Package 'sigInt'

July 23, 2025

Title Estimate the Parameters of a Discrete Crisis-Bargaining Game

Version 0.2.0

Description Provides pseudo-likelihood methods for empirically analyzing common signaling games in international relations as described in Crisman-Cox and Gibilisco (2019) <doi:10.1017/psrm.2019.58>.

Depends R (>= 3.4.0)

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL `https://github.com/ccrismancox/sigint`

Maintainer Casey Crisman-Cox <`ccrismancox@gmail.com`>

Author Casey Crisman-Cox [aut, cre],
Michael Gibilisco [aut]

RoxygenNote 7.0.2

Imports randomForest, stringr, pbivnorm, Formula, maxLik, xtable, MASS

Suggests parallel

Collate 'convergence.r' 'doc.r' 'estimationfunctions.r' 'generate.eq.r' 'grfunctions.r' 'plot.sigProb.r' 'predict.sigfit.r' 'print.sigfit.r' 'sigint.r' 'summary.sigfit.r' 'toLatex.sigfit.r'

NeedsCompilation no

Repository CRAN

Date/Publication 2019-12-17 13:20:02 UTC

Contents

`generate.eq` *Equilibrium analysis of the empirical crisis signaling game*

Description

This method uses a formula and fixed data/parameters to allow for analysis of the crisis signaling game under specific settings. This function is very similar to `predict.sigfit`, but it is designed for analysis outside of conducting counterfactuals on a fitted model.

Usage

```
generate.eq(
  formulas,
  data,
  theta,
  type = c("actions", "outcomes"),
  na.action = na.omit,
  control = list(),
  parallel = FALSE
)
```

Arguments

`formulas`	a Formula object with no left-hand side and seven separate (7) right-hand sides. See "Details" and examples below.
`data`	a data frame containing the variables in the model Each row of the data frame describes an individual game $d = 1, 2, ..., D$. Each row d should be a summary of all of the within-game observations for game d. If the model is all constants, then this argument should be left empty.
`theta`	a data frame with one or more rows where each row is a parameter vector.
`type`	whether to provide probabilities over actions (default, returns p_C, p_R, and p_F) or outcomes (returns SQ, CD, SF, and BD).
`na.action`	how to deal with `NA`s in `data`.
`control`	list of options describing the grid search method. See "Details" for more information
`parallel`	logical. Should the comparative statics be computed in parallel, requires the `parallel` package be installed. Parallelization is done using `parSapply`.

Details

This function is used to consider comparative statics in the crisis signaling game, where the model of interest has pre-defined parameters. As such, it requires, at minimum, a seven-part formula and parameters. How this function behaves has to do with how `data` and `theta` are specified.

When the model is all constants (every part of the `formula` argument is either `0` or `1`), then `data` is ignored. In these cases, equilibria are computed for every parameter vector, which are supplied as rows in a data frame to `theta`.

When there is one or more covariate in the model, then a data frame must be supplied to `data`. In these cases both `data` or `theta` must have at least one row. However, only one of these arguments can have multiple rows. In other words, only `data` or `theta` may vary, but not both.

For additional implementation details see `predict.sigfit`.

Value

An object of class `sigProb` containing three elements:

`predicted` data frame of predicted probabilities. The first column of this data frame is called `Row`, which corresponds to the rows in either `model` or `par`. In the event of multiple equilibria, this column allows for mapping data and parameters to all computed equilibria.

`model` data frame of covariates used to produce the predicted probabilities.

`par` data frame of parameters used to produce the predicted probabilities.

See Also

`plot.sigProb`, `predict.sigfit`

Examples

```
## An example with one covariate
ftest1 <-  ~ 0 | #SA
             1 | #VA
             0 | #CB
             1 | #barWA
             x1 | #barWB
             1| #bara
             1 #VB

theta <- data.frame(VA = 1, barWA = -1.9, barWB = -2.9,
                    barWB1 = 0.1, bara = -1.2, VB = 1)
data <- data.frame(x1 = seq(from = -1,to = 2, length.out = 101))
test <- generate.eq(ftest1, data = data, theta = theta)
plot(test, prob = "pr")

## An example with all constants
ftest2 <-  ~ 0 | #SA
             1 | #VA
             0 | #CB
             1 | #barWA
             1 | #barWB
```

```
                1 | #bara
                1 #VB

theta <- data.frame(VA = 1, barWA = -1.9,
                    barWB = seq(-2.9, -2.2, length.out = 15),
                    bara = -1.2,
                    VB = 1)
test <- generate.eq(ftest2, theta = theta)
plot(test, prob = "pr")
```

`plot.sigProb` *Plot predicted probabilities from a* `sigProb` *object.*

Description

This method takes a `sigProb` object produced by `predict.sigfit` and plots the comparative static(s) of interest.

Usage

```
## S3 method for class 'sigProb'
plot(x, prob, xvar, main = "", ylab, xlab, col = "blue", pch = 16, ...)
```

Arguments

`x`	an object of class `sigProb`, which is obtained by using `predict.sigfit` on a model fit using `sigint`.
`prob`	A string providing the column name for the column of `object$predicted` should be used as the outcome or probability of interest.
`xvar`	A string providing the column name of the column (from either object$model or object$par) that provides the "x-variable" in the plot.
`main`	The title of the plot
`ylab`	The y-axis label
`xlab`	The x-axis label
`col`	The color of the plot
`pch`	An integer or character used to choose the type of points used in the plot
`...`	Additional arguments and graphical parameters used by `plot` and `par`

See Also

`predict.sigfit` `generate.eq` `plot` `par`

Examples

```
data(sanctionsData)

f1 <- sq+cd+sf+bd ~ sqrt(senderecondep) + senderdemocracy + contig + ally -1|#SA
                    anticipatedsendercosts|#VA
                    sqrt(targetecondep) + anticipatedtargetcosts + contig + ally|#CB
                    sqrt(senderecondep) + senderdemocracy + lncaprat | #barWA
                    targetdemocracy + lncaprat| #barWB
                    senderdemocracy| #bara
                    -1#VB
## Outcome probabilities for first five using NPL probabilities
Phat <- list(PRhat=sanctionsData$PRnpl, PFhat=sanctionsData$PFnpl)
fit2 <- sigint(f1, data=sanctionsData, method="pl", phat=Phat)

## comparative static on \bar{a}, compute more precise equilibria with uniroot
new.theta <- data.frame(t(replicate(25, coef(fit2))))
new.theta[,19] <- seq(-6, 0, length=25)
pout <- predict(fit2, newdata=sanctionsData[93,], new.theta=new.theta,
                control=list(gridsize=500))


plot(pout, prob="pc", ylab="Pr Challenge", xlab="Audience Costs")
```

`predict.sigfit` *Predicted probabilities and comparative statics for signaling games*

Description

This method uses a fitted model of class `sigfit` to compute predicted probabilities or comparative statics. Users can provide either new data or a new parameters to generate counterfactuals of interest.

Usage

```
## S3 method for class 'sigfit'
predict(
  object,
  newdata,
  new.theta,
  type = c("actions", "outcomes"),
  na.action = na.pass,
  control = list(),
  parallel = FALSE,
  ...
)
```

Arguments

`object`	a fitted model of class `sigfit`.
`newdata`	data frame of covariates used to produce the predicted probabilities. If this is left empty, the entire original data set is used. When `newdata` is specified, `new.theta` should be either missing (use the coefficients from the `sigfit` object) or a one row data frame. See "Details" for more information. As with other `predict` methods, variable names must match those used to fit the model.
`new.theta`	a data frame of alternative parameters for comparative statics. When missing, the coefficients from the `object` are used. When specified, each row should be a complete parameter vector. If `new.theta` is specified, then `newdata` must be a data frame with only one row. Unlike other `predict` methods, column names do not matter here. Instead, the columns must be the same order as the coefficients in `object`. See "Details" and "Examples" for more information.
`type`	whether to provide probabilities over actions (default, returns p_C, p_R, and p_F) or outcomes (returns SQ, CD, SF, and BD).
`na.action`	how to deal with `NA`s in `newdata`.
`control`	list of options describing the grid search method. See "Details" for more information
`parallel`	logical. Should the comparative statics be computed in parallel, requires the `parallel` package be installed. Parallelization is done using `parSapply`.
`...`	Additional arguments (not currently used)

Details

This function is used to consider comparative statics in the crisis signaling game. The model of interest is fit using `sigint`. How this function behaves largely depends on how `newdata` and `new.theta` are specified.

When both `newdata` and `new.theta` are missing, all equilibria for every observation used to fit the model are computed. These equilibria are then used to calculated either choice probabilities (default, `type = "action"`) the distribution over outcomes (`type = "outcomes"`).

When only `newdata` is specified, then all equilibria are computed using the data frame in `newdata` and the coefficients from `object`. This produces standard comparative statics with respect to observed covariates.

When `newdata` is specified and `new.theta` is a one row data frame, then all equilibria are computed using the data frame in `newdata` and the coefficients from `new.theta`.

When `newdata` is a one row data frame and `new.theta` is specified, then all equilibria are computed using the data frame in `newdata` and the coefficients from `new.theta`. This is a comparative static on changing a structural parameter in the model.

If `new.theta` has more than one row, then `newdata` must be specified as a data frame with only one row. Anything else returns an error.

Equilibria are computed using a line search method. The `control` argument allows for user control over this process. Users can specify a list with the following named elements

gridsize Integer. The number of points considered in the line search (default, 1e4). More points makes it more likely that all equilibria are discovered, but can slow down the search.

comp Logical. Should an equilibrium be computed when discovered? When `comp = FALSE` (default), the mean of the grid points surrounding the equilibrium is used as an approximate solution. When `comp = TRUE` the function `uniroot` is called to find a more precise solution to the equilibrium constraint problem.

tol Numeric. When `comp = TRUE`, this is the tolerance used by `uniroot`.

When dealing with a larger problem, such as computing all equilibria for every observation, it can be helpful to parallelize process. If the user has the (suggested) `parallel` package, then the option `parallel = TRUE` will use the function `parSapply` is used.

Value

An object of class `sigProb` containing three elements:

`predicted` data frame of predicted probabilities. The first column of this data frame is called `Row`, which corresponds to the rows in either `model` or `par`. In the event of multiple equilibria, this column allows for mapping data and parameters to all computed equilibria.

`model` data frame of covariates used to produce the predicted probabilities.

`par` data frame of parameters used to produce the predicted probabilities.

See Also

`plot.sigProb`, `generate.eq`

Examples

```
data(sanctionsData)

f1 <- sq+cd+sf+bd ~ sqrt(senderecondep) + senderdemocracy + contig + ally -1|#SA
                    anticipatedsendercosts|#VA
                    sqrt(targetecondep) + anticipatedtargetcosts + contig + ally|#CB
                    sqrt(senderecondep) + senderdemocracy + lncaprat | #barWA
                    targetdemocracy + lncaprat| #barWB
                    senderdemocracy| #bara
                    -1#VB
## Using Nested-Pseudo Likelihood  with default first stage
## Not run:
fit1 <- sigint(f1, data=sanctionsData, npl.trace=TRUE)
p.out <- predict(fit2, parallel=TRUE) #fitted choice probabilites for all observations

## End(Not run)

## Outcome probabilities for first five using PL method
Phat <- list(PRhat=sanctionsData$PRhat, PFhat=sanctionsData$PFhat)
fit2 <- sigint(f1, data=sanctionsData, method="pl", phat=Phat)
p1 <- predict(fit2, newdata=sanctionsData[1:5,], type="outcome")

## comparative static on \bar{a}, compute more precise equilibria with uniroot
new.theta <- data.frame(t(replicate(25, coef(fit2))))
new.theta[,19] <- seq(-6, 0, length=25)
p2 <- predict(fit2, newdata=sanctionsData[1,], new.theta=new.theta, control=list(comp=TRUE))
```

print.summary.sigfit *Print the summary table for a* sigfit *object.*

Description

Prints the summary regression table for a model fitted with sigint.

Usage

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

Arguments

x	a summary.sigfit object.
...	Additional arguments (not currently used)

Details

Prints the standard regression results table from a fitted strategic model, along with the log-likelihood and number of games used in estimation.

sanctionsData *Economic Sanctions Threats and Outcomes*

Description

Dataset on economic sanctions threats and outcomes from 1970-2000

Usage

```
data(sanctionsData)
```

Details

These data were compiled using the Threat and Imposition of Sanctions (TIES), data project (Morgan, Bapat, and Kobayashi 2014), with additional data from the Correlates of War (COW), and Polity IV datasets. See Crisman-Cox and Gibilisco (2018) for more information. The unit of observation is the dyad-decade, and the variables are:

gameID	A dyad-decade identifier composed of COW country codes and the decade observed.
dyadID	A dyad identifier composed of COW country codes
tenyear	The observed decade

`code1` Challenger's COW code

`code2` Target's COW code

`sq` The number of status quo observations in this dyad decade

`cd` The number of times that the game ends with Challenge-Concede (Outcome CD)

`sf` The number of times that the game ends with Challenge-Resist -Stand Firm (Outcome SF)

`bd` The number of times that the game ends with Challenge-Resist-Back Down (Outcome BD)

`senderecondep` Challenger's economic dependence (dyadic trade / Challenger's GDP per capita)

`senderdemocracy` Challenger's Polity score

`contig` Contiguity between states

`ally` Are these state allied? (indicator)

`anticipatedsendercosts` The Challenger's anticipated costs for enacting sanctions

`anticipatedtargetcosts` The Target's anticipated costs for being sanctions

`targetecondep` Target's economic dependence (dyadic trade / Target's GDP per capita)

`lncaprat` Ratio of the Challenger's military capability to the Target's (logged)

`targetdemocracy` Target's Polity score

`PRhat` Estimated probability that the Target resists a challenge (fit using a random forest)

`PFhat` Estimated probability that the Challenger stands firm given that it challenged (fit using a random forest)

`PRnpl` Estimated probability that the Target resists a challenge (taken from the last stage of NPL iteration)

`PFnpl` Estimated probability that the Challenger stands firm given that it challenged (taken from the last stage of NPL iteration)

References

Barbieri, Katherine, Omar M. G. Keshk, and Brian Pollins. 2009. "TRADING DATA: Evaluating our Assumptions and Coding Rules." Conflict Management and Peace Science. 26(5): 471-491.

Casey Crisman-Cox and Michael Gibilisco. 2018. "Estimating Signaling Games in International Relations: Problems and Solutions." Unpublished Manuscript.

Gibler, Douglas M. 2009. International military alliances, 1648-2008. CQ Press.

Marshall, Monty G., and Keith Jaggers. 2013. "Polity IV Project." `http://www.systemicpeace.org/polity/polity4.htm`.

Morgan, T. Clifton, Navin Bapat, and Yoshi Kobayashi. 2014. "The Threat and Imposition of Sanctions: Updating the TIES dataset." Conflict Management and Peace Science 31(5): 541-558.

Singer, J. David, Stuart Bremer, and John Stuckey. 1972. "Capability Distribution, Uncertainty, and Major Power War, 1820-1965." in Bruce Russett (ed) Peace, War, and Numbers, Beverly Hills: Sage, 19-48.

Stinnett, Douglas M., Jaroslav Tir, Philip Schafer, Paul F. Diehl, and Charles Gochman. 2002. "The Correlates of War Project Direct Contiguity Data, Version 3." Conflict Management and Peace Science 19(2):58-66.

See Also

`sigint`

sigint *Estimating the parameters of the canonical discrete crisis bargaining game.*

Description

This function fits the Lewis and Schultz (2003) model to data using either the pseudo-likelihood (PL) or nested-pseudo likelihood (NPL) method from Crisman-Cox and Gibilisco (2018). Throughout, we refer to the data as containing D games, where each game is observed one or more times.

Usage

```
sigint(
  formulas,
  data,
  subset,
  na.action,
  fixed.par = list(),
  method = c("npl", "pl"),
  npl.maxit = 25,
  npl.tol = 1e-07,
  npl.trace = FALSE,
  start.beta,
  maxlik.method = "NR",
  phat,
  phat.formulas,
  pl.vcov = FALSE,
  phat.vcov,
  seed = 12345,
  maxlik.options = list()
)
```

Arguments

`formulas`	a `Formula` object four variables on the left-hand side and seven (7) separate right-hand sides. See "Details" and examples below.
`data`	a data frame containing the variables used to fit the model. Each row of the data frame describes an individual game $d = 1, 2, ..., D$. Each row d should be a summary of all of the within-game observations for game d. See "Details" for more information.
`subset`	an optional logical expression to specify a subset of observations to be used in fitting the model.
`na.action`	how do deal with missing data (`NA`s). Defaults to the `na.action` setting of `options` (typically `na.omit`).

`fixed.par`	a list with up to seven (7) named elements for normalizing payoffs to non-zero values. Names must match a payoff name as listed in "Details." Each named element should contain a single number that is the fixed (not estimated) value of that payoff. For example, to fix each side's victory-without-fighting payoff to 1 use `fixed.par=list(VA=1, VB=1)` and set their portions of the `formulas` to zero. To normalize a payoff to zero, you only need to specify it has a zero in the `formulas`.
`method`	whether to use the nested-pseudo-likelihood (`"npl"`, default) or the pseudo-likelihood method for fitting the model. See "Details" for more information.
`npl.maxit`	maximum number of outer-loop iterations to be used when fitting the NPL. See "Details" for more information.
`npl.tol`	Convergence criteria for the NPL. When the estimates change by less than this amount, convergence is considered successful.
`npl.trace`	logical. Should the NPL's progress be printed to screen?
`start.beta`	starting values for the model coefficients as a single vector. If missing, random values are drawn from a normal distribution with mean zero and standard deviation 0.05.
`maxlik.method`	method used by `maxLik` to fit the model. Default is Newton-Raphson (`"NR"`). See `maxLik` for additional details. At this time only `"NR"`, `"BFGS"`, and `"Nelder-Mead"` are available.
`phat`	a list containing two vectors: `PRhat` and `PFhat`. These are the first-stage estimates that B resists a threat and that A follows through on a threat, respectively. If missing, they will be estimated by a `randomForest` with default options. See "Details" for more information.
`phat.formulas`	if `phat` are missing, you can supply formulas to estimate them. Should be a Formulas object containing no left-hand side and 1-2 right hand sides. If one right-hand side is given, the same covariates are used to estimate both `PRhat` and `PFhat`. Otherwise, the first RHS is used to generate `PRhat`, while the second RHS generates `PFhat`. If no formulas are provided and `phat` is missing, all the covariates used in formulas argument and used here. See "Details" for more information.
`pl.vcov`	number of bootstrap iterations to generate `phat.vcov`. If less than `0` or `FALSE` (default), the pseudo-likelihood covariance is not estimated. Only used if `method = "pl"`.
`phat.vcov`	a covariance matrix for the estimates `PRhat` and `PFhat`. If missing and `pl.vcov = TRUE` and `phat` is missing, it will be estimated by bootstrapping the random forest used to fit `phat`.
`seed`	integer. Used to set the seed for the random forest and for drawing the the starting values. The PL can be sensitive to starting value, so this makes results reproducible. The NPL is less sensitive, but we always recommend checking the first order conditions.
`maxlik.options`	a list of options to be passed to `maxLik` for fitting the model.

Details

The model corresponds to an extensive-form, discrete-crisis-bargaining game from Lewis and Schultz (2003):

```
.       A
.      / \
.     /   \
.    /     \
.   S_A     B
.    0     / \
.         /   \
.        /     \
.      V_A      A
.      C_B     / \
.             /   \
.            /     \
.     W_A + e_A    a + e_a
.     W_B + e_B    V_B
```

If A chooses not to challenge B, then the game ends at the leftmost node (SQ) and payoffs are S_A and 0 to players A and B, respectively. If A challenges B, B can concede or resist. If B concedes, the game ends at CD with payoffs V_A and C_B. However, if B resists, A decides to stand firm, which ends the game at SF with payoffs $W_A + \epsilon_A$ and $W_B + \epsilon_B$. Finally, if A decides to back down in the face of B's resistance, then the game ends at the rightmost node BD, with payoffs $a + \epsilon_a$ and V_B.

The seven right-hand formulas that are specified in the formula argument correspond to the regressors to be placed in $S_A, V_A, C_B, W_A, W_B, a$, and V_B, respectively. The model is unidentified if any regressor (including a constant term) is included in all the formulas for each player (Lewis and Schultz 2003). Often the easiest way to meet this requirement is set one formula per player to 0. When an identification problem is detected, an error is issued. For example, the syntax for the formula argument could be:

`formulas = sq + cd + sf + bd ~ x1 + 0 | x2 | x2 | x1 + x2 | x1 | 1 | 0)`

Where:

- `sq + cd + sf + bd` are the tallies of how many times each outcome is observed for each observation. When the game is only observed once, that observation will be a 1 and three 0s. When the game is observed multiple times, these variables should count the number of times each outcome is observed. They need to be in the order of SQ, CD, SF, BD.
- S_A is a function of the variable `x1` and no constant term.
- V_A is a function of the variable `x2` and a constant term.
- C_B is a function of the variable `x2` and a constant term.
- W_A is a function of the variables `x1`, `x2` and a constant term.
- W_B is a function of the variable `x1` and a constant term.
- a is a constant term.
- V_B is fixed to 0 (or a non-zero value set by `fixed.par`.

Each row of the data frame should be a summary of the covariates and outcomes associated with that particular game. When each game is observed only once, then this will resemble an ordinary dyad-time data frame. However, if there are multiple observations per game, then each row should be a summary of all the data associated with that game. For example, if there are D games in the data, where each is observed T_d times, then the data frame should have D rows. The four columns making up the dependent variable will denote the frequencies of each outcome for game d, such that $\texttt{sq}_d + \texttt{cd}_d + \texttt{sf}_d + \texttt{bd}_d = T_d$. The covariates in row d should be summary statistics for the exogenous variables (e.g., mean, median, mode, first observation).

The model is first fit using a pseudo-likelihood estimator. This approach requires first stage estimation of the probability that B resists and the probability that A fights conditional on B choosing to resist. These first stage estimates should be flexible and we recommend that users fit a flexible semi-parametric or non-parametric model to produce them. If these estimates are produced by the analyst prior to using this function, then they can be provided by providing a list to the `phat` argument. This list should contain two named elements

- `PRhat` is the probability that B resists. This should be a vector of probabilities with one estimated probability for each observation.
- `PFhat` is the probability that A stands firm conditional on B resisting. This should be a vector of probabilities with one estimated probability for each observation.

If the user leaves the `phat` argument empty, then these first-stage estimates are produced internally using the `randomForest` function. Users wanting to use the random forest, can supply a formula for it using the argument `phat.formulas`. This argument can take a formula with nothing on the left-hand side and 1-2 right-hand sides. If two right-hand sides are provided then the first is used to generate `PRhat`, and the second is used for `PFhat`. If only one right-hand side is provided, it is used for both. Some examples:

- `phat.formulas = ~ x1 + x2` predict `PRhat` and `PFhat` using `x1` and `x2`.
- `phat.formulas = ~ x1 + x2 | x1 + x2` predict `PRhat` and `PFhat` using `x1` and `x2`
- `phat.formulas = ~ x1 + x2 | x1` predict `PRhat` using `x1` and `x2`, but predict `PFhat` using only `x1`.

If both `phat` and `phat.formula` are missing, then a random forest is fit using all the exogenous variables listed in the formulas argument

If `method = "npl"`, then estimation continues. For each iteration of the NPL, the estimates of `PRhat` and `PFhat` are updated by one best-response iteration using the current parameter estimates. The model is then refit using these updated choice probabilities. This process continues until the maximum absolute change in parameters and choice probabilities is less than `npl.tol` (default, `1e-7`), or the number of outer iterations exceeds `npl.maxit` (default, `25`). In the latter case, a warning is produced.

If pseudo-likelihood (`method="pl"`) is used, then `pl.vcov` is checked. There are four possibilities here:

- `pl.vcov = FALSE` (default), then no covariance matrix or standard errors are returned, only the point estimates.
- `pl.vcov > 0` and `phat.vcov` is supplied, then `phat.vcov` is used to estimate the PL's covariance matrix.

- `pl.vcov` > `0`, `phat.vcov` is missing, and `phat` is missing, then the random forest used to estimate `PRhat` and `PFhat` is bootstrapped (simple, nonparametric bootstrap) `pl.vcov` times.
- `pl.vcov` > `0`, `phat.vcov` is missing, and `phat` is not missing, then an error is returned.

Value

An object of class `sigfit`, containing:

`coefficients` A vector of estimated model parameters.

`vcov` Estimated variance-covariance matrix. When `pl.vcov = FALSE`, this slot is omitted.

`utilities` Each actor's utilities at the estimated values.

`fixed.par` The fixed utilities if specified in the call.

`logLik` Final log-likelihood value of the model.

`gradient` First derivative values at the estimated parameters.

`Phat` List of two elements

- `PRhat` The first stage estimates of the probability that B resists (`method = "pl"`) or the final estimates that B resists (if `method = "npl"`)
- `PFhat` The first stage estimates of the probability that A stands firms given that A challenged (`method = "pl"`) or the final estimates that A stands firms given that A challenged (if `method = "npl"`)

Note that `PRhat` will only be an equilibrium if `method = "npl"` and the NPL convergences

`user.phat` Logical. Did the user provide phat?

`start.beta` The vector of starting values used in the PL optimization.

`call` The call used to produce the object.

`model` The data frame used to fit the model.

`method` The method (`"pl"` or `"npl"`) used to fit the model.

`maxlik.method` The optimization used by `maxLik` to fit the model.

`maxlik.code` The convergence code returned by `maxLik`.

`maxlik.message` The convergence message returned by `maxLik`.

Additionally, when `method = "npl"`, the following are also included in the `sigfit` object.

`npl.iter` Number of best response iterations used in fitting the NPL.

`npl.eval` Maximum difference between the parameters at the last two NPL iterations. If the NPL method converged, this should be less than `npl.tol` specified in the function call.

`eq.constraint` Maximum equilibrium constraint violation.

References

Casey Crisman-Cox and Michael Gibilisco. 2019. "Estimating Signaling Games in International Relations: Problems and Solutions." *Political Science Research and Methods*. Online First.

Jeffrey B. Lewis and Kenneth A. Schultz. 2003. "Revealing Preferences: Empirical Estimation of a Crisis Bargaining Game with Incomplete Information." *Political Analysis* 11:345–367.

Examples

```
data("sanctionsData")
f1 <- sq+cd+sf+bd ~ sqrt(senderecondep) + senderdemocracy + contig + ally -1|#SA
                    anticipatedsendercosts|#VA
                    sqrt(targetecondep) + anticipatedtargetcosts + contig + ally|#CB
                    sqrt(senderecondep) + senderdemocracy + lncaprat | #barWA
                    targetdemocracy + lncaprat| #barWB
                    senderdemocracy| #bara
                    -1#VB

## Using Nested-Pseudo Likelihood  with default first stage
## Not run:
fit1 <- sigint(f1, data=sanctionsData, npl.trace=TRUE)
summary(fit1)

## End(Not run)


## Using Pseudo Likelihood with user supplied first stage
Phat <- list(PRhat=sanctionsData$PRnpl, PFhat=sanctionsData$PFnpl)
fit2 <- sigint(f1, data=sanctionsData, method="pl", phat=Phat)
summary(fit2)

## Using Pseudo Likelihood with user made first stage and user covariance
## SIGMA is a bootstrapped first-stage covariance matrix (not provided)
## Not run:
fit3 <- sigint(f1, data=sanctionsData, method="pl", phat=Phat, phat.vcov=SIGMA, pl.vcov=TRUE)
summary(fit3)

## End(Not run)

## Using Pseudo Likelihood with default first stage and
## bootstrapped standard errors for the first stage covariance
## Not run:
fit4 <- sigint(f1, data=sanctionsData, method="pl", pl.vcov=25)
summary(fit4)

## End(Not run)
```

`summary.sigfit` *Summarize a fitted crisis signaling model*

Description

The default method for summarizing a `sigfit` object.

Usage

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

Arguments

`object`	a fitted model of class `sigfit`
`vcov`	a substitute variance covariance matrix
`...`	Additional arguments (not currently used)

Details

Forms a block regression results table from fitted crisis signaling model.

Value

An object of class `summary.sigfit`. This object contains the information needed to print the summary

See Also

`print.summary.sigfit`.

`toLatexTable` *Export a* `sigfit` *object into paper-ready LaTeX table*

Description

This method converts one or more fitted models of class `sigfit` into a publication-ready LaTeX table. This conversion is performed by reformatting the models into a format that is fed to `xtable`, which generates the actual LaTeX code.

Usage

```
toLatexTable(
  ...,
  se.list,
  stars = c("default", "all", "none"),
  caption = "",
  label,
  align,
  digits = 2,
  se.note = "Standard errors in parenthesis",
  order,
  covariate.labels,
  model.names,
  dep.varnames,
  k = 1,
  print.xtable.options = list()
)
```

Arguments

`...`	one or more models fit using `sigint`.
`se.list`	an optional list where each element contains the standard errors of the models. If included this list must include one element per model, even if that model's standard errors are unchanged. This argument should be used when standard errors have been adjusted outside the model (e.g., a bootstrap). If left empty the standard errors from the model's covariance matrix are used if available. Within each element of this list, the order of standard errors must be the same order as the coefficients in the model. Standard errors are not matched on name.
`stars`	how should significance stars be used? `stars = "default"` returns a single star when $p < 0.05$, `stars = "all"` returns flags for $p < 0.1$, $p < 0.05$, and $p < 0.01$, and `stars = "none"` returns no stars at all.
`caption`	a string to be used as the table's caption.
`label`	a string to be used as the table's LaTeX `label` argument.
`align`	a string to indicate the alignment of each column in the table. Passed directly to the LaTeX `tabular` options.
`digits`	how many digits after the decimal point should be displayed? Default 2.
`se.note`	a string containing a note for the bottom of the table. Default is the common "Standard errors in parenthesis.
`order`	a string vector describing the order that the covariates should be in the table. The default is to use the order they're listed in the `sigfit` object. When this vector is shorter than the coefficient vector, variables are first included by `order`, remaining variables are included based on their order in the model
`covariate.labels`	a string vector of "nice" names for the variables appropriate for a published work. If empty, the "ugly" names from the fitted model are used. Note that if `order` is specified then the covariate labels must match the order in `order`.
`model.names`	an optional vector of model names to include as column titles in the table. Should be either a single title or one title per model (repetition is allowed). If only one title is given, that is centered over the table.
`dep.varnames`	an optional vector to describe the dependent variable in the models. Can be either a single variable name or one per model (repetition is allowed).
`k`	an integer to start the counter for model numbers in the table.
`print.xtable.options`	a list of options for `print.xtable`.

Details

This function produces a ready-to-use LaTeX table for `sigfit` objects. Each column is its own model, along with model-based information. The generation of LaTeX code is done by `xtable` and so additional printing options can be passed via `print.xtable.options`. For a full list of these options see `print.xtable`.

See Also

`xtable`, `print.xtable`

Examples

```
data("sanctionsData")
f1 <- sq+cd+sf+bd ~ sqrt(senderecondep) + senderdemocracy + contig + ally -1|#SA
                    anticipatedsendercosts|#VA
                    sqrt(targetecondep) + anticipatedtargetcosts + contig + ally|#CB
                    sqrt(senderecondep) + senderdemocracy + lncaprat | #barWA
                    targetdemocracy + lncaprat| #barWB
                    senderdemocracy| #bara
                    -1#VB
## Using Nested-Pseudo Likelihood  with default first stage
## Not run:
fit1 <- sigint(f1, data=sanctionsData, npl.trace=TRUE)

## End(Not run)

## Using Pseudo Likelihood with user made first stage
Phat <- list(PRhat=sanctionsData$PRhat, PFhat=sanctionsData$PFhat)
fit2 <- sigint(f1, data=sanctionsData, method="pl", phat=Phat)

## Using Pseudo Likelihood with default first stage and bootstrapped standard errors
## Not run:
fit3 <- sigint(f1, data=sanctionsData, method="pl", pl.vcov=25)

## End(Not run)

## Simple regression table
toLatexTable(fit2)

## More options: multiple models and user supplied standard errors
## Not run:
toLatexTable(fit1, fit2, fit3,
        se.list=list(sqrt(diag(vcov(fit1))),
                     sqrt(diag(vcov(fit3))),
                     sqrt(diag(vcov(fit3)))),
        stars="all",
        caption = "Economic Sanctions",
        label = "tab:sanctions",
        model.names = c("NPL", "PL", "PL"))

## End(Not run)

## Not run:
## More options, from print.xtable including printing to a file
toLatexTable(fit1, fit2, fit3,
        caption = "Economic Sanctions",
        label = "tab:sanctions",
        model.names = c("NPL", "PL", "PL"),
        print.xtable.options=list(file="myTable.tex",
                                  booktabs=TRUE))

## End(Not run)
```

Index