

# Package ‘semEff’

October 1, 2020

**Type** Package

**Title** Automatic Calculation of Effects for Piecewise Structural Equation Models

**Version** 0.4.0

**Description** Provides functionality to automatically calculate direct, indirect, and total effects for piecewise structural equation models, comprising lists of fitted models representing structured equations (Lefcheck 2016 <doi:10/f8s8rb>). Confidence intervals are provided via bootstrapping.

**URL** <https://github.com/murphymv/semEff>

**BugReports** <https://github.com/murphymv/semEff/issues>

**Depends** R (>= 3.6.0)

**Imports** boot, lme4, parallel, stats, utils

**Suggests** knitr, rmarkdown

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**Language** en-GB

**NeedsCompilation** no

**Author** Mark Murphy [aut, cre]

**Maintainer** Mark Murphy <murphymv@gmail.com>

**Repository** CRAN

**Date/Publication** 2020-10-01 07:40:02 UTC

## R topics documented:

avgEst . . . . .	2
bootCI . . . . .	4
bootEff . . . . .	5

getData	8
getEff	9
getY	10
gl	11
predEff	13
print.semEff	17
pSapply	17
R2	18
rMapply	22
sdW	23
semEff	23
Shipley	26
Shipley.Growth	27
Shipley.SEM	27
Shipley.SEM.Boot	28
Shipley.SEM.Eff	29
stdCoeff	29
stdEff	30
varW	33
VIF	34
xNam	35

**Index** **37**

avgEst *Weighted Average of Model Estimates*

**Description**

Calculate a weighted average of model estimates (e.g. effects, fitted values, residuals) for a set of models.

**Usage**

```
avgEst(est, weights = "equal", est.names = NULL)
```

**Arguments**

est	A list or nested list of numeric vectors, comprising the model estimates. In the latter case, these should correspond to estimates for candidate models for each of a set of different response variables.
weights	An optional numeric vector of weights to use for model averaging, or a named list of such vectors. The former should be supplied when est is a list, and the latter when it is a nested list (with matching list names). If weights = "equal" (default), a simple average is calculated instead.
est.names	An optional vector of names used to extract and/or sort estimates from the output.

## Details

This function can be used to calculate a weighted average of model estimates such as effects, fitted values, or residuals, where models are typically competing candidate models fit to the same response variable. Weights are typically a 'weight of evidence' type metric such as Akaike model weights (Burnham & Anderson 2002, Burnham *et al.* 2011), which can be conveniently calculated in R using packages such as **MuMIn** or **AICcmodavg**. However, numeric weights of any sort can be used. If none are supplied, the simple average is calculated instead.

Averaging is performed via the 'full'/'zero' rather than 'subset'/'conditional'/'natural' method, meaning that zero is substituted for estimates for any 'missing' parameters (e.g. effects) prior to calculations. This provides a form of shrinkage and thus reduces **estimate bias** (Burnham & Anderson 2002, Grueber *et al.* 2011).

## Value

A numeric vector of the model-averaged estimates, or a list of such vectors.

## References

Burnham, K. P., & Anderson, D. R. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach* (2nd ed.). New York: Springer-Verlag. Retrieved from <https://www.springer.com/gb/book/9780387953649>

Burnham, K. P., Anderson, D. R., & Huyvaert, K. P. (2011). AIC model selection and multimodel inference in behavioral ecology: some background, observations, and comparisons. *Behavioral Ecology and Sociobiology*, **65**(1), 23-35. <https://doi.org/c4mrns>

Dormann, C. F., Calabrese, J. M., Guillera-Arroita, G., Matechou, E., Bahn, V., Barton, K., ... Hartig, F. (2018). Model averaging in ecology: a review of Bayesian, information-theoretic, and tactical approaches for predictive inference. *Ecological Monographs*, **88**(4), 485-504. <https://doi.org/gfgwrv>

Grueber, C. E., Nakagawa, S., Laws, R. J., & Jamieson, I. G. (2011). Multimodel inference in ecology and evolution: challenges and solutions. *Journal of Evolutionary Biology*, **24**(4), 699-711. <https://doi.org/b7b5d4>

Walker, J. A. (2019). Model-averaged regression coefficients have a straightforward interpretation using causal conditioning. *BioRxiv*, 133785. <https://doi.org/c8zt>

## See Also

[weighted.mean](#)

## Examples

```
## Model-averaged effects (coefficients)
m <- Shipley.Growth # candidate models
e <- lapply(m, function(i) coef(summary(i))[, 1])
avgEst(e)

## Using weights
w <- runif(length(e), 0, 1)
avgEst(e, w)
```

```
## Model-averaged predictions
f <- lapply(m, predict)
avgEst(f, w)
```

---

bootCI *Bootstrap Confidence Intervals*

---

### Description

Calculate confidence intervals from bootstrapped model effects.

### Usage

```
bootCI(mod, conf = 0.95, type = "bca", digits = 3, bci.arg = NULL, ...)
```

### Arguments

mod	A fitted model object. Alternatively, a boot object (class "boot"), containing bootstrapped model effects. Can also be a list or nested list of such objects.
conf	A numeric value specifying the confidence level for the intervals.
type	The type of confidence interval to return (defaults to "bca" - see Details). See <a href="#">boot.ci</a> for further specification details.
digits	The number of significant digits to return for numeric values.
bci.arg	A named list of any additional arguments to <code>boot.ci</code> , excepting argument <code>index</code> .
...	Arguments to <code>bootEff</code> .

### Details

This is essentially a wrapper for `boot.ci` from the **boot** package, returning confidence intervals of the specified type and level calculated from bootstrapped model effects. If a model or models is supplied, bootstrapping will first be performed via `bootEff`. Effects for which the confidence intervals do not contain zero are highlighted with an asterisk.

Nonparametric bias-corrected and accelerated confidence intervals (BCa, Efron 1987) are calculated by default, which should provide the most accurate coverage across a range of bootstrap sampling distributions (Puth *et al.* 2015). They will, however, be **inappropriate** for parametric resampling - in which case the default will be set to the bootstrap percentile method instead ("perc").

### Value

A data frame of the effects and bootstrapped confidence intervals, or a list or nested list of same.

**Note**

All bootstrapped confidence intervals will tend to underestimate the true nominal coverage to some extent when sample size is small (Chernick & Labudde 2009), so the appropriate caution should be exercised in interpretation in such cases. Comparison of different interval types may be informative. For example, normal-theory based intervals may outperform bootstrap percentile methods when  $n < 34$  (Hesterberg 2015). Ultimately however, the bootstrap is **not a solution to small sample size**.

**References**

- Chernick, M. R., & Labudde, R. A. (2009). Revisiting Qualms about Bootstrap Confidence Intervals. *American Journal of Mathematical and Management Sciences*, **29**(3–4), 437–456. <https://doi.org/c8zv>
- Efron, B. (1987). Better Bootstrap Confidence Intervals. *Journal of the American Statistical Association*, **82**(397), 171–185. <https://doi.org/gfww2z>
- Hesterberg, T. C. (2015). What Teachers Should Know About the Bootstrap: Resampling in the Undergraduate Statistics Curriculum. *The American Statistician*, **69**(4), 371–386. <https://doi.org/gd85v5>
- Puth, M.-T., Neuhäuser, M., & Ruxton, G. D. (2015). On the variety of methods for calculating confidence intervals by bootstrapping. *Journal of Animal Ecology*, **84**(4), 892–897. <https://doi.org/f8n9rq>

**See Also**

[boot.ci](#), [bootEff](#)

**Examples**

```
## CIs from bootstrapped SEM
(ShIPLEY.SEM.CI <- bootCI(ShIPLEY.SEM.Boot))

## From original SEM (models)
## (not typically recommended - better to use saved boot objects)
# system.time(
#   ShIPLEY.SEM.CI <- bootCI(ShIPLEY.SEM, ran.eff = "site", seed = 53908)
# )
```

---

bootEff

*Bootstrap Effects*


---

**Description**

Bootstrap model effects (standardised coefficients) and optional SEM correlated errors.

**Usage**

```
bootEff(
  mod,
  R = 10000,
  seed = NULL,
  ran.eff = NULL,
  cor.err = NULL,
  catch.err = TRUE,
  parallel = "snow",
  ncpus = NULL,
  cl = NULL,
  bM.arg = NULL,
  ...
)
```

**Arguments**

mod	A fitted model object, or a list or nested list of such objects.
R	Number of bootstrap replicates to generate.
seed	Seed for the random number generator. If not provided, a random five-digit integer is used (see Details).
ran.eff	For mixed models with nested random effects, the name of the variable comprising the highest-level random effect. For non-nested random effects, specify "crossed". Non-specification of this argument when mod is a mixed model(s) will result in an error.
cor.err	Optional, names of SEM correlated errors to be bootstrapped. Should be of the form: <code>c("mod1 ~~ mod2", "mod3 ~~ mod4", ...)</code> (spaces optional), with names matching model names.
catch.err	Logical, should errors generated during model fitting or estimation be caught and NA returned for estimates? If FALSE, any such errors will cause the function to exit.
parallel	The type of parallel processing to use. Can be one of "snow", "multicore", or "no" (for none).
ncpus	Number of system cores to use for parallel processing. If NULL (default), all available cores are used.
cl	Optional cluster to use if parallel = "snow". If NULL (default), a local cluster is created using the specified number of cores.
bM.arg	A named list of any additional arguments to bootMer.
...	Arguments to stdEff.

**Details**

bootEff uses the boot function (primarily) to bootstrap standardised effects from a fitted model or list of models (calculated using stdEff). Bootstrapping is typically nonparametric, i.e. effects are calculated from data where the rows have been randomly sampled with replacement. The number of

replicates is set by default to 10,000, which should provide accurate coverage for confidence intervals in most situations. To ensure that data is resampled in the same way across individual bootstrap operations within the same run (e.g. models in a list), the same seed is set per operation, with the value saved as an attribute to the bootstrapped values (for reproducibility). The seed can either be user-supplied or a randomly-generated five-digit number (default), and is always re-initialised on exit (i.e. `set.seed(NULL)`).

Where `weights` are specified, bootstrapped effects will be a weighted average across the set of candidate models for each response variable, calculated after each model is first refit to the resampled dataset (specifying `weights = "equal"` will use a simple average instead). If no weights are specified and `mod` is a nested list of models, the function will throw an error, as it will be expecting weights for a presumed model averaging scenario. If instead the user wishes to bootstrap each individual model, they should recursively apply the function using `rMapply` (remember to set a seed).

Where names of models with correlated errors are specified to `cor.err`, the function will also return bootstrapped Pearson correlated errors (weighted residuals) for those models. If `weights` are supplied and `mod` is a nested list, residuals will first be averaged across candidate models. If any two models (or candidate sets) with correlated errors were fit to different subsets of data observations, both models/sets are first refit to data containing only the observations in common.

For mixed models with nested random effects, the highest-level random effect (only) in the dataset is resampled, a procedure which should best retain the hierarchical structure of the data (Davison & Hinkley 1997, Ren *et al.* 2010). Lower-level groups or individual observations are not themselves resampled, as these are not independent. The name of this random effect must be supplied to `ran.eff`, matching the name in the data. Incidentally, this form of resampling will result in different sized datasets if observations are unbalanced across groups; however this should not generally be an issue, as the number of independent units (groups), and hence the 'degrees of freedom', remains **unchanged**. For non-nested random effects however (i.e. "crossed"), group resampling will not be appropriate, and (semi-)parametric bootstrapping is performed instead via `bootMer` in the **lme4** package. Users should think carefully about whether their random effects are **nested or not**. NOTE: As `bootMer` takes only a fitted model as its first argument, any model averaging is calculated 'post-hoc' using the estimates in `boot` objects for each candidate model, rather than during the bootstrapping process itself (i.e. the default procedure via `boot`). Results are then returned in a new `boot` object for each response variable or correlated error estimate.

Parallel processing is used by default via the **parallel** package and option `parallel = "snow"` (and is generally recommended), but users can specify the type of parallel processing to use, or none. If "snow", a cluster of workers is created using `makeCluster`, and the user can specify the number of system cores to incorporate in the cluster (defaults to all available). `bootEff` then exports all required objects and functions to this cluster using `clusterExport`, after performing a (rough) match of all objects and functions in the current global environment to those referenced in the model call(s). Users should load any required external packages prior to calling the function.

### Value

An object of class "boot" containing the bootstrapped effects, or a list/nested list of such objects.

### Note

Bootstrapping mixed (or indeed any other) models may take a very long time when the number of replicates, observations, parameters, and/or models is high. To decrease processing time, it may

be worth trying different optimizers and/or other options to generate faster estimates (always check results).

## References

- Burnham, K. P., & Anderson, D. R. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach* (2nd ed.). New York: Springer-Verlag. Retrieved from <https://www.springer.com/gb/book/9780387953649>
- Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and their Application*. Cambridge University Press.
- Ren, S., Lai, H., Tong, W., Aminzadeh, M., Hou, X., & Lai, S. (2010). Nonparametric bootstrapping for hierarchical data. *Journal of Applied Statistics*, *37*(9), 1487–1498. <https://doi.org/dvfzcn>

## See Also

[boot](#), [bootMer](#), [stdEff](#), [residuals](#), [avgEst](#)

## Examples

```
## Bootstrap Shipley SEM (test)
## (set 'site' as group for resampling - highest-level random effect)
bootEff(Shipley.SEM, ran.eff = "site", R = 1, parallel = "no")

## Estimates (use saved boot object, 10000 reps)
lapply(Shipley.SEM.Boot, "[", 1) # original
lapply(Shipley.SEM.Boot, function(i) head(i$t)) # bootstrapped
```

---

getData

*Get Model Data*

---

## Description

Extract the data used to fit a model.

## Usage

```
getData(mod, subset = FALSE, merge = FALSE, env = parent.frame())
```

## Arguments

mod	A fitted model object, or a list or nested list of such objects.
subset	Logical. If TRUE, only observations used to fit the model(s) are returned (i.e. missing observations (NA) or those with zero weight are removed).
merge	Logical. If TRUE, and mod is a list or nested list, a single dataset containing all variables used to fit models is returned.
env	Environment in which to look for data (passed to eval).



**Details**

This is a simple convenience function to return the data used to fit a model, by evaluating the 'data' slot of the model call object. If the 'data' argument of the model call was not specified, or is not a data frame (or coercible to such) containing all variables referenced in the model formula, an error will be thrown - this restriction is largely to ensure that a single coherent dataset of all model variables can be made available for resampling purposes.

If mod is a list of models and merge = TRUE, all (unique) variables used to fit models are merged into a single data frame. This will return an error if subset = TRUE results in datasets with different numbers of observations (rows).

**Value**

A data frame of the variables used to fit the model(s), or a list or nested list of same.

**See Also**

[getCall](#), [eval](#)

**Examples**

```
## Get data used to fit SEM from Shipley (2009)
getData(Shipley.SEM[[1]]) # from single model
getData(Shipley.SEM) # from SEM (list)
getData(Shipley.SEM, merge = TRUE) # from SEM (single dataset)
```

---

getEff

*Get SEM Effects*

---

**Description**

Extract SEM direct, indirect, and/or total effects from an object of class "semEff".

**Usage**

```
getEff(eff, type = "orig")

dirEff(...)

indEff(...)

totEff(...)
```

**Arguments**

eff	An object of class "semEff".
type	The type of effects to return. Must be either "orig" (default) or "boot".
...	Arguments (above) to be passed to getEff from other extractor functions.

**Details**

These are simple extractor functions for effects calculated using `semEff`, intended for convenience (e.g. for use with `predEff`).

**Value**

A list containing the original or bootstrapped effects for each response variable, as numeric vectors or matrices (respectively).

**Functions**

- `getEff`: Extract all effects.
- `dirEff`: Extract direct effects.
- `indEff`: Extract indirect effects.
- `totEff`: Extract total effects.

---

getY

*Get Model Response Variable*

---

**Description**

Extract the response variable from a fitted model on the original or link scale.

**Usage**

```
getY(mod, data = NULL, link = FALSE, offset = FALSE, env = parent.frame())
```

**Arguments**

<code>mod</code>	A fitted model object, or a list or nested list of such objects.
<code>data</code>	An optional dataset, used to first re-fit the model(s).
<code>link</code>	Logical. If TRUE, return the GLM response variable on the link scale (see Details).
<code>offset</code>	Logical. If TRUE, include model offset(s) in the response.
<code>env</code>	Environment in which to look for model data (if none supplied).

**Details**

`getY` will return the response variable from a model by summing the fitted values and the response residuals. If `link = TRUE` and the model is a GLM, the response is returned on the link scale. If this results in undefined values, it is replaced by an estimate based on the 'working' response variable of the GLM (see [glt](#)).

Any offset variables are subtracted from the response by default. This means that, for example, rates rather than raw counts will be returned for poisson GLMs (where applicable).

**Value**

A numeric vector comprising the response variable on the original or link scale, or an array, list of vectors/arrays, or nested list.

**Examples**

```
## All SEM responses (original scale)
getY(Shipley.SEM)

## Estimated response in link scale from binomial model
getY(Shipley.SEM$Live, link = TRUE)
```

---

glt	<i>Generalised Link Transformation</i>
-----	--

---

**Description**

Transform a numeric variable using a GLM link function, or return an estimate of same.

**Usage**

```
glt(x, family = NULL, force.est = FALSE)
```

**Arguments**

x	a positive numeric vector, corresponding to a variable to be transformed. Can also be a list or nested list of such vectors.
family	Optional, the error distribution family containing the link function which will be used to transform x (see <a href="#">family</a> for specification details). If not supplied, it is determined from x (see <a href="#">Details</a> ).
force.est	Logical, whether to force the return of the estimated rather than direct transformation, where the latter is available (i.e. does not contain undefined values).

**Details**

`glt` can be used to provide a 'generalised' transformation of a numeric variable, using the link function from a generalised linear model (GLM) fit to the variable. The transformation is generalised in the sense that it can extend even to cases where a standard link transformation would generate undefined values. It achieves this by using an estimate based on the 'working' response variable of the GLM (see below). If the error distribution family is not specified (default), then it is determined (roughly) from x, with `binomial(link = "logit")` used when all  $x \leq 1$  and `poisson(link = "log")` otherwise. Although the function is generally intended for binomial or poisson variables, any variable which can be fit using `glm` can be supplied. One of the key purposes of `glt` is to allow the calculation of fully standardised effects (coefficients) for GLMs (in which case  $x$  = the response variable), while it can also facilitate the proper calculation of SEM indirect effects (see below).

**Estimating the link transformation**

A key challenge in generating fully standardised effects for a GLM with a non-gaussian link function is the difficulty in calculating appropriate standardised ranges (typically the standard deviation) for the response variable in the link scale. This is because a direct transformation of the response will often produce undefined values. Although methods for circumventing this issue by indirectly estimating the variance of the response on the link scale have been proposed - including a latent-theoretic approach for binomial models (McKelvey & Zavoina 1975) and a more general variance-based method using a pseudo R-squared (Menard 2011) - here an alternative approach is used. Where transformed values are undefined, the function will instead return the synthetic 'working' response from the iteratively reweighted least squares algorithm (IRLS) of the GLM (McCullagh & Nelder 1989). This is reconstructed as the sum of the linear predictor and the working residuals - with the latter comprising the error of the model on the link scale. The advantage of this approach is that a relatively straightforward 'transformation' of any non-gaussian response is readily attainable in all cases. The standard deviation (or other relevant range) can then be calculated using values of the transformed response and used to scale the effects. An additional benefit for piecewise SEMs is that the transformed rather than original response can be specified as a predictor in other models, ensuring that standardised indirect and total effects are calculated correctly (i.e. using the same units).

To ensure a high level of 'accuracy' in the working response - in the sense that the inverse-transformation is practically indistinguishable from the original response variable - the function uses the following iterative fitting procedure to calculate a 'final' working response:

1. A new GLM of the same error family is fit with the original response variable as both predictor and response, and using a single IWLS iteration.
2. The working response is extracted from this model.
3. The inverse transformation of the working response is then calculated.
4. If the inverse transformation is 'effectively equal' to the original response (tested using `all.equal`), the working response is returned; otherwise, the GLM is re-fit with the working response now as the predictor, and steps 2-4 are repeated - each time with an additional IWLS iteration.

This approach will generate a very reasonable transformation of the response variable, which will also be practically indistinguishable from the direct transformation where this can be compared (see Examples). It also ensures that the transformed values, and hence the standard deviation, are the same for any GLM fitting the same response (provided it uses the same link function) - facilitating model comparisons, selection, and averaging.

### **Value**

A numeric vector of the transformed values, or an array, list of vectors/arrays, or nested list.

### **Note**

As we often cannot directly observe the GLM response variable on the link scale, any method estimating its values or statistics will be 'wrong' to some degree. The aim should be to try to minimise this error as far as (reasonably) possible, while also generating standardised effects whose interpretation most closely resembles those of an ordinary linear model - something which the current method achieves. The solution of using the working response from the GLM to scale effects is a purely practical, but reasonable one, and one that takes advantage of modern computing power to minimise error through iterative model fitting. An added bonus is that the estimated variance is

constant across models fit to the same response variable, which cannot be said of previous methods (Menard 2011). The overall approach would be classed as 'observed-empirical' by Grace *et al.* (2018), as it utilises model error variance (the working residuals) rather than theoretical distribution-specific variance.

## References

- Grace, J.B., Johnson, D.J., Lefcheck, J.S. and Byrnes, J.E.K. (2018) Quantifying relative importance: computing standardized effects in models with binary outcomes. *Ecosphere* **9**, e02283. <https://doi.org/gdm5bj>
- McCullagh P. and Nelder, J. A. (1989) *Generalized Linear Models* (2nd Edition). London: Chapman and Hall.
- McKelvey, R. D., & Zavoina, W. (1975). A statistical model for the analysis of ordinal level dependent variables. *The Journal of Mathematical Sociology*, **4**(1), 103-120. <https://doi.org/dqfhpp>
- Menard, S. (2011) Standards for Standardized Logistic Regression Coefficients. *Social Forces* **89**, 1409-1428. <https://doi.org/bvxb6s>

## See Also

[glm, all.equal](#)

## Examples

```
## Compare estimate with a direct link transformation
## (test with a poisson variable, log link)
set.seed(1)
y <- rpois(30, lambda = 10)
y1 <- unname(glt(y, force.est = TRUE))

## Effectively equal?
all.equal(log(y), y1)
# TRUE

## Actual difference...
all.equal(log(y), y1, tolerance = .Machine$double.eps)
# "Mean relative difference: 1.05954e-12"
```

---

predEff

*Predict Effects*

---

## Description

Generate predicted values for SEM direct, indirect, or total effects.

**Usage**

```

predEff(
  mod,
  newdata = NULL,
  effects = NULL,
  eff.boot = NULL,
  re.form = NA,
  type = "link",
  interaction = NULL,
  use.raw = FALSE,
  ci.conf = 0.95,
  ci.type = "bca",
  digits = 3,
  bci.arg = NULL,
  parallel = "no",
  ncpus = NULL,
  cl = NULL,
  ...
)

```

**Arguments**

<code>mod</code>	A fitted model object, or a list or nested list of such objects.
<code>newdata</code>	An optional data frame of new values to predict, which should contain all the variables named in <code>effects</code> or all those used to fit <code>mod</code> .
<code>effects</code>	A numeric vector of effects to predict, or a list or nested list of such vectors. These will typically have been calculated using <code>semEff</code> , <code>bootEff</code> , or <code>stdEff</code> . Alternatively, a boot object produced by <code>bootEff</code> can be supplied.
<code>eff.boot</code>	A matrix of bootstrapped effects used to calculate confidence intervals for predictions, or a list or nested list of such matrices. These will have been calculated using <code>semEff</code> or <code>bootEff</code> .
<code>re.form</code>	For mixed models of class <code>"merMod"</code> , the formula for random effects to condition on when predicting effects. Defaults to <code>NA</code> , meaning random effects are averaged over. See <a href="#">predict.merMod</a> for further specification details.
<code>type</code>	The type of prediction to return (for GLMs). Can be either <code>"link"</code> (default) or <code>"response"</code> .
<code>interaction</code>	An optional name of an interactive effect, for which to return standardised effects for a 'main' continuous variable across different values or levels of interacting variables (see Details).
<code>use.raw</code>	Logical, whether to use raw (unstandardised) effects for all calculations (if present).
<code>ci.conf</code>	A numeric value specifying the confidence level for confidence intervals on predictions (and any interactive effects).
<code>ci.type</code>	The type of confidence interval to return (defaults to <code>"bca"</code> - see Details). See <a href="#">boot.ci</a> for further specification details.
<code>digits</code>	The number of significant digits to return for interactive effects.

<code>bci.arg</code>	A named list of any additional arguments to <code>boot.ci</code> , excepting argument <code>index</code> .
<code>parallel</code>	The type of parallel processing to use for calculating confidence intervals on predictions. Can be one of "snow", "multicore", or "no" (for none - the default).
<code>ncpus</code>	Number of system cores to use for parallel processing. If NULL (default), all available cores are used.
<code>cl</code>	Optional cluster to use if <code>parallel = "snow"</code> . If NULL (default), a local cluster is created using the specified number of cores.
<code>...</code>	Arguments to <code>stdEff</code> .

## Details

Generate predicted values for SEM direct, indirect, or total effects on a response variable, which should be supplied to `effects`. These are used in place of model coefficients in the standard prediction formula, with values for predictors drawn either from the data used to fit the original model(s) (`mod`) or from `newdata`. It is assumed that effects are fully standardised; however, if this is not the case, then the same centring and scaling options originally specified to `stdEff` should be re-specified - which will then be used to standardise the data. If no effects are supplied, standardised (direct) effects will be calculated from the model and used to generate predictions. These predictions will equal the model(s) fitted values if `newdata = NULL`, `unique.x = FALSE`, and `re.form = NULL` (where applicable).

Model-averaged predictions can be generated if averaged effects are supplied to the model in `mod`, or, alternatively, if weights are specified (passed to `stdEff`) and `mod` is a list of candidate models (effects can also be passed using this latter method). For mixed model predictions where random effects are included (e.g. `re.form = NULL`), the latter approach should be used, otherwise the contribution of random effects will be taken from the single model instead of (correctly) being averaged over a candidate set.

If bootstrapped effects are supplied to `eff.boot` (or to `effects`, as part of a `boot` object), bootstrapped predictions are calculated by predicting from each effect. Confidence intervals can then be returned, for which the `type` should be appropriate for the original form of bootstrap sampling (defaults to "bca"). If the number of observations to predict is very large, parallel processing may speed up the calculation of intervals.

Predictions are always returned in the original (typically unstandardised) units of the (link-)response variable. For GLMs, they can be returned in the response scale if `type = "response"`.

Additionally, if the name of an interactive effect is supplied to `interaction`, standardised effects (and confidence intervals) can be returned for effects of a continuous 'main' variable across different values or levels of interacting variable(s). The name should be of the form "`x1:x2...`", containing all the variables involved and matching the name of an interactive effect in the model(s) terms or in `effects`. The values for all variables should be supplied in `newdata`, with the continuous variable being automatically identified as having the most unique values.

## Value

A numeric vector of the predictions, or, if bootstrapped effects are supplied, a list containing the predictions and the upper and lower confidence intervals. Optional interactive effects may also be appended. Predictions may also be returned in a list or nested list, depending on the structure of `mod` (and other arguments).

**See Also**

[predict](#), [semEff](#), [stdEff](#), [bootCI](#), [pSapply](#)

**Examples**

```
## Predict effects (direct, total)
m <- Shipley.SEM
e <- Shipley.SEM.Eff
dir <- dirEff(e); tot <- totEff(e)
f.dir <- predEff(m, effects = dir, type = "response")
f.tot <- predEff(m, effects = tot, type = "response")

## Using new data for predictors
d <- na.omit(Shipley)
xn <- c("lat", "DD", "Date", "Growth")
seq100 <- function(x) seq(min(x), max(x), length = 100)
nd <- data.frame(sapply(d[xn], seq100))
f.dir <- predEff(m, nd, dir, type = "response")
f.tot <- predEff(m, nd, tot, type = "response")
## Add CIs
# dir.b <- dirEff(e, "boot"); tot.b <- totEff(e, "boot")
# f.dir <- predEff(m, nd, dir, dir.b, type = "response")
# f.tot <- predEff(m, nd, tot, tot.b, type = "response")

## Predict an interactive effect (e.g. Live ~ Growth * DD)
xn <- c("Growth", "DD")
d[xn] <- scale(d[xn]) # scale predictors (improves fit)
m <- lme4::glmer(Live ~ Growth * DD + (1 | site) + (1 | tree),
  family = binomial, data = d)
nd <- with(d, expand.grid(
  Growth = seq100(Growth),
  DD = mean(DD) + c(-sd(DD), sd(DD)) # two levels for DD
))
f <- predEff(m, nd, type = "response", interaction = "Growth:DD")
## Add CIs (need to bootstrap model...)
# system.time(B <- bootEff(m, ran.eff = "site", R = 1000))
# f <- predEff(m, nd, B, type = "response", interaction = "Growth:DD")

## Model-averaged predictions (several approaches)
m <- Shipley.Growth # candidate models (list)
w <- runif(length(m), 0, 1) # weights
e <- stdEff(m, w) # averaged effects
f1 <- predEff(m[[1]], effects = e) # pass avg. effects
f2 <- predEff(m, weights = w) # pass weights argument
f3 <- avgEst(predEff(m), w) # use avgEst function
stopifnot(all.equal(f1, f2))
stopifnot(all.equal(f2, f3))

## Compare model fitted values: predEff() vs. fitted()
m <- Shipley.SEM$Live
f1 <- predEff(m, unique.x = FALSE, re.form = NULL, type = "response")
f2 <- fitted(m)
```



```

stopifnot(all.equal(f1, f2))

## Compare predictions for standardised vs. raw effects
f1 <- predEff(m)
f2 <- predEff(m, use.raw = TRUE)
stopifnot(all.equal(f1, f2))

```

---

```

print.semEff          Print SEM Effects

```

---

### Description

A print method for an object of class "semEff", returning summary tables of effects and confidence intervals for all responses.

### Usage

```

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

```

### Arguments

x                    An object of class "semEff".

...                   Further arguments passed to or from other methods.

---

```

pSapply              Parallel sapply

```

---

### Description

Apply a function to a vector using parallel processing.

### Usage

```

pSapply(
  X,
  FUN,
  parallel = "snow",
  ncpus = NULL,
  cl = NULL,
  add.obj = NULL,
  ...
)

```

**Arguments**

<code>X</code>	A vector object (numeric, character, or list).
<code>FUN</code>	Function to apply to the elements of <code>X</code> .
<code>parallel</code>	The type of parallel processing to use. Can be one of "snow", "multicore", or "no" (for none). If none, <code>sapply</code> is used instead.
<code>ncpus</code>	Number of system cores to use for parallel processing. If NULL (default), all available cores are used.
<code>cl</code>	Optional cluster to use if <code>parallel = "snow"</code> . If NULL (default), a local cluster is created using the specified number of cores.
<code>add.obj</code>	A character vector of any additional object names to be exported to the cluster for parallel processing. Use if a required object or function cannot be found.
<code>...</code>	Arguments to <code>parSapply</code> or <code>sapply</code> .

**Details**

This is a wrapper for `parSapply` from the **parallel** package, enabling (potentially) faster processing of a function over a vector of objects. Parallel processing via option "snow" (default) is carried out using a cluster of workers, which is automatically set up via `makeCluster` using all available system cores or a user supplied number of cores. The function then exports the required objects and functions to this cluster using `clusterExport`, after performing a (rough) match of all objects and functions in the current global environment to those referenced in the call to `FUN` (and also any calls in `X`). Any additional required objects can be supplied using `add.obj`.

**Value**

The output of `FUN` in a list, or simplified to a vector or array.

**See Also**

[parSapply](#), [sapply](#)

**Description**

Calculate R-squared or pseudo R-squared for a fitted model, defined as the squared multiple correlation between the observed and fitted values for the response variable. 'Adjusted' and 'Predicted' versions are also calculated (see Details).

**Usage**

```
R2(
  mod,
  data = NULL,
  adj = TRUE,
  pred = TRUE,
  offset = FALSE,
  re.form = NULL,
  env = parent.frame()
)
```

**Arguments**

<code>mod</code>	A fitted model object, or a list or nested list of such objects.
<code>data</code>	An optional dataset, used to first re-fit the model(s).
<code>adj, pred</code>	Logical. If TRUE (default), adjusted and/or predicted R-squared are also returned (the latter is not available for all model types).
<code>offset</code>	Logical. If TRUE, include model offset(s) in the calculations (i.e. in the response variable and fitted values).
<code>re.form</code>	For mixed models of class "merMod", the formula for random effects to condition on when generating fitted values used in the calculation of R-squared. Defaults to NULL, meaning all random effects are included. See <a href="#">predict.merMod</a> for further specification details.
<code>env</code>	Environment in which to look for model data (if none supplied).

**Details**

Various approaches to the calculation of a goodness-of-fit measure for GLMs analogous to R-squared in the ordinary linear model have been proposed. Generally termed 'pseudo R-squared' measures, they include variance-based, likelihood-based, and distribution-specific approaches. Here however, a more straightforward definition is used, which can be applied to any model for which fitted values of the response variable are generated: R-squared is calculated as the squared (weighted) correlation between the observed and fitted values of the response (in the original units). This is simply the squared version of the correlation measure advocated by Zheng & Agresti (2000), itself an intuitive measure of goodness-of-fit describing the predictive power of a model. As the measure does not depend on any specific error distribution or model estimating procedure, it is also generally comparable across many different types of model (Kvalseth 1985). In the case of the ordinary linear model, the measure equals the more traditional R-squared based on sums of squares.

If `adj = TRUE` (default), the 'adjusted' R-squared value is also returned, which provides an estimate of the population - as opposed to sample - R-squared. This is achieved via an analytical formula which adjusts R-squared for the 'degrees of freedom' of the model (i.e. the ratio of observations to parameters). Here, this is calculated via the 'Pratt' rather than standard 'Ezekiel/Wherry' formula, shown in a previous simulation to be the most effective of a range of existing formulas at estimating the population R-squared across a range of model specification scenarios (Yin & Fan 2001). Adjusted R-squared can be used to safeguard against overfitting of the model to the original sample.

If `pred = TRUE` (default), a 'predicted' R-squared is also returned, which is calculated via the same formula as for R-squared but using cross-validated rather than standard fitted values. These are obtained by dividing the model response residuals by the complement of the observation leverages (diagonals of the hat matrix), then subtracting these inflated 'predicted' residuals from the response variable. This is essentially a short cut to obtaining out-of-sample predictions, normally arising via a leave-one-out cross-validation procedure (in a GLM they will not be exactly equal to such predictions). The resulting R-squared is an estimate of the R-squared that would occur were the model to be fit to new data, and will be lower than the original - and likely also the adjusted - R-squared, highlighting the loss of explanatory power when predicting to new data. This measure is a variant of an **existing one**, calculated by substituting the 'PRESS' statistic, i.e. the sum of squares of the predicted residuals (Allen 1974), for the residual sum of squares in the classic R-squared formula.

For mixed models, the function will, by default, calculate all R-squared measures using fitted values incorporating both the fixed and random effects, thus encompassing all variation captured by the model. This is equivalent to the 'conditional' R-squared of Nakagawa *et al.* (2017). To include only some or no random effects, simply set the appropriate formula using the argument `re.form`, which is passed directly to `predict.merMod`. If `re.form = NA`, R-squared is calculated for the fixed effects only - equivalent to the 'marginal' R-squared of Nakagawa *et al.* (2017).

For models fitted with one or more offsets, these will be removed by default from the response variable and fitted values prior to calculations. Thus R-squared will measure goodness-of-fit only for the predictors with estimated, rather than fixed, coefficients. This is likely to be the intended behaviour in most circumstances, though if users wish to include variation due to the offset(s) they can set `offset = TRUE`.

R-squared values produced by this function will always be bounded between zero (no fit) and one (perfect fit), meaning that any negative values arising from calculations will be rounded up to zero. Negative values typically mean that the fit is 'worse' than the null expectation of no relationship between the variables, which can be difficult to interpret in practice and in any case usually only occurs in rare situations, such as where the intercept is suppressed. Hence, for simplicity and ease of interpretation, values less than zero are presented here as a complete lack of model fit.

### Value

A numeric vector of the R-squared value(s), or an array, list of vectors/arrays, or nested list.

### Note

Caution must be exercised in interpreting the values of any pseudo R-squared measure calculated for a GLM or mixed model (including those produced by this function), as such measures do not hold all the properties of R-squared in the ordinary linear model and as such may not always behave as expected. Care must also be taken in comparing the measures to their equivalents from ordinary linear models, particularly the adjusted and predicted versions. For example, for the adjusted R-squared for mixed models, it's not entirely clear what the sample size ( $n$ ) in the formula should represent - no. of observations? independent groups? something else? (the default interpretation of no. of observations is used). With all that being said, the value of standardised R-squared measures for even 'rough' model fit assessment and comparison may outweigh such reservations, and the adjusted and predicted versions in particular may aid the user in diagnosing and preventing overfitting. They should NOT, however, replace other measures such as AIC or BIC for formally comparing and/or ranking competing models fit to the same response variable.

## References

- Allen, D. M. (1974). The Relationship Between Variable Selection and Data Augmentation and a Method for Prediction. *Technometrics*, **16**(1), 125-127. <https://doi.org/gfgv57>
- Kvalseth, T. O. (1985) Cautionary Note about R2. *The American Statistician*, **39**(4), 279-285. <https://doi.org/b8b782>
- Nakagawa, S., Johnson, P.C.D. and Schielzeth, H. (2017) The coefficient of determination R2 and intra-class correlation coefficient from generalized linear mixed-effects models revisited and expanded. *Journal of the Royal Society Interface* **14**(134). <https://doi.org/gddpnq>
- Yin, P. and Fan, X. (2001) Estimating R2 Shrinkage in Multiple Regression: A Comparison of Different Analytical Methods. *The Journal of Experimental Education* **69**(2), 203-224. <https://doi.org/fbdq5g>
- Zheng, B. and Agresti, A. (2000) Summarizing the predictive power of a generalized linear model. *Statistics in Medicine* **19**(13), 1771-1781. <https://doi.org/db7rfv>

## Examples

```
## Pseudo R-squared for mixed models
R2(Shingley.SEM) # fixed + random ('conditional')
R2(Shingley.SEM, re.form = ~ (1 | tree)) # fixed + 'tree'
R2(Shingley.SEM, re.form = ~ (1 | site)) # fixed + 'site'
R2(Shingley.SEM, re.form = NA) # fixed only ('marginal')

## Predicted R-squared: compare cross-validated predictions calculated/
## approximated via the hat matrix to standard method (leave-one-out)

## Fit test models using Shingley data - compare lm vs glm
d <- na.omit(Shingley)
m <- lm(Live ~ Date + DD + lat, d)
# m <- glm(Live ~ Date + DD + lat, binomial, d)

## Manual CV predictions (leave-one-out)
cvf1 <- sapply(1:nrow(d), function(i) {
  m.ni <- update(m, data = d[-i, ])
  predict(m.ni, d[i, ], type = "response")
})

## Short-cut via the hat matrix
y <- getY(m)
f <- fitted(m)
cvf2 <- y - (y - f) / (1 - hatvalues(m))

## Compare predictions (not exactly equal for GLMs)
all.equal(cvf1, cvf2)
# lm: TRUE; glm: "Mean relative difference: 1.977725e-06"
cor(cvf1, cvf2)
# lm: 1; glm: 0.9999987

# NOTE: comparison not tested here for mixed models, as hierarchical data can
# complicate the choice of an appropriate leave-one-out procedure. However,
# there is no obvious reason why use of the leverage values (diagonals of the
```

```
# hat matrix) to estimate CV predictions shouldn't generalise, roughly, to
# the mixed model case (at least for LMMs). In any case, users should
# exercise the appropriate caution in interpretation.
```

---

rMapply

*Recursive mapply*


---

## Description

Recursively apply a function to a list or lists.

## Usage

```
rMapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

## Arguments

FUN	Function to apply.
...	Object(s) to which FUN can be applied, or lists of such objects to iterate over (defined narrowly, as of class "list").
MoreArgs	A list of additional arguments to FUN.
SIMPLIFY	Logical, whether to simplify the results to a vector or array.
USE.NAMES	Logical, whether to use the names of the first list object in ... for the output.

## Details

rMapply recursively applies FUN to the elements of the lists in ... via mapply. If only a single list is supplied, the function acts like a recursive version of sapply. The particular condition that determines if the function should stop recursing is if either the first or second objects in ... are not of class "list". Thus, unlike mapply, it will not iterate over non-list elements in these objects, but instead returns the output of FUN(...).

This is primarily a convenience function used internally to enable recursive application of functions to lists or nested lists. Its particular stop condition for recursing is also designed to either a) act as a wrapper for FUN if the first object in ... is not a list, or b) apply a model averaging operation if the first object is a list and the second object is a numeric vector (of weights).

## Value

The output of FUN in a list or nested list, or simplified to a vector or array (or list of arrays).

## See Also

[mapply](#)

---

sdW	<i>Weighted Standard Deviation</i>
-----	------------------------------------

---

**Description**

Calculate the weighted standard deviation of x.

**Usage**

```
sdW(...)
```

**Arguments**

```
...           Arguments to varW.
```

**Details**

This is simply a wrapper for varW, applying the square root to the output.

**Value**

A numeric value, the weighted standard deviation of x.

**See Also**

[sd](#), [varW](#)

---

semEff	<i>SEM Effects</i>
--------	--------------------

---

**Description**

Automatically calculate direct, indirect, total, and mediator effects for endogenous (response) variables in a 'piecewise' structural equation model (SEM).

**Usage**

```
semEff(  
  sem,  
  predictors = NULL,  
  mediators = NULL,  
  responses = NULL,  
  use.raw = FALSE,  
  ci.conf = 0.95,  
  ci.type = "bca",  
  digits = 3,  
)
```

```

    bci.arg = NULL,
    ...
  )

```

### Arguments

<code>sem</code>	A piecewise SEM, comprising a list of fitted model objects, or, alternatively, of boot objects (class "boot"), containing bootstrapped model effects.
<code>predictors, mediators, responses</code>	Names of variables for/through which to calculate effects. If NULL (default), all predictors, endogenous predictors (mediators), and endogenous variables (responses) will be used.
<code>use.raw</code>	Logical, whether to use raw (unstandardised) effects for all calculations (if present).
<code>ci.conf</code>	A numeric value specifying the confidence level for confidence intervals on effects.
<code>ci.type</code>	The type of confidence interval to return (defaults to "bca" - see Details). See <a href="#">boot.ci</a> for further specification details.
<code>digits</code>	The number of significant digits to return for numeric values.
<code>bci.arg</code>	A named list of any additional arguments to <code>boot.ci</code> , excepting argument <code>index</code> .
<code>...</code>	Arguments to <code>bootEff</code> .

### Details

The eponymous function of this package calculates all direct, indirect, total, and mediator effects for endogenous variables in a 'piecewise' structural equation model (SEM), that is, one where parameter estimation is local rather than global (Shipley 2000, 2009; Lefcheck 2016). The SEM simply takes the form of a list of fitted models, or bootstrapped estimates from such models, describing hypothesised causal pathways from predictors to response ('endogenous') variables. These are either direct, or operate indirectly via other response variables ('mediators'). This list should represent a directed ('acyclic') causal model, which should be named (exactly) for each response variable and ordered from 'upstream' or 'causal' variables through to 'downstream' (i.e. those at the end of the pathway). If `sem` is a list of fitted models, effects will first be bootstrapped using `bootEff` (this may take a while!).

Direct effects are calculated as fully standardised model coefficients for each response variable, while indirect effects are the product of these direct effects operating along causal pathways in the SEM. The total effects of any given predictor on a response are then the sum of its direct and (all) its indirect effects. 'Mediator' effects are also calculated, as the sum of all indirect paths which operate through each individual mediator - useful to assess the relative importance of different mediators in affecting the response. All of these effect types are calculated automatically for all (default) or a subset of predictors, mediators, or response variables in the SEM.

Confidence intervals for effects are returned for each response, with BCa intervals calculated by default using bootstrapped estimates for each effect type (MacKinnon *et al.* 2004, Cheung 2009, Hayes & Scharkow 2013). As indirect, total, and mediator effects are not directly bootstrapped using the fitted models for response variables (i.e. via `bootEff`), their equivalent 'bootstrapped' estimates are calculated instead using each bootstrapped direct effect.



Correlated errors (and confidence intervals) are also returned if their bootstrapped values are present in `sem`, or, if `sem` is a list of fitted models, if specified to argument `cor.err` (see `bootEff`). These represent residual relationships among response variables, unaccounted for by the SEM.

All effects and bootstrapped effects are returned in lists for each response variable, with all except mediator effects also including the model intercept(s) - required for prediction (this will be zero for ordinary linear models with fully standardised effects).

## Value

A list object of class "semEff", comprising:

1. all effects
2. all bootstrapped effects
3. summary tables of effects and confidence intervals

## References

Cheung, M. W. L. (2009). Comparison of methods for constructing confidence intervals of standardized indirect effects. *Behavior Research Methods*, **41**(2), 425-438. <https://doi.org/fnx7xk>

Hayes, A. F., & Scharkow, M. (2013). The Relative Trustworthiness of Inferential Tests of the Indirect Effect in Statistical Mediation Analysis: Does Method Really Matter? *Psychological Science*, **24**(10), 1918-1927. <https://doi.org/bbhr>

Lefcheck, J. S. (2016). piecewiseSEM: Piecewise structural equation modelling in R for ecology, evolution, and systematics. *Methods in Ecology and Evolution*, **7**(5), 573-579. <https://doi.org/f8s8rb>

MacKinnon, D. P., Lockwood, C. M., & Williams, J. (2004). Confidence Limits for the Indirect Effect: Distribution of the Product and Resampling Methods. *Multivariate Behavioral Research*, **39**(1), 99. <https://doi.org/chqcnx>

Shipley, B. (2000). A New Inferential Test for Path Models Based on Directed Acyclic Graphs. *Structural Equation Modeling: A Multidisciplinary Journal*, **7**(2), 206-218. <https://doi.org/cqm32d>

Shipley, B. (2009). Confirmatory path analysis in a generalized multilevel context. *Ecology*, **90**(2), 363-368. <https://doi.org/bqd43d>

## See Also

[bootEff](#), [bootCI](#)

## Examples

```
## SEM effects
(Shingley.SEM.Eff <- semEff(Shingley.SEM.Boot))

## Effects for selected variables
# semEff(Shingley.SEM.Boot, predictors = "lat")
# semEff(Shingley.SEM.Boot, mediators = "DD")
# semEff(Shingley.SEM.Boot, responses = "Live")
```

```
## Effects calculated using original SEM (models)
## (not typically recommended - better to use saved boot objects)
# system.time(
# Shipley.SEM.Eff <- semEff(Shipley.SEM, ran.eff = "site", seed = 53908)
# )
```

---

Shipley

*Simulated Data from Shipley (2009)*

---

### Description

Simulated Data from Shipley (2009)

### Usage

Shipley

### Format

A data frame with 1900 observations and 9 variables:

**site** a numeric code giving the site from which the observation comes

**tree** a numeric code giving the tree from which the observation comes

**lat** the latitude of the site

**year** the year in which the observation was taken

**Date** the Julian date when the bud burst occurs

**DD** the number of degree days when bud burst occurs

**Growth** the increase in diameter growth of the tree

**Survival** the probability of survival until the next year (used only for the simulation)

**Live** a binary value (1 = tree lived the following winter, 0 = tree died the following winter)

### Source

<https://doi.org/c886>

### References

Shipley, B. (2009). Confirmatory path analysis in a generalized multilevel context. *Ecology*, **90**(2), 363-368. <https://doi.org/bqd43d>

---

ShipleY.Growth

*Candidate Model Set from ShipleY 'Growth' Model*

---

### Description

A set of hypothetical competing models fit to the same response variable ('Growth') using the simulated data in ShipleY (2009), for which model estimates can be compared and/or averaged.

### Usage

ShipleY.Growth

### Format

A list of mixed models of class "lmer" and "glmer", fit to the same response variable.

### References

ShipleY, B. (2009). Confirmatory path analysis in a generalized multilevel context. *Ecology*, **90**(2), 363-368. <https://doi.org/bqd43d>

### Examples

```
## Specification
# ShipleY.Growth <- list(
#   lme4::lmer(Growth ~ Date + (1 | site) + (1 | tree), data = ShipleY),
#   lme4::lmer(Growth ~ Date + DD + (1 | site) + (1 | tree), data = ShipleY),
#   lme4::lmer(Growth ~ Date + DD + lat + (1 | site) + (1 | tree),
#             data = ShipleY)
# )
```

---

ShipleY.SEM

*Hypothesised SEM from ShipleY (2009)*

---

### Description

Hypothesised SEM from ShipleY (2009)

### Usage

ShipleY.SEM

### Format

A list of fitted mixed models of class "lmer" and "glmer", representing structured equations.

## References

Shipley, B. (2009). Confirmatory path analysis in a generalized multilevel context. *Ecology*, **90**(2), 363-368. <https://doi.org/bqd43d>

## Examples

```
## Specification
# Shipley.SEM <- list(
#   DD = lme4::lmer(DD ~ lat + (1 | site) + (1 | tree), data = Shipley),
#   Date = lme4::lmer(Date ~ DD + (1 | site) + (1 | tree), data = Shipley),
#   Growth = lme4::lmer(Growth ~ Date + (1 | site) + (1 | tree),
#                       data = Shipley),
#   Live = lme4::glmer(Live ~ Growth + (1 | site) + (1 | tree), binomial,
#                     data = Shipley)
# )
```

---

Shipley.SEM.Boot

*Bootstrapped Estimates for Shipley SEM*

---

## Description

Bootstrapped estimates generated from the hypothesised SEM from Shipley (2009), using bootEff.

## Usage

```
Shipley.SEM.Boot
```

## Format

A list of objects of class "boot", representing bootstrapped estimates from fitted mixed models.

## References

Shipley, B. (2009). Confirmatory path analysis in a generalized multilevel context. *Ecology*, **90**(2), 363-368. <https://doi.org/bqd43d>

## Examples

```
## Specification
# Shipley.SEM.Boot <- bootEff(Shipley.SEM, ran.eff = "site", seed = 53908)
```

---

Shipley.SEM.Eff	<i>Effects for Shipley SEM</i>
-----------------	--------------------------------

---

**Description**

SEM effects calculated from bootstrapped estimates of the hypothesised SEM from Shipley (2009), using `semEff`.

**Usage**

```
Shipley.SEM.Eff
```

**Format**

A list object of class "semEff", containing SEM effects and summary tables.

**References**

Shipley, B. (2009). Confirmatory path analysis in a generalized multilevel context. *Ecology*, **90**(2), 363-368. <https://doi.org/bqd43d>

**Examples**

```
## Specification
# Shipley.SEM.Eff <- semEff(Shipley.SEM.Boot)
```

---

stdCoeff	<i>Standardised Effects</i>
----------	-----------------------------

---

**Description**

This is a temporary function which calls the renamed original (`stdEff`).

**Usage**

```
stdCoeff(...)
```

**Arguments**

```
... Arguments to stdEff.
```

stdEff

*Standardised Effects***Description**

Calculate fully standardised effects (model coefficients) in standard deviation units, adjusted for multicollinearity among predictors.

**Usage**

```
stdEff(
  mod,
  weights = NULL,
  data = NULL,
  term.names = NULL,
  cen.x = TRUE,
  cen.y = TRUE,
  std.x = TRUE,
  std.y = TRUE,
  unique.x = TRUE,
  refit.x = TRUE,
  r.squared = FALSE,
  incl.raw = FALSE,
  env = parent.frame(),
  ...
)
```

**Arguments**

mod	A fitted model object, or a list or nested list of such objects.
weights	An optional numeric vector of weights to use for model averaging, or a named list of such vectors. The former should be supplied when mod is a list, and the latter when it is a nested list (with matching list names). If set to "equal", a simple average is calculated instead.
data	An optional dataset, used to first re-fit the model(s).
term.names	An optional vector of names used to extract and/or sort effects from the output.
cen.x, cen.y	Logical, whether effects should be calculated using mean-centred variables.
std.x, std.y	Logical, whether effects should be scaled by the standard deviations of variables.
unique.x	Logical, whether effects should be adjusted for multicollinearity among predictors.
refit.x	Logical, whether the model should be re-fit with mean-centred predictors.
r.squared	Logical, whether R-squared values should also be returned.
incl.raw	Logical, whether to append the raw (unstandardised) effects to the output.
env	Environment in which to look for model data (if none supplied).
...	Arguments to R2.

## Details

stdEff will calculate fully standardised effects (coefficients) in standard deviation units for a fitted model or list of models. It achieves this via adjusting the 'raw' model coefficients, so no standardisation of input variables is required beforehand. Users can simply specify the model with all variables in their original units and the function will do the rest. However, the user is free to scale and/or centre any input variables should they choose, which should not affect the outcome of standardisation (provided any scaling is by standard deviations). This may be desirable in some cases, such as to increase numerical stability during model fitting when variables are on widely different scales.

If arguments `cen.x` or `cen.y` are TRUE, effects will be calculated as if all predictors (x) and/or the response variable (y) were mean-centred prior to model-fitting (including any dummy variables arising from categorical predictors). Thus, for an ordinary linear model where centring of x and y is specified, the intercept will be zero - the mean (or weighted mean) of y. In addition, if `cen.x = TRUE` and there are interacting terms in the model, all effects for lower order terms of the interaction are adjusted using an expression which ensures that each main effect or lower order term is estimated at the mean values of the terms they interact with (zero in a 'centred' model) - typically improving the interpretation of effects. The expression used comprises a weighted sum of all the effects that contain the lower order term, with the weight for the term itself being zero and those for 'containing' terms being the product of the means of the other variables involved in that term (i.e. those not in the lower order term itself). For example, for a three-way interaction ( $x_1 * x_2 * x_3$ ), the expression for main effect  $\beta_1$  would be:

$$\beta_1 + \beta_{12}\bar{x}_2 + \beta_{13}\bar{x}_3 + \beta_{123}\bar{x}_2\bar{x}_3$$

(adapted from [here](#))

In addition, if `std.x = TRUE` or `unique.x = TRUE` (see below), product terms for interactive effects will be recalculated using mean-centred variables, to ensure that standard deviations and variance inflation factors (VIF) for predictors are calculated correctly (the model must be re-fit for this latter purpose, to recalculate the variance-covariance matrix).

If `std.x = TRUE`, effects are scaled by multiplying by the standard deviations of predictor variables (or terms), while if `std.y = TRUE` they are divided by the standard deviation of the response variable (minus any offsets). If the model is a GLM, this latter is calculated using the link-transformed response (or an estimate of same) generated using the function `gl.t`. If both arguments are true, the effects are regarded as 'fully' standardised in the traditional sense, often referred to as 'betas'.

If `unique.x = TRUE` (default), effects are adjusted for multicollinearity among predictors by dividing by the square root of the VIFs (Dudgeon 2016, Thompson *et al.* 2017). If they have also been scaled by the standard deviations of x and y, this converts them to semipartial correlations, i.e. the correlation between the unique components of predictors (residualised on other predictors) and the response variable. This measure of effect size is arguably much more interpretable and useful than the traditional standardised coefficient, as it is always estimated independent of other predictors and so can more readily be compared both within and across models. Values range from zero to +/- one rather than +/- infinity (as in the case of betas) - putting them on the same scale as the bivariate correlation between predictor and response. In the case of GLMs however, the measure is analogous but not exactly equal to the semipartial correlation, so its values may not always be bound between +/- one (such cases are likely rare). Importantly, for ordinary linear models, the square of the semipartial correlation equals the increase in R-squared when that variable is added

last in the model - directly linking the measure to model fit and 'variance explained'. See [here](#) for additional arguments in favour of the use of semipartial correlations.

If `refit.x`, `cen.x`, and `unique.x` are TRUE and there are interaction terms in the model, the model will be re-fit with any (newly-)centred continuous predictors, in order to calculate correct VIFs from the variance-covariance matrix. However, re-fitting may not be necessary in some circumstances, for example where predictors have already been mean-centred, and whose values will not subsequently be resampled (e.g. parametric bootstrap). Setting `refit.x = FALSE` in such cases will save time, especially with larger/more complex models and/or bootstrap runs.

If `r.squared = TRUE`, model R-squared values are appended to effects via the R2 function, with any additional arguments being passed via . . . .

If `incl.raw = TRUE`, raw (unstandardised) effects can also be appended, i.e. those with all centring and scaling options set to FALSE (though still adjusted for multicollinearity, where applicable). These may be of interest in some cases, for example to compare their bootstrapped distributions with those of standardised effects.

Finally, if `weights` are specified, the function calculates a weighted average of the standardised effects across models (Burnham & Anderson 2002).

## Value

A numeric vector of the standardised effects, or a list or nested list of such vectors.

## References

Burnham, K. P., & Anderson, D. R. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach* (2nd ed.). New York: Springer-Verlag. Retrieved from <https://www.springer.com/gb/book/9780387953649>

Dudgeon, P. (2016). A Comparative Investigation of Confidence Intervals for Independent Variables in Linear Regression. *Multivariate Behavioral Research*, **51**(2-3), 139-153. <https://doi.org/gfww3f>

Thompson, C. G., Kim, R. S., Aloe, A. M., & Becker, B. J. (2017). Extracting the Variance Inflation Factor and Other Multicollinearity Diagnostics from Typical Regression Results. *Basic and Applied Social Psychology*, **39**(2), 81-90. <https://doi.org/gfww2w>

## See Also

[coef](#), [VIF](#), [getY](#), [R2](#), [avgEst](#)

## Examples

```
library(lme4)

## Standardised (direct) effects for SEM
m <- Shipley.SEM
stdEff(m)
stdEff(m, cen.y = FALSE, std.y = FALSE) # x-only
stdEff(m, std.x = FALSE, std.y = FALSE) # centred only
stdEff(m, cen.x = FALSE, cen.y = FALSE) # scaled only
stdEff(m, unique.x = FALSE) # include multicollinearity
```



```

stdEff(m, r.squared = TRUE) # add R-squared
stdEff(m, incl.raw = TRUE) # add unstandardised

## Demonstrate equality with effects from manually-standardised variables
## (gaussian models only)
m <- Shipley.Growth[[3]]
d <- data.frame(scale(na.omit(Shipley)))
e1 <- stdEff(m, unique.x = FALSE)
e2 <- coef(summary(update(m, data = d)))[, 1]
stopifnot(all.equal(e1, e2))

## Demonstrate equality with square root of increment in R-squared
## (ordinary linear models only)
m <- lm(Growth ~ Date + DD + lat, data = Shipley)
r2 <- summary(m)$r.squared
e1 <- stdEff(m)[-1]
en <- names(e1)
e2 <- sapply(en, function(i) {
  f <- reformulate(en[!en %in% i])
  r2i <- summary(update(m, f))$r.squared
  sqrt(r2 - r2i)
})
stopifnot(all.equal(e1, e2))

## Model-averaged standardised effects
m <- Shipley.Growth # candidate models
w <- runif(length(m), 0, 1) # weights
stdEff(m, w)

```

---

varW

*Weighted Variance*


---

### Description

Calculate the weighted variance of  $x$ .

### Usage

```
varW(x, w = NULL, na.rm = FALSE)
```

### Arguments

<code>x</code>	A numeric vector.
<code>w</code>	A numeric vector of weights of the same length as <code>x</code> .
<code>na.rm</code>	Logical, whether NAs in <code>x</code> should be removed.

### Details

Calculate the weighted variance of  $x$  via the weighted covariance matrix (`cov.wt`). If no weights are supplied, the simple variance is returned instead. As in `weighted.mean`, NAs in `w` are not handled specially and will return NA as result.

**Value**

A numeric value, the weighted variance of  $x$ .

**See Also**

[var](#), [cov.wt](#), [weighted.mean](#)

**Examples**

```
## Weighted variance
x <- rnorm(30)
w <- runif(30, 0, 1)
varW(x, w)

## Simple variance
varW(x)
stopifnot(varW(x) == var(x))

## NA handling
varW(c(x[1:29], NA), w, na.rm = TRUE) # NA in x (removed)
varW(c(x[1:29], NA), w, na.rm = FALSE) # NA in x (NA returned)
varW(x[1:29], w = c(w[1:29], NA)) # NA in w (NA returned)
```

---

VIF

*Generalised Variance Inflation Factors*


---

**Description**

Calculate generalised variance inflation factors for terms in a fitted model via the variance-covariance matrix of coefficients.

**Usage**

```
VIF(mod, data = NULL, env = parent.frame())
```

**Arguments**

<code>mod</code>	A fitted model object, or a list or nested list of such objects.
<code>data</code>	An optional dataset, used to first re-fit the model(s).
<code>env</code>	Environment in which to look for model data (if none supplied).

**Details**

VIF calculates generalised variance inflation factors (GVIF) as described in Fox & Monette (1992), and also implemented in the `vif` function in the **car** package. However, whereas `vif` returns both GVIF and  $GVIF^{1/(2 \cdot Df)}$  values, VIF simply returns the squared result of the latter measure, which equals the standard VIF for single-coefficient terms and is the equivalent measure for multi-coefficient terms (e.g. categorical or polynomial). Also, while `vif` returns values per model term

(i.e. predictor variable), VIF returns values per coefficient, meaning that the same VIF will be returned per coefficient for multi-coefficient terms. Finally, NA is returned for any terms which could not be estimated in the model (e.g. aliased).

### Value

A numeric vector of the VIFs, or an array, list of vectors/arrays, or nested list.

### References

Fox, J. and Monette, G. (1992) Generalized Collinearity Diagnostics. *Journal of the American Statistical Association* **87**, 178-183. <https://doi.org/dm9wbw>

### See Also

[vif \(web\)](#)

### Examples

```
## Model with two correlated terms
m <- Shipley.Growth[[3]]
VIF(m) # Date & DD somewhat correlated
VIF(update(m, . ~ . - DD)) # drop DD

## Model with different types of predictor (some multi-coefficient terms)
d <- data.frame(
  y = rnorm(100),
  x1 = rnorm(100),
  x2 = as.factor(rep(c("a", "b", "c", "d"), each = 25)),
  x3 = rep(1, 100)
)
m <- lm(y ~ poly(x1, 2) + x2 + x3, data = d)
VIF(m)
```

---

xNam

*Get Model Term Names*

---

### Description

Extract term names from a fitted model object.

### Usage

```
xNam(mod, intercept = TRUE, aliased = TRUE, list = FALSE, env = parent.frame())
```

**Arguments**

mod	A fitted model object, or a list or nested list of such objects.
intercept	Logical, whether the intercept should be included.
aliased	Logical, whether names of aliased terms should be included (see Details).
list	Logical, whether names should be returned as a list, with all multi-coefficient terms grouped under their main term names.
env	Environment in which to look for model data (used to construct the model frame).

**Details**

Extract term names from a fitted model. Names of terms for which coefficients cannot be estimated are also included if `aliased = TRUE` (default). These may be terms which are perfectly correlated with other terms in the model, so that the model design matrix is rank deficient.

**Value**

A character vector or list/nested list of term names.

**Examples**

```
## Term names from Shipley SEM
m <- Shipley.SEM
xNam(m)
xNam(m, intercept = FALSE)

## Model with different types of predictor (some multi-coefficient terms)
d <- data.frame(
  y = rnorm(100),
  x1 = rnorm(100),
  x2 = as.factor(rep(c("a", "b", "c", "d"), each = 25)),
  x3 = rep(1, 100)
)
m <- lm(y ~ poly(x1, 2) + x2 + x3, data = d)
xNam(m)
xNam(m, aliased = FALSE) # drop term that cannot be estimated (x3)
xNam(m, aliased = FALSE, list = TRUE) # names as list
```

# Index

## \* datasets

- Shipley, 26
- Shipley.Growth, 27
- Shipley.SEM, 27
- Shipley.SEM.Boot, 28
- Shipley.SEM.Eff, 29

all.equal, 13  
avgEst, 2, 8, 32

boot, 8  
boot.ci, 4, 5, 14, 24  
bootCI, 4, 16, 25  
bootEff, 5, 5, 25  
bootMer, 8

coef, 32  
cov.wt, 34

dirEff (getEff), 9

eval, 9

family, 11

getCall, 9  
getData, 8  
getEff, 9  
getY, 10, 32  
glm, 13  
glt, 10, 11

indEff (getEff), 9

mapply, 22

parSapply, 18  
predEff, 13  
predict, 16  
predict.merMod, 14, 19  
print.semEff, 17

pSapply, 16, 17

R2, 18, 32  
residuals, 8  
rMapply, 22

sapply, 18  
sd, 23  
sdW, 23  
semEff, 16, 23  
Shipley, 26  
Shipley.Growth, 27  
Shipley.SEM, 27  
Shipley.SEM.Boot, 28  
Shipley.SEM.Eff, 29  
stdCoeff, 29  
stdEff, 8, 16, 29, 30

totEff (getEff), 9

var, 34  
varW, 23, 33  
VIF, 32, 34

weighted.mean, 3, 34

xNam, 35