Package ‘ivmte’

September 17, 2021

Title Instrumental Variables: Extrapolation by Marginal Treatment Effects

Version 1.4.0

Maintainer Joshua Shea <jkcshea@uchicago.edu>

Description The marginal treatment effect was introduced by Heckman and Vytlacil (2005) <doi:10.1111/j.1468-0262.2005.00594.x> to provide a choice-theoretic interpretation to instrumental variables models that maintain the monotonicity condition of Imbens and Angrist (1994) <doi:10.2307/2951620>. This interpretation can be used to extrapolate from the compliers to estimate treatment effects for other subpopulations. This package provides a flexible set of methods for conducting this extrapolation. It allows for parametric or nonparametric sieve estimation, and allows the user to maintain shape restrictions such as monotonicity. The package operates in the general framework developed by Mogstad, Santos and Torgovitsky (2018) <doi:10.3982/ECTA15463>, and accommodates either point identification or partial identification (bounds). In the partially identified case, bounds are computed using either linear programming or quadratically constrained quadratic programming. Support for four solvers is provided. Gurobi and the Gurobi R API can be obtained from <http://www.gurobi.com/index>. CPLEX can be obtained from <https://www.ibm.com/analytics/cplex-optimizer>. CPLEX R APIs 'Rcplex' and 'cplexAPI' are available from CRAN. MOSEK and the MOSEK R API can be obtained from <https://www.mosek.com/>. The lp_solve library is freely available from <http://lpsolve.sourceforge.net/5.5/>, and is included when installing its API 'lpSolveAPI', which is available from CRAN.

Depends R (>= 3.6.0)

Imports Formula, methods, stats, utils

Suggests gurobi (>= 8.1-0), slam (>= 0.1-42), cplexAPI (>= 1.3.3), lpSolveAPI (>= 5.5.2.0-17), Rmosek (>= 9.2.38), testthat (>= 2.0.0), data.table (>= 1.12.0), splines2 (>= 0.2.8), future.apply (>= 1.6.0), future (>= 1.18.0), Matrix, knitr, rmarkdown, pander, AER, lsei, ggplot2, gridExtra

License GPL-2 | GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Alexander Torgovitsky [aut],
Joshua Shea [aut, cre]

Repository CRAN

Date/Publication 2021-09-17 12:20:07 UTC

R topics documented:

`AE` *Angrist Evans Data*

Description

Angrist Evans Data

Usage

```
AE
```

Format

A data frame with 209,133 rows and 8 columns.

worked indicator for whether worked in the previous year

hours weekly hours worked in the previous year

morekids indicator for having more than two children vs. exactly two children.

samesex indicator for the first two children having the same sex (male-male or female-female)

yob the year the woman was born

black indicator that mother is Black

hisp indicator that mother is Hispanic

other indicator that mother is neither Black nor Hispanic

Source

Derived from Angrist and Evans (1998, The American Economic Review).

`altDefSplinesBasis` *(Alternative) Defining single splines basis functions, with interactions*

Description

This function returns a numerically integrable function corresponding to a single splines basis function. It was not implemented because it was slower than using the function from the `splines2` package.

Usage

```
altDefSplinesBasis(splineslist, j, l, v = 1)
```

Arguments

- `splineslist` a list of splines commands and names of variables that interact with the splines. This is generated using the command `removeSplines`.
- `j` the index for the spline for which to generate the basis functions.
- `l` the index for the basis.
- `v` a constant that multiplies the spline basis.

Value

a vectorized function corresponding to a single splines basis function that can be numerically integrated.

`argstring` *Auxiliary function: extract arguments from function in string form*

Description

Auxiliary function to extract arguments from a function that is in string form.

Usage

```
argstring(string)
```

Arguments

- `string` the function in string form.

Value

string of arguments.

`audit` *Audit procedure*

Description

This is the wrapper for running the entire audit procedure. This function sets up the LP/QCQP problem of minimizing criterion. for the set of IV-like estimands, while satisfying boundedness and monotonicity constraints declared by the user. Rather than enforce that boundedness and monotonicity hold across the entire support of covariates and unobservables, this procedure enforces the conditions over a grid of points. This grid corresponds to the set of values the covariates can take, and a set of values of the unobservable term. The size of this grid is specified by the user in the function arguments. The procedure first estimates the bounds while imposing the shape constraints for an initial subset of points in the grid. The procedure then goes on to check ('audit') whether the constraints are satisfied over the entire grid. Any point where either the boundedness or monotonicity constraints are violated are incorporated into the initial grid, and the process is repeated until the audit no longer finds any violations, or until some maximum number of iterations is reached.

Usage

```
audit(
  data,
  uname,
  m0,
  m1,
  pm0,
  pm1,
  splinesobj,
  vars_mtr,
  terms_mtr0,
  terms_mtr1,
  vars_data,
  initgrid.nu = 20,
  initgrid.nx = 20,
  audit.nx = 2500,
  audit.nu = 25,
  audit.add = 100,
  audit.max = 25,
  audit.tol,
  audit.grid = NULL,
  m1.ub,
  m0.ub,
  m1.lb,
  m0.lb,
  mte.ub,
  mte.lb,
  m1.ub.default = FALSE,
  m0.ub.default = FALSE,
  mte.ub.default = FALSE,
  m1.lb.default = FALSE,
  m0.lb.default = FALSE,
  mte.lb.default = FALSE,
  m0.dec = FALSE,
  m0.inc = FALSE,
  m1.dec = FALSE,
  m1.inc = FALSE,
  mte.dec = FALSE,
  mte.inc = FALSE,
  equal.coef0,
  equal.coef1,
  sset,
  gstar0,
  gstar1,
  orig.sset = NULL,
  orig.criterion = NULL,
  criterion.tol = 1e-04,
  solver,
```

```
  solver.options,
  solver.presolve,
  solver.options.criterion,
  solver.options.bounds,
  rescale = TRUE,
  smallreturnlist = FALSE,
  noisy = TRUE,
  debug = FALSE
)
```

Arguments

- `data` — `data.frame` or `data.table` used to estimate the treatment effects.
- `uname` — variable name for the unobservable used in declaring the MTRs. The name can be provided with or without quotation marks.
- `m0` — one-sided formula for the marginal treatment response function for the control group. Splines may also be incorporated using the expression `uSpline`, e.g. `uSpline(degree = 2,knots = c(0.4,0.8),intercept = TRUE)`. The `intercept` argument may be omitted, and is set to `TRUE` by default.
- `m1` — one-sided formula for the marginal treatment response function for the treated group. See `m0` for details.
- `pm0` — A list of the monomials in the MTR for the control group.
- `pm1` — A list of the monomials in the MTR for the treated group.
- `splinesobj` — list of spline components in the MTRs for treated and control groups. Spline terms are extracted using `removeSplines`. This object is supposed to be a dictionary of splines, containing the original calls of each spline in the MTRs, their specifications, and the index used for naming each basis spline.
- `vars_mtr` — character, vector of variables entering into `m0` and `m1`.
- `terms_mtr0` — character, vector of terms entering into `m0`.
- `terms_mtr1` — character, vector of terms entering into `m1`.
- `vars_data` — character, vector of variables that can be found in the data.
- `initgrid.nu` — integer determining the number of points in the open interval (0, 1) drawn from a Halton sequence. The end points 0 and 1 are additionally included. These points are always a subset of the points defining the audit grid (see `audit.nu`). These points are used to form the initial constraint grid for imposing shape restrictions on the `u` components of the MTRs.
- `initgrid.nx` — integer determining the number of points of the covariates used to form the initial constraint grid for imposing shape restrictions on the MTRs.
- `audit.nx` — integer determining the number of points on the covariates space to audit in each iteration of the audit procedure.
- `audit.nu` — integer determining the number of points in the open interval (0, 1) drawn from a Halton sequence. The end points 0 and 1 are additionally included. These points are used to audit whether the shape restrictions on the `u` components of the MTRs are satisfied. The initial grid used to impose the shape constraints in the LP/QCQP problem are constructed from a subset of these points.

`audit.add`	maximum number of points to add to the initial constraint grid for imposing each kind of shape constraint. For example, if there are 5 different kinds of shape constraints, there can be at most `audit.add * 5` additional points added to the constraint grid.
`audit.max`	maximum number of iterations in the audit procedure.
`audit.tol`	feasibility tolerance when performing the audit. By default to set to be 1e-06, which is equal to the default feasibility tolerances of Gurobi (`solver = "gurobi"`), CPLEX (`solver = "cplexapi"`), and Rmosek (`solver = "rmosek"`). This parameter should only be changed if the feasibility tolerance of the solver is changed, or if numerical issues result in discrepancies between the solver's feasibility check and the audit.
`audit.grid`	list, contains the `A` matrix used in the audit for the original sample, as well as the RHS vector used in the audit from the original sample.
`m1.ub`	numeric value for upper bound on MTR for the treated group. By default, this will be set to the largest value of the observed outcome in the estimation sample.
`m0.ub`	numeric value for upper bound on MTR for the control group. By default, this will be set to the largest value of the observed outcome in the estimation sample.
`m1.lb`	numeric value for lower bound on MTR for the treated group. By default, this will be set to the smallest value of the observed outcome in the estimation sample.
`m0.lb`	numeric value for lower bound on MTR for the control group. By default, this will be set to the smallest value of the observed outcome in the estimation sample.
`mte.ub`	numeric value for upper bound on treatment effect parameter of interest.
`mte.lb`	numeric value for lower bound on treatment effect parameter of interest.
`m1.ub.default`	boolean, default set to FALSE. Indicator for whether the value assigned was by the user, or set by default.
`m0.ub.default`	boolean, default set to FALSE. Indicator for whether the value assigned was by the user, or set by default.
`mte.ub.default`	boolean, default set to FALSE. Indicator for whether the value assigned was by the user, or set by default.
`m1.lb.default`	boolean, default set to FALSE. Indicator for whether the value assigned was by the user, or set by default.
`m0.lb.default`	boolean, default set to FALSE. Indicator for whether the value assigned was by the user, or set by default.
`mte.lb.default`	boolean, default set to FALSE. Indicator for whether the value assigned was by the user, or set by default.
`m0.dec`	logical, set to `FALSE` by default. Set equal to `TRUE` if the MTR for the control group should be weakly monotone decreasing.
`m0.inc`	logical, set to `FALSE` by default. Set equal to `TRUE` if the MTR for the control group should be weakly monotone increasing.
`m1.dec`	logical, set to `FALSE` by default. Set equal to `TRUE` if the MTR for the treated group should be weakly monotone decreasing.

`m1.inc` logical, set to `FALSE` by default. Set equal to `TRUE` if the MTR for the treated group should be weakly monotone increasing.

`mte.dec` logical, set to `FALSE` by default. Set equal to `TRUE` if the MTE should be weakly monotone decreasing.

`mte.inc` logical, set to `FALSE` by default. Set equal to `TRUE` if the MTE should be weakly monotone increasing.

`equal.coef0` character, a vector containing all the terms in `m0` that should have the same coefficients in `m1`. The order of the variables must match those of `equal.coef1`, which contains all the corresponding terms in `m1`. The reason the terms are entered separately for `m0` and `m1` is because the spline terms may be named differently across treatment and control groups.

`equal.coef1` character, a vector containing all the terms in `m1` that should have the same coefficients in `m0`. See the description for `equal.coef0` for more details.

`sset` a list containing the point estimates and gamma moments for each IV-like specification.

`gstar0` set of expectations for each terms of the MTR for the control group, corresponding to the target parameter.

`gstar1` set of expectations for each terms of the MTR for the control group, corresponding to the target parameter.

`orig.sset` list, only used for bootstraps. The list contains the gamma moments for each element in the S-set, as well as the IV-like coefficients.

`orig.criterion` numeric, only used for bootstraps. The scalar corresponds to the minimum observational equivalence criterion from the original sample.

`criterion.tol` tolerance for the criterion function, and is set to 1e-4 by default. The criterion measures how well the IV-like moments/conditional means are matched using the l1-norm. Statistical noise may prohibit the theoretical LP/QCQP problem from being feasible. That is, there may not exist a set of MTR coefficients that are able to match all the specified moments. The function thus first estimates the minimum criterion, which is reported in the output under the name 'minimum criterion', with a criterion of 0 meaning that all moments were able to be matched. The function then relaxes the constraints by tolerating a criterion up to `minimum criterion * (1 + criterion.tol)`. Set `criterion.tol` to a value greater than 0 to allow for more conservative bounds.

`solver` character, name of the programming package in R used to obtain the bounds on the treatment effect. The function supports `'gurobi'`, `'cplexapi'`, `rmosek`, `'lpsolveapi'`. The name of the solver should be provided with quotation marks.

`solver.options` list, each item of the list should correspond to an option specific to the solver selected.

`solver.presolve`
boolean, default set to `TRUE`. Set this parameter to `FALSE` if presolve should be turned off for the LP/QCQP problems.

`solver.options.criterion`
list, each item of the list should correspond to an option specific to the solver selected. These options are specific for finding the minimum criterion.

`solver.options.bounds`
: list, each item of the list should correspond to an option specific to the solver selected. These options are specific for finding the bounds.

`rescale`
: boolean, set to `TRUE` if the MTR components should be rescaled to improve stability in the LP/QCQP optimization.

`smallreturnlist`
: boolean, default set to `FALSE`. Set to `TRUE` to exclude large intermediary components (i.e. propensity score model, LP/QCQP model, bootstrap iterations) from being included in the return list.

`noisy`
: boolean, default set to `TRUE`. If `TRUE`, then messages are provided throughout the estimation procedure. Set to `FALSE` to suppress all messages, e.g. when performing the bootstrap.

`debug`
: boolean, indicates whether or not the function should provide output when obtaining bounds. The option is only applied when `solver = 'gurobi'` or `solver = 'rmosek'`. The output provided is the same as what the Gurobi API would send to the console.

Value

a list. Included in the list are estimates of the treatment effect bounds; the minimum violation of observational equivalence of the set of IV-like estimands; the list of matrices and vectors defining the LP/QCQP problem; the points used to generate the audit grid, and the points where the shape constraints were violated.

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare empty list to be updated (in the event multiple IV like
## specifications are provided
sSet <- list()

## Declare MTR formulas
formula0 = ~ 1 + u
formula1 = ~ 1 + u

## Construct object that separates out non-spline components of MTR
## formulas from the spline components. The MTR functions are
## obtained from this object by the function 'genSSet'
splinesList = list(removeSplines(formula0), removeSplines(formula1))

## If splines are interacted with other variables, the
## 'interactSplines' should be used.
## splinesList <- interactSplines(splinesobj = splinesList,
##                                m0 = formula0,
##                                m1 = formula1,
##                                data = data,
##                                uname = 'u')

## Construct MTR polynomials
```

```
polynomials0 <- polyparse(formula = formula0,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)

polynomials1 <- polyparse(formula = formula1,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)

## Generate propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Generate IV estimates
ivEstimates <- ivEstimate(formula = ey ~ d | z,
                          data = dtm,
                          components = l(intercept, d),
                          treat = d,
                          list = FALSE)

## Generate target gamma moments
targetGamma <- genTarget(treat = "d",
                         m0 = ~ 1 + u,
                         m1 = ~ 1 + u,
                         target = "atu",
                         data = dtm,
                         splinesobj = splinesList,
                         pmodobj = propensityObj,
                         pm0 = polynomials0,
                         pm1 = polynomials1)

## Construct S-set, which contains the coefficients and weights
## corresponding to various IV-like estimands
sSet <- genSSet(data = dtm,
                sset = sSet,
                sest = ivEstimates,
                splinesobj = splinesList,
                pmodobj = propensityObj$phat,
                pm0 = polynomials0,
                pm1 = polynomials1,
                ncomponents = 2,
                scount = 1,
                yvar = "ey",
                dvar = "d",
                means = TRUE)

## Perform audit procedure and return bounds
audit(data = dtm,
      uname = u,
      m0 = formula0,
      m1 = formula1,
```

```
        pm0 = polynomials0,
        pm1 = polynomials1,
        splinesobj = splinesList,
        vars_data = colnames(dtm),
        vars_mtr = "u",
        terms_mtr0 = "u",
        terms_mtr1 = "u",
        sset = sSet$sset,
        gstar0 = targetGamma$gstar0,
        gstar1 = targetGamma$gstar1,
        m0.inc = TRUE,
        m1.dec = TRUE,
        m0.lb = 0.2,
        m1.ub = 0.8,
        audit.max = 5,
        solver = "lpSolveAPI")
```

---

bound *Obtaining TE bounds*

Description

This function estimates the bounds on the target treatment effect. The LP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
bound(
  env,
  sset,
  solver,
  solver.options,
  noisy = FALSE,
  smallreturnlist = FALSE,
  rescale = FALSE,
  debug = FALSE
)
```

Arguments

`env`	environment containing the matrices defining the LP problem.
`sset`	a list containing the point estimates and gamma components associated with each element in the S-set. This object is only used to determine the names of terms. If it is no submitted, then no names are provided to the solution vector.
`solver`	string, name of the package used to solve the LP problem.
`solver.options`	list, each item of the list should correspond to an option specific to the LP solver selected.

`noisy`	boolean, set to `TRUE` if optimization results should be displayed.
`smallreturnlist`	boolean, set to `TRUE` if the LP model should not be returned.
`rescale`	boolean, set to `TRUE` if the MTR components should be rescaled to improve stability in the LP/QP/QCP optimization.
`debug`	boolean, indicates whether or not the function should provide output when obtaining bounds. The option is only applied when `solver = 'gurobi'` or `solver = 'rmosek'`. The output provided is the same as what the Gurobi API would send to the console.

Value

a list containing the bounds on the treatment effect; the coefficients on each term in the MTR associated with the upper and lower bounds, for both counterfactuals; the optimization status to the maximization and minimization problems; the LP problem that the optimizer solved.

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare empty list to be updated (in the event multiple IV like
## specifications are provided
sSet <- list()

## Declare MTR formulas
formula0 = ~ 1 + u
formula1 = ~ 1 + u

## Construct object that separates out non-spline components of MTR
## formulas from the spline components. The MTR functions are
## obtained from this object by the function 'genSSet'.
splinesList = list(removeSplines(formula0), removeSplines(formula1))

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                          data = dtm,
                          uname = u,
                          as.function = FALSE)
polynomials1 <- polyparse(formula = formula1,
                          data = dtm,
                          uname = u,
                           as.function = FALSE)

## Generate propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Generate IV estimates
ivEstimates <- ivEstimate(formula = ey ~ d | z,
                          data = dtm,
```

```
                              components = l(intercept, d),
                              treat = d,
                              list = FALSE)

## Generate target gamma moments
targetGamma <- genTarget(treat = "d",
                         m0 = ~ 1 + u,
                         m1 = ~ 1 + u,
                         target = "atu",
                         data = dtm,
                         splinesobj = splinesList,
                         pmodobj = propensityObj,
                         pm0 = polynomials0,
                         pm1 = polynomials1)

## Construct S-set. which contains the coefficients and weights
## corresponding to various IV-like estimands
sSet <- genSSet(data = dtm,
                sset = sSet,
                sest = ivEstimates,
                splinesobj = splinesList,
                pmodobj = propensityObj$phat,
                pm0 = polynomials0,
                pm1 = polynomials1,
                ncomponents = 2,
                scount = 1,
                yvar = "ey",
                dvar = "d",
                means = TRUE)
## Only the entry $sset is required
sSet <- sSet$sset

## Define additional upper- and lower-bound constraints for the LP
## problem
A <- matrix(0, nrow = 22, ncol = 4)
A <- cbind(A, rbind(cbind(1, seq(0, 1, 0.1)),
                    matrix(0, nrow = 11, ncol = 2)))
A <- cbind(A, rbind(matrix(0, nrow = 11, ncol = 2),
                    cbind(1, seq(0, 1, 0.1))))
sense <- c(rep(">", 11), rep("<", 11))
rhs <- c(rep(0.2, 11), rep(0.8, 11))

## Construct LP object to be interpreted and solved by
## lpSolveAPI. Note that an environment has to be created for the LP
## object. The matrices defining the shape restrictions must be stored
## as a list under the entry \code{$mbobj} in the environment.
modelEnv <- new.env()
modelEnv$mbobj <- list(mbA = A,
                    mbs = sense,
                    mbrhs = rhs)
## Convert the matrices defining the shape constraints into a format
## that is suitable for the LP solver.
lpSetup(env = modelEnv,
```

```
        sset = sSet,
        solver = "lpsolveapi")
## Setup LP model so that it is solving for the bounds.
lpSetupBound(env = modelEnv,
             g0 = targetGamma$gstar0,
             g1 = targetGamma$gstar1,
             sset = sSet,
             criterion.tol = 0,
             criterion.min = 0,
             solver = "lpsolveapi")
## Declare any LP solver options as a list.
lpOptions <- optionsLpSolveAPI(list(epslevel = "tight"))
## Obtain the bounds.
bounds <- bound(env = modelEnv,
                sset = sSet,
                solver = "lpsolveapi",
                solver.options = lpOptions)
cat("The bounds are [",  bounds$min, ",", bounds$max, "].\n")
```

---

boundCI *Construct confidence intervals for treatment effects under partial identification*

Description

This function constructs the forward and backward confidence intervals for the treatment effect under partial identification.

Usage

```
boundCI(bounds, bounds.resamples, n, m, levels, type)
```

Arguments

`bounds`	vector, bounds of the treatment effects under partial identification.
`bounds.resamples`	matrix, stacked bounds of the treatment effects under partial identification. Each row corresponds to a subset resampled from the original data set.
`n`	integer, size of original data set.
`m`	integer, size of resampled data sets.
`levels`	vector, real numbers between 0 and 1. Values correspond to the level of the confidence intervals constructed via bootstrap.
`type`	character. Set to 'forward' to construct the forward confidence interval for the treatment effect bounds. Set to 'backward' to construct the backward confidence interval for the treatment effect bounds. Set to 'both' to construct both types of confidence intervals.

Value

if `type` is 'forward' or 'backward', then the corresponding type of confidence interval for each level is returned. The output is in the form of a matrix, with each row corresponding to a level. If `type` is 'both', then a list is returned. One element of the list is the matrix of backward confidence intervals, and the other element of the list is the matrix of forward confidence intervals.

`boundPvalue` *Construct p-values for treatment effects under partial identification*

Description

This function estimates the p-value for the treatment effect under partial identification. p-values corresponding to forward and backward confidence intervals can be returned.

Usage

```
boundPvalue(bounds, bounds.resamples, n, m, type)
```

Arguments

`bounds` vector, bounds of the treatment effects under partial identification.

`bounds.resamples`
matrix, stacked bounds of the treatment effects under partial identification. Each row corresponds to a subset resampled from the original data set.

`n` integer, size of original data set.

`m` integer, size of resampled data sets.

`type` character. Set to 'forward' to construct the forward confidence interval for the treatment effect bounds. Set to 'backward' to construct the backward confidence interval for the treatment effect bounds. Set to 'both' to construct both types of confidence intervals.

Value

If `type` is 'forward' or 'backward', a scalar p-value corresponding to the type of confidence interval is returned. If `type` is 'both', a vector of p-values corresponding to the forward and backward confidence intervals is returned.

bX — *Spline basis function of order 1*

Description

This function is the splines basis function of order 1. This function was coded in accordance to Carl de Boor's set of notes on splines, "B(asic)-Spline Basics".

Usage

```
bX(x, knots, i)
```

Arguments

`x`	vector, the values at which to evaluate the basis function.
`knots`	vector, the internal knots.
`i`	integer, the basis component to be evaluated.

Value

scalar.

checkU — *Check polynomial form of the u-term*

Description

This function ensures that the unobservable term enters into the MTR in the correct manner. That is, it enters as a polynomial.

Usage

```
checkU(formula, uname)
```

Arguments

`formula`	a formula.
`uname`	name of the unobserved variable.

Value

If the unobservable term is entered correctly into the formula, then `NULL` is returned. Otherwise, the vector of incorrect terms is returned.

`classFormula` *Auxiliary function: test if object is a formula*

Description

Auxiliary function to test if an object is a formula. Warnings are suppressed.

Usage

```
classFormula(obj)
```

Arguments

`obj`	the object to be checked.

Value

boolean expression.

`classList` *Auxiliary function: test if object is a list*

Description

Auxiliary function to test if an object is a list. Warnings are suppressed.

Usage

```
classList(obj)
```

Arguments

`obj`	the object to be checked.

Value

boolean expression.

combinemonobound *Combining the boundedness and monotonicity constraint objects*

Description

This function simply combines the objects associated with the boundedness constraints and the monotonicity constraints.

Usage

```
combinemonobound(bdA, monoA)
```

Arguments

`bdA`	list containing the constraint matrix, vector of inequalities, and RHS vector associated with the boundedness constraints.
`monoA`	list containing the constraint matrix, vector on inequalities, and RHS vector associated with the monotonicity constraints.

Value

a list containing a unified constraint matrix, unified vector of inequalities, and unified RHS vector for the boundedness and monotonicity constraints of an LP/QCQP problem.

constructConstant *Construct constant function*

Description

This function constructs another function that returns a constant. It is used for constructing weight/knot functions.

Usage

```
constructConstant(x)
```

Arguments

`x`	scalar, the constant the function evaluates to.

Value

a function.

criterionMin *Minimizing violation of observational equivalence*

Description

Given a set of IV-like estimates and the set of matrices/vectors defining an LP problem, this function minimizes the violation of observational equivalence under the L1 norm. The LP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
criterionMin(env, sset, solver, solver.options, rescale = FALSE, debug = FALSE)
```

Arguments

`env`	environment containing the matrices defining the LP problem.
`sset`	A list of IV-like estimates and the corresponding gamma terms.
`solver`	string, name of the package used to solve the LP problem.
`solver.options`	list, each item of the list should correspond to an option specific to the LP solver selected.
`rescale`	boolean, set to `TRUE` if the MTR components should be rescaled to improve stability in the LP/QP/QCP optimization.
`debug`	boolean, indicates whether or not the function should provide output when obtaining bounds. The option is only applied when `solver = 'gurobi'` or `solver = 'rmosek'`. The output provided is the same as what the Gurobi API would send to the console.

Value

A list including the minimum violation of observational equivalence, the solution to the LP problem, and the status of the solution.

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare empty list to be updated (in the event multiple IV like
## specifications are provided
sSet <- list()

## Declare MTR formulas
formula0 = ~ 1 + u
formula1 = ~ 1 + u

## Construct object that separates out non-spline components of MTR
## formulas from the spline components. The MTR functions are
## obtained from this object by the function 'genSSet'.
```

```
splinesList = list(removeSplines(formula0), removeSplines(formula1))

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                          data = dtm,
                          uname = u,
                          as.function = FALSE)
polynomials1 <- polyparse(formula = formula1,
                          data = dtm,
                          uname = u,
                           as.function = FALSE)

## Generate propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Generate IV estimates
ivEstimates <- ivEstimate(formula = ey ~ d | z,
                          data = dtm,
                          components = l(intercept, d),
                          treat = d,
                          list = FALSE)

## Generate target gamma moments
targetGamma <- genTarget(treat = "d",
                         m0 = ~ 1 + u,
                         m1 = ~ 1 + u,
                         target = "atu",
                         data = dtm,
                         splinesobj = splinesList,
                         pmodobj = propensityObj,
                         pm0 = polynomials0,
                         pm1 = polynomials1)

## Construct S-set. which contains the coefficients and weights
## corresponding to various IV-like estimands
sSet <- genSSet(data = dtm,
                sset = sSet,
                sest = ivEstimates,
                splinesobj = splinesList,
                pmodobj = propensityObj$phat,
                pm0 = polynomials0,
                pm1 = polynomials1,
                ncomponents = 2,
                scount = 1,
                yvar = "ey",
                dvar = "d",
                means = TRUE)
## Only the entry $sset is required
sSet <- sSet$sset

## Define additional upper- and lower-bound constraints for the LP
```

```
## problem.  The code below imposes a lower bound of 0.2 and upper
## bound of 0.8 on the MTRs.
A <- matrix(0, nrow = 22, ncol = 4)
A <- cbind(A, rbind(cbind(1, seq(0, 1, 0.1)),
                    matrix(0, nrow = 11, ncol = 2)))
A <- cbind(A, rbind(matrix(0, nrow = 11, ncol = 2),
                    cbind(1, seq(0, 1, 0.1))))
sense <- c(rep(">", 11), rep("<", 11))
rhs <- c(rep(0.2, 11), rep(0.8, 11))

## Construct LP object to be interpreted and solved by
## lpSolveAPI. Note that an environment has to be created for the LP
## object. The matrices defining the shape restrictions must be stored
## as a list under the entry \code{$mbobj} in the environment.
modelEnv <- new.env()
modelEnv$mbobj <- list(mbA = A,
                    mbs = sense,
                    mbrhs = rhs)
## Convert the matrices defining the shape constraints into a format
## that is suitable for the LP solver.
lpSetup(env = modelEnv,
        sset = sSet,
        solver = "lpsolveapi")
## Setup LP model so that it will minimize the criterion
lpSetupCriterion(env = modelEnv,
                sset = sSet)
## Declare any LP solver options as a list.
lpOptions <- optionsLpSolveAPI(list(epslevel = "tight"))
## Minimize the criterion.
obseqMin <- criterionMin(env = modelEnv,
                         sset = sSet,
                         solver = "lpsolveapi",
                         solver.options = lpOptions)
obseqMin
cat("The minimum criterion is",  obseqMin$obj, "\n")
```

`design` *Generating design matrices*

Description

This function generates the design matrix given an IV specification.

Usage

```
design(formula, data, subset, treat, orig.names)
```

Arguments

formula	Formula with which to generate the design matrix.
data	`data.frame` with which to generate the design matrix.
subset	Condition to select subset of data.
treat	The name of the treatment variable. This should only be passed when constructing OLS weights.
orig.names	character vector of the terms in the final design matrix. This is required when the user declares an IV-like formula where the treatment variable is passed into the `factor` function. Since the treatment variable has to be fixed to 0 or 1, the design matrix will be unable to construct the contrasts. The argument `orig.names` is a vector of the terms in the IV-like specification prior to fixing the treatment variable.

Value

Three matrices are returned: one for the outcome variable, Y; one for the second stage covariates, X; and one for the first stage covariates, Z.

Examples

```
dtm <- ivmte:::gendistMosquito()
design(formula = ey ~ d | z,
           data = dtm,
           subset = z %in% c(1, 2))
```

---

extractcols *Auxiliary function: extracting columns by component names*

Description

Auxiliary function to extract columns from a matrix based on column names.

Usage

```
extractcols(M, components)
```

Arguments

M	The matrix to extract from.
components	The vector of variable names.

fmtResult *Format result for display*

Description

This function simply takes a number and formats it for being displayed. Numbers less than 1 in absolute value are rounded to 6 significant figure. Numbers larger than

Usage

```
fmtResult(x)
```

Arguments

x	The scalar to be formated

Value

A scalar.

funEval *Evaluate a particular function*

Description

This function evaluates a single function in a list of functions.

Usage

```
funEval(fun, values = NULL, argnames = NULL)
```

Arguments

fun	the function to be evaluated.
values	the values of the arguments to the function. Ordering is assumed to be the same as in `argnames`.
argnames	the argument names corresponding to `values`.

Value

the output of the function evaluated.

genBasisSplines *Generate basis matrix for splines*

Description

The user can declare that the unobservable enters into the MTRs in the form of splines. This function generates the basis matrix for the splines. The specifications for the spline must be passed as the `$splineslist` object generated by `removeSplines`. Note that this function does not account for any interactions between the splines and the covariates. Interactions can be added simply by sweeping the basis matrix by a vector for the values of the covariates.

Usage

```
genBasisSplines(splines, x, d = NULL)
```

Arguments

`splines`	a list. The name of each element should be the spline command, and each element should be a vector. Each entry of the vector is a covariate that the spline should be interacted with. Such an object can be generated by `removeSplines`, and accessed using `$splineslist`.
`x`	the values of the unobservable at which the splines basis should be evaluated.
`d`	either 0 or 1, indicating the treatment status.

Value

a matrix. The number of rows is equal to the length of `x`, and the number of columns depends on the specifications of the spline. The name of each column takes the following form: "u[d]S[j].[b]", where "u" and "S" are fixed and stand for "unobservable" and "Splines" respectively. "[d]" will be either 0 or 1, depending on the treatment status. "[j]" will be an integer indicating which element of the list `splines` the column pertains to. "[b]" will be an integer reflect which component of the basis the column pertains to.

genboundA *Generating the constraint matrix*

Description

This function generates the component of the constraint matrix in the LP/QCQP problem pertaining to the lower and upper bounds on the MTRs and MTEs. These bounds are declared by the user.

Usage

```
genboundA(
  A0,
  A1,
  sset,
  gridobj,
  uname,
  m0.lb,
  m0.ub,
  m1.lb,
  m1.ub,
  mte.lb,
  mte.ub,
  solution.m0.min = NULL,
  solution.m1.min = NULL,
  solution.m0.max = NULL,
  solution.m1.max = NULL,
  audit.tol,
  direct = FALSE
)
```

Arguments

- `A0` — the matrix of values from evaluating the MTR for control observations over the grid generated to perform the audit. This matrix will be incorporated into the final constraint matrix for the bounds.
- `A1` — the matrix of values from evaluating the MTR for control observations over the grid generated to perform the audit. This matrix will be incorporated into the final constraint matrix for the bounds.
- `sset` — a list containing the point estimates and gamma components associated with each element in the S-set.
- `gridobj` — a list containing the grid over which the monotonicity and boundedness conditions are imposed on.
- `uname` — name declared by user to represent the unobservable term.
- `m0.lb` — scalar, lower bound on MTR for control group.
- `m0.ub` — scalar, upper bound on MTR for control group.
- `m1.lb` — scalar, lower bound on MTR for treated group.
- `m1.ub` — scalar, upper bound on MTR for treated group.
- `mte.lb` — scalar, lower bound on MTE.
- `mte.ub` — scalar, upper bound on MTE.
- `solution.m0.min` — vector, the coefficients for the MTR for `D = 0` corresponding to the lower bound of the target parameter. If passed, this will initiate checks of shape constraints.
- `solution.m1.min` — vector, the coefficients for the MTR for `D = 1` corresponding to the lower bound of the target parameter. If passed, this will initiate checks of shape constraints.

solution.m0.max
: vector, the coefficients for the MTR for `D = 0` corresponding to the upper bound of the target parameter. If passed, this will initiate checks of shape constraints.

solution.m1.max
: vector, the coefficients for the MTR for `D = 1` corresponding to the upper bound of the target parameter. If passed, this will initiate checks of shape constraints.

audit.tol
: feasibility tolerance when performing the audit. By default to set to be equal `1e-06`. This parameter should only be changed if the feasibility tolerance of the solver is changed, or if numerical issues result in discrepancies between the solver's feasibility check and the audit.

direct
: boolean, set to `TRUE` if the direct MTR regression is used.

Value

a constraint matrix for the LP/QCQP problem, the associated vector of inequalities, and the RHS vector in the inequality constraint. The objects pertain only to the boundedness constraints declared by the user.

---

gendist1 *Generate test distribution 1*

Description

This function generates a data set for testing purposes. There is a single instrument that takes on values of 1 or 2, and the distribution of the values for the binary instrument is uniform. The MTRs are m0 ~ 0 + u and m1 ~ 1 + u. All unobservables u are integrated out.

Usage

```
gendist1(subN = 5, p1 = 0.4, p2 = 0.6)
```

Arguments

subN
: integer, default set to 5. This is the number of individuals possessing each value of the instrument. So the total number of observations is subN * 2.

p1
: the probability of treatment for those with the instrument Z = 1.

p2
: the probability of treatment for those with the instrument Z = 2.

Value

a data.frame.

`gendist1e` *Generate test distribution 1 with errors*

Description

This function generates a data set for testing purposes. There is a single instrument that takes on values of 1 or 2, and the distribution of the values for the binary instrument is uniform. The MTRs are m0 ~ 0 + u and m1 ~ 1 + u.

Usage

```
gendist1e(N = 100, subN = 0.5, p1 = 0.4, p2 = 0.6, v0.sd = 0.5, v1.sd = 0.75)
```

Arguments

`N`	integer, default set to 100. Total number of observations in the data.
`subN`	, default set to 0.5. This is the probability the agent will have Z = 1.
`p1`	the probability of treatment for those with the instrument Z = 1.
`p2`	the probability of treatment for those with the instrument Z = 2.
`v0.sd`	numeric, standard deviation of error term for counterfactual `D = 0`
`v1.sd`	numeric, standard deviation of error term for counterfactual `D = 1`

Value

a data.frame.

`gendist2` *Generate test distribution 2*

Description

This function generates a data set for testing purposes. There is a single instrument that takes on values of 1, 2, or 3, and the distribution of the values for the binary instrument is uniform. The MTRs are m0 ~ 1 + u and m1 ~ 1 + u. All unobservables u are integrated out.

Usage

```
gendist2(subN = 5, p1 = 0.4, p2 = 0.6, p3 = 0.8)
```

Arguments

`subN`	integer, default set to 5. This is the number of individuals possessing each value of the instrument. So the total number of observations is subN * 2.
`p1`	the probability of treatment for those with the instrument Z = 1.
`p2`	the probability of treatment for those with the instrument Z = 2.
`p3`	the probability of treatment for those with the instrument Z = 3.

Value

a data.frame.

---

gendist3 *Generate test distribution 3*

Description

This function generates a data set for testing purposes. There is a single instrument that takes on values of 1 and 2, and the distribution of the values for the binary instrument is uniform. The MTRs are m0 ~ 1 and m1 ~ 1. All unobservables u are integrated out.

Usage

```
gendist3(subN = 5, p1 = 0.4, p2 = 0.6)
```

Arguments

`subN`	integer, default set to 5. This is the number of individuals possessing each value of the instrument. So the total number of observations is subN * 2.
`p1`	the probability of treatment for those with the instrument Z = 1.
`p2`	the probability of treatment for those with the instrument Z = 2.

Value

a data.frame.

---

gendist3e *Generate test distribution 3 with errors*

Description

This function generates a data set for testing purposes. There is a single instrument that takes on values of 1 or 2, and the distribution of the values for the binary instrument is uniform. The MTRs are m0 ~ 0 + u and m1 ~ 1 + u.

Usage

```
gendist3e(N = 100, subN = 0.5, p1 = 0.4, p2 = 0.6, v0.sd = 0.5, v1.sd = 0.75)
```

Arguments

`N`	integer, default set to 100. Total number of observations in the data.
`subN`	, default set to 0.5. This is the probability the agent will have Z = 1.
`p1`	the probability of treatment for those with the instrument Z = 1.
`p2`	the probability of treatment for those with the instrument Z = 2.
`v0.sd`	numeric, standard deviation of error term for counterfactual `D = 0`
`v1.sd`	numeric, standard deviation of error term for counterfactual `D = 1`

Value

a data.frame.

---

gendist4 *Generate test distribution 4*

Description

This function generates a data set for testing purposes. There is a single instrument that takes on values of 1, 2, and 3, and the distribution of the values for the binary instrument is uniform. The MTRs are m0 ~ 1 and m1 ~ 1. All unobservables u are integrated out.

Usage

```
gendist4(subN = 5, p1 = 0.4, p2 = 0.6, p3 = 0.8)
```

Arguments

`subN`	integer, default set to 5. This is the number of individuals possessing each value of the instrument. So the total number of observations is subN * 2.
`p1`	the probability of treatment for those with the instrument Z = 1.
`p2`	the probability of treatment for those with the instrument Z = 2.
`p3`	the probability of treatment for those with the instrument Z = 3.

Value

a data.frame.

`gendist5e` *Generate test distribution 5 (has errors and a covariate)*

Description

This function generates a data set for testing purposes. There is a single instrument that takes on values of 1 or 2, and the distribution of the values for the binary instrument is uniform. The MTRs are both of the form m ~ 1 + x + u.

Usage

```
gendist5e(N = 100, subN = 0.5, p1 = 0.4, p2 = 0.6, v0.sd = 1, v1.sd = 1.55)
```

Arguments

`N`	integer, default set to 100. Total number of observations in the data.
`subN`	, default set to 0.5. This is the probability the agent will have Z = 1.
`p1`	the probability of treatment for those with the instrument Z = 1.
`p2`	the probability of treatment for those with the instrument Z = 2.
`v0.sd`	numeric, standard deviation of error term for counterfactual `D = 0`
`v1.sd`	numeric, standard deviation of error term for counterfactual `D = 1`

Value

a data.frame.

`gendist6e` *Generate test distribution 6 (has errors and a covariate)*

Description

This function generates a data set for testing purposes. There is a single instrument that is uniformly distributed over [0, 1]. The MTRs are both of the form m ~ 1 + x + x:u.

Usage

```
gendist6e(N = 100, v0.sd = 1, v1.sd = 1.55)
```

Arguments

`N`	integer, default set to 100. Total number of observations in the data.
`v0.sd`	numeric, standard deviation of error term for counterfactual `D = 0`
`v1.sd`	numeric, standard deviation of error term for counterfactual `D = 1`

Value

a data.frame.

`gendistBasic` *Generate basic data set for testing*

Description

This code generates population level data to test the estimation function. This is a simpler dataset, one in which we can more easily estimate a correctly specified model. The data presented below will have already integrated over the # unobservable terms U, where U | X, Z ~ Unif[0, 1].

Usage

```
gendistBasic()
```

Value

a list of two data.frame objects. One is the distribution of the simulated data, the other is the full simulated data set.

`gendistCovariates` *Generate test data set with covariates*

Description

This code generates population level data to test the estimation function. This data includes covariates. The data generated will have already integrated over the unobservable terms U, where U | X, Z ~ Unif[0, 1].

Usage

```
gendistCovariates()
```

Value

a list of two data.frame objects. One is the distribution of the simulated data, the other is the full simulated data set.

`gendistMosquito` *Generate mosquito data set*

Description

This code generates the population level data in Mogstad, Santos, Torgovitsky (2018), i.e. the mosquito data set used as the running example.

Usage

```
gendistMosquito()
```

Value

data.frame.

`gendistSplines` *Generate test data set with splines*

Description

This code generates population level data to test the estimation function. This data set incorporates splines in the MTRs.

Usage

```
gendistSplines()
```

Details

The distribution of the data is as follows

| Z X/Z | 0 1 _______|___________ -1 | 0.1 0.1 | X 0 | 0.2 0.2 | 1 | 0.1 0.2

The data presented below will have already integrated over the unobservable terms U, and U | X, Z ~ Unif[0, 1].

The propensity scores are generated according to the model

p(x, z) = 0.5 - 0.1 * x + 0.2 * z

| Z p(X,Z) | 0 1 _______|___________ -1 | 0.6 0.8 | X 0 | 0.5 0.7 | 1 | 0.4 0.6

The lowest common multiple of the first table is 12. The lowest common multiple of the second table is 84. It turns out that 840 * 5 = 4200 observations is enough to generate the population data set, such that each group has a whole-number of observations.

The MTRs are defined as follows:

y1 ~ beta0 + beta1 * x + uSpline(degree = 2, knots = c(0.3, 0.6), intercept = FALSE)

The coefficients (beta1, beta2), and the coefficients on the splines, will be defined below.

y0 = x : uSpline(degree = 0, knots = c(0.2, 0.5, 0.8), intercept = TRUE) + uSpline(degree = 1, knots = c(0.4), intercept = TRUE) + beta3 * I(u ^ 2)

The coefficient beta3, and the coefficients on the splines, will be defined below.

Value

a list of two data.frame objects. One is the distribution of the simulated data, the other is the full simulated data set.

---

genej — *Auxiliary function: generating basis vectors*

Description

Auxiliary function to generate standard basis vectors.

Usage

```
genej(pos, length)
```

Arguments

`pos`	The position of the non-zero entry/dimension the basis vector corresponds to
`length`	Number of dimensions in total/length of vector.

Value

Vector containing 1 in a single position, and 0 elsewhere.

---

genGamma — *Estimating expectations of terms in the MTR (gamma objects)*

Description

This function generates the gamma objects defined in the paper, i.e. each additive term in E[md], where md is a MTR.

Usage

```
genGamma(
  monomials,
  lb,
  ub,
  multiplier = 1,
  subset = NULL,
  means = TRUE,
  late.rows = NULL
)
```

Arguments

`monomials`	[UPDATE DESCRIPTION] object containing list of list of monomials. Each element of the outer list represents an observation in the data set, each element in the inner list is a monomial from the MTR. The variable is the unobservable u, and the coefficient is the evaluation of any interactions with u.
`lb`	vector of lower bounds for the interval of integration. Each element corresponds to an observation.
`ub`	vector of upper bounds for the interval of integration. Each element corresponds to an observation.
`multiplier`	a vector of the weights that enter into the integral. Each element corresponds to an observation.
`subset`	The row names/numbers of the subset of observations to use.
`means`	logical, if TRUE then function returns the terms of E[md]. If FALSE, then function instead returns each term of E[md | D, X, Z]. This is useful for testing the code, i.e. obtaining population estimates.
`late.rows`	Boolean vector indicating which observations to include when conditioning on covariates X.

Value

If `means = TRUE`, then the function returns a vector of the additive terms in Gamma (i.e. the expectation is over D, X, Z, and u). If `means = FALSE`, then the function returns a matrix, where each row corresponds to an observation, and each column corresponds to an additive term in E[md | D, X, Z] (i.e. only the integral with respect to u is performed).

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare MTR formula
formula0 = ~ 1 + u

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                data = dtm,
                uname = u,
                as.function = FALSE)

## Construct propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Generate gamma moments, with S-weight equal to its default value
## of 1
genGamma(monomials = polynomials0,
         lb = 0,
         ub = propensityObj$phat)
```

genGammaSplines — *Generate Gamma moments for splines*

Description

The user can declare that the unobservable enters into the MTRs in the form of splines. This function generates the gamma moments for the splines. The specifications for the spline must be passed as an element generated by `removeSplines`. This function accounts for the interaction between covariates and splines.

Usage

```
genGammaSplines(
  splinesobj,
  data,
  lb,
  ub,
  multiplier = 1,
  subset,
  d = NULL,
  means = TRUE,
  late.rows = NULL
)
```

Arguments

`splinesobj`	a list generated by `removeSplines` applied to either the `m0` and `m1` argument.
`data`	a `data.frame` object containing all the variables that interact with the spline components.
`lb`	vector of lower bounds for the interval of integration. Each element corresponds to an observation.
`ub`	vector of upper bounds for the interval of integration. Each element corresponds to an observation.
`multiplier`	a vector of the weights that enter into the integral. Each element corresponds to an observation.
`subset`	Subset condition used to select observations with which to estimate gamma.
`d`	either 0 or 1, indicating the treatment status.
`means`	boolean, default set to `TRUE`. Set to `TRUE` if estimates of the gamma moments should be returned. Set to `FALSE` if the gamma estimates for each observation should be returned.
`late.rows`	Boolean vector indicating which observations to include when conditioning on covariates X.

Value

a matrix, corresponding to the splines being integrated over the region specified by `lb` and `ub`, accounting for the interaction terms. The number of rows is equal to the number of rows in `data`. The number of columns depends on the specifications of the spline. The name of each column takes the following form: "u[d]S[j].[b]", where "u" and "S" are fixed and stand for "unobservable" and "Splines" respectively. "[d]" will be either 0 or 1, depending on the treatment status. "[j]" will be an integer indicating which element of the list `splines` the column pertains to. "[b]" will be an integer reflect which component of the basis the column pertains to.

`genGammaSplinesTT` *Generating the Gamma moments for splines, for 'testthat'*

Description

This function generates the Gamma moments for a given set of weights. This funciton is written specifically for tests.

Usage

```
genGammaSplinesTT(distr, weight, zvars, u1s1, u0s1, u0s2, target = FALSE, ...)
```

Arguments

`distr`	data.frame, the distribution of the data.
`weight`	function, the S-function corresponding to a particular IV-like estimand.
`zvars`	vector, string names of the covariates, other than the intercept and treatment variable.
`u1s1`	matrix, the spline basis for the treated group ("u1") corresponding to the first (and only) spline specification ("s1").
`u0s1`	matrix, the spline basis for the control group ("u0") corresponding to the first spline specification ("s1").
`u0s2`	matrix, the spline basis for the control group ("u0") corresponding to the second spline specification ("s2").
`target`	boolean, set to `TRUE` if the gamma moment being generated corresponds to the target parameter.
`...`	all other arguments that enter into `weight`, excluding the argument `d` for treatment indicator.

Value

vector, the Gamma moments associated with `weight`.

`genGammaTT` *Function to generate gamma moments for 'testthat'*

Description

This function generates the gamma moments from a population level data set. This is specifically constructed to carry out tests.

Usage

```
genGammaTT(data, s0, s1, lb, ub)
```

Arguments

`data`	data.table.
`s0`	variable name (contained in the data) for the S-weight used to generate the Gamma moments for the control group.
`s1`	variable name (contained in the data) for the S-weight used to generate the Gamma moments for the treated group.
`lb`	scalar, lower bound for integration.
`ub`	scalar, upper bound for integration.

Value

list, contains the vectors of the Gamma moments for control and treated observations.

`gengrid` *Generating the grid for the audit procedure*

Description

This function takes in a matrix summarizing the support of the covariates, as well as set of points summarizing the support of the unobservable variable. A Cartesian product of the subset of the support of the covariates and the points in the support of the unobservable generates the grid that is used for the audit procedure.

Usage

```
gengrid(index, xsupport, usupport, uname)
```

Arguments

- `index` — a vector whose elements indicate the rows in the matrix `xsupport` to include in the grid.
- `xsupport` — a matrix containing all the unique combinations of the covariates included in the MTRs.
- `usupport` — a vector of points in the interval [0, 1], including 0 and 1. The number of points is decided by the user. The function generates these points using a Halton sequence.
- `uname` — name declared by user to represent the unobservable term.

Value

a list containing the grid used in the audit; a vector mapping the elements in the support of the covariates to `index`.

---

genmonoA — *Generate components of the monotonicity constraints*

Description

This function generates the matrix and vectors associated with the monotonicity constraints declared by the user. It takes in a grid of the covariates on which the shape constraints are defined, and then calculates the values of the MTR and MTE over the grid. The matrices characterizing the monotonicity conditions can then be obtained by taking first differences over the grid of the unobservable term, within each set of values in the grid of covariate values.

Usage

```
genmonoA(
  A0,
  A1,
  sset,
  uname,
  gridobj,
  gstar0,
  gstar1,
  m0.dec,
  m0.inc,
  m1.dec,
  m1.inc,
  mte.dec,
  mte.inc,
  solution.m0.min = NULL,
  solution.m1.min = NULL,
  solution.m0.max = NULL,
  solution.m1.max = NULL,
```

```
    audit.tol,
    direct
  )
```

Arguments

A0	the matrix of values from evaluating the MTR for control observations over the grid generated to perform the audit. This matrix will be incorporated into the final constraint matrix for the monotonicity conditions.
A1	the matrix of values from evaluating the MTR for control observations over the grid generated to perform the audit. This matrix will be incorporated into the final constraint matrix for the monotonicity conditions.
sset	a list containing the point estimates and gamma components associated with each element in the S-set.
uname	Name of unobserved variable.
gridobj	a list containing the grid over which the monotonicity and boundedness conditions are imposed on.
gstar0	set of expectations for each terms of the MTR for the control group.
gstar1	set of expectations for each terms of the MTR for the control group.
m0.dec	boolean, indicating whether the MTR for the control group is monotone decreasing.
m0.inc	boolean, indicating whether the MTR for the control group is monotone increasing.
m1.dec	boolean, indicating whether the MTR for the treated group is monotone decreasing.
m1.inc	boolean, indicating whether the MTR for the treated group is monotone increasing.
mte.dec	boolean, indicating whether the MTE is monotone decreasing.
mte.inc	boolean, indicating whether the MTE is monotone increasing.
solution.m0.min	vector, the coefficients for the MTR for D = 0 corresponding to the lower bound of the target parameter. If passed, this will initiate checks of shape constraints.
solution.m1.min	vector, the coefficients for the MTR for D = 1 corresponding to the lower bound of the target parameter. If passed, this will initiate checks of shape constraints.
solution.m0.max	vector, the coefficients for the MTR for D = 0 corresponding to the upper bound of the target parameter. If passed, this will initiate checks of shape constraints.
solution.m1.max	vector, the coefficients for the MTR for D = 1 corresponding to the upper bound of the target parameter. If passed, this will initiate checks of shape constraints.
audit.tol	feasibility tolerance when performing the audit. By default to set to be equal 1e-06. This parameter should only be changed if the feasibility tolerance of the solver is changed, or if numerical issues result in discrepancies between the solver's feasibility check and the audit.
direct	boolean, set to TRUE if the direct MTR regression is used.

Value

constraint matrix for the LP/QCQP problem. The matrix pertains only to the monotonicity conditions on the MTR and MTE declared by the user.

---

genmonoboundA *Generating monotonicity and boundedness constraints*

Description

This is a wrapper function generating the matrices and vectors associated with the monotonicity and boundedness constraints declared by the user. Since this function generates all the components required for the shape constraints, it is also the function that performs the audit. That is, MTR coefficients are passed, then this function will verify whether they satisfy the shape constraints.

Usage

```
genmonoboundA(
  pm0,
  pm1,
  support,
  grid_index,
  uvec,
  splinesobj,
  monov,
  uname,
  m0,
  m1,
  sset,
  gstar0,
  gstar1,
  m0.lb,
  m0.ub,
  m1.lb,
  m1.ub,
  mte.lb,
  mte.ub,
  m0.dec,
  m0.inc,
  m1.dec,
  m1.inc,
  mte.dec,
  mte.inc,
  solution.m0.min = NULL,
  solution.m1.min = NULL,
  solution.m0.max = NULL,
  solution.m1.max = NULL,
```

```
    audit.tol,
    direct
)
```

Arguments

pm0	A list of the monomials in the MTR for d = 0.
pm1	A list of the monomials in the MTR for d = 1.
support	a matrix for the support of all variables that enter into the MTRs.
grid_index	a vector, the row numbers of `support` used to generate the grid preceding the audit.
uvec	a vector, the points in the interval [0, 1] that the unobservable takes on.
splinesobj	a list of lists. Each of the inner lists contains details on the splines declared in the MTRs.
monov	name of variable for which the monotonicity conditions applies to.
uname	name declared by user to represent the unobservable term in the MTRs.
m0	one-sided formula for marginal treatment response function for the control group. The formula may differ from what the user originally input in `ivmte`, as the spline components should have been removed. This formula is simply a linear combination of all covariates that enter into the original `m0` declared by the user in `ivmte`.
m1	one-sided formula for marginal treatment response function for the treated group. The formula may differ from what the user originally input in `ivmte`, as the spline components should have been removed. This formula is simply a linear combination of all covariates that enter into the original `m1` declared by the user in `ivmte`.
sset	a list containing the point estimates and gamma components associated with each element in the S-set.
gstar0	set of expectations for each terms of the MTR for the control group.
gstar1	set of expectations for each terms of the MTR for the control group.
m0.lb	scalar, lower bound on MTR for control group.
m0.ub	scalar, upper bound on MTR for control group.
m1.lb	scalar, lower bound on MTR for treated group.
m1.ub	scalar, upper bound on MTR for treated group.
mte.lb	scalar, lower bound on MTE.
mte.ub	scalar, upper bound on MTE.
m0.dec	boolean, indicating whether the MTR for the control group is monotone decreasing.
m0.inc	boolean, indicating whether the MTR for the control group is monotone increasing.
m1.dec	boolean, indicating whether the MTR for the treated group is monotone decreasing.

`m1.inc` boolean, indicating whether the MTR for the treated group is monotone increasing.

`mte.dec` boolean, indicating whether the MTE is monotone decreasing.

`mte.inc` boolean, indicating whether the MTE is monotone increasing.

`solution.m0.min` vector, the coefficients for the MTR for `D = 0` corresponding to the lower bound of the target parameter. If passed, this will initiate checks of shape constraints.

`solution.m1.min` vector, the coefficients for the MTR for `D = 1` corresponding to the lower bound of the target parameter. If passed, this will initiate checks of shape constraints.

`solution.m0.max` vector, the coefficients for the MTR for `D = 0` corresponding to the upper bound of the target parameter. If passed, this will initiate checks of shape constraints.

`solution.m1.max` vector, the coefficients for the MTR for `D = 1` corresponding to the upper bound of the target parameter. If passed, this will initiate checks of shape constraints.

`audit.tol` feasibility tolerance when performing the audit. By default to set to be equal `1e-06`. This parameter should only be changed if the feasibility tolerance of the solver is changed, or if numerical issues result in discrepancies between the solver's feasibility check and the audit.

`direct` boolean, set to `TRUE` if the direct MTR regression is used.

Value

a list containing a unified constraint matrix, unified vector of inequalities, and unified RHS vector for the boundedness and monotonicity constraints of an LP/QCQP problem.

`genSSet` *Generating moments/data for IV-like estimands*

Description

This function takes in the IV estimate and its IV-like specification, and generates a list containing the corresponding IV-like point estimate, and the corresponding moments (gammas) that will enter into the constraint matrix of the LP problem. If the option `means = FALSE`, then the data are not averaged to generate the gamma moments and may be used for GMM. The function requires the user to provide a list (i.e. the list the point estimates and moments corresponding to other IV-like specifications; or an empty list) to append these point estimates and moments to.

Usage

```
genSSet(
  data,
  sset,
  sest,
```

```
    splinesobj,
    pmodobj,
    pm0,
    pm1,
    ncomponents,
    scount,
    subset_index,
    means = TRUE,
    yvar,
    dvar,
    noisy = TRUE,
    ivn = NULL,
    redundant = NULL
)
```

Arguments

data	data.frame used to estimate the treatment effects.
sset	list, which is modified and returned as the output. This object will contain all the information from the IV-like specifications that can be used for estimating the treatment effect.
sest	list containing the point estimates and S-weights corresponding to a particular IV-like estimand.
splinesobj	list of spline components in the MTRs for treated and control groups. Spline terms are extracted using removeSplines.
pmodobj	vector of propensity scores.
pm0	list of the monomials in the MTR for the control group.
pm1	list of the monomials in the MTR for the treated group.
ncomponents	The number of components from the IV regression to include in the S-set.
scount	integer, an index for the elements in the S-set.
subset_index	vector of integers, a row index for the subset of the data the IV regression is restricted to.
means	boolean, set to TRUE by default. If set to TRUE, then the gamma moments are returned, i.e. sample averages are taken. If set to FALSE, then no sample averages are taken, and a matrix is returned. The sample average of each column of the matrix corresponds to a particular gamma moment.
yvar	name of outcome variable. This is only used if means = FALSE, which occurs when the user believes the treatment effect is point identified.
dvar	name of treatment indicator. This is only used if means = FALSE, which occurs when the user believes the treatment effect is point identified.
noisy	boolean, default set to TRUE. If TRUE, then messages are provided throughout the estimation procedure. Set to FALSE to suppress all messages, e.g. when performing the bootstrap.
ivn	integer, the number indicating which IV specification the component corresponds to.
redundant	vector of integers indicating which components in the S-set are redundant.

Value

A list containing the point estimate for the IV regression, and the expectation of each monomial term in the MTR.

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare empty list to be updated (in the event multiple IV like
## specifications are provided)
sSet <- list()

## Declare MTR formulas
formula1 = ~ 1 + u
formula0 = ~ 1 + u

## Construct object that separates out non-spline components of MTR
## formulas from the spline components. The MTR functions are
## obtained from this object by the function 'genSSet'.
splinesList = list(removeSplines(formula0), removeSplines(formula1))

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)

polynomials1 <- polyparse(formula = formula0,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)

## Generate propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Generate IV estimates
ivEstimates <- ivEstimate(formula = ey ~ d | z,
                          data = dtm,
                          components = l(d),
                          treat = d,
                          list = FALSE)

## Construct S-set, which contains the coefficients and weights
## corresponding to various IV-like estimands
genSSet(data = dtm,
        sset = sSet,
        sest = ivEstimates,
        splinesobj = splinesList,
        pmodobj = propensityObj$phat,
        pm0 = polynomials0,
```

```
        pm1 = polynomials1,
        ncomponents = 1,
        scount = 1)
```

---

genTarget *Generating target MTR moments*

Description

This function estimates the moment of each MTR term under the target weight.

Usage

```
genTarget(
  treat,
  m0,
  m1,
  target,
  target.weight0,
  target.weight1,
  target.knots0,
  target.knots1,
  late.Z,
  late.from,
  late.to,
  late.X,
  eval.X,
  genlate.lb,
  genlate.ub,
  data,
  splinesobj,
  pmodobj,
  pm0,
  pm1,
  noisy = TRUE
)
```

Arguments

`treat` variable name for treatment indicator. The name can be provided with or without quotation marks.

`m0` one-sided formula for the marginal treatment response function for the control group. Splines may also be incorporated using the expression `uSpline`, e.g. `uSpline(degree = 2,knots = c(0.4,0.8),intercept = TRUE)`. The `intercept` argument may be omitted, and is set to `TRUE` by default.

m1	one-sided formula for the marginal treatment response function for the treated group. See m0 for details.
target	character, target parameter to be estimated. The function allows for ATE ('ate'), ATT ('att'), ATU ('atu'), LATE ('late'), and generalized LATE ('genlate').
target.weight0	user-defined weight function for the control group defining the target parameter. A list of functions can be submitted if the weighting function is in fact a spline. The arguments of the function should be variable names in data. If the weight is constant across all observations, then the user can instead submit the value of the weight instead of a function.
target.weight1	user-defined weight function for the treated group defining the target parameter. See target.weight0 for details.
target.knots0	user-defined set of functions defining the knots associated with spline weights for the control group. The arguments of the function should consist only of variable names in data. If the knots are constant across all observations, then the user can instead submit the vector of knots instead of a function.
target.knots1	user-defined set of functions defining the knots associated with spline weights for the treated group. See target.knots0 for details.
late.Z	vector of variable names used to define the LATE.
late.from	baseline set of values of Z used to define the LATE.
late.to	comparison set of values of Z used to define the LATE.
late.X	vector of variable names of covariates to condition on when defining the LATE.
eval.X	numeric vector of the values to condition variables in late.X on when estimating the LATE.
genlate.lb	lower bound value of unobservable u for estimating the generalized LATE.
genlate.ub	upper bound value of unobservable u for estimating the generalized LATE.
data	data.frame or data.table used to estimate the treatment effects.
splinesobj	list of spline components in the MTRs for treated and control groups. Spline terms are extracted using removeSplines. This object is supposed to be a dictionary of splines, containing the original calls of each spline in the MTRs, their specifications, and the index used for naming each basis spline.
pmodobj	A vector of propensity scores.
pm0	A list of the monomials in the MTR for d = 0.
pm1	A list of the monomials in the MTR for d = 1.
noisy	boolean, default set to TRUE. If TRUE, then messages are provided throughout the estimation procedure. Set to FALSE to suppress all messages, e.g. when performing the bootstrap.

Value

A list containing either the vectors of gamma moments for D = 0 and D = 1, or a matrix of individual gamma values for D = 0 and D = 1. Additoinally, two vectors are returned. xindex0 and xindex1 list the variables that interact with the unobservable u in m0 and m1. uexporder0 and uexporder1 lists the exponents of the unobservable u in each term it appears in.

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare MTR functions
formula1 = ~ 1 + u
formula0 = ~ 1 + u
splinesList = list(removeSplines(formula0), removeSplines(formula1))

## Declare propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)

polynomials1 <- polyparse(formula = formula0,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)

## Generate target gamma moments
genTarget(treat = "d",
          m0 = ~ 1 + u,
          m1 = ~ 1 + u,
          target = "atu",
          data = dtm,
          splinesobj = splinesList,
          pmodobj = propensityObj,
          pm0 = polynomials0,
          pm1 = polynomials1)
```

`genWeight` *Generating list of target weight functions*

Description

This function takes in the user-defined target weight functions and the data set, and generates the weight functions for each observation.

Usage

```
genWeight(fun, fun.name, uname, data)
```

Arguments

`fun`	custom weight function defined by the user. Arguments of the weight function must only be names of variables entering into the function, and can include the unobserved variable.
`fun.name`	string, name of function.
`uname`	the name assigned to the unobserved variable entering into the MTR.
`data`	a named vector containing the values of the variables defining the 'fun', excluding the value of the unobservable (generated from applying split() to a data.frame).

Value

The weight function 'fun', where all arguments other than that of the unobserved variable are fixed according to the vector 'data'.

`getXZ` *Auxiliary function: extract X and Z covariates from a formula*

Description

Auxiliary function that takes in a two-sided formula, and extracts the variable names of either the covariates or instruments. The function returns an error if the formula includes a variable called 'intercept'.

Usage

```
getXZ(fm, inst = FALSE, terms = FALSE, components = FALSE)
```

Arguments

`fm`	the formula.
`inst`	boolean expression, set to TRUE if the instrument names are to be extracted. Otherwise, the covariate names are extracted.
`terms`	boolean expression, set to TRUE if the terms in the formula `fm` should be returned instead of the variable names.
`components`	boolean expression, set to FALSE by default. Indicates that the formula being considered is constructed from a list of components, and thus the term 'intercept' is permitted.

Value

vector of variable names.

gmmEstimate *GMM estimate of TE under point identification*

Description

If the user sets the argument `point = TRUE` in the function `ivmte`, then it is assumed that the treatment effect parameter is point identified. The observational equivalence condition is then set up as a two-step GMM problem. Solving this GMM problem recovers the coefficients on the MTR functions m0 and m1. Combining these coefficients with the target gamma moments allows one to estimate the target treatment effect.

Usage

```
gmmEstimate(
  sset,
  gstar0,
  gstar1,
  center = NULL,
  subsetList = NULL,
  n = NULL,
  redundant = NULL,
  identity = FALSE,
  nMoments,
  splines,
  noisy = TRUE
)
```

Arguments

`sset`	a list of lists constructed from the function genSSet. Each inner list should include a coefficient corresponding to a term in an IV specification, a matrix of the estimates of the gamma moments conditional on (X, Z) for the control group, and a matrix of the estimates of the gamma moments conditional on (X, Z) for the treated group. The column means of the last two matrices is what is used to generate the gamma moments.
`gstar0`	vector, the target gamma moments for the control group.
`gstar1`	vector, the target gamma moments for the treated group.
`center`	numeric, the GMM moment equations from the original sample. When bootstrapping, the solution to the point identified case obtained from the original sample can be passed through this argument to recenter the bootstrap distribution of the J-statistic.
`subsetList`	list of subset indexes, one for each IV-like specification.
`n`	number of observations in the data. This option is only used when subsetting is involved.
`redundant`	vector of integers indicating which components in the S-set are redundant.

`identity` boolean, default set to `FALSE`. Set to `TRUE` if GMM point estimate should use the identity weighting matrix (i.e. one-step GMM).

`nMoments` number of linearly independent moments. This option is used to determine the cause of underidentified cases.

`splines` boolean, set to `TRUE` if the MTRs involve splines. This option is used to determine the cause of underidentified cases.

`noisy` boolean, default set to `TRUE`. If `TRUE`, then messages are provided throughout the estimation procedure. Set to `FALSE` to suppress all messages, e.g. when performing the bootstrap.

Value

a list containing the point estimate of the treatment effects, and the MTR coefficient estimates. The moment conditions evaluated at the solution are also returned, along with the J-test results. However, if the option `center` is passed, then the moment conditions and J-test are centered (this is to perform the J-test via bootstrap).

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare empty list to be updated (in the event multiple IV like
## specifications are provided
sSet <- list()

## Declare MTR formulas
formula1 = ~ 0 + u
formula0 = ~ 0 + u

## Construct object that separates out non-spline components of MTR
## formulas from the spline components. The MTR functions are
## obtained from this object by the function 'genSSet'.
splinesList = list(removeSplines(formula0), removeSplines(formula1))

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)
polynomials1 <- polyparse(formula = formula0,
                 data = dtm,
                 uname = u,
                 as.function = FALSE)

## Generate propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Generate IV estimates
ivEstimates <- ivEstimate(formula = ey ~ d | z,
```

```
                         data = dtm,
                         components = l(intercept, d),
                         treat = d,
                         list = FALSE)

## Generate target gamma moments
targetGamma <- genTarget(treat = "d",
                        m0 = ~ 1 + u,
                        m1 = ~ 1 + u,
                        target = "atu",
                        data = dtm,
                        splinesobj = splinesList,
                        pmodobj = propensityObj,
                        pm0 = polynomials0,
                        pm1 = polynomials1)

## Construct S-set. which contains the coefficients and weights
## corresponding to various IV-like estimands
sSet <- genSSet(data = dtm,
                sset = sSet,
                sest = ivEstimates,
                splinesobj = splinesList,
                pmodobj = propensityObj$phat,
                pm0 = polynomials0,
                pm1 = polynomials1,
                ncomponents = 2,
                scount = 1,
                yvar = "ey",
                dvar = "d",
                means = FALSE)

## Obtain point estimates using GMM
gmmEstimate(sset = sSet$sset,
            gstar0 = targetGamma$gstar0,
            gstar1 = targetGamma$gstar1)
```

`interactSplines` *Update splines object with list of interactions*

Description

Certain interactions between factor variables and splines should be dropped to avoid collinearity. Albeit collinearity in the MTR specification will not impact the bounds, it can substantially impact how costly it is to carry out the estimation. What this function does is map each spline to a temporary variable. A design matrix is then constructed using these temporary variables in place the splines. If an interaction involving one of the temporary variables is dropped, then one knows to also drop the corresponding interaction with the spline. Note that only interaction terms need to be omitted, so one does not need to worry about the formula contained in removeSplines$formula.

Usage

```
interactSplines(splinesobj, m0, m1, data, uname)
```

Arguments

`splinesobj`	list, consists of two elelments. The first is `removeSplines(m0)`, the second is `removeSplines(m1)`.
`m0`	one-sided formula for the marginal treatment response function for the control group. This should be the full MTR specificaiton (i.e. not the specification after removing the splines).
`m1`	one-sided formula for the marginal treatment response function for the treated group. This should be the full MTR specificaiton (i.e. not the specification after removing the splines).
`data`	data.frame, restricted to complete observations.
`uname`	string, name of the unobserved variable.

Value

An updated version of `splinesobj`.

`isfunctionstring` *Auxiliary function: check if string is command*

Description

Auxiliary function to check if a string is in fact a command, but in string form.

Usage

```
isfunctionstring(string)
```

Arguments

`string`	the string object to be checked.

Value

boolean expression.

ivEstimate *Obtaining IV-like specifications*

Description

This function estimates the IV-like estimands, as well as generates the weights associated with the IV-like specifications.

Usage

```
ivEstimate(
  formula,
  data,
  subset,
  components,
  treat,
  list = FALSE,
  order = NULL
)
```

Arguments

`formula`	formula to be estimated using OLS/IV.
`data`	`data.frame` with which to perform the estimation.
`subset`	subset condition with which to perform the estimate.
`components`	vector of variable names whose coefficients we want to include in the set of IV-like estimands.
`treat`	name of treatment indicator variable.
`list`	logical, set to TRUE if this function is being used to loop over a list of formulas.
`order`	integer, default set to `NULL`. This is simply an index of which IV-like specification the estimate corresponds to.

Value

Returns a list containing the matrices of IV-like specifications for `D = 0` and `D = 1`; and the estimates of the IV-like estimands.

Examples

```
dtm <- ivmte:::gendistMosquito()
ivEstimate(formula = ey ~ d | z,
           data = dtm,
           components = l(d),
           treat = d,
           list = FALSE)
```

ivmte *Instrumental Variables: Extrapolation by Marginal Treatment Effects*

Description

This function provides a general framework for using the marginal treatment effect (MTE) to extrapolate. The model is the same binary treatment instrumental variable (IV) model considered by Imbens and Angrist (1994) (doi: 10.2307/2951620) and Heckman and Vytlacil (2005) (doi: 10.1111/j.14680262.2005.00594.x). The framework on which this function is based was developed by Mogstad, Santos and Torgovitsky (2018) (doi: 10.3982/ECTA15463). See also the recent survey paper on extrapolation in IV models by Mogstad and Torgovitsky (2018) (doi: 10.1146/annurev-economics101617041813). A detailed description of the module and its features can be found in Shea and Torgovitsky (2021).

Usage

```
ivmte(
  data,
  target,
  late.from,
  late.to,
  late.X,
  genlate.lb,
  genlate.ub,
  target.weight0 = NULL,
  target.weight1 = NULL,
  target.knots0 = NULL,
  target.knots1 = NULL,
  m0,
  m1,
  uname = u,
  m1.ub,
  m0.ub,
  m1.lb,
  m0.lb,
  mte.ub,
  mte.lb,
  m0.dec,
  m0.inc,
  m1.dec,
  m1.inc,
  mte.dec,
  mte.inc,
  equal.coef,
  ivlike,
  components,
  subset,
```

```
  propensity,
  link = "logit",
  treat,
  outcome,
  solver,
  solver.options,
  solver.presolve,
  solver.options.criterion,
  solver.options.bounds,
  lpsolver,
  lpsolver.options,
  lpsolver.presolve,
  lpsolver.options.criterion,
  lpsolver.options.bounds,
  criterion.tol = 1e-04,
  initgrid.nx = 20,
  initgrid.nu = 20,
  audit.nx = 2500,
  audit.nu = 25,
  audit.add = 100,
  audit.max = 25,
  audit.tol,
  rescale,
  point,
  point.eyeweight = FALSE,
  bootstraps = 0,
  bootstraps.m,
  bootstraps.replace = TRUE,
  levels = c(0.99, 0.95, 0.9),
  ci.type = "backward",
  specification.test = TRUE,
  noisy = FALSE,
  smallreturnlist = FALSE,
  debug = FALSE
)
```

Arguments

`data`	`data.frame` or `data.table` used to estimate the treatment effects.
`target`	character, target parameter to be estimated. The function allows for ATE (`'ate'`), ATT (`'att'`), ATU (`'atu'`), LATE (`'late'`), and generalized LATE (`'genlate'`).
`late.from`	a named vector or a list declaring the baseline values of Z used to define the LATE. The name associated with each value should be the name of the corresponding variable.
`late.to`	a named vector or a list declaring the comparison set of values of Z used to define the LATE. The name associated with each value should be the name of the corresponding variable.

`late.X`	a named vector or a list declaring the values to condition on. The name associated with each value should be the name of the corresponding variable.
`genlate.lb`	lower bound value of unobservable `u` for estimating the generalized LATE.
`genlate.ub`	upper bound value of unobservable `u` for estimating the generalized LATE.
`target.weight0`	user-defined weight function for the control group defining the target parameter. A list of functions can be submitted if the weighting function is in fact a spline. The arguments of the function should be variable names in `data`. If the weight is constant across all observations, then the user can instead submit the value of the weight instead of a function.
`target.weight1`	user-defined weight function for the treated group defining the target parameter. See `target.weight0` for details.
`target.knots0`	user-defined set of functions defining the knots associated with spline weights for the control group. The arguments of the function should consist only of variable names in `data`. If the knots are constant across all observations, then the user can instead submit the vector of knots instead of a function.
`target.knots1`	user-defined set of functions defining the knots associated with spline weights for the treated group. See `target.knots0` for details.
`m0`	one-sided formula for the marginal treatment response function for the control group. Splines may also be incorporated using the expression `uSpline`, e.g. `uSpline(degree = 2, knots = c(0.4, 0.8), intercept = TRUE)`. The `intercept` argument may be omitted, and is set to `TRUE` by default.
`m1`	one-sided formula for the marginal treatment response function for the treated group. See `m0` for details.
`uname`	variable name for the unobservable used in declaring the MTRs. The name can be provided with or without quotation marks.
`m1.ub`	numeric value for upper bound on MTR for the treated group. By default, this will be set to the largest value of the observed outcome in the estimation sample.
`m0.ub`	numeric value for upper bound on MTR for the control group. By default, this will be set to the largest value of the observed outcome in the estimation sample.
`m1.lb`	numeric value for lower bound on MTR for the treated group. By default, this will be set to the smallest value of the observed outcome in the estimation sample.
`m0.lb`	numeric value for lower bound on MTR for the control group. By default, this will be set to the smallest value of the observed outcome in the estimation sample.
`mte.ub`	numeric value for upper bound on treatment effect parameter of interest.
`mte.lb`	numeric value for lower bound on treatment effect parameter of interest.
`m0.dec`	logical, set to `FALSE` by default. Set equal to `TRUE` if the MTR for the control group should be weakly monotone decreasing.
`m0.inc`	logical, set to `FALSE` by default. Set equal to `TRUE` if the MTR for the control group should be weakly monotone increasing.
`m1.dec`	logical, set to `FALSE` by default. Set equal to `TRUE` if the MTR for the treated group should be weakly monotone decreasing.

m1.inc
: logical, set to FALSE by default. Set equal to TRUE if the MTR for the treated group should be weakly monotone increasing.

mte.dec
: logical, set to FALSE by default. Set equal to TRUE if the MTE should be weakly monotone decreasing.

mte.inc
: logical, set to FALSE by default. Set equal to TRUE if the MTE should be weakly monotone increasing.

equal.coef
: one-sided formula to indicate which terms in m0 and m1 should be constrained to have the same coefficients. These terms therefore have no effect on the MTE.

ivlike
: formula or vector of formulas specifying the regressions for the IV-like estimands. Which coefficients to use to define the constraints determining the treatment effect bounds (alternatively, the moments determining the treatment effect point estimate) can be selected in the argument components. If no argument is passed, then a linear regression will be performed to estimate the MTR coefficients.

components
: a list of vectors of the terms in the regression specifications to include in the set of IV-like estimands. No terms should be in quotes. To select the intercept term, include the name intercept. If the factorized counterpart of a variable is included in the IV-like specifications, e.g. factor(x) where x = 1,2,3, the user can select the coefficients for specific factors by declaring the components factor(x)-1,factor(x)-2,factor(x)-3. See l on how to input the argument. If no components for a IV specification are given, then all coefficients from that IV specification will be used to define constraints in the partially identified case, or to define moments in the point identified case.

subset
: a single subset condition or list of subset conditions corresponding to each regression specified in ivlike. The input must be logical. See l on how to input the argument. If the user wishes to select specific rows, construct a binary variable in the data set, and set the condition to use only those observations for which the binary variable is 1, e.g. the binary variable is use, and the subset condition is use == 1.

propensity
: formula or variable name corresponding to propensity to take up treatment. If a formula is declared, then the function estimates the propensity score according to the formula and link specified in link. If a variable name is declared, then the corresponding column in the data is taken as the vector of propensity scores. A variable name can be passed either as a string (e.g propensity = 'p'), a variable (e.g. propensity = p), or a one-sided formula (e.g. propensity = ~p).

link
: character, name of link function to estimate propensity score. Can be chosen from 'linear', 'probit', or 'logit'. Default is set to 'logit'. The link should be provided with quoation marks.

treat
: variable name for treatment indicator. The name can be provided with or without quotation marks.

outcome
: variable name for outcome variable. The name can be provided with or without quotation marks.

solver
: character, name of the programming package in R used to obtain the bounds on the treatment effect. The function supports 'gurobi', 'cplexapi', rmosek, 'lpsolveapi'. The name of the solver should be provided with quotation marks.

`solver.options` list, each item of the list should correspond to an option specific to the solver selected.

`solver.presolve`
: boolean, default set to `TRUE`. Set this parameter to `FALSE` if presolve should be turned off for the LP/QCQP problems.

`solver.options.criterion`
: list, each item of the list should correspond to an option specific to the solver selected. These options are specific for finding the minimum criterion.

`solver.options.bounds`
: list, each item of the list should correspond to an option specific to the solver selected. These options are specific for finding the bounds.

`lpsolver` character, deprecated argument for `lpsolver`.

`lpsolver.options`
: list, deprecated argument for `solver.options`.

`lpsolver.presolve`
: boolean, deprecated argument for `solver.presolve`.

`lpsolver.options.criterion`
: list, deprecated argument for `solver.options.criterion`.

`lpsolver.options.bounds`
: list, deprecated argument for `solver.options.bounds`.

`criterion.tol` tolerance for the criterion function, and is set to 1e-4 by default. The criterion measures how well the IV-like moments/conditional means are matched using the l1-norm. Statistical noise may prohibit the theoretical LP/QCQP problem from being feasible. That is, there may not exist a set of MTR coefficients that are able to match all the specified moments. The function thus first estimates the minimum criterion, which is reported in the output under the name 'minimum criterion', with a criterion of 0 meaning that all moments were able to be matched. The function then relaxes the constraints by tolerating a criterion up to `minimum criterion * (1 + criterion.tol)`. Set `criterion.tol` to a value greater than 0 to allow for more conservative bounds.

`initgrid.nx` integer determining the number of points of the covariates used to form the initial constraint grid for imposing shape restrictions on the MTRs.

`initgrid.nu` integer determining the number of points in the open interval (0, 1) drawn from a Halton sequence. The end points 0 and 1 are additionally included. These points are always a subset of the points defining the audit grid (see `audit.nu`). These points are used to form the initial constraint grid for imposing shape restrictions on the `u` components of the MTRs.

`audit.nx` integer determining the number of points on the covariates space to audit in each iteration of the audit procedure.

`audit.nu` integer determining the number of points in the open interval (0, 1) drawn from a Halton sequence. The end points 0 and 1 are additionally included. These points are used to audit whether the shape restrictions on the `u` components of the MTRs are satisfied. The initial grid used to impose the shape constraints in the LP/QCQP problem are constructed from a subset of these points.

audit.add
: maximum number of points to add to the initial constraint grid for imposing each kind of shape constraint. For example, if there are 5 different kinds of shape constraints, there can be at most `audit.add * 5` additional points added to the constraint grid.

audit.max
: maximum number of iterations in the audit procedure.

audit.tol
: feasibility tolerance when performing the audit. By default to set to be 1e-06, which is equal to the default feasibility tolerances of Gurobi (`solver = "gurobi"`), CPLEX (`solver = "cplexapi"`), and Rmosek (`solver = "rmosek"`). This parameter should only be changed if the feasibility tolerance of the solver is changed, or if numerical issues result in discrepancies between the solver's feasibility check and the audit.

rescale
: boolean, set to `TRUE` by default. This rescalels the MTR components to improve stability in the LP/QCQP optimization.

point
: boolean. Set to `TRUE` if it is believed that the treatment effects are point identified. If set to `TRUE` and IV-like formulas are passed, then a two-step GMM procedure is implemented to estimate the treatment effects. Shape constraints on the MTRs will be ignored under point identification. If set to `TRUE` and the regression-based criteria is used instead, then OLS will be used to estimate the MTR coefficients used to estimate the treatment effect. If not declared, then the function will determine whether or not the target parameter is point identified.

point.eyeweight
: boolean, default set to `FALSE`. Set to `TRUE` if the GMM point estimate should use the identity weighting matrix (i.e. one-step GMM).

bootstraps
: integer, default set to 0. This determines the number of bootstraps used to perform statistical inference.

bootstraps.m
: integer, default set to size of data set. Determines the size of the subsample drawn from the original data set when performing inference via the bootstrap. This option applies only to the case of constructing confidence intervals for treatment effect bounds, i.e. it does not apply when `point = TRUE`.

bootstraps.replace
: boolean, default set to `TRUE`. This determines whether the resampling procedure used for inference will sample with replacement.

levels
: vector of real numbers between 0 and 1. Values correspond to the level of the confidence intervals constructed via bootstrap.

ci.type
: character, default set to `'both'`. Set to `'forward'` to construct the forward confidence interval for the treatment effect bound. Set to `'backward'` to construct the backward confidence interval for the treatment effect bound. Set to `'both'` to construct both types of confidence intervals.

specification.test
: boolean, default set to `TRUE`. Function performs a specification test for the partially identified case when `bootstraps > 0`.

noisy
: boolean, default set to `TRUE`. If `TRUE`, then messages are provided throughout the estimation procedure. Set to `FALSE` to suppress all messages, e.g. when performing the bootstrap.

smallreturnlist
: boolean, default set to FALSE. Set to TRUE to exclude large intermediary components (i.e. propensity score model, LP/QCQP model, bootstrap iterations) from being included in the return list.

debug
: boolean, indicates whether or not the function should provide output when obtaining bounds. The option is only applied when solver = 'gurobi' or solver = 'rmosek'. The output provided is the same as what the Gurobi API would send to the console.

Details

When the function is used to estimate bounds, and statistical inference is not performed, the function returns the following objects.

audit.count the number of audits required until there were no more violations; or the number of audits performed before the audit procedure was terminated.

audit.criterion the minimum criterion.

audit.grid a list containing the points used to define the audit grid, as well as a table of points where the shape constraints were violated.

bounds a vector with the estimated lower and upper bounds of the target treatment effect.

call.options a list containing all the model specifications and call options generating the results.

gstar a list containing the estimate of the weighted means for each component in the MTRs. The weights are determined by the target parameter declared in target, or the weights defined by target.weight1, target.knots1, target.weight0, target.knots0.

gstar.coef a list containing the coefficients on the treated and control group MTRs.

gstar.weights a list containing the target weights used to estimate gstar.

result a list containing the LP/QCQP model, and the full output from solving the problem.

solver the solver used in estimation.

moments the number of elements in the S-set used to generate achieve (partial) identification.

propensity the propensity score model. If a variable is fed to the propensity argument when calling ivmte, then the returned object is a list containing the name of variable given by the user, and the values of that variable used in estimation.

s.set a list of all the coefficient estimates and weights corresponding to each element in the S-set.

splines.dict a list including the specifications of each spline declared in each MTR.

messages a vector of character strings logging the output of the estimation procedure.

If bootstraps is greater than 0, then statistical inference will be performed and the output will additionally contain the following objects.

bootstraps the number of bootstraps.

bootstraps.failed the number of bootstraps that failed (e.g. due to collinearity) and had to be repeated.

bounds.bootstraps the estimates of the bounds from every bootstrap draw.

bounds.ci forward and/or backward confidence intervals for the bound estimates at the levels specified in levels.

bounds.se bootstrap standard errors on the lower and upper bound estimates.

p.value p-value for the estimated bounds. p-values are constructed by finding the level at which the confidence interval no longer contains 0.

propensity.ci confidence interval for coefficient estimates of the propensity score model.

propensity.se standard errors for the coefficient estimates of the propensity score model.

specification.p.value p-value from a specification test. The specification test is only performed if the minimum criterion is not 0.

If `point = TRUE` and `bootstraps = 0`, then point estimation is performed using two-step GMM. The output will contain the following objects.

j.test test statistic and results from the asymptotic J-test.

moments a vector. Each element is the GMM criterion for each moment condition used in estimation.

mtr.coef coefficient estimates for the MTRs.

point.estimate point estimate of the treatment effect.

redundant indexes for the moment conditions (i.e. elements in the S set) that were linearly independent and could be dropped.

If `point = TRUE` and `bootstraps` is not 0, then point estimation is performed using two-step GMM, and additional statistical inference is performed using the bootstrap samples. The output will contain the following additional objects.

bootstraps the number of bootstraps.

bootstraps.failed the number of bootstraps that failed (e.g. due to collinearity) and had to be repeated.

j.test test statistic and result from the J-test performed using the bootstrap samples.

j.test.bootstraps J-test statistic from each bootstrap.

mtr.bootstraps coefficient estimates for the MTRs from each bootstrap sample. These are used to construct the confidence intervals and standard errors for the MTR coefficients.

mtr.ci confidence intervals for each MTR coefficient.

mtr.se standard errors for each MTR coefficient estimate.

p.value p-value for the treatment effect point estimate estimated using the bootstrap.

point.estimate.bootstraps treatment effect point estimate from each bootstrap sample. These are used to construct the confidence interval, standard error, and p-value for the treatment effect.

point.estimate.ci confidence interval for the treatment effect.

point.estimate.se standard error for the treatment effect estimate.

propensity.ci confidence interval for the coefficients in the propensity score model, constructed using the bootstrap.

propensity.se standard errors for the coefficient estimates of the propensity score model.

Value

Returns a list of results from throughout the estimation procedure. This includes all IV-like estimands; the propensity score model; bounds on the treatment effect; the estimated expectations of each term in the MTRs; the components and results of the LP/QCQP problem.

Examples

```
dtm <- ivmte:::gendistMosquito()

ivlikespecs <- c(ey ~ d | z,
                 ey ~ d | factor(z),
                 ey ~ d,
                 ey ~ d | factor(z))
jvec <- l(d, d, d, d)
svec <- l(, , , z %in% c(2, 4))

ivmte(ivlike = ivlikespecs,
      data = dtm,
      components = jvec,
      propensity = d ~ z,
      subset = svec,
      m0 = ~  u + I(u ^ 2),
      m1 = ~  u + I(u ^ 2),
      uname = u,
      target = "att",
      m0.dec = TRUE,
      m1.dec = TRUE,
      bootstraps = 0,
      solver = "lpSolveAPI")
```

`ivmteEstimate` *Single iteration of estimation procedure from Mogstad, Torgovitsky, Santos (2018)*

Description

This function estimates the treatment effect parameters, following the procedure described in Mogstad, Santos and Torgovitsky (2018) (doi: 10.3982/ECTA15463). A detailed description of the module and its features can be found in Shea and Torgovitsky (2021). However, this is not the main function of the module. See `ivmte` for the main function. For examples of how to use the package, see the vignette, which is available on the module's GitHub page.

Usage

```
ivmteEstimate(
  data,
  target,
  late.Z,
  late.from,
  late.to,
  late.X,
  eval.X,
  genlate.lb,
```

```
genlate.ub,
target.weight0,
target.weight1,
target.knots0 = NULL,
target.knots1 = NULL,
m0,
m1,
uname = u,
m1.ub,
m0.ub,
m1.lb,
m0.lb,
mte.ub,
mte.lb,
m0.dec,
m0.inc,
m1.dec,
m1.inc,
mte.dec,
mte.inc,
equal.coef,
ivlike,
components,
subset,
propensity,
link = "logit",
treat,
solver,
solver.options,
solver.presolve,
solver.options.criterion,
solver.options.bounds,
criterion.tol = 0.01,
initgrid.nx = 20,
initgrid.nu = 20,
audit.nx = 2500,
audit.nu = 25,
audit.add = 100,
audit.max = 25,
audit.tol,
audit.grid = NULL,
rescale = TRUE,
point = FALSE,
point.eyeweight = FALSE,
point.center = NULL,
point.redundant = NULL,
bootstrap = FALSE,
count.moments = TRUE,
```

```
  orig.sset = NULL,
  orig.criterion = NULL,
  vars_y,
  vars_mtr,
  terms_mtr0,
  terms_mtr1,
  vars_data,
  splinesobj,
  splinesobj.equal,
  noisy = TRUE,
  smallreturnlist = FALSE,
  debug = FALSE,
  environments
)
```

Arguments

- `data` — `data.frame` or `data.table` used to estimate the treatment effects.
- `target` — character, target parameter to be estimated. The function allows for ATE (`'ate'`), ATT (`'att'`), ATU (`'atu'`), LATE (`'late'`), and generalized LATE (`'genlate'`).
- `late.Z` — vector of variable names used to define the LATE.
- `late.from` — baseline set of values of Z used to define the LATE.
- `late.to` — comparison set of values of Z used to define the LATE.
- `late.X` — vector of variable names of covariates to condition on when defining the LATE.
- `eval.X` — numeric vector of the values to condition variables in `late.X` on when estimating the LATE.
- `genlate.lb` — lower bound value of unobservable `u` for estimating the generalized LATE.
- `genlate.ub` — upper bound value of unobservable `u` for estimating the generalized LATE.
- `target.weight0` — user-defined weight function for the control group defining the target parameter. A list of functions can be submitted if the weighting function is in fact a spline. The arguments of the function should be variable names in `data`. If the weight is constant across all observations, then the user can instead submit the value of the weight instead of a function.
- `target.weight1` — user-defined weight function for the treated group defining the target parameter. See `target.weight0` for details.
- `target.knots0` — user-defined set of functions defining the knots associated with spline weights for the control group. The arguments of the function should consist only of variable names in `data`. If the knots are constant across all observations, then the user can instead submit the vector of knots instead of a function.
- `target.knots1` — user-defined set of functions defining the knots associated with spline weights for the treated group. See `target.knots0` for details.
- `m0` — one-sided formula for the marginal treatment response function for the control group. Splines may also be incorporated using the expression `uSpline`, e.g. `uSpline(degree = 2,knots = c(0.4,0.8),intercept = TRUE)`. The `intercept` argument may be omitted, and is set to `TRUE` by default.

m1	one-sided formula for the marginal treatment response function for the treated group. See m0 for details.
uname	variable name for the unobservable used in declaring the MTRs. The name can be provided with or without quotation marks.
m1.ub	numeric value for upper bound on MTR for the treated group. By default, this will be set to the largest value of the observed outcome in the estimation sample.
m0.ub	numeric value for upper bound on MTR for the control group. By default, this will be set to the largest value of the observed outcome in the estimation sample.
m1.lb	numeric value for lower bound on MTR for the treated group. By default, this will be set to the smallest value of the observed outcome in the estimation sample.
m0.lb	numeric value for lower bound on MTR for the control group. By default, this will be set to the smallest value of the observed outcome in the estimation sample.
mte.ub	numeric value for upper bound on treatment effect parameter of interest.
mte.lb	numeric value for lower bound on treatment effect parameter of interest.
m0.dec	logical, set to FALSE by default. Set equal to TRUE if the MTR for the control group should be weakly monotone decreasing.
m0.inc	logical, set to FALSE by default. Set equal to TRUE if the MTR for the control group should be weakly monotone increasing.
m1.dec	logical, set to FALSE by default. Set equal to TRUE if the MTR for the treated group should be weakly monotone decreasing.
m1.inc	logical, set to FALSE by default. Set equal to TRUE if the MTR for the treated group should be weakly monotone increasing.
mte.dec	logical, set to FALSE by default. Set equal to TRUE if the MTE should be weakly monotone decreasing.
mte.inc	logical, set to FALSE by default. Set equal to TRUE if the MTE should be weakly monotone increasing.
equal.coef	one-sided formula to indicate which terms in m0 and m1 should be constrained to have the same coefficients. These terms therefore have no effect on the MTE.
ivlike	formula or vector of formulas specifying the regressions for the IV-like estimands. Which coefficients to use to define the constraints determining the treatment effect bounds (alternatively, the moments determining the treatment effect point estimate) can be selected in the argument components. If no argument is passed, then a linear regression will be performed to estimate the MTR coefficients.
components	a list of vectors of the terms in the regression specifications to include in the set of IV-like estimands. No terms should be in quotes. To select the intercept term, include the name intercept. If the factorized counterpart of a variable is included in the IV-like specifications, e.g. factor(x) where x = 1,2,3, the user can select the coefficients for specific factors by declaring the components factor(x)-1,factor(x)-2,factor(x)-3. See l on how to input the argument. If no components for a IV specification are given, then all coefficients from that IV specification will be used to define constraints in the partially identified case, or to define moments in the point identified case.

subset
: a single subset condition or list of subset conditions corresponding to each regression specified in `ivlike`. The input must be logical. See `l` on how to input the argument. If the user wishes to select specific rows, construct a binary variable in the data set, and set the condition to use only those observations for which the binary variable is 1, e.g. the binary variable is `use`, and the subset condition is `use == 1`.

propensity
: formula or variable name corresponding to propensity to take up treatment. If a formula is declared, then the function estimates the propensity score according to the formula and link specified in `link`. If a variable name is declared, then the corresponding column in the data is taken as the vector of propensity scores. A variable name can be passed either as a string (e.g `propensity = 'p'`), a variable (e.g. `propensity = p`), or a one-sided formula (e.g. `propensity = ~p`).

link
: character, name of link function to estimate propensity score. Can be chosen from `'linear'`, `'probit'`, or `'logit'`. Default is set to `'logit'`. The link should be provided with quoation marks.

treat
: variable name for treatment indicator. The name can be provided with or without quotation marks.

solver
: character, name of the programming package in R used to obtain the bounds on the treatment effect. The function supports `'gurobi'`, `'cplexapi'`, `rmosek`, `'lpsolveapi'`. The name of the solver should be provided with quotation marks.

solver.options
: list, each item of the list should correspond to an option specific to the solver selected.

solver.presolve
: boolean, default set to `TRUE`. Set this parameter to `FALSE` if presolve should be turned off for the LP/QCQP problems.

solver.options.criterion
: list, each item of the list should correspond to an option specific to the solver selected. These options are specific for finding the minimum criterion.

solver.options.bounds
: list, each item of the list should correspond to an option specific to the solver selected. These options are specific for finding the bounds.

criterion.tol
: tolerance for the criterion function, and is set to 1e-4 by default. The criterion measures how well the IV-like moments/conditional means are matched using the l1-norm. Statistical noise may prohibit the theoretical LP/QCQP problem from being feasible. That is, there may not exist a set of MTR coefficients that are able to match all the specified moments. The function thus first estimates the minimum criterion, which is reported in the output under the name 'minimum criterion', with a criterion of 0 meaning that all moments were able to be matched. The function then relaxes the constraints by tolerating a criterion up to `minimum criterion * (1 + criterion.tol)`. Set `criterion.tol` to a value greater than 0 to allow for more conservative bounds.

initgrid.nx
: integer determining the number of points of the covariates used to form the initial constraint grid for imposing shape restrictions on the MTRs.

initgrid.nu
: integer determining the number of points in the open interval (0, 1) drawn from a Halton sequence. The end points 0 and 1 are additionally included. These points

are always a subset of the points defining the audit grid (see `audit.nu`). These points are used to form the initial constraint grid for imposing shape restrictions on the `u` components of the MTRs.

`audit.nx` integer determining the number of points on the covariates space to audit in each iteration of the audit procedure.

`audit.nu` integer determining the number of points in the open interval (0, 1) drawn from a Halton sequence. The end points 0 and 1 are additionally included. These points are used to audit whether the shape restrictions on the `u` components of the MTRs are satisfied. The initial grid used to impose the shape constraints in the LP/QCQP problem are constructed from a subset of these points.

`audit.add` maximum number of points to add to the initial constraint grid for imposing each kind of shape constraint. For example, if there are 5 different kinds of shape constraints, there can be at most `audit.add * 5` additional points added to the constraint grid.

`audit.max` maximum number of iterations in the audit procedure.

`audit.tol` feasibility tolerance when performing the audit. By default to set to be 1e-06, which is equal to the default feasibility tolerances of Gurobi (`solver = "gurobi"`), CPLEX (`solver = "cplexapi"`), and Rmosek (`solver = "rmosek"`). This parameter should only be changed if the feasibility tolerance of the solver is changed, or if numerical issues result in discrepancies between the solver's feasibility check and the audit.

`audit.grid` list, contains the `A` matrix used in the audit for the original sample, as well as the RHS vector used in the audit from the original sample.

`rescale` boolean, set to `TRUE` by default. This rescalels the MTR components to improve stability in the LP/QCQP optimization.

`point` boolean. Set to `TRUE` if it is believed that the treatment effects are point identified. If set to `TRUE` and IV-like formulas are passed, then a two-step GMM procedure is implemented to estimate the treatment effects. Shape constraints on the MTRs will be ignored under point identification. If set to `TRUE` and the regression-based criteria is used instead, then OLS will be used to estimate the MTR coefficients used to estimate the treatment effect. If not declared, then the function will determine whether or not the target parameter is point identified.

`point.eyeweight` boolean, default set to `FALSE`. Set to `TRUE` if the GMM point estimate should use the identity weighting matrix (i.e. one-step GMM).

`point.center` numeric, a vector of GMM moment conditions evaluated at a solution. When bootstrapping, the moment conditions from the original sample can be passed through this argument to recenter the bootstrap distribution of the J-statistic.

`point.redundant` vector of integers indicating which components in the S-set are redundant.

`bootstrap` boolean, indicates whether the estimate is for the bootstrap.

`count.moments` boolean, indicate if number of linearly independent moments should be counted.

`orig.sset` list, only used for bootstraps. The list contains the gamma moments for each element in the S-set, as well as the IV-like coefficients.

- `orig.criterion` numeric, only used for bootstraps. The scalar corresponds to the minimum observational equivalence criterion from the original sample.
- `vars_y` character, variable name of observed outcome variable.
- `vars_mtr` character, vector of variables entering into `m0` and `m1`.
- `terms_mtr0` character, vector of terms entering into `m0`.
- `terms_mtr1` character, vector of terms entering into `m1`.
- `vars_data` character, vector of variables that can be found in the data.
- `splinesobj` list of spline components in the MTRs for treated and control groups. Spline terms are extracted using `removeSplines`. This object is supposed to be a dictionary of splines, containing the original calls of each spline in the MTRs, their specifications, and the index used for naming each basis spline.
- `splinesobj.equal` list of spline components in the MTRs for treated and control groups. The structure of `splinesobj.equal` is the same as `splinesobj`, except the splines are restricted to those whose MTR cofficients should be constrained to be equal across treatment groups.
- `noisy` boolean, default set to `TRUE`. If `TRUE`, then messages are provided throughout the estimation procedure. Set to `FALSE` to suppress all messages, e.g. when performing the bootstrap.
- `smallreturnlist` boolean, default set to `FALSE`. Set to `TRUE` to exclude large intermediary components (i.e. propensity score model, LP/QCQP model, bootstrap iterations) from being included in the return list.
- `debug` boolean, indicates whether or not the function should provide output when obtaining bounds. The option is only applied when `solver = 'gurobi'` or `solver = 'rmosek'`. The output provided is the same as what the Gurobi API would send to the console.
- `environments` a list containing the environments of the MTR formulas, the IV-like formulas, and the propensity score formulas. If a formula is not provided, and thus no environment can be found, then the parent.frame() is assigned by default.

Details

The treatment effects parameters the user can choose from are the ATE, ATT, ATU, LATE, and generalized LATE. The user is required to provide a polynomial expression for the marginal treatment responses (MTR), as well as a set of regressions.

There are two approaches to estimating the treatment effect parameters. The first approach restricts the set of MTR coefficients on each term of the MTRs to be consistent with the regression estimates from the specifications passed through `ivlike`. The bounds on the treatment effect parameter correspond to finding coefficients on the MTRs that maximize their average difference. If the model is point identified, then GMM is used for estimation. Otherwise, the function solves an LP problem. The second approach restricts the set of MTR coefficients to fit the conditional mean of the outcome variable. If the model is point identified, then constrained least squares is used for estimation. Otherwise, the function solves a QCQP.

The estimation procedure relies on the propensity to take up treatment. The propensity scores can either be estimated as part of the estimation procedure, or the user can specify a variable in the data set already containing the propensity scores.

Constraints on the shape of the MTRs and marginal treatment effects (MTE) can be imposed by the user. Specifically, bounds and monotonicity restrictions are permitted. These constraints are first enforced over a subset of points in the data. An iterative audit procedure is then performed to ensure the constraints hold more generally.

Value

Returns a list of results from throughout the estimation procedure. This includes all IV-like estimands; the propensity score model; bounds on the treatment effect; the estimated expectations of each term in the MTRs; the components and results of the LP/QCQP problem.

`ivmteSimData` *ivmte Simulated Data*

Description

ivmte Simulated Data

Usage

```
ivmteSimData
```

Format

A data frame with 5,000 rows and 14 columns.

y binary outcome variable

d binary treatment variable

z instrument that takes the value 0, 1, 2, or 3

x covariate x that takes integer values from 1 to 10

Source

Simulated — see code in data/ivmteSimData.R.

l — *Listing subsets and components*

Description

This function allows the user to declare a list of variable names in non-character form and subsetting conditions. This is used to ensure clean entry of arguments into the `components` and `subset` arguments of the function. When selecting components to include in the S set, selecting the intercept term and factor variables requires special treatment. To select the intercept term, include in the vector of variable names, 'intercept'. If the the factorized counterpart of a variable `x = 1,2,3` is included in the IV-like specifications via `factor(x)`, the user can select the coefficients for specific factors by declaring the components `factor(x)-1,factor(x)-2,factor(x)-3`.

Usage

```
l(...)
```

Arguments

`...`	subset conditions or variable names

Value

list.

Examples

```
components <- l(d, x1, intercept, factor(x)-2)
subsets <- l(, z %in% c(2, 4))
```

lpSetup — *Constructing LP problem*

Description

If the user passes IV-like moments to the function, then the function constructs the components of the LP problem. If no IV-like moments are passed, then the function constructs the linear constraints of the QCQP problem. Note that the LP/QCQP model will be saved inside an environment variable, which is to be passed through the argument `env`. This is done for efficient use of memory. The environment `env` is supposed to already contain a list under the entry `$mbobj` containing the matrices defining the shape constraints. This list of shape constraints `$mbobj` should contain three entries corresponding to a system of linear equations of the form `Ax <=> b`: `mbA`, the matrix defining the constraints, `A`; `mbs`, a vector indicating whether a row in `mbA` is an equality or inequality constraint (for Gurobi and MOSEK, use '<=', '>=', '='; for CPLEX, use 'L', 'G', and 'E'); `mbrhs`, a vector of the right hand side values defining the constraint of the form i.e. the vector `b`. Depending on the linear programming solver used, this function will return different output specific to the solver.

Usage

```
lpSetup(
  env,
  sset,
  orig.sset = NULL,
  equal.coef0 = NULL,
  equal.coef1 = NULL,
  shape = TRUE,
  direct = FALSE,
  rescale = TRUE,
  solver
)
```

Arguments

env	environment containing the matrices defining the LP/QCQP problem.
sset	List of IV-like estimates and the corresponding gamma terms.
orig.sset	list, only used for bootstraps. The list contains the gamma moments for each element in the S-set, as well as the IV-like coefficients.
equal.coef0	character, name of terms in m0 that should have common coefficients with the corresponding terms in m1.
equal.coef1	character, name of terms in m1 that should have common coefficients with the corresponding terms in m0.
shape	boolean, default set to TRUE. Switch to determine whether or not to include shape restrictions in the LP/QCQP problem.
direct	boolean, set to TRUE if the direct MTR regression is used.
rescale	boolean, set to TRUE if the MTR components should be rescaled to improve stability in the LP/QCQP optimization.
solver	string, name of the package used to solve the LP/QCQP problem.

Value

A list of matrices and vectors necessary to define an LP/QCQP problem.

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare empty list to be updated (in the event multiple IV like
## specifications are provided
sSet <- list()

## Declare MTR formulas
formula0 = ~ 1 + u
formula1 = ~ 1 + u

## Construct object that separates out non-spline components of MTR
```

```
## formulas from the spline components. The MTR functions are
## obtained from this object by the function 'genSSet'.
splinesList = list(removeSplines(formula0), removeSplines(formula1))

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                          data = dtm,
                          uname = u,
                          as.function = FALSE)
polynomials1 <- polyparse(formula = formula1,
                          data = dtm,
                          uname = u,
                           as.function = FALSE)

## Generate propensity score model
propensityObj <- propensity(formula = d ~ z,
                            data = dtm,
                            link = "linear")

## Generate IV estimates
ivEstimates <- ivEstimate(formula = ey ~ d | z,
                          data = dtm,
                          components = l(intercept, d),
                          treat = d,
                          list = FALSE)

## Generate target gamma moments
targetGamma <- genTarget(treat = "d",
                         m0 = ~ 1 + u,
                         m1 = ~ 1 + u,
                         target = "atu",
                         data = dtm,
                         splinesobj = splinesList,
                         pmodobj = propensityObj,
                         pm0 = polynomials0,
                         pm1 = polynomials1)

## Construct S-set. which contains the coefficients and weights
## corresponding to various IV-like estimands
sSet <- genSSet(data = dtm,
                sset = sSet,
                sest = ivEstimates,
                splinesobj = splinesList,
                pmodobj = propensityObj$phat,
                pm0 = polynomials0,
                pm1 = polynomials1,
                ncomponents = 2,
                scount = 1,
                yvar = "ey",
                dvar = "d",
                means = TRUE)
## Only the entry $sset is required
sSet <- sSet$sset
```

```
## Define additional upper- and lower-bound constraints for the LP
## problem.  The code below imposes a lower bound of 0.2 and upper
## bound of 0.8 on the MTRs.
A <- matrix(0, nrow = 22, ncol = 4)
A <- cbind(A, rbind(cbind(1, seq(0, 1, 0.1)),
                    matrix(0, nrow = 11, ncol = 2)))
A <- cbind(A, rbind(matrix(0, nrow = 11, ncol = 2),
                    cbind(1, seq(0, 1, 0.1))))
sense <- c(rep(">", 11), rep("<", 11))
rhs <- c(rep(0.2, 11), rep(0.8, 11))

## Construct LP object to be interpreted and solved by
## lpSolveAPI. Note that an environment has to be created for the LP
## object. The matrices defining the shape restrictions must be stored
## as a list under the entry \code{$mbobj} in the environment.
modelEnv <- new.env()
modelEnv$mbobj <- list(mbA = A,
                    mbs = sense,
                    mbrhs = rhs)
## Convert the matrices defining the shape constraints into a format
## that is suitable for the LP solver.
lpSetup(env = modelEnv,
        sset = sSet,
        solver = "lpsolveapi")
## Setup LP model so that it is solving for the bounds.
lpSetupBound(env = modelEnv,
             g0 = targetGamma$gstar0,
             g1 = targetGamma$gstar1,
             sset = sSet,
             criterion.tol = 0,
             criterion.min = 0,
             solver = "lpsolveapi")
## Declare any LP solver options as a list.
lpOptions <- optionsLpSolveAPI(list(epslevel = "tight"))
## Obtain the bounds.
bounds <- bound(env = modelEnv,
                sset = sSet,
                solver = "lpsolveapi",
                solver.options = lpOptions)
cat("The bounds are [",  bounds$min, ",", bounds$max, "].\n")
```

`lpSetupBound` *Configure LP environment for obtaining the bounds*

Description

This function sets up the LP model so that the bounds can be obtained. The LP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
lpSetupBound(
  env,
  g0,
  g1,
  sset,
  criterion.tol,
  criterion.min,
  solver,
  setup = TRUE
)
```

Arguments

`env`	the environment containing the LP model.
`g0`	set of expectations for each terms of the MTR for the control group.
`g1`	set of expectations for each terms of the MTR for the control group.
`sset`	a list containing the point estimates and gamma components associated with each element in the S-set. This object is only used to determine the names of terms. If it is no submitted, then no names are provided to the solution vector.
`criterion.tol`	additional multiplicative factor for how much more the solution is permitted to violate observational equivalence of the IV-like estimands, i.e. `1 + criterion.tol` will multiply `criterion.min` directly.
`criterion.min`	minimum criterion, i.e. minimum deviation from observational equivalence while satisfying shape constraints.
`solver`	string, name of the package used to solve the LP problem.
`setup`	boolean. If `TRUE`, the function will modify the LP environment so that the LP solver can obtain the bounds. If `FALSE`, then it will undo the changes made by the function if `setup = TRUE`.

Value

Nothing, as this modifies an environment variable to save memory.

`lpSetupCriterion` *Configure LP environment for minimizing the criterion*

Description

This function sets up the objective function for minimizing the criterion. The LP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
lpSetupCriterion(env, sset)
```

Arguments

`env`	The LP environment
`sset`	List of IV-like estimates and the corresponding gamma terms.

Value

Nothing, as this modifies an environment variable to save memory.

`lpSetupCriterionBoot` *Configure LP environment for specification testing*

Description

This function re-centers various objects in the LP environment so that a specification test can be performed via the bootstrap. The LP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
lpSetupCriterionBoot(
  env,
  sset,
  orig.sset,
  orig.criterion,
  criterion.tol = 0,
  setup = TRUE
)
```

Arguments

`env`	the LP environment
`sset`	list of IV-like estimates and the corresponding gamma terms.
`orig.sset`	list, only used for bootstraps. The list caontains the gamma moments for each element in the S-set, as well as the IV-like coefficients.
`orig.criterion`	scalar, only used for bootstraps. This is the minimum criterion from the original sample.
`criterion.tol`	tolerance for violation of observational equivalence, set to 0 by default.
`setup`	boolean. If `TRUE`, the function will modify the LP environment so that the LP solver can obtain the test statistic for the specification test. If `FALSE`, then it will undo the changes made by the function if `setup = TRUE`.

Value

Nothing, as this modifies an environment variable to save memory.

`lpSetupEqualCoef`	*Generate equality constraints*

Description

This function generates the linear constraints to ensure that certain MTR coefficients are constant across the treatment and control group.

Usage

```
lpSetupEqualCoef(equal.coef0, equal.coef1, ANames)
```

Arguments

`equal.coef0`	character, name of terms in `m0` that should have common coefficients with the corresponding terms in `m1`.
`equal.coef1`	character, name of terms in `m1` that should have common coefficients with the corresponding terms in `m0`.
`ANames`	character, name of all terms in `m0` and `m1`. The names of the terms corresponding to the treatment and control groups should be distinguishable. For example, all terms for `m0` may contain a prefix '[m0]', and all terms for `m1` may contain a prefix '[m1]'. All the terms in `equal.coef0` and `equal.coef1` should be contained in `ANames`.

Value

A list, containing the matrix of linear equality constraints, a vector of equal signs, and a vector of 0s.

`lpSetupInfeasible`	*Configure LP environment for diagnostics*

Description

This function separates the shape constraints from the LP environment. That way, the model can be solved without any shape constraints, which is the primary cause of infeasibility. This is done in order to check which shape constraints are causing the model to be infeasible. The LP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
lpSetupInfeasible(env, sset)
```

Arguments

`env`	The LP environment
`sset`	List of IV-like estimates and the corresponding gamma terms.

Value

Nothing, as this modifies an environment variable to save memory.

`lpSetupSolver` *Configure LP environment to be compatible with solvers*

Description

This alters the LP environment so the model will be compatible with specific solvers. The LP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
lpSetupSolver(env, solver)
```

Arguments

`env`	The LP environment
`solver`	Character, the LP solver.

Value

Nothing, as this modifies an environment variable to save memory.

`magnitude` *Check magnitude of real number*

Description

This function returns the order of magnitude of a a number.

Usage

```
magnitude(x)
```

Arguments

`x`	The number to be checked.

Value

An integer indicating the order of magnitude.

matrixTriplets *Convert matrix into triplet form*

Description

This function converts matrices into triplet form for Mosek. This is required in order to declare quadratic programming problems and second-order cone programming problems.

Usage

```
matrixTriplets(mat, lower = TRUE)
```

Arguments

mat	A matrix.
lower	Boolean, set to TRUE if matrix is symmetric, and only its lower triangle should be returned.

Value

A list containing vectors of row and column indexes, and matrix values.

mInt *Function to generate integral of m0 and m1*

Description

Function carries out integral for a polynomial of degree 3.

Usage

```
mInt(ub, lb, coef)
```

Arguments

ub	scalar, upper bound of the integral.
lb	scalar, lower bound of the integral.
coef	vector, polynomial coefficients.

Value

scalar.

modcall — *Auxiliary function: modifying calls*

Description

This function can be used to modify calls in several ways.

Usage

```
modcall(call, newcall, newargs, keepargs, dropargs)
```

Arguments

`call`	Call object to be modified.
`newcall`	New function to be called.
`newargs`	List, new arguments and their values.
`keepargs`	List, arguments in original call to keep, with the rest being dropped.
`dropargs`	List, arguments in original call to drop, with the rest being kept.

Value

New call object.

momentMatrix — *Construct pre-meaned moment matrix*

Description

This function constructs the matrix to be fed into the GMM estimator to construct the moment conditions.

Usage

```
momentMatrix(sset, gn0, gn1, subsetList = NULL, n = NULL)
```

Arguments

`sset`	a list of lists constructed from the function genSSet. Each inner list should include a coefficient corresponding to a term in an IV specification, a matrix of the estimates of the gamma moments conditional on (X, Z) for d = 0, and a matrix of the estimates of the gamma moments conditional on (X, Z) for d = 1. The column means of the last two matrices is what is used to generate the gamma moments.
`gn0`	integer, number of terms in the MTR for control group.

gn1	integer, number of terms in the MTR for treated group.
subsetList	list of subset indexes, one for each IV-like specification.
n	number of observations in the data. This option is only used when subsets are involved.

Value

matrix whose column means can be used to carry out the GMM estimation.

---

monoIntegral *Integrating and evaluating monomials*

Description

Analytically integrates monomials and evalates them at a given point. It is assumed that there is no constant multiplying the monomial.

Usage

```
monoIntegral(u, exp)
```

Arguments

u	scalar, the point at which to evaluate the integral. If a vector is passed, then the integral is evaluated at all the elements of the vector.
exp	The exponent of the monomial.

Value

scalar or vector, depending on what u is.

---

negationCheck *Check if custom weights are negations of each other*

Description

This function checks whether the user-declared weights for treated and control groups are in fact negations of each other. This is problematic for the GMM procedure when accounting for estimation error of the target weights.

Usage

```
negationCheck(
  data,
  target.knots0,
  target.knots1,
  target.weight0,
  target.weight1,
  N = 20
)
```

Arguments

data	data set used for estimation. The comparisons are made only on values in the support of the data set.
target.knots0	user-defined set of functions defining the knots associated with splines weights for the control group. The arguments of the function should consist only of variable names in `data`. If the knot is constant across all observations, then the user can instead submit the value of the weight instead of a function.
target.knots1	user-defined set of functions defining the knots associated with splines weights for the treated group. The arguments of the function should be variable names in `data`. If the knot is constant across all observations, then the user can instead submit the value of the weight instead of a function.
target.weight0	user-defined weight function for the control group defining the target parameter. A list of functions can be submitted if the weighting function is in fact a spline. The arguments of the function should be variable names in `data`. If the weight is constant across all observations, then the user can instead submit the value of the weight instead of a function.
target.weight1	user-defined weight function for the treated group defining the target parameter. A list of functions can be submitted if the weighting function is in fact a spline. The arguments of the function should be variable names in `data`. If the weight is constant across all observations, then the user can instead submit the value of the weight instead of a function.
N	integer, default set to 20. This is the maxmimum number of points between treated and control groups to compare and determine whether or not the weights are indeed negations of one another. If the data set contains fewer than `N` unique values for a given set of variables, then all those unique values are used for the comparison.

Value

boolean. If the weights are negations of each other, `TRUE` is returned.

`olsj` *OLS weights*

Description

Function generating the S-weights for OLS estimand, with controls.

Usage

```
olsj(X, X0, X1, components, treat, order = NULL)
```

Arguments

`X`	Matrix of covariates, including the treatment indicator.
`X0`	Matrix of covariates, once fixing treatment to be 0.
`X1`	Matrix of covariates, once fixing treatment to be 1.
`components`	Vector of variable names of which user wants the S-weights for.
`treat`	Variable name for the treatment indicator.
`order`	integer, default set to `NULL`. This is simply an index of which IV-like specification the estimate corresponds to.

Value

A list of two vectors: one is the weight for D = 0, the other is the weight for D = 1.

`optionsCplexAPI` *Function to parse options for CPLEX*

Description

This function constructs a list of options to be parsed when `solver` is set to `cplexapi`.

Usage

```
optionsCplexAPI(options)
```

Arguments

options — list. The name of each item must be the name of the function to set the option, and is case sensitive. The value assigned to each item is the value to set the option to. The `env` argument should always be omitted. If the option accepts a list of parameters, then these parameters should be passed as using a named vector (e.g. `list(setLogFileNameCPLEX = c(filename = "cpx.log",mode = "w"))`). If the function to set the option can be used multiple times, then the value submitted should be a a list, with each entry being a named vector (e.g. `list(setDblParmCPLEX = list(c(parm = 1016,value = 1e-04),c(parm = 1084,value = 2)))`). If the option only requires the `env` parameter, then an `NA` should be passed as the parameter value (e.g. `list(setDefaultParm = NA)`).

Value

list, each element being the command to evaluate to implement an option.

`optionsCplexAPISingle` *Function to parse a single set of options for CPLEX*

Description

This function constructs a string to be parsed when `solver` is set to `cplexapi`.

Usage

```
optionsCplexAPISingle(name, vector)
```

Arguments

name — string, name of the `cplexapi` function to call to implement the option.

vector — a named vector, contains the argument names and values of the options. The `env` argument in the `cplexapi` documentation should always be omitted.

Value

string, the command to be evaluated to implement a single option.

`optionsCplexAPITol` *Function to extract feasibility tolerance from CPLEX options*

Description

This function parses through the user-submitted CPLEX options to determine what the feasibility tolerance is. This tolerance can then be used for the audit. If the user does not set the CPLEX feasibility tolerance, then a default value of `1e-06` is returned.

Usage

```
optionsCplexAPITol(options)
```

Arguments

`options`	list, the set of options submitted by the user.

Value

scalar, the level to set the audit tolerance at.

`optionsGurobi` *Function to parse options for Gurobi*

Description

This function constructs a list of options to be parsed when `solver` is set to `Gurobi`. This function really implements some default values, and accounts for the `debug` option.

Usage

```
optionsGurobi(options, debug)
```

Arguments

`options`	list. The list should be structured the same way as if one were using the `gurobi` library directly. That is, the name of each item must be the name of the option, and is case sensitive. The value assigned to each item is the value to set the option to.
`debug`	boolean, indicates whether or not the function should provide output when obtaining bounds. The output provided is the same as what the Gurobi API would send to the console.

Value

list, the set of options declared by the user, including some additional default values (if not assigned by the user) and accounting for `debug`.

`optionsLpSolveAPI` *Function to parse options for lp_solve*

Description

This function constructs a list of options to be parsed when `solver` is set to `lpsolveapi`. The options permitted are those that can be set via `lpSolveAPI::lp.control`, and should be passed as a named list (e.g. `list(epslevel = "tight")`).

Usage

```
optionsLpSolveAPI(options)
```

Arguments

`options`	list. The name of each item must be the name of the option, and is case sensitive. The value assigned to each item is the value to set the option to. The `lprec` argument should always be omitted.

Value

string, the command to be evaluated to implement the options.

`optionsRmosek` *Function to parse options for Gurobi*

Description

This function constructs a list of options to be parsed when `solver` is set to `Rmosek`. This function really implements the default feasibility tolerances.

Usage

```
optionsRmosek(options, debug)
```

Arguments

`options`	list. Each set of options should be passed as a list, with the name of each entry being the name of the class of options. For example, options for double parameters should be contained in the entry`dparam = list(BASIS_TOL_X = 1e-06)`.
`debug`	boolean, indicates whether or not the function should provide output when obtaining bounds. The output provided is the same as what Mosek would send to the console.

Value

list, the set of options declared by the user, including some additional default values.

parenthBoolean — *Correct boolean expressions in terms lists*

Description

This function takes a vector of terms and places parentheses around boolean expressions.

Usage

```
parenthBoolean(termsList)
```

Arguments

`termsList`	character vector, the vector of terms.

Value

character vector.

permute — *Auxiliary function: generate all permutations of a vector*

Description

This function generates every permutation of the elements in a vector.

Usage

```
permute(vector)
```

Arguments

`vector`	The vector whose elements are to be permuted.

Value

a list of all the permutations of `vector`.

`permuteN` *Auxiliary function: generate all permutation orderings*

Description

This function generates every permutation of the first n natural numbers.

Usage

```
permuteN(n)
```

Arguments

`n`	integer, the first n natural numbers one wishes to permute.

Value

a list of all the permutations of the first n natural numbers.

`piv` *Obtaining IV-like estimands*

Description

This function performs TSLS to obtain the estimates for the IV-like estimands.

Usage

```
piv(
  Y,
  X,
  Z,
  lmcomponents = NULL,
  weights = NULL,
  order = NULL,
  excluded = TRUE
)
```

Arguments

`Y`	the vector of outcomes.
`X`	the matrix of covariates (includes endogenous and exogenous covariates).
`Z`	the matrix of instruments (includes exogenous covariates in the second stage).

`lmcomponents`	vector of variable names from the second stage that we want to include in the S-set of IV-like estimands. If `NULL` is submitted, then all components will be included.
`weights`	vector of weights.
`order`	integer, the counter for which IV-like specification and component the regression is for.
`excluded`	boolean, to indicate whether or not the regression involves excluded variables.

Value

vector of select coefficient estimates.

`polyparse` *Parsing marginal treatment response formulas*

Description

This function takes in an MTR formula, and then parses the formula such that it becomes a polynomial in the unobservable `u`. It then breaks these polynomials into monomials, and then integrates each of them with respect to `u`. Each integral corresponds to E[md | D, X, Z].

Usage

```
polyparse(
  formula,
  data,
  uname = "u",
  env = parent.frame(),
  as.function = FALSE
)
```

Arguments

`formula`	the MTR.
`data`	`data.frame` for which we obtain E[md	D, X, Z] for each observation.
`uname`	variable name for unobservable used in declaring the MTR.
`env`	environment, the original environment in which the formula was declared.
`as.function`	boolean, if `FALSE` then a list of the polynomial terms are returned; if `TRUE` then a list of functions corresponding to the polynomials are returned.

Value

A list (of lists) of monomials corresponding to the original MTR (for each observation); a list (of lists) of the integrated monomials; a vector for the degree of each of the original monomials in the MTR; and a vector for the names of each variable entering into the MTR (note `x^2 + x` has only one term, `x`).

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declare MTR functions
formula1 = ~ 1 + u
formula0 = ~ 1 + u

## Construct MTR polynomials
polynomials0 <- polyparse(formula = formula0,
                          data = dtm,
                          uname = u,
                          as.function = FALSE)

polynomials1 <- polyparse(formula = formula0,
                          data = dtm,
                          uname = u,
                          as.function = FALSE)
```

---

polyProduct *Function to multiply polynomials*

Description

This function takes in two vectors characterizing polynomials. It then returns a vector characterizing the product of the two polynomials.

Usage

```
polyProduct(poly1, poly2)
```

Arguments

poly1	vector, characerizing a polynomial.
poly2	vector, characerizing a polynomial.

Value

vector, characterizing the product of the two polynomials characterized `poly1` and `poly2`.

popmean — *Calulating population mean*

Description

Given a distribution, this function calculates the population mean for each term in a formula.

Usage

```
popmean(formula, distribution, density = "f")
```

Arguments

`formula`	formula, each term of which will have its mean calculated.
`distribution`	data.table, characterizing the distribution of the variables entering into `formula`.
`density`	string, name of the variable `data` characterizing the density.

Value

vector, the means for each term in `formula`.

print.ivmte — *Print results*

Description

This function uses the print method on the ivmte return list.

Usage

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

Arguments

`x`	an object returned from '`ivmte`'.
`...`	additional arguments.

Value

basic set of results.

propensity *Estimating propensity scores*

Description

This function estimates the propensity of taking up treatment. The user can choose from fitting a linear probability model, a logit model, or a probit model. The function can also be used to generate a table of propensity scores for a given set of covariates and excluded variables. This was incorporated to account for the LATE being a target parameter. Specifically, if the argument `formula` is the name of a variable in `data`, but the target parameter is not the LATE, then no propensity model is returned. If the target parameter is the LATE, then then the propensity model is simply the empirical distribution of propensity scores in the data conditioned on the set of covariates declared in `late.X` and `late.Z`.

Usage

```
propensity(formula, data, link = "logit", late.Z, late.X, env = parent.frame())
```

Arguments

`formula`	Formula characterizing probability model. If a variable in the data already contains the propensity scores, input the variable as a one-sided formula. For example, if the variable `pz` contains the propensity score, input `formula = ~ pz`.
`data`	`data.frame` with which to estimate the model.
`link`	Link function with which to estimate probability model. Can be chosen from "linear", "logit", or "probit".
`late.Z`	A vector of variable names of excluded variables. This is required when the target parameter is the LATE.
`late.X`	A vector of variable names of non-excluded variables. This is required when the target parameter is the LATE, and the estimation procedure will condition on these variables.
`env`	environment, the environment for the original propensity score formula.

Value

A vector of propensity scores for each observation, as well as a 'model'. If the user inputs a formula characterizing the model for taking up treatment, then the `lm`/`glm` object is returned. If the user declares a variable in the data set to be used as the propensity score, then a `data.frame` containing the propensity score for each value of the covariates in the probability model is returned.

Examples

```
dtm <- ivmte:::gendistMosquito()

## Declaring a probability model.
propensity(formula = d ~ z,
```

```
               data = dtm,
               link = "linear")

## Declaring a variable to be used instead
propensity(formula = ~ pz,
               data = dtm,
               link = "linear")
```

---

qpSetup *Constructing QCQP problem*

Description

This function is only used when the direct MTR regression procedure is used. This function simply constructs the quadratic constraint, and adds it to the LP problem defined by the linear optimization problem for the bounds and the linear shape constraints.

Usage

```
qpSetup(env, sset, rescale = TRUE)
```

Arguments

`env`	environment containing the matrices defining the LP problem.
`sset`	A list containing the covariats and outcome variable for the direct MTR regression.
`rescale`	boolean, set to `TRUE` if the MTR components should be rescaled to improve stability in the LP/QP/QCP optimization.

---

qpSetupBound *Constructing QCQP problem for bounding*

Description

This function is only used when the direct MTR regression procedure is used. This function simply constructs the quadratic constraint, and adds it to the LP problem defined by the linear optimization problem for the bounds and the linear shape constraints.

Usage

```
qpSetupBound(
  env,
  g0,
  g1,
  criterion.tol,
  criterion.min,
  rescale = FALSE,
  setup = TRUE
)
```

Arguments

`env`	environment containing the matrices defining the LP problem.
`g0`	set of expectations for each terms of the MTR for the control group.
`g1`	set of expectations for each terms of the MTR for the control group.
`criterion.tol`	non-negative scalar, determines how much the quadratic constraint should be relaxed by. If set to 0, the constraint is not relaxed at all.
`criterion.min`	minimum of (SSR - SSY) of a linear regression with shape constraints.
`rescale`	boolean, set to `TRUE` if the MTR components should be rescaled to improve stability in the LP/QP/QCP optimization.
`setup`	boolean, set to `TRUE` if the QP problem should be set up for solving the bounds, which includes the quadratic constraint. Set to `FALSE` if the quadratic constraint should be removed.

Value

A list of matrices and vectors necessary to define an LP problem for Gurobi or MOSEK.

`qpSetupCriterion` *Configure QCQP problem to find minimum criterion*

Description

This function sets up the objective function for minimizing the criterion. The QCQP model must be passed as an environment variable, under the entry `$model`. See `qpSetup`.

Usage

```
qpSetupCriterion(env)
```

Arguments

`env`	The LP environment

Value

Nothing, as this modifies an environment variable to save memory.

---

qpSetupInfeasible *Configure QP environment for diagnostics*

Description

This function separates the shape constraints from the QP environment. That way, the model can be solved without any shape constraints, which is the primary cause of infeasibility. This is done in order to check which shape constraints are causing the model to be infeasible. The QP model must be passed as an environment variable, under the entry `$model`. See `lpSetup`.

Usage

```
qpSetupInfeasible(env, rescale)
```

Arguments

`env`	The LP environment
`rescale`	boolean, set to `TRUE` if the MTR components should be rescaled to improve stability in the LP/QP/QCP optimization.

Value

Nothing, as this modifies an environment variable to save memory.

---

removeSplines *Separating splines from MTR formulas*

Description

This function separates out the function calls `uSpline()` and `uSplines()` potentially embedded in the MTR formulas from the rest of the formula. The terms involving splines are treated separately from the terms that do not involve splines when creating the gamma moments.

Usage

```
removeSplines(formula, env = parent.frame())
```

Arguments

`formula`	the formula that is to be parsed.
`env`	environment in which to formulas. This is necessary as splines may be declared using objects, e.g. `knots = x`, where `x = c(0.3,0.64,0.9)`.

Value

a list containing two objects. One object is `formula` but with the spline components removed. The second object is a list. The name of each element is the `uSpline()`/`uSplines()` command, and the elements are a vector of the names of covariates that were interacted with the `uSpline()`/`uSplines()` command.

Examples

```
## Declare and MTR with a sline component.
m0 = ~ x1 + x1 : uSpline(degree = 2,
                         knots = c(0.2, 0.4)) +
            x2 : uSpline(degree = 2,
                         knots = c(0.2, 0.4)) +
            x1 : x2 : uSpline(degree = 2,
                              knots = c(0.2, 0.4)) +
            uSpline(degree = 3,
                    knots = c(0.2, 0.4),
                    intercept = FALSE)

## Now separate the spline component from the non-spline component
removeSplines(m0)
```

---

rescaleX — *Function to implement rescaling procedure*

Description

This function rescales the matrix of covariates used in the direct regression to improve the conditioning number and the stability of the estimation procedure.

Usage

```
rescaleX(sset, dVec, drY, drN)
```

Arguments

`sset`	a list of lists constructed from the function genSSet. In the case of a direct regression, 'sset' contains only one inner list. This list contains the gamma moment at the individual level.
`dVec`	Vector of treatment statuses from the data.
`drY`	Vector of outcomes from the data.
`drN`	Scalar, number of observations in the data.

Value

List of rescaled covariates.

`restring` *Auxiliary function that converts an expression of variable names into a vector of strings.*

Description

Auxiliary function that converts an expression of variable names into a vector of strings.

Usage

```
restring(vector, substitute = TRUE, command = "c")
```

Arguments

`vector`	An expression of a list of variable names.
`substitute`	Boolean option of whether or not we wish to use the `substitute` command when implementing this function. Note that this substitutes the argument of the function. If `substitute = FALSE`, then the function will instead treat the arguments as variables, and substitute in their values.
`command`	character, the name of the function defining the vector or list, e.g. "c", "list", "l". This let's the function determine how many characters in front to remove.

Value

A vector of variable names (strings).

Examples

```
a <- 4
b <- 5
ivmte:::restring(c(a, b), substitute = TRUE)
ivmte:::restring(c(a, b), substitute = FALSE)
```

`rhalton` *Generate Halton sequence*

Description

This function generates a one dimensional Halton sequence.

Usage

```
rhalton(n, base = 2)
```

Arguments

`n`	Number of draws.
`base`	Base used for the Halton sequence, set to 2 by default.

Value

A sequence of randomly drawn numbers.

`runCplexAPI` *Running cplexAPI solver*

Description

This function solves the LP problem using the cplexAPI package. The object generated by `lpSetup` is not compatible with the `cplexAPI` functions. This function adapts the object to solve the LP problem. See `runGurobi` for additional error code labels.

Usage

```
runCplexAPI(model, lpdir, solver.options)
```

Arguments

`model`	list of matrices and vectors defining the linear programming problem.
`lpdir`	input either CPX_MAX or CPX_MIN, which sets the LP problem as a maximization or minimization problem.
`solver.options`	list, each item of the list should correspond to an option specific to the LP solver selected.

Value

a list of the output from CPLEX. This includes the objective value, the solution vector, and the optimization status (status of 1 indicates successful optimization).

runGurobi *Running Gurobi solver*

Description

This function solves the LP/QCQP problem using the Gurobi package. The object generated by `lpSetup` is compatible with the `gurobi` function. See `runCplexAPI` for additional error code labels.

Usage

```
runGurobi(model, solver.options)
```

Arguments

`model`	list of matrices and vectors defining the linear programming problem.
`solver.options`	list, each item of the list should correspond to an option specific to the LP solver selected.

Value

a list of the output from Gurobi. This includes the objective value, the solution vector, and the optimization status (status of `1` indicates successful optimization) .

runLpSolveAPI *Running lpSolveAPI*

Description

This function solves the LP problem using the `lpSolveAPI` package. The object generated by `lpSetup` is not compatible with the `lpSolveAPI` functions. This function adapts the object to solve the LP problem. See `runGurobi` and `runCplexAPI` for additional error code labels.

Usage

```
runLpSolveAPI(model, modelsense, solver.options)
```

Arguments

`model`	list of matrices and vectors defining the linear programming problem.
`modelsense`	input either 'max' or 'min' which sets the LP problem as a maximization or minimization problem.
`solver.options`	list, each item of the list should correspond to an option specific to the LP solver selected.

Value

a list of the output from `lpSolveAPI`. This includes the objective value, the solution vector, and the optimization status (status of `1` indicates successful optimization).

`runMosek` *Running Rmosek*

Description

This function solves the LP/QCQP problem using the `Rmosek` package. The object generated by `lpSetup` is not compatible with the `Rmosek` functions. This function adapts the object to solve the LP problem. See `runGurobi` and `runCplexAPI` for additional error code labels.

Usage

```
runMosek(model, modelsense, solver.options, debug = FALSE)
```

Arguments

`model`	list of matrices and vectors defining the linear programming problem.
`modelsense`	input either 'max' or 'min' which sets the LP problem as a maximization or minimization problem.
`solver.options`	list, each item of the list should correspond to an option specific to the LP solver selected.
`debug`	boolean, indicates whether or not the function should provide output when obtaining bounds. The output provided is the same as what the Mosek would send to the console.

Value

a list of the output from `Rmosek`. This includes the objective value, the solution vector, and the optimization status (status of `1` indicates successful optimization).

`selectViolations` *Select points from audit grid to add to the constraint grid*

Description

This function selects which points from the audit grid should be included into the original grid. Both the constraint grid and audit grid are represented as constraints in an LP/QCQP problem. This function selects which points in the audit grid (i.e. which rows in the audit constraint matrix) should be added to the constraint grid (i.e. should be appended to the constraint matrix).

Usage

```
selectViolations(
  diffVec,
  audit.add,
  lb0seq,
  lb1seq,
  lbteseq,
  ub0seq,
  ub1seq,
  ubteseq,
  mono0seq,
  mono1seq,
  monoteseq,
  mbmap
)
```

Arguments

`diffVec`	numeric vector, with a positive value indicating a violation of a shape constraint.
`audit.add`	integer, the number of points from the audit grid to add to the initial for each constraint type. For instance, if there are 5 different kinds of constraints imposed, and `audit.add = 5`, then up to 30 points may be added to the constraint grid.
`lb0seq`	integer vector, indicates which rows in the audit constraint matrix correspond to the lower bound for m0.
`lb1seq`	integer vector, indicates which rows in the audit constraint matrix correspond to the lower bound for m1.
`lbteseq`	integer vector, indicates which rows in the audit constriant matrix correspond to the lower bound for the treatment effect.
`ub0seq`	integer vector, indicates which rows in the audit constraint matrix correspond to the upper bound for m0.
`ub1seq`	integer vector, indicates which rows in the audit constraint matrix correspond to the upper bound for m1.
`ubteseq`	integer vector, indicates which rows in the audit constriant matrix correspond to the upper bound for the treatment effect.
`mono0seq`	integer matrix, indicates which rows in the audit constraint matrix correspond to the monotonicity conditions for m0, and whether the constraint is increasing (+1) or decreasing (-1).
`mono1seq`	integer matrix, indicates which rows in the audit constraint matrix correspond to the monotonicity conditions for m1, and whether the constraint is increasing (+1) or decreasing (-1).
`monoteseq`	integer matrix, indicates which rows in the audit constraint matrix correspond to the monotonicity conditions for the treatment effect, and whether the constraint is increasing (+1) or decreasing (-1).
`mbmap`	integer vector, indexes the X-value associated with each row in the audit constraint matrix.

Value

The audit grid is represented using a set of constraint matrices. Each point in the audit grid corresponds to a set of rows in the constraint matrices. The function simply returns the vector of row numbers for the points from the audit grid whose corresponding constraints should be added to the original LP/QCQP problem (i.e. the points to add to the original grid).

---

sOls1d — *IV-like weighting function, OLS specification 1*

Description

IV-like weighting function for OLS specification 1.

Usage

```
sOls1d(d, exx)
```

Arguments

d	0 or 1, indicating treatment or control.
exx	the matrix E[XX']

Value

scalar.

---

sOls2d — *IV-like weighting function, OLS specification 2*

Description

IV-like weighting function for OLS specification 2.

Usage

```
sOls2d(x, d, exx)
```

Arguments

x	vector, the value of the covariates other than the intercept and the treatment indicator.
d	0 or 1, indicating treatment or control.
exx	the matrix E[XX']

Value

scalar.

sOls3 *IV-like weighting function, OLS specification 3*

Description

IV-like weighting function for OLS specification 3.

Usage

```
sOls3(x, d, j, exx)
```

Arguments

x	vector, the value of the covariates other than the intercept and the treatment indicator.
d	0 or 1, indicating treatment or control.
j	scalar, position of the component one is interested in constructing the IV-like weight for.
exx	the matrix E[XX']

Value

scalar.

sOlsSplines *IV-like weighting function, OLS specifications*

Description

IV-like weighting function for OLS specifications.

Usage

```
sOlsSplines(x = NULL, d, j, exx)
```

Arguments

x	vector, the value of the covariates other than the intercept and the treatment indicator.
d	0 or 1, indicating treatment or control.
j	scalar, position of the component one is interested in constructing the IV-like weight for.
exx	matrix corresponding to E[XX'].

Value

scalar.

splineInt *Integrating splines*

Description

This function simply integrates the splines.

Usage

```
splineInt(ub, lb, knots, degree, intercept = FALSE)
```

Arguments

ub	scalar, upperbound of integral.
lb	scalar, lowerbound of integral.
knots	vector, knots of the spline.
degree	scalar, degre of spline.
intercept	boolean, set to TRUE if spline basis should include a component so that the basis sums to 1.

Value

vector, each component being the integral of a basis.

splinesBasis *Evaluating splines basis functions*

Description

This function evaluates the splines basis functions. Unlike the `bSpline` in the `splines2` package, this function returns the value of a single spline basis, rather than a vector of values for all the spline basis functions.

Usage

```
splinesBasis(x, knots, degree, intercept = TRUE, i, boundary.knots = c(0, 1))
```

Arguments

`x`	vector, the values at which to evaluate the basis function.
`knots`	vector, the internal knots.
`degree`	integer, the degree of the splines.
`intercept`	boolean, default set to `TRUE`. This includes an additional component to the basis splines so that the splines are a partition of unity (i.e. the sum of all components equal to 1).
`i`	integer, the basis component to be evaluated.
`boundary.knots`	vector, default is `c(0,1)`.

Value

scalar.

`splineUpdate` *Constructing higher order splines*

Description

This function recursively constructs the higher order splines basis. Note that the function does not take into consideration the order of the final basis function. The dimensions of the inputs dicate this, and are updated in each iteration of the recursion. The recursion ends once the row number of argument `bmat` reaches 1. This function was coded in accordance to Carl de Boor's set of notes on splines, "B(asic)-Spline Basics".

Usage

```
splineUpdate(x, bmat, knots, i, current.order)
```

Arguments

`x`	vector, the values at which to evaluate the basis function.
`bmat`	matrix. Each column of `bmat` corresponds to an element of argument `x`. Each row corresponds to the evaluation of basis component `i`, `i + 1`, ... The recursive nature of splines requires that we initially evaluate the basis functions for components `i`, ..., `i + degree` of `spline`. Each iteration of the recursion reduces the row of `bmat` by 1. The recursion terminates once `bmat` has only a single row.
`knots`	vector, the internal knots.
`i`	integer, the basis component of interest.
`current.order`	integer, the current order associated with the argument `bmat`.

Value

vector, the evaluation of the spline at each value in vector `x`.

`statusString` *Convert status code to string*

Description

This function returns the status code specific to a solver.

Usage

```
statusString(status, solver)
```

Arguments

`status`	Status code.
`solver`	Name of solver, either 'gurobi', 'cplexapi', or 'lpsolveapi'.

Value

Status specific to solver, e.g. 'OPTIMAL (2)'.

`sTsls` *IV-like weighting function, TSLS specification*

Description

IV-like weighting function for TSLS specification.

Usage

```
sTsls(z, j, exz, pi)
```

Arguments

`z`	vector, the value of the instrument.
`j`	scalar, position of the component one is interested in constructing the IV-like weight for.
`exz`	the matrix E[XZ']
`pi`	the matrix E[XZ']E[ZZ']^-1

Value

scalar.

`sTslsSplines` *IV-like weighting function, TSLS specification*

Description

IV-like weighting function for TSLS specification.

Usage

```
sTslsSplines(z, d, j, exz, pi)
```

Arguments

`z`	vector, the value of the instrument.
`d`	0 or 1, indicating treatment or control (redundant in this function; included to exploit apply()).
`j`	scalar, position of the component one is interested in constructing the IV-like weight for.
`exz`	matrix, corresponds to E[XZ'].
`pi`	matrix, corresponds to E[XZ']E[ZZ']^-1, the first stage regression.

Value

scalar.

`subsetclean` *Auxiliary function: remove extraneous spaces*

Description

Auxiliary function to remove extraneous spaces from strings.

Usage

```
subsetclean(string)
```

Arguments

`string`	the string object to be cleaned.

Value

a string

`summary.ivmte` *Summarize results*

Description

This function uses the summary method on the ivmte return list.

Usage

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

Arguments

`object`	an object returned from '`ivmte`'.
`...`	additional arguments.

Value

summarized results.

`sWald` *IV-like weighting function, Wald specification*

Description

IV-like weighting function for OLS specification 2.

Usage

```
sWald(z, p.to, p.from, e.to, e.from)
```

Arguments

`z`	vector, the value of the instrument.
`p.to`	P[Z = z'], where z' is value of the instrument the agent is switching to.
`p.from`	P[Z = z], where z is the value of the instrument the agent is switching from.
`e.to`	E[D	Z = z'], where z' is the value of the instrument the agent is switching to.
`e.from`	E[D	Z = z], where z is the value of the instrument the agent is switching from.

Value

scalar.

symat — *Generate symmetric matrix*

Description

Function takes in a vector of values, and constructs a symmetric matrix from it. Diagonals must be included. The length of the vector must also be consistent with the number of "unique" entries in the symmetric matrix. Note that entries are filled in along the columns (i.e. equivalent to byrow = FALSE).

Usage

```
symat(values)
```

Arguments

`values`	vector, the values that enter into the symmetric matrix. Dimensions will be determined automatically.

Value

matrix.

tsls — *TSLS weights, with controls*

Description

Function generating the S-weights for TSLS estimand, with controls.

Usage

```
tsls(X, Z, Z0, Z1, components, treat, order = NULL)
```

Arguments

`X`	Matrix of covariates, including the treatment indicator.
`Z`	Matrix of instruments.
`Z0`	Matrix of instruments, fixing treatment to 0.
`Z1`	Matrix of instruments, fixing treatment to 1.
`components`	Vector of variable names of which user wants the S-weights for.
`treat`	Variable name for the treatment indicator.
`order`	integer, default set to `NULL`. This is simply an index of which IV-like specification the estimate corresponds to.

Value

A list of two vectors: one is the weight for D = 0, the other is the weight for D = 1.

`unstring` *Auxiliary function that converts a vector of strings into an expression containing variable names.*

Description

Auxiliary function that converts a vector of strings into an expression containing variable names.

Usage

```
unstring(vector)
```

Arguments

`vector`	Vector of variable names (strings).

Value

An expression for the list of variable names that are not strings.

Examples

```
ivmte:::unstring(c("a", "b"))
```

`uSplineBasis` *Spline basis function*

Description

This function evaluates the splines that the user specifies when declaring the MTRs. This is to be used for auditing, namely when checking the boundedness and monotonicity conditions.

Usage

```
uSplineBasis(x, knots, degree = 0, intercept = TRUE)
```

Arguments

`x`	the points to evaluate the integral of the the splines.
`knots`	the knots of the spline.
`degree`	the degree of the spline; default is set to 0 (constant splines).
`intercept`	boolean, set to TRUE if intercept term is to be included (i.e. an additional basis such that the sum of the splines at every point in `x` is equal to 1).

Value

a matrix, the values of the integrated splines. Each row corresponds to a value of x; each column corresponds to a basis defined by the degrees and knots.

Examples

```
## Since the splines are declared as part of the MTR, you will need
## to have parsed out the spline command. Thus, this command will be
## called via eval(parse(text = .)). In the examples below, the
## commands are parsed from the object \code{splineslist} generated
## by \code{\link[MST]{removeSplines}}. The names of the elements in
## the list are the spline commands, and the elements themselves are
## the terms that interact with the splines.

## Declare MTR function
m0 = ~ x1 + x1 : uSpline(degree = 2,
                          knots = c(0.2, 0.4)) +
    x2 : uSpline(degree = 2,
                  knots = c(0.2, 0.4)) +
    x1 : x2 : uSpline(degree = 2,
                       knots = c(0.2, 0.4)) +
    uSpline(degree = 3,
             knots = c(0.2, 0.4),
             intercept = FALSE)

## Extract spline functions from MTR function
splineslist <- removeSplines(m0)$splineslist

## Declare points at which we wish to evaluate the spline functions
x <- seq(0, 1, 0.2)

## Evaluate the splines
eval(parse(text = gsub("uSpline\\(",
                       "ivmte:::uSplineBasis(x = x, ",
                        names(splineslist)[1])))

eval(parse(text = gsub("uSpline\\(",
                       "ivmte:::uSplineBasis(x = x, ",
                       names(splineslist)[2])))
```

---

uSplineInt *Integrated splines*

Description

This function integrates out splines that the user specifies when declaring the MTRs. This is to be used when generating the gamma moments.

Usage

```
uSplineInt(x, knots, degree = 0, intercept = TRUE)
```

Arguments

x	the points to evaluate the integral of the the splines.
knots	the knots of the spline.
degree	the degree of the spline; default is set to 0 (constant splines).
intercept	boolean, set to TRUE if intercept term is to be included (i.e. an additional basis such that the sum of the splines at every point in x is equal to 1).

Value

a matrix, the values of the integrated splines. Each row corresponds to a value of x; each column corresponds to a basis defined by the degrees and knots.

Examples

```
## Since the splines are declared as part of the MTR, you will need
## to have parsed out the spline command. Thus, this command will be
## called via eval(parse(text = .)). In the examples below, the
## commands are parsed from the object \code{splineslist} generated
## by \code{\link[MST]{removeSplines}}. The names of the elements in
## the list are the spline commands, and the elements themselves are
## the terms that interact with the splines.

## Declare MTR function
m0 = ~ x1 + x1 : uSpline(degree = 2,
                          knots = c(0.2, 0.4)) +
    x2 : uSpline(degree = 2,
                  knots = c(0.2, 0.4)) +
    x1 : x2 : uSpline(degree = 2,
                       knots = c(0.2, 0.4)) +
    uSpline(degree = 3,
             knots = c(0.2, 0.4),
             intercept = FALSE)

## Separate the spline components from the MTR function
splineslist <- removeSplines(m0)$splineslist

## Delcare the points at which we wish to evaluate the integrals
x <- seq(0, 1, 0.2)

## Evaluate the splines integrals
eval(parse(text = gsub("uSpline\\(",
                       "ivmte:::uSplineInt(x = x, ",
                       names(splineslist)[1])))


eval(parse(text = gsub("uSpline\\(",
```

```
"ivmte:::uSplineInt(x = x, ",
names(splineslist)[2])))
```

---

vecextract *Auxiliary function: extracting elements from strings*

Description

This auxiliary function extracts the (string) element in the `position` argument of the `vector` argument.

Usage

```
vecextract(vector, position, truncation = 0)
```

Arguments

`vector`	the vector from which we want to extract the elements.
`position`	the position in `vector` to extract.
`truncation`	the number of characters from the front of the element being extracted that should be dropped.

Value

A chracter/string.

---

wate1 *Target weight for ATE*

Description

Function generates the target weight for the ATE.

Usage

```
wate1(data)
```

Arguments

`data`	`data.frame` on which the estimation is performed.

Value

The bounds of integration over unobservable `u`, as well as the multiplier in the weight.

`watt1` *Target weight for ATT*

Description

Function generates the target weight for the ATT.

Usage

```
watt1(data, expd1, propensity)
```

Arguments

`data`	`data.frame` on which the estimation is performed.
`expd1`	Scalar, the probability that treatment is received.
`propensity`	Vector of propensity to take up treatment.

Value

The bounds of integration over unobservable `u`, as well as the multiplier in the weight.

`wAttSplines` *Target weighting function, for ATT*

Description

Target weighting function, for the ATT.

Usage

```
wAttSplines(z, d, ed)
```

Arguments

`z`	vector, the value of the instrument (redundant in this function; included to exploit apply()).
`d`	0 or 1, indicating treatment or control (redundant in this function; included to exploit apply()).
`ed`	scalar, unconditional probability of taking up treatment.

Value

scalar.

watu1 — *Target weight for ATU*

Description

Function generates the target weight for the ATT.

Usage

```
watu1(data, expd0, propensity)
```

Arguments

`data`	`data.frame` on which the estimation is performed.
`expd0`	Scalar, the probability that treatment is not recieved.
`propensity`	Vector of propensity to take up treatment.

Value

The bounds of integration over unobservable `u`, as well as the multiplier in the weight.

weights — *Generating splines weights*

Description

This function generates the weights required to construct splines of higher order. This function was coded in accordance to Carl de Boor's set of notes on splines, "B(asic)-Spline Basics".

Usage

```
weights(x, knots, i, order)
```

Arguments

`x`	vector, the values at which to evaluate the basis function.
`knots`	vector, the internal knots.
`i`	integer, the basis component to be evaluated.
`order`	integer, the order of the basis. Do not confuse this with the degree of the splines, i.e. order = degree + 1.

Value

scalar.

`wgenlate1` *Target weight for generalized LATE*

Description

Function generates the target weight for the generalized LATE, where the user can specify the interval of propensity scores defining the compliers.

Usage

```
wgenlate1(data, ulb, uub)
```

Arguments

`data`	`data.frame` on which the estimation is performed.
`ulb`	Numeric, lower bound of interval.
`uub`	Numeric, upper bound of interval.

Value

The bounds of integration over unobservable `u`, as well as the multiplier in the weight.

`whichforlist` *Auxiliary function:* `which` *for lists*

Description

Auxiliary function that makes it possible to use `which` with a list.

Usage

```
whichforlist(vector, obj)
```

Arguments

`vector`	the vector for which we want to check the entries of
`obj`	the value for which we want the vector to match on.

Value

a vector of positions where the elements in `vector` are equal to `obj`.

wlate1 *Target weight for LATE*

Description

Function generates the target weight for the LATE, conditioned on a specific value of the covariates.

Usage

```
wlate1(data, from, to, Z, model, X, eval.X)
```

Arguments

`data`	`data.frame` on which the estimation is performed.
`from`	Vector of baseline values for the instruments.
`to`	Vector of comparison values for the instruments.
`Z`	Character vector of names of instruments.
`model`	A `lm` or `glm` object, or a `data.frame`, which can be used to estimate the propensity to take up treatment for the specified values of the instruments.
`X`	Character vector of variable names for the non-excluded variables the user wishes to condition the LATE on.
`eval.X`	Vector of values the user wishes to condition the `X` variables on.

Value

The bounds of integration over unobservable `u`, as well as the multiplier in the weight.

Index