes
doc <- ct12xml(codes)
lookup(c('TH1', 'SG1', 'OM1', NA), within=doc, by='lattice')
lookup(c('THETA1', 'SIGMA1', 'OMEGA1'), within=doc, as='lattice')
```

---

ltable

*Create a Latex Table*


---

**Description**

Create a latex table.

**Usage**

```
ltable(x, ...)
```

**Arguments**

<code>x</code>	dispatch argument
<code>...</code>	ignored

**Details**

`ltable` is generic.

**Value**

character

**Author(s)**

Tim Bergsma

**References**<http://mifuns.googlecode.com>**See Also**

- [ltable.data.frame](#)

---

ltable.data.frame	<i>Convert a Data Frame to a Latex Table</i>
-------------------	----------------------------------------------

---

**Description**

Convert data.frame to latex table.

**Usage**

```
## S3 method for class 'data.frame'
ltable(
  x,
  caption = NULL,
  cap = caption,
  cap.top = TRUE,
  label = NULL,
  options = "!htpb",
  environments = "center",
  file = NULL,
  ...
)
```

**Arguments**

x	data.frame
caption	full version of the caption
cap	short version of the caption, for list of tables
cap.top	Should caption be placed at the top, instead of bottom?
label	optional label
options	options for latex table environment
environments	extra environments to nest between 'table' and 'tabular'.
file	optional file name
...	passed to tabular

**Details**

Converts data.frame to tabular, then wraps it in specified environments, then wraps result in a latex table environment. Result is returned visibly or printed to file and returned invisibly.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [tabular.data.frame](#)

**Examples**

```
ltable(head(Theoph))
```

---

map

*Map One Set of Values to Another*

---

**Description**

Systematically substitute specific values from one set using corresponding values from another set.

**Usage**

```
map(x, from, to)
```

**Arguments**

x	vector
from	vector (elements usually unique)
to	vector with same length as from

**Details**

Occasionally one wants to recode a set of categories using some other idiom. `factor` supports recoding by creative use of the arguments `levels` and `labels`. However, the result is a factor, and may need more transformation. Furthermore, `factor` allows one-to-one reclassification but not many-to-one reclassification (repeated levels is not supported; i.e., it is not directly possible to collapse two levels to a single replacement code).

Here, `from` is the discrete set of values we expect in `x`, and `to` is the element-wise corresponding values with which we wish to re-present elements in `x`. Values in `x` not found in `from` will be represented as `NA` (which is itself a legitimate value for `to`). It is an error if `from` and `to` have different lengths.

**Value**

vector of same class as `to` and same length as `x`

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [match](#)

**Examples**

```
map(
  c('white', 'Asian', 'Indian'),
  from=c('white', 'White', 'black', 'Black', 'asian', 'Asian'),
  to= c( 1,      1,      2,      2,      3,      3 )
)
```

---

metaSub.character

*Create Replicate Text files with Systematic Substitutions*

---

**Description**

`metaSub` is generic. A method is defined for `character`; a convenience wrapper is provided for passing names of text files to be read and then resampled.

`metaSub` collapses a character vector to one line of text. The vector is replicated as many times as there are elements in `names`, with flexible substitution of text fragments. If `out` is supplied, the replicates are saved as text files identified with `names` and a `suffix`.

`metaSub.filename` will process multiple filenames, if `x` is as long as `names`.

**Usage**

```

## S3 method for class 'character'
as.filename(x, ...)
## S3 method for class 'filename'
metaSub(x, names, pattern=NULL, replacement=NULL, ...)
## S3 method for class 'character'
metaSub(
  x,
  names,
  pattern = NULL,
  replacement = NULL,
  out = NULL,
  suffix = '.txt',
  fixed = TRUE,
  ...
)

```

**Arguments**

x	scalar character, or (second form) filename(s). Multi-element character will be collapsed to one element, with newline as the separator.
names	a list of (unique) names for resulting output elements. A vector of names will be coerced to character and to list.
pattern	a character vector of text fragments to replace, optionally encoded as regular expressions ( <code>fixed==FALSE</code> , See <code>?gsub</code> , <code>?regex</code> ). Can also be a list. See details.
replacement	A character vector of substitutions for patterns. The wildcard <code>*</code> is available to represent the corresponding value of names. Can also be a list with as many elements as <code>pattern</code> (see details). Expressions are supported (see details).
out	(path and) directory in which to write the resulting control streams
suffix	file extension for filenames, if out is supplied
fixed	passed to <code>gsub</code> : use <code>FALSE</code> if <code>pattern</code> contains regular expressions. Scalar, or same length as <code>pattern</code> .
...	extra arguments, available to expressions or passed to <code>gsub</code>

**Details**

Typical usages are

```

metaSub(x, names, ...)
metaSub(as.filename(x), names, ...)

```

Replacement is performed by `gsub`, so an element of `pattern` will be replaced everywhere it occurs in a line.

if `pattern` or `replacement` is a list, each element should be of length one, or as long as `names`. In the latter case, substitutions can be specific on a per-name basis. The wild card `*` is still available.

It is necessary that pattern and replacement be of the same length, but it is not necessary that their corresponding elements have equal lengths. Thus, one can specify name-specific replacements for a single pattern, or a single replacement for name-specific patterns.

An expression can be specified for replacement itself, or one of its pattern-wise elements, or one of the name-wise elements of a pattern-wise element. Expressions are evaluated in an environment containing “name” (same meaning as ‘\*’ above) and all other ... arguments. This is useful if extra arguments have a dimension indexed, at least loosely, by names. The evaluated expression is treated as character, and wildcard substitution is attempted. Use \\* for a literal asterisk: in R: \\\*.

NOTE: be very careful not to have trailing commas in your lists! An error results that is very hard to track. e.g. c(a,b,c,).

### Value

an invisible named character vector. If out is supplied, elements are written as files with corresponding names.

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [gsub](#)
- [regex](#)

### Examples

```
data(ctl)
dir()
e <- metaSub(
  ctl,
  names=1:3,
  pattern=c(
    'PROBLEM 8',
    'data8.csv',
    '8.MSF'
  ),
  replacement=c(
    'PROBLEM *',
    '*.csv',
    '*.MSF'
  ),
  out='.',
  suffix='.ctl'
)
t <- metaSub(
```

```

ctl,
names=c('test1','test2'),
pattern=c('PROBLEM 8','data8.csv','METH=0'),
replacement=c('PROBLEM *','*.csv','METH=1'),
)

t <- metaSub(
ctl,
names=c('test1','test2'),
pattern=c(
'PROBLEM 8',
'data8.csv',
'METH=0'
),
replacement=list(
'PROBLEM *',
'*.csv',
c('METH=1','METH=2')
),
out='.',
suffix='.ctl'
)
#just a file copy...
metaSub(as.filename('1.ctl'),names='4',out='.',suffix='.ctl')

#using a (nonsense) replacement expression...
options <- data.frame(var=c(8,9),alt=c(10,11))
a <- metaSub(
ctl,
names=rownames(options),
pattern='9999',
replacement=expression(
options$var[rownames(options)==name]
),
options=options
)
cat(a[[2]])

#replacement expression in a 'mixed' list...
b <- metaSub(
ctl,
names=rownames(options),
pattern=list(
'PRINT=2',
'9999'
),
replacement=list(
'PRINT=3',
expression(options$var[rownames(options)==name])
),
options=options
)
cat(b[[2]])

```

```

#replacement expressions on a per-name basis
d <- metaSub(
  ctl,
  names=rownames(options),
  pattern='9999',
  replacement=list( #so that replacement is as long as pattern
    list( #with different options for each 'name'
      expression(options$var[rownames(options)==name]),
      expression(options$alt[rownames(options)==name])
    )
  ),
  options=options
)
cat(d[[2]])

```

---

miTemporal

*Temporal Classes with Selective Defaults*


---

## Description

miTemporal is an abstract superclass of miTime, miDate, and miDateTime. These latter are convenience classes that store temporal information as seconds since the start of 1970-01-01. They rely on POSIXlt and POSIXct, giving full access to the format constructs of `strftime`. However, the concepts of ‘origin’ and ‘timezone’ are deconstructed (fixed to 1970-01-01 and GMT). Default formats are suitably chosen for inputs (as `.character` methods) and outputs (format methods) and may be overridden. By default, format will append a ‘+’ symbol to temporals with dangling seconds (not multiples of 60): seconds are not displayed by default but still operate (perhaps dangerously) in comparisons.

## Usage

```

as.miTime(x, ...)
## S3 method for class 'character'
as.miTime(x, format = '%H:%M',...)
## S3 method for class 'numeric'
as.miTime(x,...)
## S3 method for class 'miTime'
as.miTime(x, ...)
## S3 method for class 'times'
as.miTime(x, ...)
as.miDate(x, ...)
## S3 method for class 'character'
as.miDate(x, format = '%Y-%m-%d',...)
## S3 method for class 'numeric'
as.miDate(x,...)
## S3 method for class 'Date'
as.miDate(x,...)

```

```

## S3 method for class 'dates'
as.miDate(x,...)
as.miDateTime(x, ...)
## S3 method for class 'character'
as.miDateTime(x, format = '%Y-%m-%d %H:%M',...)
## S3 method for class 'numeric'
as.miDateTime(x,...)
## S3 method for class 'miDate'
as.miDateTime(x, y = 0,...)
## S3 method for class 'miDateTime'
as.miDateTime(x, ...)
## S3 method for class 'POSIXct'
as.miDateTime(x, ...)
## S3 method for class 'POSIXlt'
as.miDateTime(x, ...)
## S3 method for class 'chron'
as.miDateTime(x, ...)
## S3 method for class 'miTime'
format(x, format = '%H:%M', mark=TRUE,...)
## S3 method for class 'miDate'
format(x, format = '%Y-%m-%d', mark=TRUE,...)
## S3 method for class 'miDateTime'
format(x, format = '%Y-%m-%d %H:%M', mark=TRUE,...)
## S3 method for class 'miTemporal'
unique(x, incomparables=FALSE,...)

```

### Arguments

x	character time as per format, numeric seconds since 1970-01-01, or miTemporal subclass
...	other arguments, usually ignored
y	optional time for constructing miDateTime from miDate (defaults to midnight)
format	character, as per <a href="#">strftime</a>
mark	boolean: mark times with dangling seconds using '+'
incomparables	passed to unique

### Details

Creating a temporal object with these methods ultimately calls one of the `.numeric` methods, each of which round their first argument to zero places. This means that all comparisons are based on whole numbers, and therefore not subject to rounding errors.

Seconds that are not multiples of 60 can be stored in `miTime` and `miDateTime` objects, but will not be displayed by default (see above). `miDate` can only store numbers of seconds that correspond to midnight, so adding an `miDate` and `miTime` gives a predictable result.

The `miTemporal` classes are all subclasses of `numeric`, so numeric methods are available (but may not preserve class information; try `max` for example).

miTemporal classes support NA, Inf, -Inf, as.data.frame, seq, subset, element selection, element assignment, and interconversion.

**Value**

format	character
as.miTime	object with class c('miTime', 'miTemporal', 'numeric')
as.miDate	object with class c('miDate', 'miTemporal', 'numeric')
as.miDateTime	object with class c('miDateTime', 'miTemporal', 'numeric')

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [c.miTemporal](#)

**Examples**

```
#numeric to temporal
as.miTime(0)
# 00:00
as.miTime(1)
# 00:00+
as.miTime(-1)
# 23:59+
as.miTime(60)
# 00:01
as.miTime(-60)
# 23:59
as.miTime(86400)
# 00:00
as.miTime(86460)
# 00:01
as.miDate(0)
# 1970-01-01
as.miDate(1)
# 1970-01-01
as.miDate(-1)
# 1969-12-31
as.miDate(-86400)
# 1969-12-31
as.miDate(-86401)
# 1969-12-30
as.miDateTime(0)
# 1970-01-01 00:00
```

```
as.miDateTime(60)
# 1970-01-01 00:01
as.miDateTime(61)
# 1970-01-01 00:01+
as.miDateTime(-1)
# 1969-12-31 23:59+

#character to temporal
as.miTime('00:00')
# 00:00
as.miTime('23:59')
# 23:59
as.miTime('23:59:00')
# 23:59
as.miTime('23:59:01')
# 23:59
as.miTime('23:59:01',format='%H:%M:%S')
# 23:59+
as.miTime('24:00')
# NA
as.miDate('1970-01-02')
# 1970-01-02
as.miDate('01/02/1970',format='%m/%d/%Y')
# 1970-01-02
as.miDateTime('01/02/1970 12:30',format='%m/%d/%Y %H:%M')
# 1970-01-02 12:30
as.miDateTime('01/02/1970 12:30:15',format='%m/%d/%Y %H:%M:%S')
# 1970-01-02 12:30+

#temporal to numeric
as.numeric(as.miTime(0))
# 0
as.numeric(as.miTime(1))
# 1
as.numeric(as.miTime(-1))
# 86399
as.numeric(as.miTime(60))
# 60
as.numeric(as.miTime(-60))
# 86340
as.numeric(as.miTime(86400))
# 0
as.numeric(as.miTime(86460))
# 60
as.numeric(as.miDate(0))
# 0
as.numeric(as.miDate(1))
# 0
as.numeric(as.miDate(-1))
# -86400
as.numeric(as.miDate(-86400))
# -86400
as.numeric(as.miDate(-86401))
```

```

# -172800
as.numeric(as.miDateTime(0))
# 0
as.numeric(as.miDateTime(60))
# 60
as.numeric(as.miDateTime(61))
# 61
as.numeric(as.miDateTime(-1))
# -1
as.numeric(as.miTime('00:00'))
# 0
as.numeric(as.miTime('23:59'))
# 86340
as.numeric(as.miTime('23:59:00'))
# 86340
as.numeric(as.miTime('23:59:01'))
# 86340
as.numeric(as.miTime('23:59:01',format='%H:%M:%S'))
# 86341
as.numeric(as.miTime('24:00'))
# NA
as.numeric(as.miDate('1970-01-02'))
# 86400
as.numeric(as.miDate('01/02/1970',format='%m/%d/%Y'))
# 86400
as.numeric(as.miDateTime('01/02/1970 12:30',format='%m/%d/%Y %H:%M'))
# 131400
as.numeric(as.miDateTime('01/02/1970 12:30:15',format='%m/%d/%Y %H:%M:%S'))
# 131415

#temporal to character
as.character(as.miTime(0))
# '00:00'
as.character(as.miDate(0))
# '1970-01-01'
as.character(as.miDateTime(0))
# '1970-01-01 00:00'

#non-default printout
format(as.miTime(30000),format='%H')
# '08'
format(as.miDate('1970-01-01'),format='%d%b%y')
# '01Jan70'
format(as.miDateTime('1970-01-02 23:30'),format='[%d/%m/%y %H:%M:%S]')
# '[02/01/70 23:30:00]'
format(as.miTime(1))
# '00:00+'
format(as.miTime(1),mark=FALSE)
# '00:00'

#sequence
seq(from=as.miTime('8:00'),to=as.miTime('12:00'))
# 08:00 09:00 10:00 11:00 12:00

```

```

seq(from=as.miDate('2008-01-01'),to=as.miDate('2008-01-04'))
# 2008-01-01 2008-01-02 2008-01-03 2008-01-04
seq(from=as.miDateTime('2008-01-01 12:00'),to=as.miDateTime('2008-01-04 12:30'))
# 2008-01-01 12:00 2008-01-02 12:00 2008-01-03 12:00 2008-01-04 12:00

#interconversion
as.miTime(as.miDate('2008-10-14'))
# 00:00
as.miTime(as.miDateTime('2008-10-14 08:00'))
# 08:00
as.miDate(as.miTime('23:59'))
# 1970-01-01
as.miDate(as.miDateTime('2008-10-14 08:00'))
# 2008-10-14
as.miDateTime(as.miTime(6000000))
# 1970-01-01 10:40
as.miDateTime(as.miDate('2008-10-14'))
# 2008-10-14 00:00

#interconversion from other systems
as.miDate(as.Date('1970-01-01'))
# 1970-01-01
as.miDateTime(as.POSIXct('1970-01-01',tz='GMT'))
# 1970-01-01 00:00
as.miDateTime(as.POSIXlt('1970-01-01',tz='GMT'))
# 1970-01-01 00:00
library(chron)
as.miTime(times('12:30:00'))
# 12:30
as.miDate(dates('01/01/1970'))
# 1970-01-01
as.miDateTime(chron(dates='01/01/1970',times='12:30:00'))
# 1970-01-01 12:30

#NA
as.miTime(as.numeric(NA))
# <NA>

#infinity
as.miTime(Inf)
# Inf
as.miDate(Inf)
# Inf
as.miDateTime(Inf)
# Inf
as.miTime(-Inf)
# -Inf
as.miDateTime(Inf) + as.miTime(Inf)
# Inf
as.miDateTime(Inf) + as.miTime(-Inf)
# <NA>

#comparison

```

```

as.miTime('08:00') < as.miTime('09:00')
# TRUE
as.miDate('1970-01-01') > as.miDate('2008-01-01')
# FALSE
as.miDateTime('1970-01-01 08:00') > as.miDate('1970-01-01')
# TRUE

#statistics
max(as.miDate(c('1970-01-01', '1980-01-01', '1975-01-01')))
# 315532800

#operations
as.miDateTime(0) + as.miTime(60)
# 1970-01-01 00:01
as.miTime(60) + as.miDateTime(0)
# 00:01

#unary operations
-as.miTime(1)
# 23:59+

#sorting
sort(unique(as.miTime(c(180,120,60))))
# 00:01 00:02 00:03

```

---

naInContext

*Display Missing Values in Context*


---

## Description

Display rows of data with missing values, as well as other rows with the same key.

## Usage

```
naInContext(x, context, search)
```

## Arguments

x	data.frame.
context	character vector of column names representing a key
search	character vector of column names representing search fields.

## Details

Often one wants to view not just records with missing values, but other related records as well, e.g. “view all the records for any subject with missing doses”. `naInContext` searches for NAs in columns of `x` named by `search`. It then determines keys for these rows, using columns in `x` named by `context`. Finally, it displays all rows with matching keys.

**Value**

data.frame

**Author(s)**

Tim Bergsma

**Examples**

```
test <- Theoph
test$conc[[5]] <- NA
naInContext(test, context='Subject', search='conc')
```

---

nest

*Nest an XML Fragment in a Parent Element*

---

**Description**

Create matching open and close tags and embed an XML fragment between them.

**Usage**

```
nest(x, tag, ...)
```

**Arguments**

x	an XML fragment
tag	an element name
...	ignored

**Details**

Attributes are not supported.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [bracket](#)
- [as.xml.data.frame](#)
- [as.pxml.ext](#)

**Examples**

```
nest('some xml', tag='content')
```

---

nix

*Identify Unix-like Platforms*

---

**Description**

Tells whether the platform OS is of type 'unix'.

**Usage**

```
nix()
```

**Value**

logical

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [NONR](#)

**Examples**

```
nix()
```

nm.pl

*Build Commands to Invoke NONMEM***Description**

Format a set of commandlines, suitable for submission to a command shell, to operate the executable for an NMQual-mediated NONMEM installation. nm.pl is the default for NONMEM 7.1.2 and earlier. autolog.pl is the default for NONMEM 7.2.0 and later.

**Usage**

```
nm.pl(
  command,
  ctlfile,
  outfile=NULL,
  perl='perl',
  checksum=TRUE,
  compile=TRUE,
  execute=TRUE,
  split=FALSE,
  ...
)
autolog.pl(
  command,
  rdir,
  run,
  perl = "perl",
  compile = TRUE,
  execute = TRUE,
  split = FALSE,
  config = file.path(dirname(command), "log.xml"),
  mode = "run",
  ...
)
```

**Arguments**

command	path for the installation-specific variant of 'nm.pl'
ctlfile	path for a control stream
outfile	path for output
perl	how to invoke Perl on this system
checksum	whether to conduct checksums
compile	whether to compile NONMEM
execute	whether to execute the compiled NONMEM
split	whether to compile and execute as separate steps

rdir	run directory
run	run name
config	path to a configuration file (sensu NMQual 8)
mode	process mode
...	ignored

### Details

If `split` is `TRUE`, two commands are returned, having the mode flags 'c' and 'e', respectively. `nm.pl` drops the 'c' or 'e' argument if `split` is `FALSE` and both `compile` and `execute` are `TRUE`. In the identical case, `autolog.pl` passes 'ce'.

`autolog.pl` returns a run directory and run name, rather than input and output file paths as in `nm.pl`. `autolog.pl` does not support `checksum`. By default, `autolog.pl` expects a config file called 'log.xml' in the same directory as `command`; it also assumes the mode is 'run'. See also <http://nmqual.googlecode.com>.

### Value

character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [runCommand](#)

### Examples

```
nm.pl('/nm6/test/nm6.pl', '1.ct1')
nm.pl('/nm6/test/nm6.pl', '1.ct1', '1.out', checksum=FALSE)
nm.pl('/nm6/test/nm6.pl', '1.ct1', execute=FALSE)
nm.pl('/nm6/test/nm6.pl', '1.ct1', compile=FALSE)
nm.pl('/nm6/test/nm6.pl', '1.ct1', split=TRUE)
autolog.pl('/nm72/nmqual/autolog.pl', '/home/ubuntu/test/3', 3)
```

---

nmPlots

*An Extensible List of Diagnostic Plots for nm Objects*


---

### Description

Default diagnostics for plotting an nm object.

**kinetics** dose and concentration data

**constantContinuous** continuous variables that are constant within SUBJ

**varyingContinuous** continuous variables that vary within SUBJ

**constantCategorical** categorical variables that are constant within SUBJ

**varyingCategorical** categorical variables that vary within SUBJ

### Details

plot.nm preprocesses an nm dataset and passes it to a subset of nmPlots. Extra arguments can be passed to plot.nm to exert fine control on the end results.

### See Also

- [plot.nm](#)

---

nmVersion

*Extract the NONMEM Version from an NMQual Configuration File*


---

### Description

This function reads the configuration file and extracts the version number from the 'nonmem' element.

### Usage

```
nmVersion(config, ...)
```

### Arguments

config	path for a configuration file
...	ignored

### Value

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [runNonmem](#)

**Examples**

```
## Not run: nmVersion('../NONMEM/nm6osx1/test/config.xml')
```

---

NONR

*Run NONMEM*

---

**Description**

NONR executes, from R, a NONMEM run for which there exists a compatible (see note) control stream. Supported platforms include MS Windows and Unix-like operating systems (Nix). Special support is included for Sun Grid Engine with Nix. Generally, NONMEM needs to be installed using NMQual (<http://nmqual.googlecode.com>). Execution on the Nix grid requires the installation of Sun Grid Engine v6. Following a successful NONMEM run NONR can call PLOTR to create diagnostic plots (pdf format) and/or run a user-written script. NONR72 is a wrapper designed to invoke an NMQual-mediated installation of NONMEM 7.2.0 or above. See [autolog.pl](#).

**Usage**

```
NONR(  
  run,  
  command,  
  project = getwd(),  
  boot = FALSE,  
  grid = boot,  
  concurrent = grid,  
  urgent = !boot,  
  udef= FALSE,  
  invisible=udef,  
  compile = TRUE,  
  execute = TRUE,  
  split = grid & compile & execute,  
  checkrunno = TRUE,  
  checksum = TRUE,  
  diag = !boot,  
  fdata = TRUE,
```

```
logtrans = FALSE,
nice= TRUE,
epilog = NULL,
dvname = NULL,
grp = NULL,
grpnames = NULL,
cont.cov = NULL,
cat.cov = NULL,
par.list = NULL,
eta.list = NULL,
missing = -99,
...
)
NONR72(
run,
command,
project = getwd(),
boot = FALSE,
grid = boot,
concurrent = grid,
urgent = !boot,
undef= FALSE,
invisible=undef,
compile = TRUE,
execute = TRUE,
split = FALSE,
checkrunno = TRUE,
checksum = TRUE,
diag = !boot,
fdata = TRUE,
logtrans = FALSE,
nice= TRUE,
epilog = NULL,
dvname = NULL,
grp = NULL,
grpnames = NULL,
cont.cov = NULL,
cat.cov = NULL,
par.list = NULL,
eta.list = NULL,
missing = -99,
...,
interface='autolog.pl',
q='all.q'
)
```

**Arguments**

run	vector of control stream names. Typically integer. e.g. 1:10 corresponds to 1.ctl, 2.ctl, ... 10.ctl.
command	path for a script to run NONMEM, e.g., <code>command='C:/nm6/test/nm6.pl'</code> . The script is assumed to be an NMQual-generated wrapper for NONMEM execution, but may also be arbitrarily defined by the user ( <code>undef=TRUE</code> ). In the former case, appropriate arguments and modifiers are added to build a typical NMQual commandline ( <code>nm.pl</code> ), conditionally wrapped in the arguments for a typical grid commandline ( <code>qsub</code> ). The path may be given relative to the current directory. <code>command</code> may be a character vector of length greater than one: each element will be run in succession.
project	directory in which run-specific subdirectories will be created. Typically also contains original control streams. Use an absolute path (safer) or specify relative to the current working directory.
boot	If TRUE, autogenerated run directories will end in <code>boot</code> . If <code>grid</code> is also TRUE, The job will run asynchronously and no cleanup or plotting will be attempted. Default: FALSE.
grid	whether NONMEM execution should be wrapped in a call to Sun Grid Engine's <code>qsub</code> utility. TRUE if <code>boot</code> is TRUE, by default. Set to TRUE explicitly for regular runs using a grid. Coerced to FALSE for Windows. Autogenerated run directories will end in <code>lock</code> until run is complete.
concurrent	Should the runs represented by elements of <code>run</code> be run concurrently as separate processes? TRUE if <code>grid</code> is TRUE, by default, since this is the primary advantage of using a grid. Coerced to FALSE on Windows. Concurrency is implemented using the package <b>fork</b> .
urgent	Should jobs use as many resources as possible? Only meaningful in the grid environment, where all available processors will be used if <code>urgent</code> , and a limited number of processors if not. In practice, this dissolves to a choice between <code>q=all.q</code> and <code>q=bootstrap.q</code> . Passing <code>q</code> explicitly will override this choice.
undef	a logical indicating whether <code>command</code> is user defined. If so, the command is simply wrapped in a system call rather than pre-converted to a (possibly grid-ready) NMQual commandline. Default: FALSE
invisible	<code>intern</code> , <code>minimized</code> , and <code>invisible</code> can be specified for the system call that runs NONMEM. Mifuns will largely ignore these on a unix-like platform. For Windows, <code>invisible</code> is TRUE by default when <code>undef</code> is TRUE, and FALSE otherwise. If FALSE, NONMEM output will be displayed in a command window during the run. Set <code>invisible=TRUE</code> to suppress this display. See the Windows help for <code>system</code> . To explore interactively, use <code>debug(runCommand)</code> .
compile	whether to compile NONMEM (i.e., make <code>nonmem.exe</code> ). Set to FALSE if you already did this in a previous step and merely want to execute.
execute	whether to execute the compiled NONMEM (i.e., invoke <code>nonmem.exe</code> ). Set to FALSE if you merely want to compile, e.g. to verify compile success without committing to a run. You can run the compiled object later with <code>compile=FALSE</code> .

<code>split</code>	whether to compile and execute as separate steps. If both compile and execute are TRUE, the run can be split into two separate steps (this is the default if grid is TRUE. Though split, compile, and execute can be manipulated independently, it makes no sense to split a run unless compile and execute are both TRUE. The greatest advantage to splitting is realized on a grid, with grid=TRUE: runs will be split and compile steps will by default be run in their own queue, for efficiency.
<code>checkrunno</code>	whether to check/correct the control stream to ensure that standard uses of the NONMEM control stream name match the name listed in run vector. Default: TRUE. For files, e.g. <code>../3.TAB</code> , path separator must be that recognized by <code>dirname</code> and <code>basename</code> . Only one usage per line of text will be fixed. 'Standard' uses currently include: the first word after <code>\\$PROB(LEM)</code> (not counting <code>RUN</code> or <code>RUN\#</code> ); <code>*.tab</code> (case insensitive); <code>*par.tab</code> (case insensitive); <code>*.msf</code> (case insensitive) except on a line that starts with <code>\\$MSFI</code> .
<code>checksum</code>	If TRUE (default), checksums are requested from <code>nm.pl</code> when wrapping NONMEM execution. FALSE is useful for development.
<code>diag</code>	whether to plot standard diagnostic plots. Default: TRUE for non-bootstrap runs. You can override the default file name by passing <code>plotfile</code> . See <code>PLOTR</code> for details.
<code>fdata</code>	whether to attempt deletion of NONMEM files <code>FDATA</code> and <code>PRDERR</code> . Default: TRUE.
<code>logtrans</code>	passed to <code>PLOTR</code>
<code>nice</code>	whether the NONMEM run directory, based on <code>run</code> , is deleted or simply emptied prior to the execution of NONMEM. If <code>nice=TRUE</code> (default), hidden files (as defined for <code>all.files</code> in the function <code>dir</code> ) are not deleted, e.g., any <code>.svn</code> directory.
<code>epilog</code>	user-defined function or script to call at end of NONR or <code>PLOTR</code> . A non-null argument that cannot be coerced by <code>match.fun</code> to a function will be treated as a filename to be sourced. All the arguments normally available to <code>PLOTR</code> ( <code>run</code> , <code>project</code> , <code>dvname</code> , <code>logtrans</code> , <code>grp</code> , <code>grpnames</code> , <code>cont.cov</code> , <code>at.cov</code> , <code>par.list</code> , <code>eta.list</code> , <code>missing</code> , etc) will be available to <code>epilog</code> , as well as any extra arguments you pass to NONR or <code>PLOTR</code> . A function can declare any of these, but should at minimum contain the <code>...</code> argument. A script can expect them to be present. See also <code>'inst/example/epilog.R'</code> in this package. To see exactly what is available, try <code>epilog=function(...){cat(names(list(...)))}</code> .
<code>dvname</code>	name of the dependent variable to use as a label for the diagnostic plots. Default: <code>DV</code> .
<code>grp</code>	passed to <code>PLOTR</code>
<code>grpnames</code>	passed to <code>PLOTR</code>
<code>cont.cov</code>	passed to <code>PLOTR</code>
<code>at.cov</code>	passed to <code>PLOTR</code>
<code>par.list</code>	passed to <code>PLOTR</code>
<code>eta.list</code>	passed to <code>PLOTR</code>
<code>missing</code>	passed to <code>PLOTR</code>
<code>interface</code>	the name of a function to prepare NONMEM command lines

q	choice of queue, passed to qsub
...	additional arguments passed to functions that accept them. For example, you can pass <code>outfile=FALSE</code> to <code>pdf</code> , called by <code>PLOTR</code> . You can also pass additional lattice arguments to modify the diagnostic plots. Flags recognized by Sun Grid Engine's <code>qsub</code> are accepted by a like-named wrapper (this effect replaces the older <code>SGEflgs</code> argument). Example: to set a maximum run length of one hour and get an email if the run goes over one hour and is killed, include <code>l='h_rt=1:0:0'</code> , <code>m='e'</code> , <code>M='name@email.address'</code> . Passing <code>N</code> , <code>o</code> , <code>e</code> , <code>V</code> , <code>j</code> , <code>q</code> , <code>sync</code> , <code>shell</code> , <code>b</code> , or <code>cwd</code> will override defaults. You can pass multiple instances of an argument if <code>qsub</code> allows it. passing <code>L</code> will override a constitutive instance of <code>l</code> in <code>runCommand</code> . Values are character vectors. Lengths greater than one result in element-wise paste operations: a vector of corresponding length will be returned.

### Details

NONR is currently implemented as a wrapper that calls `runNonmem` for each element in `run`. By default, it creates a like-named subdirectory for each run in `project`, attaching an appropriate extension if necessary (`.boot` for bootstrap runs, `.lock` for grid runs). It looks for `<run>.ctl` in `project`, cleans it up as necessary, and writes it to the subdirectory, in which context `NONMEM` is invoked (using `runCommand`). The output of `NONMEM` will be `<run>.lst` in the subdirectory. Files matching patterns in `runNonmem:remove` will be removed. A `*.lock` directory, if any, will be converted to `*`. Attempts will be made to run `PLOTR` and `epilog`. Then a 'complete' message is printed.

File names and locations can be manipulated somewhat, i.e., by over-riding `runNonmem`'s defaults for `runext`, `rundir`, `outfile`, `streams`, `ctlfile`, and `remove`.

### Value

used for side effects

### Note

Specific control stream syntax is expected when running `NONR` and the other functions present in the `MIFuns` package. The list of syntax requirements are as follows.

- `\$PROB` should be followed by `'RUN#'` then a number representing the control stream number. No commas should be used in the `\$PROB` statement. `'PROBLEM'` is an alternative for `'PROB'`. The run number is the first space-delimited alphanumeric sequence after `PROB(LEM)`, ignoring `RUN(\#)`.
- The datafile name and relative path needs to be on the first line of the `\$DATA` record immediately following `'\$DATA'`.
- A relative datafile path must be specified relative to `runNonmem:rundir`, typically a subdirectory of `project`. For example, `'3.ctl'` will actually run as `'path/project/3/3.ctl'`; if its datafile is `'path/data/DATA3'`, then `3.ctl` should specify `"\$DATA ../data/DATA3"`.
- The `NONMEM` datafile must contain a `'C'` column containing only `C`'s or periods(`'.'`). Typically this is the first column.

- The ‘\*.TAB’ file in the “\TABLE” step must contain an EVID column for plotting purposes. In the case of “\SPRED” models, this can be a dummy column in the data file.
- The NONMEM data file must contain a column with a header of ‘ID’ for plotting purposes.
- File paths for ‘MSF’ files and ‘TAB’ files, etc., are generally expected to be relative paths, using ‘/’ as the directory separator. File paths with no separator are assumed to be relative to the directory in which the control stream is run: ‘./’ may be prepended. E.g. “TABLE EVID FILE=100.tab” is taken as “TABLE EVID FILE=./100.tab”.

PLOTR will automatically generate CWRES plots if required files are present in NONMEM run directory. See help for compute.cwres for instructions on generating the files required for CWRES plots. PLOTR expects etas and model parameters to be output in the ‘\*par.TAB’ file and variables for diagnostic plots to be output in the ‘\*.TAB’ file, where ‘\*’ represents the control stream number. Additional \TABLE records can be present in the control stream but these are not used/needed by PLOTR. Mechanisms and expectations are somewhat different for NONMEM7.

### Author(s)

written by Bill Knebel; modified by Tim Bergsma.

### References

<http://mifuns.googlecode.com>

### See Also

- [runNonmem](#)
- [PLOTR](#)

---

omegacor

*Convert NONMEM Omega Covariance Matrix to Correlation Matrix*

---

### Description

These functions acquire omega or sigma covariance matrix elements for a NONMEM 7 run and calculate the correlation matrix. Run and project are used to calculate the .ext file name, which is read into memory as a pxml object, and then converted to unilog format. Directly supplying extfile, pxml, or unilog renders all previous arguments moot. Not vectorized: run should be scalar.

### Usage

```
unilogcor(
  pattern,
  run=0,
  project=getwd(),
  tool='nm7',
  extfile=file.path(project,run,paste(run,'ext',sep='.')),
  pxml=as.pxml.ext(extfile),
```

```

unilog=as.unilog.pxml(x=pxml,run=run,tool=tool,...),
...
)
omegacor(
run,
project=getwd(),
...
)
sigmacor(
run,
project=getwd(),
...
)

```

### Arguments

pattern	a pattern to seek in the parameter column of a unilog (anchored at the start of each string)
run	name (number) of a run
project	project directory (parent of run directory)
tool	largely irrelevant; only 'nm7' is supported
extfile	name of .ext file, e.g. run.ext
pxml	internal xml format for parameter data
unilog	unilog format; as returned invisibly by rlog
...	passed to unilogcor, else unused

### Value

correlation matrix

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [cov2cor](#)
- [rlog](#)

**Description**

These functions implement a concise syntax for joining objects of class `c('keyed', 'data frame')`. `+` produces an outer join, `&` produces an inner join, and `|` gives a left join. By default, `-` drops rows in `x` that have matching rows in `y`. The syntax can be extended to subclasses of `keyed` by writing additional proxy methods.

**Usage**

```
## S3 method for class 'keyed'
Ops(e1, e2)
## S3 method for class 'keyed'
plus(x, y)
## S3 method for class 'keyed'
and(x, y)
## S3 method for class 'keyed'
minus(x, y)
## S3 method for class 'keyed'
left(x, y)
```

**Arguments**

<code>e1</code>	left argument to <code>Ops</code>
<code>e2</code>	right argument to <code>Ops</code>
<code>x</code>	left argument to proxies
<code>y</code>	right argument to proxies

**Details**

A concise syntax for joining of `data.frames` facilitates dynamic assembly of data. This system leverages existing operators and dispatch mechanisms. Under `Ops` dispatch rules, if both left and right operands resolve to the same the method, that method is used. Operator methods are already defined for `data.frame`, but the existence of class `'keyed'` creates an opportunity for syntax specification.

`Ops.keyed` selects a *proxy* method, and dispatches on the right hand operand. Proxy methods are text equivalents of simple operators, e.g. the proxy `plus` corresponds to `+`. They are necessary because direct specification of, say, `+.keyed` could not be overridden for subclasses of `keyed` (`Ops` mechanism would detect conflicting methods for the two operands, and would default to primitives.)

Dispatch on the right hand operand is consistent with a general syntax of data assembly. For `a + b + c . . .`, results accumulate on the left under a somewhat fixed typology (class), and the next operand on the right controls the next step of assembly, perhaps invoking a specialized method by means of its class (S3 dispatch mechanism).

Operators have been chosen to coordinate intuition about their effects with existing operator precedence and a general data assembly pattern. `+` gives an outer join (`merge`, with `all=TRUE`). `&` gives

an inner join (`all=FALSE`): mnemonically, rows must match in the left AND right operands to contribute to the result. `-` suggests removal: it is used for methods that drop rows. `|` gives a left join (`merge, all.x=TRUE`). Mnemonically, it suggests conditioning (as with formulas): use of rows on the right is conditional on existence of matches on the left. Right joins are currently not implemented, but most can be expressed as left joins by rearrangement.

Operator pair `+` and `-` has higher precedence than the pair `&` and `|`. Within pairs, operators have equal precedence and resolve left to right (see `?S3groupGeneric`). A common assembly sequence is one or more outer joins followed by one or more left joins. Correspondence to the existing order of operations minimizes the need for parenthetical grouping of terms (which is available nonetheless).

Proxy methods have been defined on these operators for class `'keyed'`. Users can implement differing assembly operations by creating additional classes and defining proxy methods for them. `plus.rigged` and `minus.moot` are examples. See examples for `as.nm`.

### Value

keyed data.frame

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [as.nm](#)

---

ord.matrix

*Give the Order of a Symmetric Matrix or Half Matrix*

---

### Description

The dimensions of a symmetric matrix are equal; this value is, by definition, the order of the matrix. For class `halfmatrix`, the order is found using a quadratic expression with `length` as an argument.

### Usage

```
## S3 method for class 'matrix'
ord(x, ...)
## S3 method for class 'halfmatrix'
ord(x, ...)
```

### Arguments

<code>x</code>	symmetric matrix or halfmatrix
<code>...</code>	extra arguments, ignored

**Details**

It is an error if `x` is a matrix but not symmetric. If `x` is a vector of appropriate length its order can be found by specifying the method explicitly, even if `x` is not classed as `halfmatrix`: `ord.halfmatrix(x)`.

**Value**

scalar numeric

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

[is.square.matrix](#), [isSymmetric.matrix](#) [half.matrix](#)

**Examples**

```
ord(diag(4))
ord(half(diag(4)))
ord.halfmatrix(1:10)
```

---

packageCheck

*Load a Package and Run Package-level Examples for Testing Purposes*

---

**Description**

`packageCheck` attempts to load the specified package and to run `example` with the package name as an argument.

**Usage**

```
packageCheck(x, lib.loc=NULL)
```

**Arguments**

`x` scalar character, the name of a single package  
`lib.loc` the library to check, passed to `library`

**Details**

Many packages do not have package-level examples; the call to `example` in such cases does nothing (warnings are suppressed).

**Value**

an scalar character string: zero if the package does not load or if example generates an error; otherwise, the package version.

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [accept](#)
- [acceptance](#)
- [library](#)

---

panel.densitystrip      *Display Distributions with Respect to Reference Values*

---

**Description**

These are panel functions that compare distributions with respect to reference values. `panel.densitystrip` plots a filled polygon for each unique value of `y`. `panel.cuts` calculates the portion of each distribution (unique `y`) falling within specified limits. `panel.ref` shades a swath of the panel between specified limits of the primary axis. `covplot` uses the other panel functions to assemble a sample display of covariate data.

**Usage**

```
panel.densitystrip(  
  x,  
  y,  
  horizontal,  
  col.line,  
  fill,  
  factor,  
  border=col.line,  
  col=fill,  
  ...  
)  
panel.ref(  
  x,  
  y,  
  col = 'grey90',
```

```

    horizontal,
    rlim,
    ...
)
panel.cuts(
  x,
  y,
  cuts,
  col,
  col.line,
  text=col.line,
  horizontal = TRUE,
  offset = -0.2,
  increment = 0,
  format = function(x,...) as.numeric(round(x/sum(x) * 100)),
  include.range = TRUE,
  zero.rm = TRUE,
  cex=0.7,
  ...
)
panel.covplot(
  x,
  y,
  ref=1,
  rlim=ref * c(0.75,1.25),
  cuts=ref * c(0.75,1,1.25),
  horizontal=TRUE,
  border='black',
  fill='grey',
  text='black',
  shade='grey90',
  col='white',
  ...
)
unitDensity(x,...)

```

### Arguments

<code>x</code>	numeric
<code>y</code>	numeric
<code>horizontal</code>	if TRUE, strips run horizontally ( <code>x</code> is primary axis)
<code>col.line</code>	<code>panel.density</code> : polygon border color if border not specified; <code>panel.cuts</code> : text color if text not specified
<code>fill</code>	polygon fill color if <code>col</code> not specified
<code>factor</code>	relative height of the density polygon
<code>border</code>	polygon border color

col	panel.density: polygon fill color; panel.ref: reference area fill color; panel.covplot: color of cut lines panel.cuts: ignored
...	extra arguments passed to other functions
rlim	length 2 vector: the limits of the shaded region
cuts	the values at which to divide the primary axis
text	text color for cut statistics
offset	distance from nominal value on secondary axis, at which to plot cut statistics
increment	distance from nominal value on primary axis, at which to plot cut statistics
format	a function to post-process counts of elements between cuts (should accept ...)
include.range	if TRUE, cuts is supplemented with the range of the data
zero.rm	If TRUE, zeros are not printed
cex	scale factor for text
ref	position of a black reference line; may be NULL
shade	reference fill color

## Details

Unlike `panel.densityplot`, `panel.densitystrip` has both `x` and `y` arguments. In `panel.stripplot` and `panel.bwplot`, panel data is implicitly subset by the discrete values on the secondary axis. I.e., visual elements are panel subsets. Here, panel data is *explicitly* subset using `panel.stratify`. (The same pertains to `panel.cuts`). Densities are calculated using the default arguments of `density` (alternatives will be passed to `density` if supplied). `unitDensity` rescales densities so that the maximum value is one. `panel.densitystrip` uses `factor` (traditionally used to control jitter) to rescale densities again.

A grouping variable can be used at the ‘panel subset’ level to give the same graphical parameters to several subsets, e.g., several polygons can share a color.

For `panel.cuts`, calculated values are by default converted to character and printed below (`horizontal==TRUE`) the density strips. Zeros are not printed by default, for less visual clutter; zeros are stripped before the conversion to character. The actual values calculated are ‘bin counts’, i.e., the number of elements in each vector that fall between adjacent cut points. Only inner cuts need be specified, as the limits of the data are included by default as the outer limits. Cuts are converted to percent by default; use `format=function(x)x` to get actual counts. `cex` will control the size of the text.

`panel.covplot` is optimized for a sample presentation of covariate effects. Values are assumed to be relative, so the center cut is 1 with +/- .25 region. Five color features are specifiable. `ref`, `rlim`, and `cuts` can be specified independently.

Formally, these panel functions are alternatives to `panel.stripplot`, and thus can be passed to `stripplot`. `xplot` gives similar results, and `bwplot` seems to give identical results.

## Value

used for side-effects

## Author(s)

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [densityplot](#)
- [bin](#)
- [panel.stratify](#)

## Examples

```
#a bootstrap of a pharmacokinetic model
set.seed(0)
boot <- data.frame(
  CL =      exp(rnorm(100,mean=3,   sd=0.25)),
  WT =      exp(rnorm(100,mean=-0.25,sd=0.25)),
  MALE =    exp(rnorm(100,mean=0.3, sd=0.25)),
  ASIAN =   exp(rnorm(100,mean=-.1, sd=0.05))
)

#Model: CL = theta1 * (WT/70)**theta2 * theta3**MALE * theta4**ASIAN

#Normalize the structural parameter.
boot <- within(boot, CL <- CL/median(CL))

#Realize the submodel for non-normal instances of the continuous covariate.
boot <- within(boot, WT_35 <- (35/70) ** WT)
boot <- within(boot, WT_140 <- (140/70) ** WT)
boot$WT <- NULL

#(Categorical covariates are already expressed proportionally.)

#Reorganize the table. rev() anticipates bottom-up plotting.
boot <- melt(rev(boot))

#Limit to 90% of the data, for independence from number of bootstraps.
boot <- boot[
  with(
    boot,
    value >= reapply(value,variable,quantile,0.05) &
    value <= reapply(value,variable,quantile,0.95)
  ),
]

#plot using panel.covplot().
stripplot(
  variable~value,
  boot,
  panel=panel.covplot,
  xlab='relative clearance'
)
```

```

#with groups
stripplot(
  variable~value,
  boot,
  groups=contains('WT',variable),
  panel=panel.covplot,
  xlab='relative clearance'
)

#variations
data(crabs)
soup <- melt(crabs,id.var=c('sp','sex','index'))
soup$relative <- with(soup,value/mean(value[variable=='CL']))
soup$grp <- 'depth'
soup$grp[contains('W',soup$variable)] <- 'width'
soup$grp[contains('L',soup$variable)] <- 'length'
stripplot(variable~value,soup,panel=panel.stratify,panel.levels=panel.densitystrip)
stripplot(
  variable~relative|sp+sex,
  soup,
  panel=panel.stratify,
  panel.levels=panel.densitystrip
)
stripplot(
  variable~relative|sp+sex,
  soup,
  panel=panel.covplot
)
stripplot(
  variable~relative|sp+sex,
  soup,
  panel=panel.covplot,
  groups=grp,
  auto.key=TRUE,
  cex=0.5,
  offset=0.2
)
stripplot(
  value~variable|sp+sex,
  soup,
  groups=grp,
  panel=panel.covplot,
  auto.key=TRUE,
  horizontal=FALSE,
  ref=NULL,
  rlim=c(10,50),
  lty=3,
  lwd=2,
  border='transparent',
  text='blue',
  shade='turquoise',
  col='magenta',
  cuts=c(10,20,30,40,50)
)

```

)

panel.hist

*Plot Histograms Flexibly***Description**

histogram and panel.histogram in package **lattice** (expected) are univariate. Here, we present a bivariate histogram panel function for use, e.g., with stripplot. panel.bar adds line segments, appropriate for reference lines in a histogram. unitHist calls hist but returns histogram specifications that have been rescaled so that maximum bar height is 1.

**Usage**

```
panel.hist(x,y,level,horizontal,col,line,fill,factor,border=col,line,col=fill,offset=-0.5,font,fontface,...)
panel.bar(x,y,level,horizontal,col,col,line,fill,factor,font,fontface,...)
unitHist(x, plot = FALSE, ...)
```

**Arguments**

x	vector
y	vector
level	the level at which to draw the histogram, typically supplied by panel.stratify
horizontal	typically TRUE
col	fill color (ignored for panel.bar)
col.line	border color if border not supplied
fill	fill color if col not supplied
factor	factor by which to expand or attenuate bar heights relative to unity
border	border color
offset	amount to raise the base of each bar relative to the level
...	passed to other functions
font	ignored
fontface	ignored
plot	passed to hist

**Value**

unitHist returns class histogram with an extra element named 'heights'. 'heights' is proportional to density, but normalized relative to maximum density. Panel functions are used for side effects.

**Author(s)**

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [panel.stratify](#)

---

panel.stratify	<i>Handle Each Level of a Stripplot Separately</i>
----------------	----------------------------------------------------

---

## Description

Just as `panel.superpose` handles each group of data separately, `panel.stratify` handles each 'level' of data separately. Typically, levels are the unique values of `y` (`horizontal==TRUE`) that result from a call to `stripplot` or `bwplot`. The default panel functions treat all levels simultaneously. Plotting some transformation of the data (e.g. density polygons for each level) is much easier if the levels are presented individually.

## Usage

```
panel.stratify(  
  x,  
  y,  
  type = 'p',  
  groups = NULL,  
  pch = if (is.null(groups))  
  plot.symbol$pch else superpose.symbol$pch,  
  col,  
  col.line = if (is.null(groups))  
  plot.line$col else superpose.line$col,  
  col.symbol = if (is.null(groups)) plot.symbol$col else superpose.symbol$col,  
  font = if (is.null(groups))  
  plot.symbol$font else superpose.symbol$font,  
  fontfamily = if (is.null(groups)) plot.symbol$fontfamily else superpose.symbol$fontfamily,  
  fontface = if (is.null(groups)) plot.symbol$fontface else superpose.symbol$fontface,  
  lty = if (is.null(groups))  
  plot.line$lty else superpose.line$lty,  
  cex = if (is.null(groups)) plot.symbol$cex else superpose.symbol$cex,  
  fill = if (is.null(groups)) plot.symbol$fill else superpose.symbol$fill,  
  lwd = if (is.null(groups)) plot.line$lwd else superpose.line$lwd,  
  horizontal = FALSE,  
  panel.levels = 'panel.xyplot',  
  ...,  
  jitter.x = FALSE,  
  jitter.y = FALSE,  
  factor = 0.5,  
  amount = NULL  
)
```

**Arguments**

<code>x</code>	See <code>panel.xyplot</code>
<code>y</code>	See <code>panel.xyplot</code>
<code>type</code>	See <code>panel.xyplot</code>
<code>groups</code>	See <code>panel.xyplot</code>
<code>pch</code>	See <code>panel.xyplot</code>
<code>col</code>	See <code>panel.xyplot</code>
<code>col.line</code>	See <code>panel.xyplot</code>
<code>col.symbol</code>	See <code>panel.xyplot</code>
<code>font</code>	See <code>panel.xyplot</code>
<code>fontfamily</code>	See <code>panel.xyplot</code>
<code>fontface</code>	See <code>panel.xyplot</code>
<code>lty</code>	See <code>panel.xyplot</code>
<code>cex</code>	See <code>panel.xyplot</code>
<code>fill</code>	See <code>panel.xyplot</code>
<code>lwd</code>	See <code>panel.xyplot</code>
<code>horizontal</code>	See <code>panel.xyplot</code>
<code>panel.levels</code>	a function to handle each unique level of the data
<code>...</code>	See <code>panel.xyplot</code>
<code>jitter.x</code>	See <code>panel.xyplot</code>
<code>jitter.y</code>	See <code>panel.xyplot</code>
<code>factor</code>	See <code>panel.xyplot</code>
<code>amount</code>	See <code>panel.xyplot</code>

**Details**

`panel.stratify` is defined almost identically to `panel.xyplot`. `panel.levels` is analogous to `panel.groups`. `panel.levels` may want to handle special cases of `col`, which may be missing if `groups` is `NULL` and may be `NA` if `groups` is not `NULL` (set to `NA` by `panel.superpose`).

`x` and `y` are split into subsets by whichever of them represents levels (`y` if `horizontal` is `TRUE`, `x` otherwise). Corresponding subsets of `x` and `y` are forwarded one at a time, along with the other arguments, to `panel.levels`. Additionally, the current value of level as well as the complete vector of levels are available to `panel.levels`.

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [panel.covplot](#)
- [panel.densitystrip](#)
- [panel.hist](#)
- [panel.xyplot](#)

---

params

*List Documented Parameters*

---

**Description**

Returns a vector of parameters available in a specially-formatted XML document, preserving the order.

**Usage**

```
params(within, by = "name", type = "parameter", ...)
```

**Arguments**

within	an XML 'document' containing parameter elements (character)
by	the parameter attribute by which to interpret x
type	the element type to seek in 'within'
...	ignored

**Details**

This function is argumented nearly identically to `lookup`, except that `x` is missing. Whereas in `lookup`, one is looking for details about elements of `x`, in `params`, one is asking what those elements may be.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [lookup](#)
- [partab](#)

---

parens

*Wrap Text in Parentheses*

---

**Description**

Wrap text in parentheses. Useful when formatting latex tables.

**Usage**

```
parens(x, ...)
```

**Arguments**

x	character
...	ignored

**Details**

Text is wrapped in parentheses, with no intermediate spaces.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [glue](#)

**Examples**

```
parens('x')
```

partab

*Construct a Parameter Table***Description**

Merges data from rlog and lookup to describe model run parameters.

**Usage**

```
partab(
  run,
  project = getwd(),
  boot = FALSE,
  tool = "nm6",
  file = filename(rundir, run, ".ctl"),
  rundir = filename(project, run, if (boot) ".boot" else ""),
  nmlog = file.path(rundir, "NonmemRunLog.csv"),
  nmout = filename(rundir, run, ".lst"),
  x = params(within = within, by = by, type = type, ...),
  within = ctl2xml(readLines(file)),
  by = "name",
  as = c(NA, "estimate", "unit", "prse"),
  type = "parameter",
  append = NULL,
  pattern = NULL,
  ...
)
wikitab(
  run,
  project = getwd(),
  boot = FALSE,
  tool = "nm7",
  file = filename(rundir, run, ".ctl"),
  rundir = filename(project, run, if (boot) ".boot" else ""),
  nmlog = file.path(rundir, "NonmemRunLog.csv"),
  nmout = filename(rundir, run, ".lst"),
  x = params(within = within, by = by, type = type, ...),
  within = ctl2xml(readLines(file)),
  by = "model",
  type = "wiki",
  append = NULL,
  pattern = NULL,
  ...
)
```

**Arguments**

run                    scalar run name (number)

project	path to the directory containing run subdirectories
boot	Were these runs in ‘.boot’ directories?
tool	‘nm6’ or ‘nm7’: controls methodology
file	the run-specific input file to read, contra rlog
rundir	path to run subdirectory
nmlog	The run-specific input file to read (nm6 only). See rlog.
nmout	The run-specific NONMEM output file to read (nm7 only). See rlog.
x	a vector of parameter names (character)
within	an XML ‘document’ containing parameter elements (character)
by	the parameter attribute by which to interpret x
as	vector of parameter attributes as which to describe x. Where NA, the text value of the element (if any) is returned.
type	the element type to seek in ‘within’
append	ignored; NULL is passed to rlog
pattern	ignored; NULL is passed to rlog
...	passed to lookup and rlog

### Details

The idea here is to associate pre-specified model-specific identifiers with model output. Identifiers derive from some text file, such as a NONMEM model control stream. `partab` calls `lookup` to recover the identifiers, and `rlog` to access the results. The common key is ‘name’ by default in `lookup`, and ‘parameter’ by default in `rlog`. Caution is advisable when using these terms in other ways.

`lookup` only allows one value for `as`. Here, `as` can be a vector. The special value NA returns the text element for the parameter (cs. NULL in `lookup`).

This mechanism allows messy model details to be abstracted to a more appropriate location than a modeling script. Output is suitable as an argument to `latex` in package `Hmisc`.

`wikitab` is like `partab`, but only returns the ‘model’ attribute (default) and the element text for elements of type ‘wiki’ (default). It tries to guess the parameter names from the model attribute, using wikimath conventions (see [tos](#)), and then fold in the related details from the run log. `wikitab` is more restrictive than `partab`, but it lets all the usual parameter details arise from a single specification thereof.

### Value

data.frame

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

**See Also**

- [lookup](#)
- [params](#)
- [rlog](#)
- [wikimath](#)
- [tos](#)

plot.nm

*Plot An Object of Class nm***Description**

This specializes the generic plot for nm objects. Standard arguments are derived and passed, together with extra arguments, to a suite of diagnostic plots nmPlots.

**Usage**

```
## S3 method for class 'nm'
plot(
  x,
  which = NULL,
  dvname = 'DV',
  ivname = 'AMT',
  covariates = NULL,
  categorical = NULL,
  continuous = NULL,
  by = NULL,
  ...
)
```

**Arguments**

x	nm
which	names or numbers for subsetting nmPlots
dvname	column name for the dependent variable
ivname	column name for the independent variable
covariates	a list of covariates, guessed by default
categorical	a list of categorical covariates, guessed by default, ignored where not in covariates
continuous	a list of continuous covariates, guessed by default, ignored where not in covariates
by	column name(s) for extra plotting hierarchy ( <b>lattice</b> levels); currently unimplemented
...	extra arguments passed to nmPlots

**Details**

Time within subject is rescaled to start at zero. Covariates are guessed to be everything that is non-structural. By default, covariates with 7 or more unique values are continuous, and the rest are categorical. But you can reallocate, even assigning the same column as both categorical and continuous.

**Value**

a list of trellis objects

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [nmPlots](#)
- [as.nm](#)

---

plotfilename

*Make a Name for a Diagnostic Plot File*

---

**Description**

Make a name for a diagnostic plot file, given supplied arguments.

**Usage**

```
plotfilename(  
  run,  
  dir=getwd(),  
  grp=NULL,  
  onefile=TRUE,  
  stem='DiagnosticPlotReview',  
  pext=if(onefile) '.pdf' else '_%03d.pdf',  
  ...  
)
```

**Arguments**

run	run name
dir	directory name
grp	filename modifiers, see NONR
onefile	includes a counter if FALSE
stem	filename stem
pext	extension, typically 'pdf'
...	ignored

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTTR](#)

---

PLOTTR

*Create Diagnostic Plots for a NONMEM Run*

---

**Description**

PLOTTR is a function to generate diagnostic plots and/or covariate evaluation plots following a successful NONMEM run. It is called by NONR if diagnostic plots are requested or can be called independently following a NONMEM run. Plots use the pdf device.

**Usage**

```
PLOTTR(  
  run,  
  project=getwd(),  
  rundir=filename(project,run),  
  grp = NULL,  
  onefile=TRUE,  
  plotfile=plotfilename(run,project,grp,onefile),  
  logtrans = FALSE,  
  dvname = 'DV',
```

```

epilog=NULL,
grpnames = NULL,
cont.cov = NULL,
cat.cov = NULL,
par.list = NULL,
eta.list = NULL,
missing = -99,
estimated = NULL,
...
)

```

### Arguments

run	a control stream name, typically integer
project	the directory containing run subdirectories
rundir	path for the NONMEM run directory ('project/*'); passed to dataSynthesis
grp	item in NONMEM data file or output table file (default) that can be used to condition plots generated by PLOTTR. Default value is NULL. Example: grp=c('SEX'). Can be more than one, e.g., grp=c('SEX', 'TRT').
onefile	whether to produce one output file or many; passed to pdf
plotfile	file name for diagnostic plots, if any. Default combines 'DiagnosticPlotReview' with the run number, and grp if present. A counter is supplied if onefile=FALSE. You can specify an arbitrary name, coordinating with the value of onefile. Any asterisk will be replaced with the run name.
logtrans	Whether to transform the NONMEM output variables DV, PRED, and IPRED. Default: FALSE.
dvname	name of the dependent variable to use as a label for the diagnostic plots; default: DV
epilog	User-defined function or script to call at end of NONR or PLOTTR. A non-null argument that cannot be coerced by match.fun to a function will be treated as a filename to be sourced. All the arguments normally available to PLOTTR (run, project, dvname, logtrans, grp, grpnames, cont.cov, at.cov, par.list, eta.list, missing, etc) will be available to epilog, as well as any extra arguments you pass to NONR or PLOTTR. A function can declare any of these, but should at minimum contain the ... argument. A script can expect them to be present. See also 'inst/example/epilog.R' in this package. To see exactly what is available, try epilog=function(...)cat(names(list(...))).
grpnames	optional vector of names for grp item. Vector length must equal number of conditions in grp and must have an order corresponding to an increasing sort of grp. Default value is NULL. Example: grpnames=c('Male', 'Female')
cont.cov	vector of continuous covariate names. Names must match those used as column headers in the data file (identified in the control stream). Values are retrieved from the data file if not part of the NONMEM \$TABLE step. Default value is NULL. Example: cont.cov=c('AGE', 'WT', 'CLCR')

<code>cat.cov</code>	vector of categorical covariate names. Names must match those used as column headers in <code>DataFile</code> . Values are retrieved from the data file if not part of the <code>NONMEM\TABLE</code> step. Default value is <code>NULL</code> . Example: <code>cat.cov=c('SEX', 'FOOD')</code>
<code>par.list</code>	vector of <code>NONMEM</code> model parameter names. Values are retrieved from <code>*par.TAB</code> created in <code>NONMEM</code> if not in <code>*.TAB</code> . Default value is <code>NULL</code> . This can be a superset of parameters but only those present in <code>NONMEM</code> output table will be used. Example: <code>par.list=c('CL', 'V', 'V2', 'Q')</code>
<code>eta.list</code>	vector of <code>NONMEM</code> model random effect names. Values are retrieved from <code>*par.TAB</code> created in <code>NONMEM</code> if not in <code>*.TAB</code> . Default value is <code>NULL</code> . This can be a superset of random parameters but only those present in <code>NONMEM</code> output table will be used. Example: <code>eta.list=c('ETA1', 'ETA2', 'ETA3', 'ETA4')</code>
<code>missing</code>	numeric item that defines value used to represent missing items in the <code>NONMEM</code> data file. Default value is <code>'-99'</code> .
<code>estimated</code>	character vector of names for items that <code>NONMEM</code> estimates. Used to annotate parameter and gradient search plots.
<code>...</code>	additional arguments passed to functions that accept them. For example, you can pass <code>onfile=FALSE</code> to <code>pdf</code> . You can also pass additional lattice arguments to modify the diagnostic plots.

### Details

`PLOTTR` creates a plotting dataset using `dataSynthesis`. It passes this dataset to each of `diagnosticPlots`, `covariatePlots`, and `cwresPlots`. It then calls the function named by the `epilog` argument, if any. The example `epilog` (`'example/epilog.R'`) calls `dataSynthesis` itself, and does additional plotting.

The `...` argument can be used creatively, with appropriate caution. Extra arguments are generally passed to secondary functions that accept them, and even some that don't. All the lattice plotting functions accept extra arguments, so you may be able to modify the diagnostic plots judiciously (with the caveat that the same arguments will be passed to `diagnosticPlots`, `covariatePlots`, and `cwresPlots`). `pdf` technically does not accept extra arguments, but any specified argument that it does accept will be passed.

File names deserve special consideration. `plotfile` gives the name (or naming strategy) for the diagnostic plots; it is passed to `pdf` and has a suitable default. If an alternative is specified, `*` will be replaced with run, the run name. For example, the default `'pdf'` name is `'project/DiagnosticPlotReview\*.pdf'`, but you could change it to `'project/*/DiagnosticPlotReview.pdf'`.

### Value

Used for side effects.

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

**See Also**

- [NONR](#)
- [dataSynthesis](#)
- [diagnosticPlots](#)
- [covariatePlots](#)
- [cwresPlots](#)

---

posmat

*Coerce a Matrix to be Positive Definite*

---

**Description**

For a square matrix with an all-positive diagonal, elements are limited to 6 significant digits by rounding the diagonal and shrinking off diagonal elements toward zero. The off-diagonals are reduced by 3 percent as necessary until the determinant is positive.

**Usage**

```
posmat(x, ...)
```

**Arguments**

x	matrix with only positive diagonal elements
...	extra arguments, ignored

**Value**

matrix

**Author(s)**

Leonid Gibianski, modified by Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**Examples**

```
posmat(matrix(c(10.00006, 20.00006, -30, 40), 2, 2))  
posmat(matrix(rep(100, 4), 2, 2))
```

**Description**

`maxChar` returns the number of printed characters for the widest element of `x`. `prev` calculates the previous element for each element in `x`. `nxt` calculates the next element for each element in `x`. `runhead` determines whether each element is the ‘head’ (start) of a run.

**Usage**

```
maxChar(x)
prev(x)
nxt(x)
runhead(x)
```

**Arguments**

`x`                      vector

**Details**

If you forget to round or `signif` a numeric column, you may get more digits than desired in your output file; `maxChar` can warn you. See examples.

`prev` is used by `runhead`. Note that there is no element previous to the first element in a vector, so `prev` returns `NA` in that position. `nxt` is the reverse of `prev` (literally). Note that `next` is a reserved language word in R.

If a ‘run’ is a sequence (possibly of length one) of identical successive values in a vector, `runhead` determines whether an element is the first in such a sequence. Note that by definition, the first element is the start of a run; thus `runhead` returns `TRUE` in that position, even though `prev` returns `NA`.

**Value**

`maxChar`: a scalar integer `prev`: a vector of the same class `nxt`: a vector of the same class `runhead`: a vector of logicals

**Note**

`NAs` in the argument to `runhead` give surprising but reasonable results. It cannot be known whether they are the heads of runs, nor can it be known whether values immediately following them are heads of runs. To treat `NAs` deterministically, convert to some definite value first.

**Author(s)**

Tim Bergsma

**Examples**

```
maxChar(c(1.2, 1.234))
prev(c(1, 2, NA, 3, 3, NA, 4))
nxt(c(1, 2, NA, 3, 3, NA, 4))
runhead(c(1, 2, NA, 3, 3, NA, 4))
```

---

purge.dir

*Purge a Directory*

---

**Description**

Purge a directory, perhaps nicely.

**Usage**

```
purge.dir(dir, nice = FALSE)
```

**Arguments**

dir	a directory
nice	whether to purge system files

**Details**

If nice==TRUE, system files (as defined for all.files in dir) are not removed.

**Value**

used for side effects.

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [runNonmem](#)

---

purge.files

*Purge Files*

---

### Description

If dir is a directory, remove files matching the pattern.

### Usage

```
purge.files(pattern, dir = '.')
```

### Arguments

pattern	a regular expression
dir	a directory path

### Value

used for side effects

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [purge.dir](#)

---

qsub

*Build Commands to Invoke SGE qsub*

---

### Description

Builds one or more commands suitable for invoking Sun Grid Engine's qsub in a shell.

### Usage

```
qsub(command, ...)
```

### Arguments

command	character
...	extra arguments

**Details**

This is almost a wrapper, except that it does not actually make the system call. All argument values are character. Values are concatenated using paste, with the usual effects if any arguments have length greater than one. Arguments may be used more than once, and will be represented in the order received. Passing NA results in the argument instance being dropped. Use an empty string as the value for a flag that does not take a value, e.g. 'soft'. Quote the '@' argument in backticks.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [runCommand](#)

**Examples**

```
qsub('nm.pl', '@='options', hard='', i='stdin', soft='', hard='', N=c('Run3c', 'Run3e'), hold_jid=c(NA, 'Run3c'))
#[1] 'qsub -@ options -hard -i stdin -soft -hard -N Run3c nm.pl'
#[2] 'qsub -@ options -hard -i stdin -soft -hard -N Run3e -hold_jid Run3c nm.pl'
```

---

reapply

*Apply a Function Across Cells of an Indexed Vector, Giving an Identically-indexed Result*

---

**Description**

Like calling tapply but stretching the result to dimensions of x.

**Usage**

```
reapply(x, INDEX, FUN, ...)
```

**Arguments**

x	an atomic object, typically a vector
INDEX	list of factors, each of same length as x
FUN	the function to be applied. In the case of functions like +, %*%, etc., the function name must be quoted.
...	optional arguments to FUN

**Details**

The function `tapply` applies FUN to each cell of a vector, as specified by levels of INDEX. `reapply` repeats that result as necessary to match the number of input elements per cell, and restores the order to that of the original index. Regardless of the length of the value of FUN, the length of the value of `reapply` is always identical to that of `x`.

**Value**

an atomic object, typically a vector

**Note**

NA is returned wherever a component of INDEX is NA.

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [tapply](#)

**Examples**

```
theoph <- Theoph[c(1,12,2,13,3,14),]  
theoph$avg <- with(theoph, reapply(conc, Subject, mean))  
theoph$sum <- with(theoph, reapply(conc, Subject, cumsum))  
theoph
```

---

resample.data.frame     *Create Replicate Data Sets by Stratified Sampling*

---

**Description**

`resample` is generic. A method is defined for `data.frame`; a convenience wrapper is provided for passing names of files to be read and then resampled.

**Usage**

```
## S3 method for class 'character'
as.csv.filename(x, ...)
## S3 method for class 'csv.filename'
resample(x, ...)
## S3 method for class 'data.frame'
resample(
  x,
  names,
  key = NULL,
  rekey = FALSE,
  out = NULL,
  stratify = NULL,
  ext = '.csv',
  row.names = FALSE,
  quote = FALSE,
  sep = ',',
  replace = TRUE,
  ...
)
```

**Arguments**

<code>x</code>	a <code>data.frame</code> , or (second/third form) a file name for a file to read
<code>names</code>	a list of names for replicate data sets; can be a simple vector
<code>key</code>	a scalar character value naming the column in <code>x</code> that distinguishes unique individuals, (resampling targets); defaults to row names
<code>rekey</code>	If true, key values in resampled data sets will have unique values of key replaced with consecutive integers, starting at 1.
<code>out</code>	a (path and) directory in which to write resulting data sets
<code>stratify</code>	A list of factors, the interactions of which will be the levels of stratification. Each factor must have the same length as <code>nrow(x)</code> . Or a character vector of names in <code>names(x)</code> .
<code>ext</code>	a file extension
<code>row.names</code>	passed to <code>write.table</code>
<code>quote</code>	passed to <code>write.table</code>
<code>sep</code>	passed to <code>write.table</code>
<code>replace</code>	passed to <code>sample</code>
<code>...</code>	extra arguments, passed to <code>sample</code> and <code>write.table</code>

**Details**

Typical usages are

```
resample(x, names, ...)  
resample(as.csv.filename(x), names, ...)  
resample(as.filename(x), names, ...)
```

The argument `key` gives the name of the column in `x` to identify unique experimental units (individuals). If not supplied, a temporary key is constructed from the row names, and sampling occurs at the row level.

The number of resamplings is controlled by the length of `names`. `names` is coerced to character, and each value is used to name a `'*.csv'` file, if `out` is supplied. If `out` is omitted, a list of `data.frames` is returned.

`stratify` is a list of factors, or items that can be coerced to factors. Currently `stratify` is coerced to a `data.frame` for convenient manipulation. Empty levels are dropped. If `stratify` is not supplied, the whole data set is treated as a single level. Otherwise, each resulting data set has as many keys in each level as the original. An error results if `key` is not nested within `stratify`.

The default behavior is to sample with replacement (`replace=TRUE`.) This and other arguments to `sample` can be modified.

### Value

A list of `data.frames`, or if `out` is supplied, an invisible list of the numbers of rows of each `data.frame` written to file.

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [metaSub.character](#)
- [sample](#)

### Examples

```
b <- resample(Theoph, key='Subject', names=1:3)  
d <- resample(  
  Theoph,  
  key='Subject',  
  rekey=TRUE,  
  names=1:3,  
  out='.',  
  stratify=Theoph$Dose < mean(Theoph$Dose)  
)  
e <- resample(as.csv.filename('1.csv'), names='theoph')
```

rinvchisq

*Generate Inverse Chi-square Distribution*

---

**Description**

Generate inverse chi-square distribution given n, df, and covariance matrix

**Usage**

```
rinvchisq(n, df, cov)
```

**Arguments**

n	the number of values to return
df	the degrees of freedom for the distribution. This value reflects the uncertainty in the distribution. A reasonable starting point would be a value between the number of subjects and the total number of observations used to obtain the estimates of the variance-covariance matrix.
cov	a single parameter

**Value**

The rinvchisq function returns n parameter deviates for use as simulation parameters.

**Author(s)**

written by Leonid Gibianski; modified by Tim Bergsma.

**References**

<http://mifuns.googlecode.com>

---

riwish

*Generate Inverse Wishart Distribution*

---

**Description**

Generates an inverse Wishart distribution for a variance of a random effect. The functions requires the specification of degrees of freedom, scale matrix, and precision for each random effect.

**Usage**

```
riwish(s, df, prec)
```

**Arguments**

s	scale matrix for inverse Wishart distribution
df	degrees of freedom
prec	precision matrix: the mean of riwish is proportional to prec

**Value**

The variance of the random effect can be drawn from the returned inverse Wishart distribution.

**Author(s)**

written by Leonid Gibiansky; modified by Tim Bergsma.

**References**

<http://mifuns.googlecode.com>

---

rlog

*Generate a Combined Run Log from Multiple NONMEM Runs*

---

**Description**

Generates a combined run log across specified runs. Appends/overwrites existing file. Attempts to purge certain files in run directory.

**Usage**

```
rlog(  
  run,  
  project = getwd(),  
  boot = FALSE,  
  append = TRUE,  
  tool = 'nm6',  
  file = filename(project, 'CombRunLog.csv'),  
  rundir = filename(project, run, if(boot) '.boot' else ''),  
  nmlog = file.path(rundir, 'NonmemRunLog.csv'),  
  nmout = filename(rundir, run, '.lst'),  
  pattern = if(boot)c('^F', '^nonmem.exe', '^P', '^O', '^Run') else '^FD',  
  ...  
)
```

### Arguments

run	vector of run names (numbers)
project	path to the directory containing run subdirectories
boot	Were these runs in ‘.boot’ directories?
append	TRUE: append file; FALSE: overwrite file
tool	‘nm6’ or ‘nm7’: controls methodology
file	The run-generic output file to write. See details.
rundir	path to run subdirectory
nmlog	The run-specific input file to read (nm6 only). See details.
nmout	The run-specific NONMEM output file to read (nm7 only). See details.
pattern	search patterns for files to delete (regular expressions)
...	passed to other functions, such as runstate

### Details

rundir, nmlog, and nmout can be vector; or scalar, in which case they may contain ‘\*’, to be replaced with run names on a per-run basis.

To suppress file deletion, supply a zero-length argument such as NULL for pattern.

To suppress log creation on disk, supply a zero length argument for file.

If test is present, rlog assumes the run is in progress and does not attempt file deletion.

### Value

Combined runlog is returned invisibly in the unilog format. Side effect: if file is specified, runlog format is written to disk.

### Author(s)

written by Bill Knebel; modified by Tim Bergsma.

### References

<http://mifuns.googlecode.com>

### See Also

- [as.unilog.run](#)
- [as.runlog.file](#)
- [runstate](#)

---

`row2tabular`*Format Vector for Latex Tabular Row*

---

**Description**

Format a vector for use as a row in latex tabular environment.

**Usage**

```
row2tabular(x, ...)
```

**Arguments**

<code>x</code>	vector, coerced to character with <code>paste</code>
<code>...</code>	ignored

**Details**

Elements are collapsed into a string, with ampersand as the separator.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [tabular.data.frame](#)

**Examples**

```
row2tabular(names(Theoph))
```

runCommand

*Issue the System Call That Invokes NONMEM***Description**

If `undef` is `FALSE`, `command` is passed to `nm.pl` with supporting arguments. If `grid` is `TRUE`, the result is passed to `qsub` with supporting arguments. If `boot` is `TRUE`, `'&'` is appended to the result. Each element of the result is executed in succession using a system call customized by platform.

**Usage**

```
runCommand(
  command,
  ...,
  run,
  rdir,
  boot,
  urgent,
  checksum,
  grid,
  undef = FALSE,
  ctlfile,
  outfile,
  perl = if (nix()) 'perl -S' else if (!invisible) 'cmd /K perl -S' else 'cmd /C perl -S',
  intern = invisible,
  minimized = invisible,
  invisible = FALSE,
  split = grid,
  N = glue('Run', run, if (split) c('c', 'e') else NULL),
  o = rdir,
  e = rdir,
  L = if (split & interface == 'nm.pl') c(compileflag(compiler(config(dirname(command))))), NA) else NA,
  hold_jid = if (split) c(NA, glue('Run', run, 'c')) else NA,
  V = '',
  j = 'y',
  q = if (split)
    c(
      'compile.q',
      if (urgent) 'all.q' else 'bootstrap.q'
    )
  else
    if (!execute)
      'compile.q'
      else if (urgent) 'all.q' else 'bootstrap.q',
  sync = if (boot) 'n' else 'y',
  shell = 'n',
  b = 'y',
```

```

    cwd = '',
    compile = TRUE,
    execute = TRUE,
    background=FALSE,
    interface = 'nm.pl'
)

```

### Arguments

command	a command to pass to system
run	run name
rdir	run directory
boot	see NONR
urgent	see NONR
checksum	see NONR
grid	see NONR
undef	see NONR
ctlfile	see runNonmem
outfile	see runNonmem
perl	a character string to invoke perl
intern	see NONR, passed to system
minimized	see NONR, passed to system
invisible	see runNonmem
split	whether compile and execute should be run separately
N	passed to qsub
o	passed to qsub
e	passed to qsub
L	passed to qsub as an instance of 'l'
hold\_jid	passed to qsub
V	passed to qsub
j	passed to qsub
q	passed to qsub
sync	passed to qsub. Bootstrap runs usually occur in large quantities, whereas SGE has an internal limitation on number of synchronized processes. Therefore, no attempt is currently made to synchronize bootstrap runs.
shell	passed to qsub
b	passed to qsub
cwd	passed to qsub
compile	passed to nm.pl
execute	passed to nm.pl, influences default for q

background	TRUE appends 'l' to command lines to put the process in the background. Defunct?
interface	the name of a function to prepare NONMEM command lines
...	passed to nm.pl and qsub

### Details

The argument 'L' represents a possibly-constitutive instance of qsub's 'l', but is not called 'l' so that other instances of 'l' (multiple are allowed) will not accidentally override it. Users can override intentionally, of course.

N, L, and hold\_jid are coordinated so that if a run is split, compile status is flagged on the compile run, and the execute run waits for compile to finish.

'q' is handled specially. When overriding, be sure to pass a character vector of length one for a normal run, and of length two if split is TRUE (the default when grid is TRUE). By default, all standalone compile-only runs are diverted to 'compile.q', as well as all compile halves of split runs. By default, all execute-only runs as well as the execute halves of split runs are diverted to 'all.q' if urgent, and 'bootstrap.q' otherwise.

### Value

Used for side effects

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [runNonmem](#)

---

runlog

*Convert Model Metadata to Various Formats*

---

### Description

Metadata from successful model runs, e.g. NONMEM, comes in various formats, some of which are version dependent. NONMEM6 and 7 outputs are interconverted with support for the conventional run log format as well as a universal format.

**Usage**

```

runlog()
unilog()
as.runlog.file(file, ...)
as.unilog.lst(file, run, tool, ...)
as.unilog.pxml(x, run, tool = 'nm7', ...)
as.unilog.runlog(x, tool = 'nm6', ...)
as.runlog.unilog(x, ...)
as.file.runlog(x, file = 'NonmemRunLog.csv', header = FALSE, quote = FALSE, na = '.', ...)

```

**Arguments**

<code>x</code>	data.frame in runlog or unilog format
<code>file</code>	file name for metadata file, e.g. 'NonmemRunLog.csv', '3.lst'
<code>header</code>	whether to include column names in output
<code>quote</code>	whether to quote cell contents
<code>na</code>	string to represent NA in output
<code>...</code>	passed arguments
<code>run</code>	name (number) of the model run corresponding to the data
<code>tool</code>	the tool that created the metadata: currently 'nm6' or 'nm7'

**Details**

These functions are not typically needed by the user, since `rlog` serves as an interface.

Metadata may reside in 'lst' files (NONMEM primary output), in 'NonmemRunLog' files (created for NONMEM6 by Mifuns INFN routine) or in 'ext' files (NONMEM7 secondary output). `as.unilog.lst`, `as.runlog.file`, and `as.pxml.ext` (documented elsewhere) read these formats. `pxml` is an internal xml format produced by `as.pxml.ext` and converted to unilog by `as.unilog.pxml`.

The *runlog* format by convention has the columns: prob, moment, min, cov, mvof, p1...pn, and (possibly) run. p1 through pn are (an arbitrary number of) parameters for that run. The others give, respectively, the problem statement, a flag to identify relative standard error percent, minimization status, covariance status, minimum value of the objective function, and run name (number). The primary values for each variable are given in a single record. Where available, a second record gives 'PRSE', with non-informative entries as necessary. Usually the header is not present in files.

The *unilog* format is fully normalized. It has the columns: tool, run, parameter, moment, value. Moment is, for instance, 'estimate' or 'prse'. Typically all cells are filled and meaningful. The value column is handled as text for maximum compatibility across data types. The term 'parameter' is used informally: several items typically captured are not really parameters *per se*.

Unilog and runlog formats are interconvertible via `as.runlog.unilog` and `as.unilog.runlog`. `as.file.runlog` creates the traditional disk file from the runlog format; writing unilog to disk is left to conventional strategies.

`runlog` and `unilog` return zero-row data.frames with the corresponding formats.



```
close(file)
rlg
as.runlog.unilog(unip)
as.unilog.runlog(rlg)
```

---

runNonmem

*Process a Request for NONMEM Invocation*

---

### **Description**

The heart of NONR, this function handles file-level details and dispatches the other major functions, especially runCommand and PLOTR.

### **Usage**

```
runNonmem(
run,
command,
project,
boot,
urgent,
checkrunno,
diag,
fdata,
epilog,
dvname,
logtrans,
grp,
grpnames,
cont.cov,
cat.cov,
par.list,
eta.list,
missing,
invisible,
checksum,
grid,
nice,
undef,
compile,
execute,
split,
plotfile=plotfilename(run,project,grp),
runext = if(boot) '.boot' else if(grid) '.lock' else '',
rudir = filename(project,run,runext),
outfile = filename(rudir,run, '.lst'),
streams = project,
```

```

ctlfile = filename(streams,run,'.ctl'),
remove = c(
  "^F[ISRCMP]", "^OU", "^nonmem", "^nul$",
  "WK", "LNK$", "fort", "^nm", "lnk$", "set$",
  "^gar", "INT", "^temp", "^tr", "^new",
  if(fdata)c('^FD', '^PR')
),
sync=if(boot)'n'else'y',
interface='nm.pl',
...,
perm.cond=NULL
)

```

### Arguments

run	see NONR
command	see NONR
project	see NONR
boot	see NONR
urgent	see NONR
checkrunno	see NONR
diag	see NONR
fdata	see NONR
epilog	see NONR
dvname	see NONR
logtrans	see NONR
grp	see NONR
grpnames	see NONR
cont.cov	see NONR
cat.cov	see NONR
par.list	see NONR
eta.list	see NONR
missing	see NONR
invisible	see NONR
checksum	see NONR
grid	see NONR
nice	see NONR
udef	see NONR
compile	see NONR
execute	see NONR
split	see NONR

plotfile	see PLOTR
runext	an extension for the run directory
rundir	the directory in which run will occur
outfile	see dataSynthesis
streams	where to find control streams
ctlfile	the original control stream
remove	regular expressions for files to purge
sync	whether the R process should wait for the run to complete
interface	the name of a function to prepare NONMEM command lines
...	extra arguments for other functions
perm.cond	passed to PLOTR; defined here to prevent partial matching of pe argument to qsub

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [NONR](#)
- [dataSynthesis](#)
- [runCommand](#)

---

runstate

*Detect State of a Run Using File Existence Criteria*

---

**Description**

In real time, determines the status of a run generated by NONR.

**Usage**

```

runstate(
  run,
  project=getwd(),
  rundir=file.path(project,run),
  testfile=c('FCON','FILE10','OUTPUT'),
  queued=  c(0,0,0),
  compiled= c(1,0,0),
  running= c(1,1,1),
  done=    c(1,0,1),
  ...
)

```

**Arguments**

run	a run name or number (scalar)
project	path to the directory containing run subdirectories
rundir	the path to the run directory
testfile	vector of filenames possibly present in rundir
queued	logical vector (or coercible); see details
compiled	logical vector (or coercible); see details
running	logical vector (or coercible); see details
done	logical vector (or coercible); see details
...	ignored

**Details**

The status of a run is either queued, compiled, running, done, or indeterminate. The last occurs if the run directory does not exist. It also occurs if the more than one or fewer than one of the other states is detected. queued, compiled, running, and done must have the same length as testfile and must be coercible to logical. They indicate the subset of testfile that is present in rundir when that state applies. Obviously, they should be different from eachother.

runstate only analyzes one run at a time, but can be easily wrapped in sapply.

The defaults for testfile are defined by NONMEM. FCON: nm/ABLOCK.f. FILE10: nm/BLKDAT.f. OUTPUT: nm/BEGIN.f

**Value**

scalar character: one of 'queued', 'compiled', 'running', 'done', or 'indeterminate'.

**Author(s)**

Tim Bergsma.

**References**

<http://mifuns.googlecode.com>

**See Also**

- [rlog](#)

**Examples**

```
runstate(1)
```

---

`safe.call`*Call a Function Safely*

---

**Description**

Some functions do not accept extra arguments. `safe.call` passes only those arguments that will be recognized.

**Usage**

```
safe.call(what, ...)
```

**Arguments**

<code>what</code>	a function
<code>...</code>	extra arguments, to be filtered

**Value**

the result of the called function

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTR](#)

---

`safeQuote`*Single-quote Conditionally* ~~

---

**Description**

Place single quotes around text that contains space and is not already quoted. ~~

**Usage**

```
safeQuote(x)
```

**Arguments**

x                    character~~

**Details**

Text is already quoted if it begins with a single or double quote.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [contains](#)
- [isSubversionedFile](#)

**Examples**

```
safeQuote(c("text", "more text", "'already quoted'"))
```

---

setCwres	<i>Append Conditional Weighted Residuals to an Appropriate File</i>
----------	---------------------------------------------------------------------

---

**Description**

Appends conditional weighted residuals to an appropriate file.

**Usage**

```
setCwres(cwres, file)
```

**Arguments**

cwres	vector of conditional weighted residuals
file	file to append

**Value**

Used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTTR](#)

---

shuffle	<i>Move the Columns of a Data Frame Relative to Each Other</i>
---------	----------------------------------------------------------------

---

**Description**

It can be tedious to reorder the columns in a data.frame. This function lets you move specific columns relative to some other named column.

**Usage**

```
shuffle(x, who, after = NA)
```

**Arguments**

x	data.frame
who	a character vector of column names to move
after	character: the column after which to put who

**Details**

If after is NA, the named columns are moved to the front (before the first column).

**Value**

data.frame: a variant of x

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [match](#)

**Examples**

```
head(Theoph)
head(shuffle(Theoph, 'Time'))
```

---

simblock

*Generate Random Effect Deviates for Simulation*

---

**Description**

Generate random effect values given the number of simulations, degrees of freedom, and a variance-covariance matrix (block).

**Usage**

```
simblock(n, df, cov)
```

**Arguments**

n	the number of simulations that will be performed
df	vector of degrees of freedom of block elements
cov	variance-covariance matrix

**Details**

df and cov should have the same length.

**Value**

matrix, with one row per simulation and one column per element in the corresponding triangular version of cov.

**Author(s)**

written by Leonid Gibiansky; modified by Tim Bergsma.

**References**

<http://mifuns.googlecode.com>

**See Also**

- [simpar](#)

---

simpar

*Create Parameters for Simulation with Uncertainty*

---

**Description**

Given the parameter estimates of a mixed effect model, this function generates sets of parameters for simulation. Each set takes into account the relevant variance-covariance structure of the fixed effects and random effects. Two levels of random effects are supported, following the naming conventions of NONMEM.

**Usage**

```
simpar(  
  nsim,  
  theta,  
  covar,  
  omega,  
  sigma,  
  odf = NULL,  
  sdf = NULL,  
  digits = 4,  
  min = -Inf,  
  max = Inf  
)
```

**Arguments**

<code>nsim</code>	scalar numeric specifying the number of sets to attempt
<code>theta</code>	vector of point estimates of fixed effect parameters
<code>covar</code>	variance-covariance matrix for fixed effect parameters
<code>omega</code>	list of variance-covariance matrices for first level random effects
<code>sigma</code>	list of variance-covariance matrices for second level random effects
<code>odf</code>	vector of omega degrees of freedom, one per matrix
<code>sdf</code>	vector of sigma degrees of freedom, one per matrix
<code>digits</code>	number of significant digits to include in output
<code>min</code>	lower limit for parameter estimates
<code>max</code>	upper limit for parameter estimates

**Details**

If `min` or `max` are non-default (see below), you may want to set `nsim` marginally higher to allow for dropped sets.

`covar` is coerced to matrix using `as.matrix`.

If `omega` and `sigma` are not lists, they are coerced using `list`. Then each element is coerced using `as.matrix`.

By default, each element in `odf` and `sdf` will be the length (number of elements) in the corresponding matrix.

`min` and `max` may be given as scalar values, in which case they apply to all parameters (as do the defaults). Alternatively, the first  $n$  limits may be specified as a vector, in which case the remaining (if any) will be the default. If any simulated parameter does not fall between its limits, inclusive, the entire parameter set (row) is dropped from the result, with warning.

**Value**

matrix, with column names indicating parameters, and row names indicating set number before filtering by `min` and `max`.

**See Also**

- [posmat](#)
- [simblock](#)
- [as.matrix.halfmatrix](#)

**Examples**

```
set.seed(100)
simpar(
  nsim=10,
  theta=c(13,75,1),
  covar=matrix(c(10,7,2,7,30,1,2,1,0.5),ncol=3,nrow=3),
  omega=list(
```

```

0.1,
matrix(c(0.04,0.02,0.02,0.04),ncol=2,nrow=2)
),
odf=c(50,20),
sigma=list(0.04,1),
sdf=c(99,99),
min=rep(0,3),
max=rep(90,3)
)
simpar(
  nsim=1,
  theta=c(13,75,1),
  covar=matrix(c(10,7,2,7,30,1,2,1,0.5),ncol=3,nrow=3),
  omega=list(
0.1,
matrix(c(0.04,0.02,0.02,0.04),ncol=2,nrow=2)
),
odf=c(50,20),
sigma=list(0.04,1),
sdf=c(99,99),
min=rep(0,3),
max=rep(90,3)
)
simpar(
  nsim=1,
  theta=c(13,75,1),
  covar=matrix(c(10,7,2,7,30,1,2,1,0.5),ncol=3,nrow=3),
  omega=list(
0.1,
matrix(c(0.04,0.02,0.02,0.04),ncol=2,nrow=2)
),
odf=c(50,20),
sigma=list(0.04,1),
sdf=c(99,99),
min=Inf,
max=-1
)
)

```

---

snap

*Coerce Values to Nearest of Candidates*


---

### Description

For each value in a numeric vector, return the closest match from a vector of candidate values.

### Usage

```

snap(x, rule = 1, left = TRUE, ...)

```

**Arguments**

<code>x</code>	finite numeric
<code>rule</code>	a vector of (finite numeric) candidates, or a single value giving candidate interval on the real number line
<code>left</code>	whether to return the lesser of two equidistant candidates
<code>...</code>	ignored

**Details**

If `rule` is scalar, it must be positive; a rule will be constructed as a sequence of rule-spaced values that includes zero and includes values at least as extreme as the extremes of `x`. In some sense, this function is the complement to `cut`: whereas in `cut` one specifies the "breaks", with `snap` one specifies a set of "attractors" (breaks are the implied midpoints); both functions map their primary argument to a (possibly) smaller set of values.

**Value**

numeric

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [findInterval](#)
- [cut](#)

**Examples**

```
snap(c(0.0, 1.0, 1.2, 2.0, 2.9, 3))
snap(-10:10, 0.3)
xyplot(conc~Time, data=Theoph, groups=Subject)
times <- c(0, .25, .5, 1, 2, 4, 5, 7, 9, 12, 24)
xyplot(conc~snap(Time, times), data=Theoph, groups=Subject)
```

---

`spaces`*Generate a String of Spaces*

---

**Description**

Generate a string of spaces.

**Usage**

```
spaces(x)
```

**Arguments**

`x` scalar numeric

**Details**

generates a string of `x` spaces

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [command](#)
- [wrap](#)

**Examples**

```
spaces(3)
```

sqrtm

*Help Calculate CWRES*

---

**Description**

This is a function called by [compute.cwres](#) to calculate CWRES.

**Usage**

```
sqrtm(x)
```

**Arguments**

x                    a matrix.

**See Also**

- [compute.cwres](#)
- 

stableMerge

*Execute a Stable Left Join*

---

**Description**

Merge Two Data Sets Without Altering Material from the Left Argument.

**Usage**

```
stableMerge(x, y)
```

**Arguments**

x                    the main data.frame  
y                    a second data.frame containing related information

**Details**

merge is unpredictable, because it can add, drop, and reorder rows, as well as reorder columns and recreate row names. `stableMerge` expects a primary data.frame and a secondary data.frame, and performs a 'stable' left join (`all.x=TRUE`, `all.y=FALSE`). Absence of extra arguments is by design, for integrity. Keys are defined by common columns, and are repeatable in the primary, but must be unique in the secondary data.frame. New columns in the secondary data.frame are appended to the primary, supplying secondary information by key match. No rows are added, or dropped, row/column order is unaffected, and row names are preserved.

**Value**

A dataframe with as many rows as `nrow(x)` but possibly more columns.

**Author(s)**

Tim Bergsma

**See Also**

- [merge](#)

**Examples**

```
#a nonsense example
stableMerge(Theoph, BOD)
```

---

star	<i>Replace Asterisk in x With y</i>
------	-------------------------------------

---

**Description**

helper function to dereference '\*'.

**Usage**

```
star(x, y)
```

**Arguments**

x	character
y	character

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOT](#)

strain

*Reduce x to Those Elements that Occur in Options*

---

**Description**

a helper function that filters x on options

**Usage**

```
strain(x, options)
```

**Arguments**

x	character
options	character

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [PLOTTR](#)

---

summary.nm*Analyze an NM Object*

---

**Description**

These methods test data set features related to use with NONMEM software((c) Icon Development Solutions). The tests are performed after removing comments.

**Usage**

```
## S3 method for class 'nm'
naKeys(x, ...)
## S3 method for class 'nm'
dupKeys(x, ...)
## S3 method for class 'nm'
badAmt(x, ...)
## S3 method for class 'nm'
badDv(x, ...)
## S3 method for class 'nm'
badII(x, ...)
## S3 method for class 'nm'
falseAmt(x, ...)
## S3 method for class 'nm'
falseDv(x, ...)
## S3 method for class 'nm'
noPk(x, ...)
## S3 method for class 'nm'
predoseDv(x, ...)
## S3 method for class 'nm'
zeroAmt(x, ...)
## S3 method for class 'nm'
zeroDv(x, ...)
## S3 method for class 'nm'
summary(object, by = NULL, ...)
## S3 method for class 'nm.summary'
print(x, ...)
```

**Arguments**

x	object of class nm
object	object of class nm
by	character: column names in object on which to table row counts
...	extra arguments, ignored

**Details**

**badAmt** AMT is NA where EVID is 1

**falseAmt** AMT defined where EVID is not 1

**zeroAmt** AMT is zero where EVID is 1

**badDv** DV is NA where EVID is 0

**falseDv** DV is defined where EVID is not 0

**zeroDv** DV is zero where EVID is zero

**predoseDv** DV is defined before the first record within SUBJ where EVID is 1; NA if no EVID records.

**noPk** rows where EVID is never 0 within SUBJ

**badII** II is greater than 0 where ADDL is NA or 0 (unless SS > 0)  
**summary** all of the above, plus some other diagnostics (see value)  
**print** pretty-printing for return value of summary

### Value

logical of length `nrow(x)` unless otherwise stated.

`print` method used for side effects.

`summary` method: class `nm.summary`.

<code>rows</code>	<code>nrow(x)</code>
<code>records</code>	number of active (non-commented) rows
<code>comments</code>	number of commented rows
<code>subjects</code>	number of unique ID, active rows
<code>longestCase</code>	maximum time range among ID, active rows
<code>naKeys</code>	as defined elsewhere, active rows
<code>dupKeys</code>	as defined elsewhere, active rows
<code>badDv</code>	as defined, active rows
<code>falseDv</code>	as defined, active rows
<code>zeroDv</code>	as defined, active rows
<code>predoseDv</code>	as defined, active rows
<code>badAmt</code>	as defined, active rows
<code>falseAmt</code>	as defined, active rows
<code>zeroAmt</code>	as defined, active rows
<code>noPk</code>	as defined, active rows
<code>badII</code>	as defined, active rows
<code>table</code>	counts of active rows tabled by by, if supplied

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [as.nm](#)

---

`svnIsText`*Check Whether Subversioned Files are Text.*

---

**Description**

Check whether Subversion considers a file to be text.

**Usage**

```
svnIsText(x, ...)
```

**Arguments**

<code>x</code>	character vector of file names.
<code>...</code>	ignored

**Details**

See <http://subversion.apache.org/faq.html#binary-files>. When a file is added, Subversion guesses whether it is text or binary, storing the decision implicitly as the value (or lack thereof) of the `svn:mime-type` property. This function makes that decision explicit. Returns NA for files that are not Subversioned. If you don't agree with Subversion's determination, you can alter it manually, or with `svnMarkAsText` or `svnMarkAsNonText`.

**Value**

logical

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [svnMimeType](#)
- [svnMarkAsText](#)
- [svnMarkAsNonText](#)

---

svnMarkAsNonText      *Mark Subversioned Files as Non-text.*

---

**Description**

Mark Subversioned files as non-text.

**Usage**

```
svnMarkAsNonText(x, ...)
```

**Arguments**

x	character vector of file names
...	ignored

**Details**

For each Subversioned file in x, the property 'svn:mime-type' is set to 'application/octet-stream'.

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [isSubversioned](#)
- [svnIsText](#)
- [svnMarkAsText](#)
- [svnPropSet](#)

---

svnMarkAsText	<i>Mark Subversioned Files as Text</i>
---------------	----------------------------------------

---

**Description**

Mark subversioned files as text, rather than binary.

**Usage**

```
svnMarkAsText(x, ...)
```

**Arguments**

x	character vector of file names
...	ignored

**Details**

For each Subversioned file in x, the property 'svn:mime-type' is set to 'text'.

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [isSubversioned](#)
- [svnIsText](#)
- [svnMarkAsNonText](#)
- [svnPropSet](#)

---

svnMimeType	<i>Determine Mime Type for Multiple Files</i>
-------------	-----------------------------------------------

---

**Description**

Determine Subversion mime type for multiple files.

**Usage**

```
svnMimeType(x, ...)
```

**Arguments**

x	character vector of file names
...	ignored

**Details**

Checks the property 'svn:mime-type' for subversioned files. NA where file is not subversioned.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [svnPropSet](#)

---

svnPropGet	<i>Get Subversion Property for a Vector of Files</i>
------------	------------------------------------------------------

---

**Description**

Retrieve a given Subversion property for multiple files.

**Usage**

```
svnPropGet(x,prop,...)
```

**Arguments**

x	character vector of file names
prop	a property
...	ignored arguments

**Details**

A vectorized version of svnPropGetFile.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [svnPropGetFile](#)

---

svnPropGetFile	<i>Get Subversion Property for a Single File</i>
----------------	--------------------------------------------------

---

**Description**

Get a subversion property for a single file.

**Usage**

```
svnPropGetFile(file, prop)
```

**Arguments**

file	scalar character file name
prop	scalar character property name

**Details**

Returns NA if file is not subversioned. Returns an empty string if the property is not defined.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [isSubversionedFile](#)

---

svnPropSet                      *Set a Property on a Vector of Subversioned Files*

---

### Description

Set a property on multiple files.

### Usage

```
svnPropSet(x, prop, value, ...)
```

### Arguments

x	character vector of file names
prop	scalar character property name
value	scalar character property value
...	ignored arguments

### Details

This is a vectorized version of `svnPropSetFile`.

### Value

used for side effects

### Author(s)

Tim Bergsma

### References

<http://metruminstitute.org>

### See Also

- [svnPropSetFile](#)

---

svnPropSetFile	<i>Set a Property on a Subversioned File</i>
----------------	----------------------------------------------

---

**Description**

Set a property on a subversioned file.

**Usage**

```
svnPropSetFile(file, prop, value)
```

**Arguments**

file	scalar character file name
prop	scalar character property name
value	scalar character property value

**Details**

It is an error if the file is not subversioned.

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [isSubversionedFile](#)

---

svnSetMimeType	<i>Set Subversion Mime Type</i>
----------------	---------------------------------

---

**Description**

Set Subversion mime type for multiple files.

**Usage**

```
svnSetMimeType(x, type, ...)
```

**Arguments**

x	character vector
type	scalar character
...	ignored

**Details**

Sets Subversion property 'svn:mime-type' to type, e.g. '/text' or 'application/octet-stream'. Just a wrapper for svnPropSet.

**Value**

used for side effects

**Author(s)**

Tim Bergsma

**References**

<http://metruminstitute.org>

**See Also**

- [svnPropSet](#)

---

synthesis	<i>Sequentially Left-join an Arbitrary Number of Data Frames, Picking Up Novel Columns</i>
-----------	--------------------------------------------------------------------------------------------

---

**Description**

Integrates specified columns from among a list of data.frames.

**Usage**

```
synthesis(x, key = character(0), frames, ...)
```

**Arguments**

x	A character vector of column names to seek
key	column names on which to merge
frames	a list of data frames to search
...	ignored

**Details**

For each data.frame, sought columns as well as key columns are preserved, and the result is left-joined to the previous result (if any) after removing duplicate rows.

**Value**

data.frame

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [dataSynthesis](#)

---

tabular	<i>Create a Tabular Environment</i>
---------	-------------------------------------

---

**Description**

Create a tabular environment.

**Usage**

```
tabular(x, ...)
```

**Arguments**

x	dispatch argument
...	ignored

**Details**

Generic

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [tabular.data.frame](#)

---

tabular.data.frame      *Convert a Data Frame to a Latex Tabular Environment*

---

## Description

Convert a data.frame to a latex tabular environment

## Usage

```
## S3 method for class 'data.frame'
tabular(
  x,
  rules = c(2, 1, 1),
  walls = 0,
  grid = FALSE,
  rowgroups = rownames(x),
  colgroups = names(x),
  rowbreaks = if (grid) breaks(rowgroups, ...) else 0,
  colbreaks = if (grid) breaks(colgroups, ...) else 0,
  charjust = "left",
  numjust = "right",
  justify = ifelse(sapply(x, is.numeric), numjust, charjust),
  colwidth = NA,
  paralign = "top",
  na = "",
  verbatim = ifelse(sapply(x, is.numeric), TRUE, FALSE),
  escape = "#",
  trim = TRUE,
  ...
)
```

## Arguments

x	data.frame
rules	numeric; will be recycled to length 3. indicates number of horizontal lines above and below the header, and below the last row.
walls	numeric, recycled to length 2. Number of vertical lines on left and right of table.
grid	logical, whether to have lines between rows and columns
rowgroups	a vector as long as names(x), non-repeats trigger vertical lines
colgroups	a vector as long as nrow(x), non-repeats trigger horizontal lines
rowbreaks	numeric: a manual way to specify numbers of lines between rows (ignores grid and rowgroups)
colbreaks	numeric: a manual way to specify numbers of lines between columns (ignores grid and colgroups)
charjust	default justification for character columns

<code>numjust</code>	default justification for numeric columns
<code>justify</code>	manual specification of column justifications: left, right, center, or decimal (vector as long as <code>ncol(x)</code> )
<code>colwidth</code>	manual specification of column width. (vector of length <code>nrow(x)</code> .) Overrides <code>justify</code> where not NA.
<code>paralign</code>	used with <code>colwidth</code> to align paragraphs: top, middle, or bottom.
<code>na</code>	string to replace NA elements
<code>verbatim</code>	whether to use verbatim environment for numeric fields. Makes sense for decimal justification; interacts with <code>trim</code> and <code>justify</code> .
<code>escape</code>	symbol used by ‘verb’ command as delimiter. A warning is issued if it is found in non-NA text.
<code>trim</code>	passed to the format command: true by default, so that alignment is the responsibility of just the tabular environment arguments
<code>...</code>	passed to format

### Details

Principal choices here are the number of lines above and below the header, number of lines at the end of the table (rules), and whether to have lines between rows and columns (grid). If you do want the latter, you can modify their placement easily with `rowgroups` and `colgroups`: factor-like objects that show implicitly which sets of columns or rows go together. Neighboring groups will be separated with a line. For multiple lines at a given position, explicit control is offered by `rowbreaks` and `colbreaks`. These latter have lengths one less than their respective dimensions.

Rownames are ignored. If informative, capture them as a column.

### Value

character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [align.decimal](#)
- [breaks](#)
- [row2tabular](#)
- [tabularformat](#)
- [wrap](#)

**Examples**

```
tabular.data.frame(head(Theoph))
```

---

`tabularformat`*Create the Format String for Latex Tabular Environment*

---

**Description**

Create the format string for latex tabular environment.

**Usage**

```
tabularformat(justify, breaks, walls)
```

**Arguments**

<code>justify</code>	character vector of latex alignment symbols
<code>breaks</code>	numeric vector: number of vertical dividers at each column break
<code>walls</code>	number of vertical lines for left and right edge of table

**Details**

Used to set column alignment and aesthetics.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [tabular.data.frame](#)

---

tagvalue

*Convert List to Tag and Value Format*

---

### Description

Convert a list to tag and value format.

### Usage

```
tagvalue(x, sep = "=", collapse = ",", ...)
```

### Arguments

x	list
sep	the symbol to associate tags and values
collapse	the character to concatenated tag-value pairs
...	ignored

### Details

List names, where present, prefix list values, with the indicated separator.

### Value

Character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [latex.options](#)

### Examples

```
tagvalue(list(a=1,b=2,3))
```

---

`text2decimal`*Convert Mixed Text to Decimal*

---

**Description**

Convert character to numeric value where possible.

**Usage**

```
text2decimal(x)
```

**Arguments**

x                    character

**Details**

x is coerced to character; leading/trailing text is dropped; and the result is coerced to numeric. Useful where measurements are reported with units. Leading text is characters not among 0:9, +,-, or .. The numeric region may have a leading sign, must have one or more digits (0:9), may have a decimal point, and may have any number of trailing digits.

**Value**

numeric

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**Examples**

```
text2decimal('every 3.5 hours')
text2decimal('-17 units')
```

---

Tmax	<i>Calculate Tmax</i>
------	-----------------------

---

**Description**

Calculate the Time Associated with the Maximum Concentration for each Subject

**Usage**

```
Tmax(data, id = 'ID', dv = 'DV', time = 'TIME')
```

**Arguments**

data	the name of the R data.frame containing the data to use for the Tmax calculation.
time	chronologically ordered time variable present in data
id	variable in data defining subject level data
dv	dependent variable used to calculate Tmax present in data

**Details**

The Tmax function performs the calculation based on the variables `id`, `time`, and `dv` present in the R data object.

**Value**

One Tmax is returned for each subject.

**Author(s)**

Leonid Gibianski

**References**

<http://mifuns.googlecode.com>

---

**Tmin***Calculate Tmin*

---

**Description**

Calculate the Time Associated with the Minimum Concentration for Each Subject

**Usage**

```
Tmin(data, id = 'ID', dv = 'DV', time = 'TIME')
```

**Arguments**

<code>data</code>	the name of the R data.frame containing the data to use for the Tmin calculation.
<code>time</code>	chronologically ordered time variable present in data
<code>id</code>	variable in data defining subject level data
<code>dv</code>	dependent variable used to calculate Tmin present in data

**Details**

The Tmin function performs the calculation based on the variables `id`, `time`, and `dv` present in the R data object.

**Value**

One Tmin is returned for each subject

**Author(s)**

Leonid Gibianski

**References**

<http://mifuns.googlecode.com>

---

variants

*Locate Variants of a File in Distinctive Subdirectories*

---

### Description

Give the paths to identically named files distinguished by enclosing subdirectories, themselves enclosed by a directory bearing the file name.

### Usage

```
variants(x, path)
```

### Arguments

x	a vector of file names
path	a directory enclosing subdirectories bearing the name(s) in x

### Details

Consider a file tree where identically named variants of a file are stored in distinguishing subdirectories, e.g. directories whose names contain timestamps. The enclosing directory itself bears the same name as the file in question. Thus, one variant of the file may be located at path/file/var1/file and a second at path/file/var2/file. Given file (x) and path, `variants` returns both of these. If x is of length greater than one, all variants are concatenated into a single character vector.

### Value

character

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

### See Also

- [latest](#)

wikiparse

*Parse Wikimath***Description**

Parse wikimath, optionally overriding default style arguments.

**Usage**

```
wikiparse(x, sim = "~", dot = "*", pregreek = "", sup = "^", openSup = "{", closeSup = "}", sub = "_", openSub = "{", closeSub = "}", wrap = false, ...)
wiki2latex(x, ...)
wiki2plotmath(x, ...)
wiki2label(x, ...)
wiki2parameter(x, ...)
```

**Arguments**

x	character
sim	replaces wikimath tilde
dot	replaces wikimath asterisk
pregreek	prefixes theta, omega, sigma, eta
sup	invokes superscript
openSup	begins superscript content
closeSup	ends superscript content
sub	invokes subscript
openSub	begins subscript content
closeSub	ends subscript content
wrap	wraps entire string
...	ignored

**Details**

Wikimath is a text coding convention for simple model statements of the form " $V_c / F (L * h^{-1}) \sim \theta_1 * (WT/70)^{\theta_2}$ ". The functions `wiki2latex` and `wiki2plotmath` are wrappers for `wikiparse` that intend to give nearly identical visual representations in latex and plotmath contexts, respectively. The functions `wiki2label` and `wiki2parameter` are extraction functions that give, e.g., " $V_c/F$ " and "THETA1".

The point of a wikimath statement is to express a mathematical model definition for a quantity of interest, usually in terms of a parameter. Generally, a representation of the quantity is to the left of a tilde, optionally with units in parentheses. To the right of the tilde is a submodel including relevant parameters and constants. Parameters are limited to theta, omega, sigma, and eta, which should be subscripted. The subscript operator in wikimath is the underscore. The superscript operator is the "hat". Arithmetic operators generally represent themselves.

Subscripts and superscripts may be nested to somewhat arbitrary depth. However, to avoid ambiguity, nested levels must be terminated explicitly by means of a space character. Thus “ $V_c / F$ ” is different from “ $V_{c/F}$ ”: in the latter, “/F” is erroneously part of the subscript.

**Value**

character

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [justUnits](#)
- [closers](#)
- [wikitab](#)
- [tos](#)

**Examples**

```
wiki <- "V_c / F (L * h^-1 ) ~theta_1 *(WT/70)^theta_2"  
wiki2latex(wiki)  
wiki2plotmath(wiki)  
wiki2label(wiki)  
wiki2parameter(wiki)
```

---

win

*Identify Windows Platforms*

---

**Description**

Tells whether the platform OS is of type windows

**Usage**

```
win()
```

**Value**

logical

**Author(s)**

Tim Bergsma

**References**

<http://mifuns.googlecode.com>

**See Also**

- [NONR](#)

**Examples**

```
win()
```

---

wrap

*Wrap Text in a Latex Environment*

---

**Description**

Wrap text in a latex environment.

**Usage**

```
wrap(x, environment, options = NULL, args = NULL, depth = 0)
```

**Arguments**

x	character
environment	name of environment
options	list or vector
args	list or vector
depth	integer: extra spaces on the left.

**Details**

x is wrapped in the specified environment, with options and arguments formatted appropriately.

**Value**

character

**Author(s)**

Tim Bergsma

## References

<http://mifuns.googlecode.com>

## See Also

- [command](#)
- [latex.args](#)
- [latex.options](#)
- [spaces](#)
- [ltable.data.frame](#)

## Examples

```
wrap('Hello', 'center')
```

---

xyplot.ext

*Plot the Parameter Search History for a NONMEM 7 Run*

---

## Description

NONMEM7 produces, for run *n*, the file *n.ext*, containing the values of parameter estimates at specified iterations. Terminal estimates are included, and possibly standard errors. This function plots the parameter estimates vs. iteration. 95 percent CI is plotted if standard errors are available. Terminal estimate is indicated.

## Usage

```
## S3 method for class 'ext'
xyplot(
  x,
  data = read.table(file, skip = 1, header = TRUE, check.names = FALSE),
  project = getwd(),
  rundir = filename(project, x),
  file = filename(rundir, x, '.ext'),
  as.table = TRUE,
  auto.key = TRUE,
  layout = c(1, 4),
  scales = list(relation = 'free'),
  type = 'l',
  panel = panel.superpose,
  panel.groups = function(
    x,
    y,
    group.number,
    type,
    ...
```

```
){  
  if (group.number == 3) type <- 'p'  
  panel.xyplot(x = x, y = y, type = type, ...)  
},  
...  
)
```

### Arguments

x	run name (number)
data	a data.frame representing an *.ext file
project	parent of run directory
rundir	run directory
file	path to .ext file
as.table	passed to xyplot
auto.key	passed to xyplot
layout	passed to xyplot
scales	passed to xyplot
type	passed to xyplot
panel	passed to xyplot
panel.groups	passed to xyplot
...	passed to xyplot

### Details

If data is supplied, x,project,rundir,file are irrelevant. data is reshaped, and passed to xyplot with remaining arguments, all of which may be overridden.

### Value

a trellis object

### Note

Even though xyplot.ext looks like a method, it will be more common to call it directly, as in the examples.

### Author(s)

Tim Bergsma

### References

<http://mifuns.googlecode.com>

**See Also**

- [as.pxml.ext](#)

**Examples**

```
ext <- c(
  'TABLE NO. 1: First Order: Goal Function=MINIMUM VALUE OF OBJECTIVE FUNCTION',
  ' ITERATION    THETA1      THETA2      THETA3      SIGMA(1,1)  OMEGA(1,1)  OBJ',
  '      0 1.70000E+00 1.02000E-01 2.90000E+01 0.00000E+00 1.17001E+00 11.570086639848398',
  '      2 1.78158E+00 1.06239E-01 3.05314E+01 0.00000E+00 1.08862E+00 9.377909428896904',
  '      4 1.91182E+00 1.05179E-01 3.14289E+01 0.00000E+00 8.96680E-01 8.983605357031118',
  '      6 1.94836E+00 1.01426E-01 3.20728E+01 0.00000E+00 9.06374E-01 8.940731060922468',
  '      8 1.93983E+00 1.01742E-01 3.20128E+01 0.00000E+00 8.99988E-01 8.940110966224346',
  '     10 1.94057E+00 1.01681E-01 3.20217E+01 0.00000E+00 8.99322E-01 8.940101673144566',
  '     11 1.94057E+00 1.01681E-01 3.20217E+01 0.00000E+00 8.99322E-01 8.940101673144566',
  ' -1000000000 1.94057E+00 1.01681E-01 3.20217E+01 0.00000E+00 8.99322E-01 8.940101673144566',
  ' -1000000001 6.28499E-01 7.36368E-03 1.25313E+00 0.00000E+00 5.44628E-01 0.'
)
file <- textConnection(ext)
data <- read.table(file, skip = 1, header = TRUE, check.names = FALSE)
close(file)
xyplot.ext(data=data)
```

# Index

## \*Topic **datasets**

ctl, 51

## \*Topic **manip**

accept, 6

acceptance, 7

align.decimal, 8

as.comment, 9

as.data.frame.block, 10

as.flag, 11

as.keyed, 13

as.nm, 15

as.nmctl, 18

as.pxml.ext, 20

as.unilog.run, 21

as.xml, 22

attribute, 24

AUC, 25

aug, 26

backtrans, 27

bin, 27

bracket, 29

breaks, 30

c.miTemporal, 31

check.subjects, 33

CLNR, 34

closers, 35

colname<-, 36

command, 37

compileflag, 38

compiler, 39

compute.cwres, 40

config, 47

constant, 48

contains, 49

css, 50

dataFormat, 52

dataSynthesis, 53

deranged, 55

diagnosticPlots, 57

electronicAppendix, 59

episcript, 60

filename, 61

first, 62

fixedwidth, 63

fixProblem, 65

fTable2data.frame, 67

getCovs, 68

getCwres, 69

getdname, 70

getPars, 70

getTabs, 71

glue, 72

groupnames, 73

half.matrix, 74

hash, 75

ibw, 93

inner.data.frame, 94

is.alpha, 96

is.latex.token, 97

is.one.nonalpha, 98

is.square.matrix, 99

isSubversioned, 100

isSubversionedFile, 101

iterations, 101

justUnits, 103

latest, 105

latex.args, 106

latex.options, 107

locf, 108

lookup, 109

ltable, 110

ltable.data.frame, 111

map, 112

metaSub.character, 113

MIfuncs-package, 5

miTemporal, 117

naInContext, 123

nest, 124

- nix, 125
- nm.pl, 126
- nmVersion, 128
- NONR, 129
- omegacor, 134
- Ops.keyed, 136
- ord.matrix, 137
- packageCheck, 138
- panel.densitystrip, 139
- panel.hist, 144
- panel.stratify, 145
- params, 147
- parens, 148
- partab, 149
- plot.nm, 151
- plotfilename, 152
- PLOTR, 153
- posmat, 156
- prev, 157
- purge.dir, 158
- purge.files, 159
- qsub, 159
- reapply, 160
- resample.data.frame, 161
- riwish, 164
- rlog, 165
- row2tabular, 167
- runCommand, 168
- runlog, 170
- runNonmem, 173
- runstate, 175
- safe.call, 177
- safeQuote, 178
- setCwres, 179
- shuffle, 179
- simblock, 180
- sipar, 181
- snap, 183
- spaces, 185
- sqrtn, 186
- stableMerge, 186
- star, 187
- strain, 188
- summary.nm, 188
- svnIsText, 191
- svnMarkAsNonText, 192
- svnMarkAsText, 193
- svnMimeType, 194
- svnPropGet, 195
- svnPropGetFile, 196
- svnPropSet, 197
- svnPropSetFile, 198
- svnSetMimeType, 199
- synthesis, 200
- tabular, 201
- tabular.data.frame, 202
- tabularformat, 204
- tagvalue, 205
- text2decimal, 206
- Tmax, 207
- Tmin, 208
- variants, 209
- wikiparse, 210
- win, 211
- wrap, 212
- xyplot.ext, 213
- \*Topic utilities**
  - rinvchisq, 164
  - [.comment, 78
  - [.comment (as.comment), 9
  - [.flag, 78
  - [.flag (as.flag), 11
  - [.keyed, 78
  - [.keyed (as.keyed), 13
  - [.miTemporal, 78
  - [.miTemporal (c.miTemporal), 31
  - [.nmctl, 78
  - [.nmctl (as.nmctl), 18
  - [<- .miTemporal, 78
  - [<- .miTemporal (c.miTemporal), 31
  - [ [.comment, 78
  - [ [.comment (as.comment), 9
  - [ [.flag, 78
  - [ [.flag (as.flag), 11
  - [ [.miTemporal, 78
  - [ [.miTemporal (c.miTemporal), 31
  - [ [.nmctl, 78
  - [ [.nmctl (as.nmctl), 18
  - %nested.in% (constant), 48
  - %nests% (constant), 48
  - %nested.in%, 78
  - %nests%, 78
- accept, 6, 8, 78, 139
- acceptance, 7, 7, 78, 139
- acr, 78
- acr (css), 50

- add, 77, 92
- administer, 5
- after, 78
- after (first), 62
- aggregate.keyed, 78
- aggregate.keyed (as.keyed), 13
- align.decimal, 8, 78, 203
- and.keyed, 78
- and.keyed (Ops.keyed), 136
- apply, 92
- as.character.comment, 78
- as.character.comment (as.comment), 9
- as.character.flag, 78
- as.character.flag (as.flag), 11
- as.character.miTemporal, 78
- as.character.miTemporal (c.miTemporal), 31
- as.character.nmctl, 78
- as.character.nmctl (as.nmctl), 18
- as.chartime, 78
- as.chartime (c.miTemporal), 31
- as.chartime.numeric, 78
- as.comment, 9, 78
- as.comment.comment, 78
- as.comment.default, 78
- as.csv.filename, 78
- as.csv.filename (resample.data.frame), 161
- as.csv.filename.character, 78
- as.data.frame.block, 10, 78
- as.data.frame.comment, 78
- as.data.frame.comment (as.comment), 9
- as.data.frame.flag, 78
- as.data.frame.flag (as.flag), 11
- as.file.runlog, 78
- as.file.runlog (runlog), 170
- as.filename, 78
- as.filename (metaSub.character), 113
- as.filename.character, 79
- as.flag, 11, 79
- as.flag.default, 79
- as.flag.flag, 79
- as.halfmatrix, 79
- as.halfmatrix (half.matrix), 74
- as.halfmatrix.default, 79
- as.halfmatrix.halfmatrix, 79
- as.keyed, 13, 79
- as.keyed.data.frame, 79
- as.list.nmctl, 79
- as.list.nmctl (as.nmctl), 18
- as.matrix.halfmatrix, 79, 182
- as.matrix.halfmatrix (half.matrix), 74
- as.miDate, 79
- as.miDate (miTemporal), 117
- as.miDate.character, 79
- as.miDate.Date, 79
- as.miDate.dates, 79
- as.miDate.miDate, 79
- as.miDate.numeric, 79
- as.miDateTime, 79
- as.miDateTime (miTemporal), 117
- as.miDateTime.character, 79
- as.miDateTime.chron, 79
- as.miDateTime.miDate, 79
- as.miDateTime.miDateTime, 79
- as.miDateTime.numeric, 79
- as.miDateTime.POSIXct, 79
- as.miDateTime.POSIXlt, 79
- as.miTime, 79
- as.miTime (miTemporal), 117
- as.miTime.character, 79
- as.miTime.miTime, 79
- as.miTime.numeric, 79
- as.miTime.times, 79
- as.moot, 79
- as.moot (deranged), 55
- as.nm, 15, 57, 79, 137, 152, 190
- as.nm.data.frame, 79
- as.nmctl, 18, 79
- as.nmctl.character, 79
- as.numeric.chartime, 79
- as.numeric.chartime (c.miTemporal), 31
- as.pxml (as.pxml.ext), 20
- as.pxml.ext, 20, 23, 79, 125, 172, 215
- as.rigged, 80
- as.rigged (deranged), 55
- as.runlog.file, 80, 166
- as.runlog.file (runlog), 170
- as.runlog.unilog, 22, 80
- as.runlog.unilog (runlog), 170
- as.unilog.lst, 22, 80
- as.unilog.lst (runlog), 170
- as.unilog.pxml, 22, 80
- as.unilog.pxml (runlog), 170
- as.unilog.run, 21, 80, 166, 172
- as.unilog.runlog, 80

- as.unilog.runlog (runlog), 170
- as.xml, 21, 22, 80
- as.xml.character, 29, 80
- as.xml.data.frame, 80, 125
- as.xml.default, 80
- assemble, 89
- assign, 90
- at, 80
- at (first), 62
- attribute, 24, 29, 80
- AUC, 25, 80
- auc, 80
- auc (css), 50
- aug, 26, 80
- autolog.pl, 80, 129
- autolog.pl (nm.pl), 126
  
- backtrans, 27, 80
- badAmt, 80
- badAmt (summary.nm), 188
- badAmt.nm, 80
- badDv, 80
- badDv (summary.nm), 188
- badDv.nm, 80
- badII, 80
- badII (summary.nm), 188
- badII.nm, 80
- bakfor, 80
- bakfor (locf), 108
- before, 80
- before (first), 62
- bin, 27, 80, 142
- bmi, 80
- bmi (ibw), 93
- bracket, 24, 29, 80, 125
- breaks, 30, 80, 203
- bsa, 80
- bsa (ibw), 93
  
- c.comment, 80
- c.comment (as.comment), 9
- c.flag, 80
- c.flag (as.flag), 11
- c.miTemporal, 31, 80, 119
- calculate, 5, 89
- cavg, 80
- cavg (css), 50
- check, 76, 92
- check.subjects, 33, 81
  
- clear, 81
- clear (lookup), 109
- CLNR, 34, 81
- closers, 35, 81, 211
- cmax, 81
- cmax (css), 50
- cmin, 81
- cmin (css), 50
- coerce, 90
- colname (colname<-), 36
- colname<-, 36, 81
- colnames, 37
- command, 37, 81, 106, 107, 185, 213
- compileflag, 38, 81
- compiler, 39, 81
- compute, 89
- compute.cwres, 40, 46, 81, 186
- config, 47, 81
- constant, 48, 81
- constant.default, 81
- contains, 49, 81, 178
- convert, 89, 91, 92
- cov2cor, 102, 135
- covariatePlots, 81, 156
- covariatePlots (diagnosticPlots), 57
- covplot (panel.densitystrip), 139
- crcl, 81
- crcl (ibw), 93
- create, 76, 77, 91–93
- CreateParametersForSimulation (simpar), 181
- crosses, 81
- crosses (constant), 48
- css, 50, 81
- ctl, 51
- ctl2xml, 81
- ctl2xml (lookup), 109
- cut, 28, 184
- cwresPlots, 81, 156
- cwresPlots (diagnosticPlots), 57
  
- Data, 77, 88, 89, 92
- data.frames, 90
- dataFormat, 52, 55, 81
- dataSynthesis, 53, 53, 58, 61, 66, 81, 156, 175, 200
- delete, 76
- densityplot, 142
- deranged, 55, 81

- deranged.data.frame, 81
- deranged.keyed, 81
- determine, 89
- diagnosticPlots, 55, 57, 81, 156
- dir, 60
- distance, 81
- distance (first), 62
- document, 76
- dupKeys, 81
- dupKeys (as.keyed), 13
- dupKeys.default, 81
- dupKeys.nm, 81
- dupKeys.nm (summary.nm), 188
  
- electronicAppendix, 59, 81
- embed, 76
- episcript, 60, 81
- explicitPath, 81
- explicitPath (fixProblem), 65
- extfile, 81
- extfile (fixProblem), 65
- extractPath, 82
- extractPath (fixProblem), 65
  
- f, 82
- f (as.flag), 11
- falseAmt, 82
- falseAmt (summary.nm), 188
- falseAmt.nm, 82
- falseDv, 82
- falseDv (summary.nm), 188
- falseDv.nm, 82
- file, 76
- filename, 61, 82
- findInterval, 184
- first, 62, 82
- fixedwidth, 63, 82
- fixedwidth.data.frame, 82
- fixFile, 82
- fixFile (fixProblem), 65
- fixProblem, 65, 82
- forbak, 82
- forbak (locf), 108
- format, 77
- format.comment, 82
- format.comment (as.comment), 9
- format.flag, 82
- format.flag (as.flag), 11
- format.miDate, 82
- format.miDate (miTemporal), 117
- format.miDateTime, 82
- format.miDateTime (miTemporal), 117
- format.miTime, 82
- format.miTime (miTemporal), 117
- format.nmctl, 82
- format.nmctl (as.nmctl), 18
- fTable2data.frame, 67, 82
  
- generate, 91
- getCovs, 68, 82
- getCwres, 69, 82
- getName, 70, 82
- getPars, 70, 82
- getTabs, 71, 82
- glue, 72, 82, 148
- groupnames, 73, 82
- gsub, 115
  
- half, 82
- half (half.matrix), 74
- half.matrix, 74, 82, 138
- handle, 93
- hash, 75, 82
- helpAdminister, 76
- helpClasses, 76
- helpDataFrame, 77
- helpList, 77
- helpMatrix, 88
- helpModel, 88
- helpPharmacometric, 89
- helpPrepare, 90, 96
- helpQuantify, 90
- helpReport, 91
- helpSimulate, 91
- helpStrategic, 92
- helpVector, 92
- helpVisualize, 93
- hidden, 83
- hidden (as.comment), 9
- hidden.data.frame, 83
- hide, 82
- hide (as.comment), 9
- hide.data.frame, 82
  
- ibw, 83, 93
- identify, 91, 92
- impute, 92
- ind.cwres, 83

- ind.cwres (compute.cwres), 40
- inner, 83
- inner (inner.data.frame), 94
- inner.data.frame, 83, 94
- interconvert, 88
- investigate, 90
- is.alpha, 83, 96, 97
- is.cwres.readable.file, 83
- is.cwres.readable.file (compute.cwres), 40
- is.latex.token, 83, 97, 97, 98
- is.one.nonalpha, 83, 97, 98
- is.square, 83
- is.square (is.square.matrix), 99
- is.square.matrix, 83, 99, 138
- isSubversioned, 83, 100, 101, 192, 193
- isSubversionedFile, 83, 100, 101, 178, 196, 198
- isSymmetric.matrix, 75, 99, 138
- iterations, 83, 101
- join, 77
- justUnits, 83, 103, 211
- ke, 83
- ke (css), 50
- key, 83
- key (as.keyed), 13
- key<-, 83
- key<- (as.keyed), 13
- last, 83
- last (first), 62
- latest, 83, 105, 209
- latex.args, 83, 106, 213
- latex.options, 83, 107, 205, 213
- lbm, 83
- lbm (ibw), 93
- left.keyed, 83
- left.keyed (Ops.keyed), 136
- lhs, 83
- lhs (justUnits), 103
- library, 139
- list, 5, 91
- locate, 76
- locf, 83, 108
- lookup, 83, 91, 109, 148, 151
- lookup.one, 83
- ltable, 83, 110
- ltable.data.frame, 38, 83, 111, 111, 213
- make, 89, 91, 93
- manipulate, 91
- map, 83, 92, 112
- match, 63, 113, 180
- matrices, 90
- maxChar, 83
- maxChar (prev), 157
- merge, 187
- merge.keyed, 83
- merge.keyed (as.keyed), 13
- merge.nm, 83
- merge.nm (as.nm), 15
- metaSub, 83
- metaSub (metaSub.character), 113
- metaSub.character, 84, 113, 163
- metaSub.filename, 84
- MIfuncs, 76–78, 88–93
- MIfuncs (MIfuncs-package), 5
- MIfuncs-package, 5
- minus.keyed, 84
- minus.keyed (Ops.keyed), 136
- minus.moot, 84
- minus.moot (deranged), 55
- miTemporal, 32, 117
- model, 5
- modify, 77
- moot, 84
- moot (deranged), 55
- moot.nm, 84
- move, 77
- msffile, 84
- msffile (fixProblem), 65
- naInContext, 84, 123
- naKeys, 84
- naKeys (as.keyed), 13
- naKeys.default, 84
- naKeys.nm, 84
- naKeys.nm (summary.nm), 188
- name (colname<-), 36
- name<-, 84
- name<- (colname<-), 36
- names, 37
- nest, 84, 124
- nix, 84, 125
- nm, 84
- nm (as.nm), 15

- nm.data-class (compute.cwres), 40
- nm.pl, 84, 126
- nmPlots, 84, 85, 128, 152
- nmVersion, 84, 87, 128
- nocb, 84
- nocb (locf), 108
- NONR, 84, 125, 129, 156, 175, 212
- NONR72, 84
- NONR72 (NONR), 129
- noPk, 84
- noPk (summary.nm), 188
- noPk.nm, 84
- nospace, 84
- nospace (justUnits), 103
- noUnits, 84
- noUnits (justUnits), 103
- nth, 84
- nth (first), 62
- nxt, 84
- nxt (prev), 157
  
- offdiag, 84
- offdiag (half.matrix), 74
- offdiag.halfmatrix, 84
- omegacor, 84, 134
- only, 84
- only (first), 62
- Ops.keyed, 15, 17, 57, 84, 136
- ord, 84
- ord (ord.matrix), 137
- ord.halfmatrix, 84
- ord.matrix, 75, 84, 99, 137
  
- packageCheck, 7, 8, 85, 138
- panel.bar, 85
- panel.bar (panel.hist), 144
- panel.covplot, 85, 147
- panel.covplot (panel.densitystrip), 139
- panel.cuts, 85
- panel.cuts (panel.densitystrip), 139
- panel.densitystrip, 85, 139, 147
- panel.hist, 85, 144, 147
- panel.ref, 85
- panel.ref (panel.densitystrip), 139
- panel.stratify, 85, 142, 145, 145
- panel.xyplot, 147
- params, 85, 147, 151
- parens, 85, 148
- parfile, 85
- parfile (fixProblem), 65
- partab, 85, 148, 149
- paste, 72
- perform, 90
- plot, 5, 93
- plot.nm, 85, 128, 151
- plotfilename, 85, 152
- PLOTR, 27, 58, 69–71, 73, 85, 134, 153, 153, 177, 179, 187, 188
- plus.keyed, 85
- plus.keyed (Ops.keyed), 136
- plus.rigged, 85
- plus.rigged (deranged), 55
- posmat, 85, 156, 182
- predoseDv, 85
- predoseDv (summary.nm), 188
- predoseDv.nm, 85
- prepare, 5
- prev, 85, 157
- print.comment, 85
- print.comment (as.comment), 9
- print.flag, 85
- print.flag (as.flag), 11
- print.halfmatrix, 85
- print.halfmatrix (half.matrix), 74
- print.keyed.summary, 85
- print.keyed.summary (as.keyed), 13
- print.miTemporal, 85
- print.miTemporal (c.miTemporal), 31
- print.nm.summary, 85
- print.nm.summary (summary.nm), 188
- print.nmctl, 85
- print.nmctl (as.nmctl), 18
- purge.dir, 85, 158, 159
- purge.files, 85, 159
  
- qsub, 85, 159
- quantile, 28
  
- read.cwres.data, 85
- read.cwres.data (compute.cwres), 40
- read.nm, 85
- read.nm (as.nm), 15
- read.nmctl, 85
- read.nmctl (as.nmctl), 18
- read.table, 11
- reapply, 85, 160
- reformat, 91
- regex, 115

- regexpr, 49
- rename, 77
- rep.comment, 85
- rep.comment (as.comment), 9
- rep.flag, 86
- rep.flag (as.flag), 11
- rep.miTemporal, 86
- rep.miTemporal (c.miTemporal), 31
- report, 5
- resample, 86
- resample (resample.data.frame), 161
- resample.csv.filename, 86
- resample.data.frame, 86, 161
- resample.filename, 86
- resolve, 86
- resolve (fixProblem), 65
- rhs, 86
- rhs (justUnits), 103
- rig, 86
- rig (deranged), 55
- rig.nm, 86
- rinvchisq, 86, 164
- riwish, 86, 164
- rlog, 86, 135, 151, 165, 172, 177
- row2tabular, 86, 167, 203
- run, 89, 91
- runCommand, 38, 39, 86, 127, 160, 168, 175
- runhead, 86
- runhead (prev), 157
- runlog, 86, 170
- runNonmem, 47, 60, 66, 86, 129, 134, 158, 170, 173
- runstate, 86, 166, 175
- safe.call, 86, 177
- safeQuote, 86, 178
- sample, 163
- scavenge, 86
- scavenge (fixProblem), 65
- seq.default, 31, 32
- seq.miTemporal, 86
- seq.miTemporal (c.miTemporal), 31
- setCwres, 86, 179
- shuffle, 86, 179
- sigmacor, 86
- sigmacor (omegacor), 134
- simblock, 86, 180, 182
- simpar, 86, 181, 181
- simulate, 5
- sink, 76
- snap, 86, 183
- sort.keyed, 86
- sort.keyed (as.keyed), 13
- spaces, 38, 86, 185, 213
- sqrtm, 86, 186
- stableMerge, 86, 186
- star, 86, 187
- strain, 86, 188
- strftime, 31, 32, 117, 118
- summarize, 89
- summary.hidden.data.frame, 87
- summary.hidden.data.frame (as.comment), 9
- summary.keyed, 87
- summary.keyed (as.keyed), 13
- summary.nm, 17, 87, 188
- svnIsText, 60, 87, 191, 192, 193
- svnMarkAsNonText, 60, 87, 191, 192, 193
- svnMarkAsText, 60, 87, 191, 192, 193
- svnMimeType, 87, 191, 194
- svnPropGet, 87, 195
- svnPropGetFile, 87, 195, 196
- svnPropSet, 87, 192–194, 197, 199
- svnPropSetFile, 87, 197, 198
- svnSetMimeType, 199
- synthesis, 53, 87, 200
- tabfile, 87
- tabfile (fixProblem), 65
- table, 28
- tabular, 87, 201
- tabular.data.frame, 8, 30, 87, 112, 167, 201, 202, 204
- tabularformat, 87, 203, 204
- tagvalue, 87, 107, 205
- tapply, 161
- tell, 76
- test, 76, 92
- text2decimal, 87, 206
- Tmax, 87, 207
- tmax, 87
- tmax (css), 50
- Tmin, 87, 208
- tos, 87, 150, 151, 211
- tos (justUnits), 103
- transform, 26
- transform.keyed, 87
- transform.keyed (as.keyed), 13

- TruncateParametersForSimulation
  - (smpar), 181
  
- uniKey, 87
- uniKey (as.keyed), 13
- uniKey.keyed, 87
- unilog, 87
- unilog (runlog), 170
- unilogcor, 87
- unilogcor (omegacor), 134
- unique.miTemporal, 87
- unique.miTemporal (miTemporal), 117
- unitDensity, 87
- unitDensity (panel.densitystrip), 139
- unitHist, 87
- unitHist (panel.hist), 144
- use, 90
  
- variants, 87, 105, 209
- vectors, 90
- visualize, 89, 91
  
- wiki2label, 87
- wiki2label (wikiparse), 210
- wiki2latex, 87, 104
- wiki2latex (wikiparse), 210
- wiki2parameter, 87
- wiki2parameter (wikiparse), 210
- wiki2plotmath, 87, 104
- wiki2plotmath (wikiparse), 210
- wikimath, 151
- wikimath (wikiparse), 210
- wikiparse, 35, 88, 104, 210
- wikitab, 88, 211
- wikitab (partab), 149
- win, 88, 211
- wrap, 38, 88, 106, 107, 185, 203, 212
- write, 20
- write.nm, 88
- write.nm (as.nm), 15
- write.nmctl, 88
- write.nmctl (as.nmctl), 18
- write.table, 64
  
- xtfrm.comment, 88
- xtfrm.comment (as.comment), 9
- xtfrm.flag, 88
- xtfrm.flag (as.flag), 11
- xtfrm.miTemporal, 88
  
- xtfrm.miTemporal (miTemporal), 117
- xyplot.ext, 21, 88, 213
  
- zeroAmt, 88
- zeroAmt (summary.nm), 188
- zeroAmt.nm, 88
- zeroDv, 88
- zeroDv (summary.nm), 188
- zeroDv.nm, 88