Package 'LSDirf'

January 20, 2025

Type Package

Title Impulse-Response Function Analysis for Agent-Based Models

Version 0.1.3

Date 2024-4-4

Description
Performing impulse-response function (IRF) analysis of relevant variables of agent-based simulation models, in particular for models described in 'LSD' format. Based on the data produced by the simulation model, it performs both linear and state-dependent IRF analysis, providing the tools required by the Counterfactual Monte Carlo (CMC) methodology (Amendola and Pereira (2024) <doi:10.2139/ssrn.4740360>), including state identification and sensitivity. CMC proposes retrieving the causal effect of shocks by exploiting the opportunity to directly observe the counterfactual in a fully controlled experimental setup. 'LSD' (Laboratory for Simulation Development) is free software available at <https://www.labsimdev.org/>).

Depends R (>= 4.0.0)

Imports stats, utils, grDevices, graphics, boot, digest, gplots, abind, partykit, randomForest

Suggests LSDinterface, LSDsensitivity

License GPL-3

Language en-US

Encoding UTF-8

NeedsCompilation no

Author Marcelo C. Pereira [aut, cre] (<https://orcid.org/0000-0002-8069-2734>), Marco Amendola [aut] (<https://orcid.org/0000-0003-3056-5558>)

Maintainer Marcelo C. Pereira <mcper@unicamp.br>

Repository CRAN

Date/Publication 2024-04-04 17:53:02 UTC

Contents

`LSDirf-package` *Impulse-Response Function Analysis for Agent-Based Models*

Description

Performing impulse-response function (IRF) analysis of relevant variables of agent-based simulation models, in particular for models described in 'LSD' format. Based on the data produced by the simulation model, it performs both linear and state-dependent IRF analysis, providing the tools required by the Counterfactual Monte Carlo (CMC) methodology (Amendola and Pereira (2024) <doi:10.2139/ssrn.4740360>), including state identification and sensitivity. CMC proposes retrieving the causal effect of shocks by exploiting the opportunity to directly observe the counterfactual in a fully controlled experimental setup. 'LSD' (Laboratory for Simulation Development) is free software available at <https://www.labsimdev.org/>).

Details

The Counterfactual Monte Carlo (**CMC**) methodology (see note below) is based on the analysis of samples of seed-specific impulse-response functions (IRF's) and cumulative impulse-response functions (CIRF's) of size N. These samples are highly informative about the effects of a shock affecting a simulation running in LSD, with several statistics of interest that can be computed from them.

In particular, assuming that the mean or the median is chosen as the metric to synthesize information included in these samples, robust IRF and CIRF may be obtained by properly combining the N seed-specific IRF's and CIRF's across the different time horizons. These measures represent the mean/median dynamic effect of a designated shock. Confidence intervals can be obtained by bootstrap, thus allowing the analysis of the uncertainty around these effects.

The CMC methodology allows going beyond the linear effects. Eventual state-dependent effects of the shock can be investigated starting from the IRF and CIRF samples, exploiting the heterogeneity in the simulated system conditions of different runs of the CMC experiment.

In particular, in line with the threshold local projections models adopted in several empirical analyses (e.g., Ramey and Zubairy, 2018), the runs of the CMC experiment can be split into alternative states by comparing the value of one or more *state variable* with a (some) specific threshold(s), computed from the realizations of selected variables' time series in the periods before the shock. As in the case of linear estimates, the confidence intervals around these impulse responses can be constructed via bootstrap, which in this case are also very useful to visually assess the significance of any differences in the impulse responses between alternative states. Several standard statistical tests, such as the t-test or the Mann-Whitney U test, can then be applied to better investigate the significance of state-dependent results.

Such state-dependent analysis can be potentially conducted in two ways. The first is by testing the results against a set of relevant and distinctive system states *known* to the researcher (e.g., Auerbach

and Gorodnichenko, 2013). The second takes the alternative approach: instead of testing whether specific states significantly impact the effect of the shock, try to find such states from simulated data. A similar target, for example, is at the heart of the recent literature on *optimal policy*, which goal is to find the optimal allocation of the treatment across heterogeneous units (e.g., Kitagawa and Tetenov, 2018; Athey and Wager, 2021). We offer a data-driven heuristic to this aim that helps discover such states. It is named Random Forest State Identification Algorithm (RFSIA) as it adapts the random forest machine learning technique to our goal.

The main intuition behind the RFSIA is to use a random forest classifier to obtain a set of *meaningful* data stratifications to test for state dependency. More precisely, the idea is to test the state dependency in the final nodes of the regression trees produced by the algorithm and then recombine and rearrange this extremely detailed information to obtain a more general sense of which states have a significant impact on the effect of the shock. In particular, to make the output more understandable and bring out the more evident state patterns, the last step of RFSIA is the quantile discretization of the system states. To this aim, we divide each state variable into deciles and replace the threshold values entering each state, and grouping of similar states.

More details on the methodology, and a comprehensive application to a full LSD simulation model, can be found in Amendola and Pereira (2024).

Note

The **CMC** methodology proposes retrieving the causal effect of shocks by directly exploiting the opportunity to observe the counterfactual in a fully controlled experimental setup. Indeed, counterfactuals emerge naturally in agent-based models if two simulation runs characterized by the same values of the parameters, the same initial conditions, and the same seed of the pseudo-random number generator (PRNG) are considered, and a single shock is introduced in one of them. In these specific circumstances, the non-shocked realization acts as a direct counterfactual for the shocked one, and any difference between the shocked and non-shocked output time series can be directly traced back to the shock.

Building on this insight, the effects of shocks in an agent-based model can be rigorously studied based on the following procedure, which allows collecting a sample of size N of the (dynamic) unit treatment effect:

1. Defining a shock generating rule ($sgr \in \{0, 1\}$), which precisely defines the shock to be tested, that is, which and how model variable(s) is(are) shocked.
2. Running two simulation runs with the same values of parameters, initial conditions, and the PRNG seed but enabling the shock ($sgr = 1$) in just one of them.
3. Computing the difference in the time series of interest between the two scenarios (counterfactual: $sgr = 0$; shocked scenario: $sgr = 1$). In particular, two main metrics can be used to quantify the effects of the shock, namely the impulse-response function (IRF) and the cumulative impulse-response function (CIRF). Depending on the specific variable of interest, impulse response functions can be computed in percentage or absolute terms.
4. Repeating steps 2 and 3 N times by varying the seed of the PRNG, i.e. running an extensive Monte Carlo experiment.

The last step is crucial to robustly evaluate the effect of the shock in an agent-based model. Indeed, in these models, impulse responses are expected to vary by varying the seed. Two explanations for the cross-run variability are possible if the shock size is homogeneous across runs under different

seeds. The first one lies in the seed-specific after-shock realizations of the stochastic parts of the model, which is expected to impact the propagation of shocks in the model unpredictably. This implies that the transmission of shocks is inevitably affected by some randomness in most agent-based models. The second reason is instead related to the possibility that, in agent-based models, the effects of shocks may depend on the state of the system, i.e., the propagation of the shocks may be *structurally* amplified or weakened depending on the prevailing conditions. As the state of the system is generally different in each run of the CMC experiment (due to the seed-specific pre-shock realizations of the stochastic parts of the model), this *structural effect* is expected to vary between seeds.

As a consequence of the two cross-run variability channels, the results of a single MC experiment based on a specific seed are not informative enough. Instead, an extensive Monte Carlo experiment in needed, which allows for uncovering the distributional properties of such variability and the study of the effects of shocks in a robust way.

Author(s)

Marcelo C. Pereira [aut, cre] (<https://orcid.org/0000-0002-8069-2734>), Marco Amendola [aut] (<https://orcid.org/0000-0003-3056-5558>)

Maintainer: Marcelo C. Pereira <mcper@unicamp.br>

References

LSD documentation is available at <https://www.labsimdev.org/>

The latest LSD binaries and source code can be downloaded at <https://github.com/SantannaKS/Lsd/>.

Amendola, M., Pereira, M. C. (2024) *Linear and state-dependent impulse responses in agent-based models: a new methodology and an economic application*. SSRN pre-print <doi:10.2139/ssrn.4740360>. Available at <https://www.ssrn.com/abstract=4740360>.

Athey S., Wager, S. (2021) *Policy learning with observational data*. Econometrica 89(1):133-161

Auerbach A., Gorodnichenko, Y. (2013) *Fiscal multipliers in recession and expansion*. Fiscal Policy After the Financial Crisis, edited by Alberto Alesina and Francesco Giavazzi. University of Chicago Press

Kitagawa, T., Tetenov, A. (2018) *Who should be treated? empirical welfare maximization methods for treatment choice*. Econometrica 86(2):591-616

Ramey, V., Zubairy, S. (2018) *Government spending multipliers in good times and in bad: evidence from us historical data*. Journal of political economy 126(2):850-901

See Also

LSDinterface-package, LSDsensitivity-package

`irf.lsd` *Impulse-response function analysis*

Description

This function performs a (linear) impulse-response function (IRF) analysis on the data produced by a Monte Carlo experiment, typically from (but not restricted to) a LSD simulation model.

Usage

```
irf.lsd( data, data.shock, t.horiz, var.irf, var.shock, var.ref = NULL,
         irf.type = c( "incr.irf", "cum.irf", "peak.mult", "cum.mult", "none" ),
         stat = c( "mean", "median" ), ci.R = 999,
         ci.type = c( "basic", "perc", "bca" ),
         lim.outl = 0, alpha = 0.05, seed = 1, ... )
```

Arguments

`data`	numeric: a 3-dimensional array containing data from Monte Carlo (MC) simulation samples where the impulse (shock/treatment) was not applied/occurred. The array must have dimensions ordered as time steps x variables x MC samples. This format is automatically produced by read.3d.lsd but using it is not required. The second array dimension (variables) must be named with the names of the variables used in the analysis. The absolute minimum array dimensions are 2x1x2.
`data.shock`	numeric: a 3-dimensional array similar to `data` but containing data from MC samples where the impulse (shock/treatment) was applied/occurred. The array must have as dimensions: time steps x variables x MC samples. This can produced by read.3d.lsd, but this is not required. The first two dimensions must be similar between `data` and `data.shock`, containing the same time steps and variable names
`t.horiz`	integer: a positive value indicating the post-impulse time span to consider for analysis.
`var.irf`	string: the variable name on which perform the analysis. It must be one of the column names in `data`.
`var.shock`	string: the name of the variable containing the impulse/shock. It must be one of the column names in `data.shock`.
`var.ref`	string: the name of a reference variable to scale down (divide) the shock variable. The default is to do no scaling. If provided, it must be one of the column names in `data`. The special value `"%"` represents the same variable as the shock one, but using the corresponding non-shock values, so effectively standardizing the shock values.
`irf.type`	string: one of five options (`"incr.irf"`, `"cum.irf"`, `"peak.mult"`, `"cum.mult"`, or `"none"`) defining the type of function plot to be produced: incremental impulse-response, cumulative impulse-response, peak multiplier, cumulative multiplier,

or no plot. It also selects the type of function to be used when printing reports (response or multiplier functions). The default is `"incr.irf"`. This option only affect the plot, not the produced data, as all function values are always computed and saved in the output object.

`stat` string: one of `"mean"` or `"median"`, representing the Monte Carlo statistic used to compare samples. It also selects the corresponding deviation measure, between the standard deviation (SD) or the median absolute deviation (MAD). The default is to use the mean and SD.

`ci.R` integer: number of bootstrap replicates when computing the bootstrap confidence interval.

`ci.type` string: the type of bootstrap confidence interval to compute, must be one of `"basic"` (the default), `"perc"` (percentile interval), or `"bca"` (BCa - adjusted percentile interval).

`lim.outl` numeric: a positive outlier threshold limit multiple, applied over the distance between the first and the fourth quartiles of the Monte Carlo data. Outlier samples below or above the computed thresholds are removed from the analysis. The default (`0`), is to keep all samples (no outlier removal).

`alpha` numeric: a value between 0 and 0.5, defining the desired statistical significance level to be adopted in the analysis. The default is 0.05 (5%).

`seed` integer: a value defining the initial state of the pseudo-random number generator.

`...` additional parameters to configure printing and plotting.

Details

As a dynamic system, a simulation model may have its outputs analyzed when a brief input signal (an impulse or "shock") is applied to one of its inputs.

The function operates over `data` from multiple realizations of a Monte Carlo experiment.

Value

It returns an object of class `irf.lsd`, which has `print`- and `plot`-specific methods for presenting the analysis results. This object contains several items:

`irf` numeric: vector of length `t.horiz + 1` containing the average or median (according to `stat`) incremental impulse response function data.

`cirf` numeric: vector of length `t.horiz + 1` containing the average or median (according to `stat`) cumulative impulse response function data.

`pmf` numeric: vector of length `t.horiz + 1` containing the average or median (according to `stat`) peak-multiplier function data. The peak impulse-multiplier function evaluates the analysis variable (`var.irf`) as a multiple of the impulse variable (`var.shock`) at the shock start time.

`cmf` numeric: vector of length `t.horiz + 1` containing the average or median (according to `stat`) cumulative impulse-multiplier function data. The cumulative-multiplier function evaluates the accumulated analysis variable (`var.irf`), from impulse time, as a multiple of the accumulated impulse variable (`var.shock`) over the time horizon (`t.horiz`).

irf.ci.lo	numeric: vector of length t.horiz + 1 containing the confidence interval lower limit of incremental impulse response function data.
irf.ci.hi	numeric: vector of length t.horiz + 1 containing the confidence interval upper limit of incremental impulse response function data.
cirf.ci.lo	numeric: vector of length t.horiz + 1 containing the confidence interval lower limit of cumulative impulse response function data.
cirf.ci.hi	numeric: vector of length t.horiz + 1 containing the confidence interval upper limit of cumulative impulse response function data.
pmf.ci.lo	numeric: vector of length t.horiz + 1 containing the confidence interval lower limit of peak multiplier function data.
pmf.ci.hi	numeric: vector of length t.horiz + 1 containing the confidence interval upper limit of peak multiplier function data.
cmf.ci.lo	numeric: vector of length t.horiz + 1 containing the confidence interval lower limit of cumulative multiplier function data.
cmf.ci.hi	numeric: vector of length t.horiz + 1 containing the confidence interval upper limit of cumulative multiplier function data.
irf.ylim	numeric: vector of length two containing the absolute minimum and maximum values for the incremental impulse response function data.
cirf.ylim	numeric: vector of length two containing the absolute minimum and maximum values for the cumulative impulse response function data.
pmf.ylim	numeric: vector of length two containing the absolute minimum and maximum values for the peak multiplier function data.
cmf.ylim	numeric: vector of length two containing the absolute minimum and maximum values for the cumulative multiplier function data.
ir	numeric: data frame with nsample (see below) rows and t.horiz + 1 columns. The data frame is the set of individual incremental impulse responses from each Monte Carlo (MC) sample.
cir	numeric: data frame with nsample (see below) rows and t.horiz + 1 columns. The data frame is the set of individual cumulative impulse responses from each MC sample.
pm	numeric: data frame with nsample (see below) rows and t.horiz + 1 columns. The data frame is the set of individual peak multipliers from each MC sample.
cm	numeric: data frame with nsample (see below) rows and t.horiz + 1 columns. The data frame is the set of individual cumulative multipliers from each MC sample.
t.shock	numeric: vector of length equal to nsample (see below), containing the time the shock was detected in each MC sample.
t.horiz	integer: the time horizon used in the analysis (same as the t.horiz argument).
var.irf	character: the name of the variable used in the impulse-response analysis (same as the var.irf argument).
var.shock	character: the name of the shock variable used in the analysis (same as the var.shock argument).

`var.ref`	character: the name of the scale-reference variable used in the analysis (same as the `var.ref` argument).
`stat`	character: the Monte Carlo statistic used in the analysis (same as the `stat` argument).
`alpha`	numeric: the statistical significance level used in the analysis (same as the `alpha` argument).
`nsample`	integer: the effective number of of Monte Carlo (MC) samples effectively used for deriving the response function, after the removal of outliers if `lim.outl` > `0`.
`outliers`	integer: vector containing the number of each MC sample considered an outlier, and so removed from the analysis, or an empty vector if no outlier was excluded. The MC numbers are the indexes to the third dimension of `data`.
`data.crc`	character: an hexadecimal sting containing the 32-bit Cyclic Redundancy Check (CRC32) for the `data` used in the analysis.
`call`	character: the command line used to call the function.

Note

See the note in LSDirf-package for an methodological overview and for instructions on how to perform the (linear) impulse-response function analysis.

Author(s)

Marcelo C. Pereira [aut, cre] (<https://orcid.org/0000-0002-8069-2734>), Marco Amendola [aut] (<https://orcid.org/0000-0003-3056-5558>)

See Also

`state.irf.lsd`, `read.3d.lsd`, `read.4d.lsd`,

Examples

```
# Example data generation: Y is an AR(1) process that may receive a shock at
# t=50, S is the shock (0/1), a combination of 3 AR(1) processes (X1-X3)
# X4 is another AR(1) process, uncorrelated with S, X4sq is just X4^2
# All AR(1) processes have the same phi=0.98 coefficient, and are Monte
# Carlo sampled 500 times
set.seed( 1 )   # make results reproducible
# LSD-like arrays to store simulated time series (t x var x MC)
dataNoShock <- dataShock <-array ( 0, dim = c( 60, 7, 500 ) )
colnames( dataNoShock ) <- colnames( dataShock ) <-
  c( "Y", "S", "X1", "X2", "X3", "X4", "X4sq" )
# Monte Carlo sampling
for( n in 1 : 500 ) {
  # simulation time
  for( t in 2 : 60 ) {
    # AR process on X vars
    for( v in c( "X1", "X2", "X3", "X4" ) ) {
      dataNoShock[ t, v, n ] = dataShock[ t, v, n ] =
        0.98 * dataShock[ t - 1, v, n ] + rnorm( 1, 0, 0.1 )
```

```
    }
    # apply shock once
    if( t == 50 ) {
      dataShock[ t, "S", n ] <- 1
      shockEff <- 0.4 + 0.7 * isTRUE( dataShock[ t, "X1", n ] > 0.1 ) -
        0.4 * isTRUE( dataShock[ t, "X2", n ] > 0.1 ) +
        0.2 * isTRUE( dataShock[ t, "X3", n ] > 0.05 ) + rnorm( 1, 0, 0.2 )
    } else
      shockEff <- 0
    # AR process on Y var
    rs <- rnorm( 1, 0, 0.1 )
    dataNoShock[ t, "Y", n ] = 0.98 * dataNoShock[ t - 1, "Y", n ] + rs
    dataShock[ t, "Y", n ] = 0.98 * dataShock[ t - 1, "Y", n ] + shockEff + rs
  }
}
# another uncorrelated var
dataNoShock[ , "X4sq", ] <- dataShock[ , "X4sq", ] <- dataShock[ , "X4", ] ^ 2

# linear IRF analysis
linearIRF <- irf.lsd( data = dataNoShock,     # non-shocked MC data
                      data.shock = dataShock, # shocked data
                      t.horiz = 10,           # post-shock analysis time horizon
                      var.irf = "Y",          # variable to compute IRF
                      var.shock = "S" )       # shock variable (impulse)

plot( linearIRF, irf.type = "cum.irf" )       # cumulative IRF plot

print( linearIRF )                            # show IRF data
```

`state.ident.lsd` *IRF state Identification*

Description

This function implements the Random Forest Identification Algorithm (RFSIA) on the data produced by a Monte Carlo experiment, typically from (but not restricted to) a LSD simulation model. It exploits the random forest regression technique to obtain a series of "meaningful" stratifications of the data on which state-dependence is then tested.

Usage

```
state.ident.lsd( data, irf, state.vars = NULL, metr.irf = NULL,
                 add.vars = NULL, state.cont = FALSE,
                 ntree = 500, maxdepth = 1, nodesize  = 5,
                 mtry = max( floor( ifelse( ! is.null( state.vars ),
                                            length( state.vars ),
                                            dim( data )[ 2 ] ) / 3 ),
                             1 ),
                 quantile = 10, alpha = 0.05, seed = 1, ... )
```

Arguments

`data`	numeric: a 3-dimensional array containing data from Monte Carlo (MC) simulation samples where the impulse (shock/treatment) was not applied/occurred. The array must have dimensions ordered as time steps x variables x MC samples. This format is automatically produced by read.3d.lsd but using it is not required. The second array dimension (variables) must be named with the names of the variables used in the analysis. The absolute minimum array dimensions are 2x1x2.
`irf`	object: an object produced by a previous run of `irf.lsd` over the same dataset (as defined by `data`).
`state.vars`	character: a vector of variable names to consider as state variables.
`metr.irf`	function: a function that assigns a metric to compare each run of a Monte Carlo experiment, to be used on regressions. The function must take a cumulative impulse-response matrix, organized as runs on rows and response times (0, 1, ...,`t.horiz`) on columns. It must return a numeric vector of length equal to the number of runs, defining the metric associated with each run. Higher metric values correspond to increased impulse effect. If no function is supplied (`NULL`), the default, the mean of state variable value(s) from impulse time (`t=0`) until the time horizon (`t=t.horiz`) is used as metric.
`add.vars`	function: an optional function to add new variables to the MC dataset, before the analysis is performed. The function must take a single Monte Carlo run data frame, organized as time on rows and (original) variables on columns. It must return this data frame with new column(s) added, one per each new variable.
`state.cont`	logical: if `TRUE`, the resulting object will contain the full list of continuous states produced during the analysis. If `FALSE`, the default, the list of continuous states is not saved.
`ntree`	integer: number of trees to grow. This number should not be set to too small values, to ensure that every possible state gets predicted at least a few times.
`maxdepth`	integer: maximum depth of the trees to consider. The default (1) represents the shortest possible trees.
`nodesize`	integer: minimum number of associated data observations to a node be considered in the analysis.
`mtry`	integer: number of state variables randomly sampled as candidates at each node for the random forest algorithm. The default is to use one third of the number of considered state variables.
`quantile`	integer: number of quantiles to consider when discretizing states.
`alpha`	numeric: a value between 0 and 0.5, defining the desired statistical significance level to be adopted in the analysis. The default is 0.05 (5%).
`seed`	integer: a value defining the initial state of the pseudo-random number generator.
`...`	additional parameters to configure printing and plotting.

Details

As a dynamic system, a simulation model may have its outputs analyzed when a brief input signal (an impulse or "shock") is applied to one of its inputs. In particular, the effect of the shock may be

correlated to some system-specific state, in which it may be amplified or attenuated. This function allows for the identification of possible relevant states, that is, states which are both probable and distinguishable among them.

The function operates over `data` from multiple realizations of a Monte Carlo experiment, and a previous (linear) impulse-response function analysis (`irf`) performed by `irf.lsd`.

Value

It returns an object of class `state.ident.lsd`, which has a `print`-specific method for presenting the analysis results. This object contains several items:

`state.freq`	data frame: each row represents one of the identified discrete states, ordered in decreasing frequency. First column (`State`) identifies the state textually, in terms of state variable values in terms of the quantiles (as defined by `quantile` argument). Second column (`Prob`) lists the frequency of the state among the random forest sample used. Third column (`MetrD`) brings the mean/median (according to `stat` in `irf.lsd`) relative metric of the state. Fourth column (`MetrAD`) presents the mean/median of the absolute deviations relative to the state metric. The next columns, in groups of four, bring the mean/median threshold quantile, its standard deviation or variance absolute deviation (MAD), and absolute minimum and maximum. These groups repeat for each state variable considered in the respective identified state.
`state.vars`	character: a vector of variable names effectively available as state variables.
`t.horiz`	integer: the time horizon used in the analysis (same as the `t.horiz` argument in `irf.lsd`).
`var.irf`	character: the name of the variable used in the impulse-response analysis (same as the `var.irf` argument in `irf.lsd`).
`var.ref`	character: the name of the scale-reference variable used in the analysis (same as the `var.ref` argument in `irf.lsd`).
`stat`	character: the Monte Carlo statistic used in the analysis (same as the `stat` argument in `irf.lsd`).
`alpha`	numeric: the statistical significance level used in the analysis (same as the `alpha` argument).
`nsample`	integer: the effective number of of Monte Carlo (MC) samples effectively used for deriving the response function, after the removal of outliers if `lim.outl > 0` in `irf.lsd`.
`outliers`	integer: vector containing the number of each MC sample considered an outlier, and so removed from the analysis in `irf.lsd`, or an empty vector if no outlier was excluded. The MC numbers are the indexes to the third dimension of `data`.
`ntree`	integer: number of trees grown (same as `ntree` argument).
`maxdepth`	integer: maximum depth of the trees considered (same as `maxdepth` argument).
`nodesize`	integer: minimum number of data observations in a node considered (same as `nodesize` argument).
`mtry`	integer: number of state variables sampled per node (same as `mtry` argument).

`quantile`	integer: number of quantiles used for discretizing states (same as `quantile` argument).
`state.cont`	data frame: each row represents one of the identified continuous states, ordered by the absolute effect on the metric. Columns are organized in groups of three: state variable name (`VarN`), relation code (`RelN`), and split threshold (`VarN`). There is one column group per variable included in the corresponding state. After all column groups, there is a final column presenting the metric deviation (from non-shocked response) of each identified state.
`state.cont.num`	integer: the total number of continuous states identified.
`call`	character: the command line used to call the function.

Note

See the note in LSDirf-package for an methodological overview and for instructions on how to perform the state-dependent impulse-response function analysis.

Author(s)

Marcelo C. Pereira [aut, cre] (<https://orcid.org/0000-0002-8069-2734>), Marco Amendola [aut] (<https://orcid.org/0000-0003-3056-5558>)

See Also

`irf.lsd`, `read.3d.lsd`, `read.4d.lsd`,

Examples

```
# Example data generation: Y is an AR(1) process that may receive a shock at
# t=50, S is the shock (0/1), a combination of 3 AR(1) processes (X1-X3)
# X4 is another AR(1) process, uncorrelated with S, X4sq is just X4^2
# All AR(1) processes have the same phi=0.98 coefficient, and are Monte
# Carlo sampled 500 times
set.seed( 1 )   # make results reproducible
# LSD-like arrays to store simulated time series (t x var x MC)
dataNoShock <- dataShock <-array ( 0, dim = c( 60, 7, 500 ) )
colnames( dataNoShock ) <- colnames( dataShock ) <-
  c( "Y", "S", "X1", "X2", "X3", "X4", "X4sq" )
# Monte Carlo sampling
for( n in 1 : 500 ) {
  # simulation time
  for( t in 2 : 60 ) {
    # AR process on X vars
    for( v in c( "X1", "X2", "X3", "X4" ) ) {
      dataNoShock[ t, v, n ] = dataShock[ t, v, n ] =
        0.98 * dataShock[ t - 1, v, n ] + rnorm( 1, 0, 0.1 )
    }
    # apply shock once
    if( t == 50 ) {
      dataShock[ t, "S", n ] <- 1
      shockEff <- 0.4 + 0.7 * isTRUE( dataShock[ t, "X1", n ] > 0.1 ) -
```

```
        0.4 * isTRUE( dataShock[ t, "X2", n ] > 0.1 ) +
        0.2 * isTRUE( dataShock[ t, "X3", n ] > 0.05 ) + rnorm( 1, 0, 0.2 )
    } else
      shockEff <- 0
    # AR process on Y var
    rs <- rnorm( 1, 0, 0.1 )
    dataNoShock[ t, "Y", n ] = 0.98 * dataNoShock[ t - 1, "Y", n ] + rs
    dataShock[ t, "Y", n ] = 0.98 * dataShock[ t - 1, "Y", n ] + shockEff + rs
  }
}
# another uncorrelated var
dataNoShock[ , "X4sq", ] <- dataShock[ , "X4sq", ] <- dataShock[ , "X4", ] ^ 2

# linear IRF analysis
linearIRF <- irf.lsd( data = dataNoShock,       # non-shocked MC data
                      data.shock = dataShock,   # shocked data
                      t.horiz = 10,             # post-shock analysis t horizon
                      var.irf = "Y",            # variable to compute IRF
                      var.shock = "S",          # shock variable (impulse)
                      irf.type = "none" )       # no plot of linear IRF

# Random-forest state identification
stateId <- state.ident.lsd( data = dataNoShock, # non-shocked MC data
                            irf = linearIRF,    # linear IRF produced by irf.lsd
                            state.vars = c( "X1", "X2", "X3", "X4", "X4sq" ),
                                                # state variables to consider
                            mtry = 3 )          # number of samples per node

print( stateId )                                # show identification data

# state-dependent IRF analysis for most frequent state identified
stateIRF <- state.irf.lsd( data = dataNoShock,  # non-shocked MC data
                           irf = linearIRF,     # linear IRF produced by irf.lsd
                           states = stateId )   # object with identified states

plot( stateIRF, irf.type = "cum.irf" )          # cumulative IRF plot

print( stateIRF )                               # show IRF data
```

`state.irf.lsd` *State-dependent impulse-response function analysis*

Description

This function performs a state-dependent impulse-response function (IRF) analysis on the data produced by a Monte Carlo experiment, typically from (but not restricted to) a LSD simulation model.

Usage

```
state.irf.lsd( data, irf, states = NULL, state.num = 1,
               state.vars = NULL, eval.state = NULL,
               metr.irf = NULL, add.vars = NULL,
               irf.type = c( "incr.irf", "cum.irf", "peak.mult",
                             "cum.mult", "none" ),
               state.plot = 0, ci.R = 999,
               ci.type = c( "basic", "perc", "bca" ),
               alpha = 0.05, seed = 1, ... )
```

Arguments

`data`	numeric: a 3-dimensional array containing data from Monte Carlo (MC) simulation samples where the impulse (shock/treatment) was not applied/occurred. The array must have dimensions ordered as time steps x variables x MC samples. This format is automatically produced by read.3d.lsd but using it is not required. The second array dimension (variables) must be named with the names of the variables used in the analysis. The absolute minimum array dimensions are 2x1x2.
`irf`	object: an object produced by a previous run of `irf.lsd` over the same dataset (as defined by `data`).
`states`	object: an optional object produced by a previous run of `state.ident.lsd` containing a set of identified state candidates. Only one state candidate, as defined by `state.num` is used in each run.
`state.num`	integer: the index (1,2,...) of the state candidate in `states` to use. The default is to use the first state, which is usually the most likely one.
`state.vars`	character: a vector of variable names to use as state variables. If more than one name is provided, a proper `eval.state` state evaluation function supporting multi-variable states must be also provided.
`eval.state`	function: a function able to define the corresponding state of each run of a Monte Carlo experiment. The function must take a matrix as argument, organized as runs on rows and the state variable(s) on columns. It must return an integer vector of length equal to the number of runs, defining the state of each run. States are defined by a sequence of integer values, e.g., 0, 1 or 1, 2 ,3, The minimum number of states is two and there is no maximum. If no function is supplied (`NULL`), the default, an internal 2-state evaluation function is used, using the mean or the median, according to `stat`, to split the MC set into two states.
`metr.irf`	function: a function that assigns a metric to compare each run of a Monte Carlo experiment, to be used on regressions. The function must take a cumulative impulse-response matrix, organized as runs on rows and response times (0, 1, ...,`t.horiz`) on columns. It must return a numeric vector of length equal to the number of runs, defining the metric associated with each run. Higher metric values correspond to increased impulse effect. If no function is supplied (`NULL`), the default, the mean of state variable value(s) from impulse time (`t=0`) until the time horizon (`t=t.horiz`) is used as metric.

add.vars	function: an optional function to add new variables to the MC dataset, before the analysis is performed. The function must take a single Monte Carlo run data frame, organized as time on rows and (original) variables on columns. It must return this data frame with new column(s) added, one per each new variable.
irf.type	string: one of five options ("incr.irf", "cum.irf", "peak.mult", "cum.mult", or "none") defining the type of function plot to be produced: incremental impulse-response, cumulative impulse-response, peak multiplier, cumulative multiplier, or no plot. It also selects the type of function to be used when printing reports (response or multiplier functions). The default is "incr.irf". This option only affect the plot, not the produced data, as all function values are always computed and saved in the output object.
state.plot	integer: the relative position (1,2,...) of the state (as defined by eval.state) which data is to be used on plots. The default (0) is to plot data from all states, which allows comparing the state responses.
ci.R	integer: number of bootstrap replicates when computing the bootstrap confidence interval.
ci.type	string: the type of bootstrap confidence interval to compute, must be one of "basic" (the default), "perc" (percentile interval), or "bca" (BCa - adjusted percentile interval).
alpha	numeric: a value between 0 and 0.5, defining the desired statistical significance level to be adopted in the analysis. The default is 0.05 (5%).
seed	integer: a value defining the initial state of the pseudo-random number generator.
...	additional parameters to configure printing and plotting.

Details

As a dynamic system, a simulation model may have its outputs analyzed when a brief input signal (an impulse or "shock") is applied to one of its inputs. In particular, the effect of the shock may be correlated to some system-specific state, in which it may be amplified or attenuated. This function allows for the investigation of such differentiated effects, given an objective criterion to split the system status (i.e., the model outputs) in two or more states.

The function operates over data from multiple realizations of a Monte Carlo experiment, and a previous (linear) impulse-response function analysis (irf) performed by irf.lsd.

Value

It returns an object of class state.irf.lsd, which has print- and plot-specific methods for presenting the analysis results. This object contains several items:

irf.state	list: each list element is a vector of length t.horiz + 1 containing the average or median (according to stat) incremental impulse response function data for each identified state.
cirf.state	list: each list element is a vector of length t.horiz + 1 containing the average or median (according to stat) cumulative impulse response function data for each identified state.

`pmf.state` list: each list element is a vector of length `t.horiz + 1` containing the average or median (according to `stat`) peak impulse-multiplier function data for each identified state. The peak impulse-multiplier function evaluates the analysis variable (`var.irf`) as a multiple of the impulse variable (`var.shock`) at the shock start time.

`cmf.state` list: each list element is a vector of length `t.horiz + 1` containing the average or median (according to `stat`) cumulative impulse-multiplier function data for each identified state. The cumulative-multiplier function evaluates the accumulated analysis variable (`var.irf`), from impulse time, as a multiple of the accumulated impulse variable (`var.shock`) over the time horizon (`t.horiz`)

`irf.state.ci.lo` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval lower limit of incremental impulse response function data for each identified state.

`irf.state.ci.hi` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval upper limit of incremental impulse response function data for each identified state.

`cirf.state.ci.lo` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval lower limit of cumulative impulse response function data for each identified state.

`cirf.state.ci.hi` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval upper limit of cumulative impulse response function data for each identified state.

`pmf.state.ci.lo` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval lower limit of peak multiplier function data for each identified state.

`pmf.state.ci.hi` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval upper limit of peak multiplier function data for each identified state.

`cmf.state.ci.lo` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval lower limit of cumulative multiplier function data for each identified state.

`cmf.state.ci.hi` list: each list element is a vector of length `t.horiz + 1` containing the confidence interval upper limit of cumulative multiplier function data for each identified state.

`irf.state.ylim` list: each list element is a vector of length two containing the absolute minimum and maximum values for the incremental impulse response function data for each identified state.

`cirf.state.ylim` list: each list element is a vector of length two containing the absolute minimum and maximum values for the cumulative impulse response function data for each identified state.

`pmf.state.ylim` list: each list element is a vector of length two containing the absolute minimum and maximum values for the peak multiplier function data for each identified state.

`cmf.state.ylim` list: each list element is a vector of length two containing the absolute minimum and maximum values for the cumulative multiplier function data for each identified state.

`irf.test` object: the result of the test comparing the statistical significance of the incremental impulse-response function difference among different states. Two-state setups are evaluated with t or U tests, according to `stat` in `irf.lsd`, or H or F tests otherwise.

`cirf.test` object: the result of the test comparing the statistical significance of the cumulative impulse-response function difference among different states, considering the entire period of analysis (1, ..., `t.horiz`). Two-state setups are evaluated with t or U tests, according to `stat` in `irf.lsd`, or H or F tests otherwise.

`cirf.test.t.horiz` object: the result of the test comparing the statistical significance of the cumulative impulse-response function difference among different states just at the end of the analysis time horizon (`t.horiz`). Two-state setups are evaluated with t or U tests, according to `stat` in `irf.lsd`, or H or F tests otherwise.

`pmf.test` object: the result of the test comparing the statistical significance of the peak multiplier function difference among different states. Two-state setups are evaluated with t or U tests, according to `stat` in `irf.lsd`, or H or F tests otherwise.

`cmf.test` object: the result of the test comparing the statistical significance of the cumulative multiplier function difference among different states. Two-state setups are evaluated with t or U tests, according to `stat` in `irf.lsd`, or H or F tests otherwise.

`state` character: a textual description of the tested state.

`state.vars` character: a vector of variable names effectively available as state variables.

`t.horiz` integer: the time horizon used in the analysis (same as the `t.horiz` argument in `irf.lsd`).

`var.irf` character: the name of the variable used in the impulse-response analysis (same as the `var.irf` argument in `irf.lsd`).

`var.ref` character: the name of the scale-reference variable used in the analysis (same as the `var.ref` argument in `irf.lsd`).

`stat` character: the Monte Carlo statistic used in the analysis (same as the `stat` argument in `irf.lsd`).

`alpha` numeric: the statistical significance level used in the analysis (same as the `alpha` argument in `irf.lsd`).

`nsample` integer: the effective number of of Monte Carlo (MC) samples effectively used for deriving the response function, after the removal of outliers if `lim.outl > 0` in `irf.lsd`.

`outliers` integer: vector containing the number of each MC sample considered an outlier, and so removed from the analysis in `irf.lsd`, or an empty vector if no outlier was excluded. The MC numbers are the indexes to the third dimension of `data`.

`call` character: the command line used to call the function.

Note

See the note in LSDirf-package for an methodological overview and for instructions on how to perform the state-dependent impulse-response function analysis.

Author(s)

Marcelo C. Pereira [aut, cre] (<https://orcid.org/0000-0002-8069-2734>), Marco Amendola [aut] (<https://orcid.org/0000-0003-3056-5558>)

See Also

irf.lsd, read.3d.lsd, read.4d.lsd,

Examples

```
# Example data generation: Y is an AR(1) process that may receive a shock at
# t=50, S is the shock (0/1), a combination of 3 AR(1) processes (X1-X3)
# X4 is another AR(1) process, uncorrelated with S, X4sq is just X4^2
# All AR(1) processes have the same phi=0.98 coefficient, and are Monte
# Carlo sampled 500 times
set.seed( 1 )   # make results reproducible
# LSD-like arrays to store simulated time series (t x var x MC)
dataNoShock <- dataShock <-array ( 0, dim = c( 60, 7, 500 ) )
colnames( dataNoShock ) <- colnames( dataShock ) <-
  c( "Y", "S", "X1", "X2", "X3", "X4", "X4sq" )
# Monte Carlo sampling
for( n in 1 : 500 ) {
  # simulation time
  for( t in 2 : 60 ) {
    # AR process on X vars
    for( v in c( "X1", "X2", "X3", "X4" ) ) {
      dataNoShock[ t, v, n ] = dataShock[ t, v, n ] =
        0.98 * dataShock[ t - 1, v, n ] + rnorm( 1, 0, 0.1 )
    }
    # apply shock once
    if( t == 50 ) {
      dataShock[ t, "S", n ] <- 1
      shockEff <- 0.4 + 0.7 * isTRUE( dataShock[ t, "X1", n ] > 0.1 ) -
        0.4 * isTRUE( dataShock[ t, "X2", n ] > 0.1 ) +
        0.2 * isTRUE( dataShock[ t, "X3", n ] > 0.05 ) + rnorm( 1, 0, 0.2 )
    } else
      shockEff <- 0
    # AR process on Y var
    rs <- rnorm( 1, 0, 0.1 )
    dataNoShock[ t, "Y", n ] = 0.98 * dataNoShock[ t - 1, "Y", n ] + rs
    dataShock[ t, "Y", n ] = 0.98 * dataShock[ t - 1, "Y", n ] + shockEff + rs
  }
}
# another uncorrelated var
dataNoShock[ , "X4sq", ] <- dataShock[ , "X4sq", ] <- dataShock[ , "X4", ] ^ 2

# linear IRF analysis
```

```
linearIRF <- irf.lsd( data = dataNoShock,      # non-shocked MC data
                      data.shock = dataShock,  # shocked data
                      t.horiz = 10,            # post-shock analysis t horizon
                      var.irf = "Y",           # variable to compute IRF
                      var.shock = "S",         # shock variable (impulse)
                      irf.type = "none" )      # no plot of linear IRF

# state-dependent IRF analysis
stateIRF <- state.irf.lsd( data = dataNoShock, # non-shocked MC data
                           irf = linearIRF,    # linear IRF produced by irf.lsd
                           state.vars = "X1" ) # variable defining states

plot( stateIRF, irf.type = "cum.irf" )         # cumulative IRF plot

print( stateIRF )                              # show IRF data
```

`state.sa.lsd` *Sensitivity analysis of IRF to state variables*

Description

This function performs a sensitivity analysis of the impulse-response function (IRF) to selected state variables of data from a Monte Carlo experiment, typically from (but not restricted to) a LSD simulation model.

Usage

```
state.sa.lsd( data, irf, state.vars = NULL, metr.irf = NULL,
              add.vars = NULL, ntree = 500, nodesize = 5,
              mtry = max( floor( ifelse( ! is.null( state.vars ),
                                         length( state.vars ),
                                         dim( data )[ 2 ] ) / 3 ),
                          1 ),
              no.plot = FALSE, alpha = 0.05, seed = 1, ... )
```

Arguments

`data`	numeric: a 3-dimensional array containing data from Monte Carlo (MC) simulation samples where the impulse (shock/treatment) was not applied/occurred. The array must have dimensions ordered as time steps x variables x MC samples. This format is automatically produced by read.3d.lsd but using it is not required. The second array dimension (variables) must be named with the names of the variables used in the analysis. The absolute minimum array dimensions are 2x1x2.
`irf`	object: an object produced by a previous run of `irf.lsd` over the same dataset (as defined by `data`).
`state.vars`	character: a vector of variable names to consider as state variables.

`metr.irf`	function: a function that assigns a metric to compare each run of a Monte Carlo experiment, to be used on regressions. The function must take a cumulative impulse-response matrix, organized as runs on rows and response times (0, 1, ...,`t.horiz`) on columns. It must return a numeric vector of length equal to the number of runs, defining the metric associated with each run. Higher metric values correspond to increased impulse effect. If no function is supplied (`NULL`), the default, the sum of state variable value(s) at impulse time is used as metric.
`add.vars`	function: an optional function to add new variables to the MC dataset, before the analysis is performed. The function must take a single Monte Carlo run data frame, organized as time on rows and (original) variables on columns. It must return this data frame with new column(s) added, one per each new variable.
`ntree`	integer: number of trees to grow. This number should not be set to too small values, to ensure that every possible state gets predicted at least a few times.
`nodesize`	integer: minimum number of associated data observations to a node be considered in the analysis.
`mtry`	integer: number of state variables randomly sampled as candidates at each node for the random forest algorithm. The default is to use one third of the number of considered state variables.
`no.plot`	logical: if `TRUE`, the default, a bar plot is presented with the results. If set to `FALSE`, the bar plot is not shown.
`alpha`	numeric: a value between 0 and 0.5, defining the desired statistical significance level to be adopted in the analysis. The default is 0.05 (5%).
`seed`	integer: a value defining the initial state of the pseudo-random number generator.
`...`	additional parameters to configure printing and plotting.

Details

As a dynamic system, a simulation model may have its outputs analyzed when a brief input signal (an impulse or "shock") is applied to one of its inputs. In particular, the effect of the shock may be correlated to some system-specific state, in which it may be amplified or attenuated, associated to specific model variables. This function evaluates how sensitive such states are to each of the specified variables.

The function operates over `data` from multiple realizations of a Monte Carlo experiment, and a previous (linear) impulse-response function analysis (`irf`) performed by `irf.lsd`.

Value

It returns an object of class `state.sa.lsd`, which has `print`- and `plot`-specific methods for presenting the analysis results. This object contains several items:

`importance`	data frame: contains the state variable importance measure (mean decrease in accuracy) produced by the random forest regression, one row for each state variable. First column presents the importance measure, second column brings the measure standard error, and third, the p-value of t test comparing the measure to zero.
`state.vars`	character: a vector of variable names effectively available as state variables.

t.horiz	integer: the time horizon used in the analysis (same as the t.horiz argument in irf.lsd).
var.irf	character: the name of the variable used in the impulse-response analysis (same as the var.irf argument in irf.lsd).
var.ref	character: the name of the scale-reference variable used in the analysis (same as the var.ref argument in irf.lsd).
stat	character: the Monte Carlo statistic used in the analysis (same as the stat argument in irf.lsd).
alpha	numeric: the statistical significance level used in the analysis (same as the alpha argument).
nsample	integer: the effective number of of Monte Carlo (MC) samples effectively used for deriving the response function, after the removal of outliers if lim.outl > 0 in irf.lsd.
outliers	integer: vector containing the number of each MC sample considered an outlier, and so removed from the analysis in irf.lsd, or an empty vector if no outlier was excluded. The MC numbers are the indexes to the third dimension of data.
ntree	integer: number of trees grown (same as ntree argument).
nodesize	integer: minimum number of data observations in a node considered (same as nodesize argument).
mtry	integer: number of state variables sampled per node (same as mtry argument).
rsq	numeric: the "pseudo R-squared" (1 - MSE / Var(y)) of the random forest regression.
call	character: the command line used to call the function.

Note

See the note in LSDirf-package for an methodological overview and for instructions on how to perform the state-dependent impulse-response function analysis.

Author(s)

Marcelo C. Pereira [aut, cre] (<https://orcid.org/0000-0002-8069-2734>), Marco Amendola [aut] (<https://orcid.org/0000-0003-3056-5558>)

See Also

irf.lsd, read.3d.lsd, read.4d.lsd,

Examples

```
# Example data generation: Y is an AR(1) process that may receive a shock at
# t=50, S is the shock (0/1), a combination of 3 AR(1) processes (X1-X3)
# X4 is another AR(1) process, uncorrelated with S, X4sq is just X4^2
# All AR(1) processes have the same phi=0.98 coefficient, and are Monte
# Carlo sampled 500 times
set.seed( 1 )   # make results reproducible
# LSD-like arrays to store simulated time series (t x var x MC)
```

```
dataNoShock <- dataShock <-array ( 0, dim = c( 60, 7, 500 ) )
colnames( dataNoShock ) <- colnames( dataShock ) <-
  c( "Y", "S", "X1", "X2", "X3", "X4", "X4sq" )
# Monte Carlo sampling
for( n in 1 : 500 ) {
  # simulation time
  for( t in 2 : 60 ) {
    # AR process on X vars
    for( v in c( "X1", "X2", "X3", "X4" ) ) {
      dataNoShock[ t, v, n ] = dataShock[ t, v, n ] =
        0.98 * dataShock[ t - 1, v, n ] + rnorm( 1, 0, 0.1 )
    }
    # apply shock once
    if( t == 50 ) {
      dataShock[ t, "S", n ] <- 1
      shockEff <- 0.4 + 0.7 * isTRUE( dataShock[ t, "X1", n ] > 0.1 ) -
        0.4 * isTRUE( dataShock[ t, "X2", n ] > 0.1 ) +
        0.2 * isTRUE( dataShock[ t, "X3", n ] > 0.05 ) + rnorm( 1, 0, 0.2 )
    } else
      shockEff <- 0
    # AR process on Y var
    rs <- rnorm( 1, 0, 0.1 )
    dataNoShock[ t, "Y", n ] = 0.98 * dataNoShock[ t - 1, "Y", n ] + rs
    dataShock[ t, "Y", n ] = 0.98 * dataShock[ t - 1, "Y", n ] + shockEff + rs
  }
}
# another uncorrelated var
dataNoShock[ , "X4sq", ] <- dataShock[ , "X4sq", ] <- dataShock[ , "X4", ] ^ 2

# linear IRF analysis
linearIRF <- irf.lsd( data = dataNoShock,       # non-shocked MC data
                      data.shock = dataShock,   # shocked data
                      t.horiz = 10,             # post-shock analysis t horizon
                      var.irf = "Y",            # variable to compute IRF
                      var.shock = "S",          # shock variable (impulse)
                      irf.type = "none" )       # no plot of linear IRF

# state-variable sensitivity
stateSens <- state.sa.lsd( data = dataNoShock,  # non-shocked MC data
                           irf = linearIRF,     # linear IRF produced by irf.lsd
                           state.vars = c( "X1", "X2", "X3", "X4", "X4sq" ),
                                                # state variables to consider
                           mtry = 3 )           # number of samples per node

print( stateSens )                              # show sensitivity data
```

Index