Package 'ivmte'

May 16, 2019

Title Instrumental Variables: Extrapolation by Marginal Treatment Effects

Version 1.0.1

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 linear programming. Support for three linear programming 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. The lp_solve library is freely available from <http://lpsolve.sourceforge.net/5.5/>, and is included when installing either of its R APIs, 'lpSolve' or 'lpSolveAPI', which are available from CRAN.

Depends R (>= 3.4.0)

Imports polynom (>= 1.3-9), Formula, methods, stats, utils

Suggests gurobi (>= 7.5-1), slam (>= 0.1-42), Rcplex (>= 0.3.3), cplexAPI (>= 1.3.3), lpSolve (>= 5.6.13), lpSolveAPI (>= 3.4.4), testthat (>= 2.0.0), data.table (>= 1.11.2), splines2 (>= 0.2.8), Matrix

License GPL-2 | GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

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

Repository CRAN

Date/Publication 2019-05-16 11:10:11 UTC

R topics documented:

`alldup` *Auxiliary function: determine duplicates*

Description

Auxiliary function that takes in a data set, and then determines all the rows that are duplicates. It tags both the original row as well as the duplicates.

Usage

```
alldup(data)
```

Arguments

`data`	the data set on which to check for duplicate rows.

Value

boolean vector. Each entry indicates whether or not the corresponding row has a duplicate.

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 problem of minimizing the violation of observational equivalence for the set of IV-like estimands, while satisfying boundedness and monotonicity constraints declared by the user. Rather than enforce boundedness and monotonicity hold across the entire support of covariates and unobservables, this procedure enforces the conditions over a subset of points in a grid. This grid corresponds to the set of values the covariates can take, and a subset of values of the unobservable term. The size of this grid is specified by the user in the function arguments. The procedure then goes on to check whether the constraints are satisfied at points off the grid. Any point where either the boundedness or monotonicity constraints are violated are incorporated into the grid, and the process is repeated until the grid incorporates the entire support of the covariates, or until some a maximum number of iterations is reached.

Usage

```
audit(data, uname, m0, m1, splinesobj, vars_mtr, terms_mtr0, terms_mtr1,
  grid.nu = 20, grid.nx = 20, audit.nx = 20, audit.nu = 20,
  audit.max = 10, audit.tol = 1e-08, m1.ub, m0.ub, m1.lb, m0.lb,
  m1.ub.default = FALSE, m0.ub.default = FALSE,
  m1.lb.default = FALSE, m0.lb.default = FALSE, mte.ub, mte.lb,
  m0.dec = FALSE, m0.inc = FALSE, m1.dec = FALSE, m1.inc = FALSE,
  mte.dec = FALSE, mte.inc = FALSE, sset, gstar0, gstar1,
  obseq.tol = 0.05, lpsolver, noisy = TRUE, seed = 12345)
```

Arguments

data	data.frame used to estimate the treatment effects.
uname	name declared by user to represent the unobservable term in the MTRs.
m0	one-sided formula for marginal treatment response function for control group. The unobservable term can be entered in using splines.
m1	one-sided formula for marginal treatment response function for treated group.
splinesobj	list of spline components in the MTRs for treated and control groups. Spline terms are extracted using removeSplines.
vars_mtr	all variables entering into the MTRs for treated and control groups.
terms_mtr0	all terms entering into the MTRs for control group.
terms_mtr1	all terms entering into the MTRs for treated group.
grid.nu	number of evenly spread points in the interval [0, 1] of the unobservable u used to form the grid for imposing shape restrictions on the MTRs.
grid.nx	number of evenly spread points of the covariates to use to form the grid for imposing shape restrictions on the MTRs.

audit.nx	number of points on the covariates space to audit in each iteration of the audit procedure.
audit.nu	number of points in the interval [0, 1], corresponding to the normalized value of the unobservable term, to audit in each iteration of the audit procedure.
audit.max	maximum number of iterations in the audit procedure.
audit.tol	tolerance for determining when to end the audit procedure. Namely, if the percentage change in the upper and lower bounds both fall below audit.tol between iterations of the audit, the audit procedure ends.
m1.ub	numeric value for upper bound on MTR for treated group.
m0.ub	numeric value for upper bound on MTR for control group.
m1.lb	numeric value for lower bound on MTR for treated group.
m0.lb	numeric value for lower bound on MTR for control group.
m1.ub.default	boolean, default set to TRUE. Indicator for whether the value assigned was by the user, or set by default.
m0.ub.default	boolean, default set to TRUE. Indicator for whether the value assigned was by the user, or set by default.
m1.lb.default	boolean, default set to TRUE. Indicator for whether the value assigned was by the user, or set by default.
m0.lb.default	boolean, default set to TRUE. Indicator for whether the value assigned was by the user, or set by default.
mte.ub	numeric value for upper bound on treatment effect paramter of interest.
mte.lb	numeric value for lower bound on treatment effect paramter of interest.
m0.dec	logical, equal to TRUE if we want MTR for control group to be weakly monotone decreasing.
m0.inc	logical, equal to TRUE if we want MTR for control group to be weakly monotone increasing.
m1.dec	logical, equal to TRUE if we want MTR for treated group to be weakly monotone decreasing.
m1.inc	logical, equal to TRUE if we want MTR for treated group to be weakly monotone increasing.
mte.dec	logical, equal to TRUE if we want the MTE to be weakly monotone decreasing.
mte.inc	logical, equal to TRUE if we want the MTE to be weakly monotone decreasing.
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.
obseq.tol	tolerance level for how much more the solution is permitted to violate observational equivalence of the IV-like estimands. The threshold multiplies the violation of the observational equivalence, i.e. a threshold of 0 corresponds to the assumption that the model is correctly specified, and that any violation of observational equivalence is due to statistical noise.

lpsolver	name of the linear programming package in R used to obtain the bounds on the treatment effect.
noisy	boolean, set to TRUE by default. If TRUE, then output throughout the audit procedure is printed.
seed	integer, the seed that determines the random grid in the audit procedure.

Value

a list. Included in the list is the minimum violation of observational equivalence of the set of IV-like estimands, as well as the list of matrices and vectors associated with solving the LP problem.

Examples

```
set.seed(10L)

## 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(intercept, d),
                          treat = d,
                          list = FALSE)
```

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

## 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,
      splinesobj = splinesList,
      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,
      lpsolver = "lpSolveAPI")
```

---

bound *Obtaining TE bounds*

Description

This function estimates the bounds on the target treatment effect.

Usage

```
bound(g0, g1, sset, lpobj, obseq.factor, lpsolver, noisy = FALSE)
```

Arguments

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.
lpobj	A list of matrices and vectors defining an LP problem.
obseq.factor	overall multiplicative factor for how much more the solution is permitted to violate observational equivalence of the IV-like estimands, i.e. obseq.factor will multiply minobseq directly.
lpsolver	string, name of the package used to solve the LP problem.
noisy	boolean, set to TRUE if optimization results should be displayed.

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

```
## 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(intercept, d),
                          treat = d,
                          list = FALSE)

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

## 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)

## 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
lpObject <- lpSetup(sset = sSet$sset,
                    mbA = A,
                    mbs = sense,
                    mbrhs = rhs,
                    lpsolver = "lpSolveAPI")

## Estimate the bounds
bound(g0 = targetGamma$gstar0,
      g1 = targetGamma$gstar1,
      sset = sSet$sset,
      lpobj = lpObject,
      obseq.factor = 1,
      lpsolver = "lpSolveAPI")
```

---

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(bound, bound.resamples, n, m, levels, type)
```

Arguments

bound	vector, bound of the treatment effects under partial identification.
bound.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 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.

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(ci, bound, bound.resamples, n, m, levels, type, tol = 1e-08)
```

Arguments

ci	matrix or list. If type is set to 'forward' or 'backward', then ci should be a matrix of forward or backward confidence intervals corresponding to the levels declared in the option levels. If type is set to 'both', then ci should be a list of two elements. One element is a matrix of forward confidence intervals, and the other element is a matrix of backward confidence intervals.
bound	vector, bound of the treatment effects under partial identification.
bound.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 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.
tol	numeric, default set to 1e-08. The p-value is constructed by iteratively adjusting the confidence level to find a confidence interval that does not contain 0. When the adjustment of the confidence level falls below tol, no further iterations are performed.

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.

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 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.

design *Generating design matrices*

Description

This function generates the design matrix given an IV specification.

Usage

```
design(formula, data, subset)
```

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.

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

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

diffA *Taking first differences of constraint matrices*

Description

This function takes in the matrix of values of the MTR evaluated over the grid generated for the audit procedure. The grid is ordered according to the covariates first, and then by the unobservables (this is done in by genmonoA). This function takes the first difference of the unobservables within each set of values for the covariates. This is sufficient to generate the monotonicity constraint matrix.

Usage

```
diffA(A, monogrid, sn, d, ndcols)
```

Arguments

A	a design matrix that evaluates the MTRs over the grid generated for the audit procedure.
monogrid	the grid generated for the audit procedure, sorted by the values of the covariates, and monotone increasing in the unobservable.
sn	the number of binding constraints in the S-set.
d	indicator for treatment group (d=1) versus control group (d = 0).
ndcols	number of terms in the MTR for the other experimental group. This is used to generate a matrix that is of the correct dimension.

Value

a matrix representing the monotonicity restrictions.

dtb *Basic data set distribution*

Description

The distribution from which dtbf was generated. The distribution is defined by the (non-exported) command gendist_basic(). It is worth noting that the unobservable terms are alreayd integrated out.

Usage

```
dtb
```

Format

A data frame with 12 rows and 7 columns.

x covariate

z instrument

p probability of treatment uptake

ey1 counterfactual outcome when a recipient of treatment

ey0 counterfactual outcome when not a recipient of treatment

f density

multiplier number of observations in the data set

Source

Simulated.

`dtbf` *Basic data set*

Description

A simulated population-level data involving only one covariate. The data is constructed by the (non-exported) command `gendist_basic()`. It is worth noting that the unobservable terms are alreayd integrated out.

Usage

```
dtbf
```

Format

A data frame with 1000 rows and 10 columns.

x covariate

z instrument

p probability of treatment uptake

ey1 counterfactual outcome when a recipient of treatment

ey0 counterfactual outcome when not a recipient of treatment

f density

multiplier number of observations in the data set

d indicator for treatment (d = 1) versus control (d = 0) group

i counter for agent of each type

ey the observed outcome

Source

Simulated.

dtc *Covariates data set distribution*

Description

The distribution from which `dtcf` was generated. This distribution set is defined in the (non-exported) command `gendist_covariates()`. It is worth noting that the unobservable terms are alreayd integrated out.

Usage

```
dtc
```

Format

A data frame with 36 rows and 10 columns.

x1 covariate 1

x2 covariate 2

z1 instrument 1

z2 instrument 2

latent latent variable determining treatment participation

p probability of treatment uptake

ey0 counterfactual outcome when not a recipient of treatment

ey1 counterfactual outcome when a recipient of treatment

f density

multiplier number of observations required in data set

Source

Simulated.

dtcf *Covariates data set*

Description

A simulated population-level data set characterizing the effect of a treatment on an outcome. The data includes a treatment indicator, two covariates and two instruments. This data set is generated by the (non-exported) command `gendist_covariates()`. It is worth noting that the unobservable terms are alreayd integrated out.

Usage

```
dtcf
```

Format

A data frame with 10,000 rows and 14 columns.

x1 covariate 1

x2 covariate 2

z1 instrument 1

z2 instrument 2

latent latent variable determining treatment participation

p probability of treatment uptake

ey0 counterfactual outcome when not a recipient of treatment

ey1 counterfactual outcome when a recipient of treatment

f density

multiplier number of observations in data set

d indicator for treatment (d = 1) versus control (d = 0) group

i counter of observations within support of (X, Z)

dcut the number of treated subjects by (X, Z)

ey the observed outcome

Source

Simulated.

---

dtm *Mosquito data set*

Description

A simulated population-level data set characterizing the effect of purchasing mosquito nets, and the likelihood of contracting malaria. A randomly assigned subsidy to purchase mosquito nets was provided as part of an experiment. This data set is generated by the (non-exported) command `gendist_mosquito()`. It is worth noting that the unobservable terms are alreayd integrated out.

Usage

```
dtm
```

Format

A data frame with 400 rows and 7 columns.

i index for observation

z categorical variable for level of subsidy

pz probability of purchasing mosquito net

d indicator for whether or not mosquito net was purchased

ey0 counterfactual probability of contracting malaria conditional on not purchasing a mosquito net.

ey1 counterfactual probability of contracting malaria conditional on purchasing a mosquito net.

ey the observed probability of contracting malaria.

Source

Simulated, based on Mogstad, Torgovitsky (2017).

`dts` *Splines data set distribution*

Description

The distribution from which `dtsf` was generated. The distribution is defined by the (non-exported) command `gendist_splines()`. It is worth noting that the unobservable terms are alreayd integrated out.

Usage

```
dts
```

Format

A data frame with 6 rows and 9 columns.

group indicates combination of (X, Z)

x covariate

z instrument

f density

p probability of treatment uptake

ey1 counterfactual outcome when a recipient of treatment

ey0 counterfactual outcome when not a recipient of treatment

multiplier number of observations in the data set

controls number of controls in the data set

Source

Simulated.

`dtsf` *Splines data set*

Description

A simulated population-level data set characterizing the effect of a treatment on an outcome. The data includes a treatment indicator, a single covariate, and a single covariate. The unobservable terms generating the outcomes are generated according to the following specifications: y1 ~ beta0 + beta1 * x + uSpline(degree = 2, knots = c(0.3, 0.6), intercept = FALSE)

Usage

```
dtsf
```

Format

A data frame with 4,200 rows and 8 columns.

x covariate

z instrument

f density

p probability of treatment uptake

ey1 counterfactual outcome when a recipient of treatment

ey0 counterfactual outcome when not a recipient of treatment

d indicator for treatment (d = 1) versus control (d = 0) group

ey the observed outcome

Details

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)

This data set is generated by the (non-exported) command `gendist_splines()`.

Source

Simulated.

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 LP constraint matrix for bounds*

Description

This function generates the component of the constraint matrix in the LP problem pertaining to 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)
```

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.

Value

a constraint matrix for the LP 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.

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)
```

Arguments

monomials	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 interal. Each element corresponds to an observation.

subset	Subset condition used to select observations with which to estimate gamma.
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.

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

```
## 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(splines, data, lb, ub, multiplier = 1, subset,
  d = NULL, means = TRUE)
```

Arguments

splines	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.

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 an evenly spaced 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 evenly spaced points in the interval [0, 1], including 0 and 1. The number of points is decided by the user.
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 LP 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 we define the LP constraints, 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, gridobj, gstar0, gstar1, m0.dec, m0.inc, m1.dec,
  m1.inc, mte.dec, mte.inc, monov)
```

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.
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.
monov	name of variable for which the monotonicity conditions applies to.

Value

constraint matrix for the LP 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.

Usage

```
genmonoboundA(support, grid_index, uvec, splines, 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)
```

Arguments

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.
splines	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.

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 problem.

genmonomial *Generating monomials*

Description

This function takes in a first vector of coefficients, and a second vector declaring which univariate polynomial basis corresponds to each element ofexponents corresponding to each element of `vector`.

Usage

```
genmonomial(vector, basis, zero = FALSE, as.function = FALSE)
```

Arguments

`vector`	numeric, a vector of coefficients in a polynomial.
`basis`	integer, a vector of the polynomial degrees corresponding to the coefficients in `vector`
`zero`	logical, if `FALSE` then `vector` does not include an element for the constant term. The vector `basis` will need to be adjusted to account for this in order to generate the correct polynomial and monomials.
`as.function`	boolean, if `FALSE` then polynomials are returned; if `TRUE` then a function corresponding to the polynomial is returned.

Value

A list of monomials, in the form of the `polynom` package.

genpolynomial *Generating polynomial functions*

Description

This function takes in a first vector of coefficients, and a second vector declaring which univariate polynomial basis corresponds to each element of the first vector. Then it generates a polynomial function.

Usage

```
genpolynomial(vector, basis, zero = FALSE)
```

Arguments

vector	vector of polynomial coefficients.
basis	vector of exponents corresponding to each element of vector.
zero	logical, if FALSE then vector does not include an element for the constant term. The vector basis will need to be adjusted to account for this in order to generate the correct polynomial and monomials.

Value

A function in the form of the polynom package.

---

genSSet *Generating LP moments for IV-like estimands*

Description

This function takes in the IV estimate and its IV-like specification, and generates a list containing the corresponding point estimate, and the corresponding moments (gammas) that will enter into the constraint matrix of the LP problem.

Usage

```
genSSet(data, sset, sest, splinesobj, pmodobj, pm0, pm1, ncomponents,
  scount, subset_index, means = TRUE, yvar, dvar, noisy = TRUE)
```

Arguments

data	data.frame used to estimate the treatment effects.
sset	A list, which is modified and returned as the output.
sest	A 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	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.
ncomponents	The number of components from the IV regression we want to include in the S-set.
scount	A counter for the number of elements in the S-set.
subset_index	An 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.

Value

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

Examples

```
## 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
## coresponding 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 LP moments for IV-like estimands*

Description

This function takes in the IV estimate and its IV-like specification, and generates a list containing the corresponding point estimate, and the corresponding moments (gammas) that will enter into the constraint matrix of the LP problem.

Usage

```
genTarget(treat, m0, m1, uname, 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,
  point = FALSE, noisy = TRUE)
```

Arguments

`treat`	variable name for treatment indicator
`m0`	one-sided formula for marginal treatment response function for control group. Splines can also be incorporated using the expression "uSplines(degree, knots, intercept)". The 'intercept' argument may be omitted, and is set to `TRUE` by default.
`m1`	one-sided formula for marginal treatment response function for treated group. Splines can also be incorporated using the expression "uSplines(degree, knots, intercept)". The 'intercept' argument may be omitted, and is set to `TRUE` by default.
`uname`	variable name for unobservable used in declaring MTRs
`target`	target parameter to be estimated. Currently 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. 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.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.
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 which we condition on when defining the LATE.
eval.X	numeric vector of the values at which we condition variables in late.X on when estimating the LATE.
genlate.lb	lower bound value of unobservable u for estimating generalized LATE.
genlate.ub	upper bound value of unobservable u for estimating generalized LATE.
data	data.frame 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.
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.
point	boolean, set to FALSE by default. point refers to whether the partial or point identification is desired. If set to FALSE, then the gamma moments are returned, i.e. sample averages are taken. If set to TRUE, 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.
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

```
## 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,
          uname = u,
          target = "atu",
          data = dtm,
          splinesobj = splinesList,
          pmodobj = propensityObj,
          pm0 = polynomials0,
          pm1 = polynomials1,
          point = FALSE)
```

---

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 *Two-step 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 us to estimate the target treatment effect.

Usage

```
gmmEstimate(sset, gstar0, gstar1, 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 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.
`gstar0`	vector, the target gamma moments for d = 0.
`gstar1`	vector, the target gamma moments 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 the point estimate of the treatment effects, the standard errors, the 90 intervals, the convergence code (see `optim`), the coefficients on the MTR, and the variance/covariance matrix of the MTR coefficient estimates.

Examples

```
## 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,
                         uname = u,
                         target = "atu",
                         data = dtm,
                         splinesobj = splinesList,
                         pmodobj = propensityObj,
                         pm0 = polynomials0,
                         pm1 = polynomials1,
                         point = TRUE)

## 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)
```

---

groupby *Auxiliary function: grouping rows in data*

Description

Auxiliary function that takes in data, and assigns a group number to all observations with the same entries in the columns listed in the vector variables.

Usage

```
groupby(data, variables, groupname = ".mst.monog",
  countname = ".mst.monoc", count = TRUE)
```

Arguments

data	data.frame to which the function will assign groups to each row.
variables	vector of the variable/column names in data that will be used to determine the groups.
groupname	name of the column that will be generated to indicate the group.
countname	name of the column that will provide a cumulative count of rows in each group.
count	boolean switch, set to TRUE if the column countname should be generated.

Value

data.frame containing an additional column indicating the group each row falls under. If count is set to TRUE, then an additional column counting the rows within each group is also included.

---

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 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

```
ivEstimate(formula = ey ~ d | z,
           data = dtm,
           components = l(d),
           treat = d,
           list = FALSE)
```

ivj *IV weights*

Description

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

Usage

```
ivj(X, Z, components, treat, order = NULL)
```

Arguments

X	Matrix of covariates, including the treatment indicator.
Z	Matrix of instruments.
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.

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) and Heckman and Vytlacil (2005). The framework on which this function is based was developed by Mogstad, Santos and Torgovitsky (2018). See also the recent survey paper on extrapolation in IV models by Mogstad and Torgovitsky (2018).

Usage

```
ivmte(bootstraps = 0, bootstraps.m, bootstraps.replace = TRUE,
  levels = c(0.99, 0.95, 0.9), ci.type = "both", pvalue.tol = 1e-08,
  ivlike, data, subset, components, propensity, link = "logit", treat,
  m0, m1, uname = u, target, target.weight0 = NULL, target.weight1,
  target.knots0, target.knots1 = NULL, late.Z, late.from, late.to,
  late.X, eval.X, genlate.lb, genlate.ub, obseq.tol = 0.05,
  grid.nu = 20, grid.nx = 20, audit.nx = 20, audit.nu = 20,
  audit.max = 10, audit.tol = 1e-08, m1.ub, m0.ub, m1.lb, m0.lb,
  mte.ub, mte.lb, m0.dec, m0.inc, m1.dec, m1.inc, mte.dec, mte.inc,
  lpsolver = NULL, point = FALSE, noisy = TRUE, seed = 12345)
```

Arguments

bootstraps	integer, default set to 0.
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, 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.
pvalue.tol	numeric, default set to 1e-08. Tolerance level for determining p-value of treatment effect bound.
ivlike	formula or vector of formulas used to specify the regressions for the IV-like estimands.
data	`data.frame` used to estimate the treatment effects.
subset	single subset condition or list of subset conditions corresponding to each IV-like estimand. 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`.
components	a list of vectors of the terms/components from the regressions specifications we want to include in the set of IV-like estimands. 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`. See `l` on how to input the argument. If no components for a IV specification are given, then all components from that IV specification will be included.
propensity	formula or variable name corresponding to propensity to take up treatment. If a formula is declared, then the function estimates propensity score according to the formula and link specified. If a variable name is declared, then the corresponding column in the data is taken as the vector of propensity scores.
link	name of link function to estimate propensity score. Can be chosen from `linear`, `probit`, or `logit`. Default is set to "logit".
treat	variable name for treatment indicator
m0	one-sided formula for marginal treatment response function for control group. Splines can also be incorporated using the expression "uSplines(degree, knots, intercept)". The 'intercept' argument may be omitted, and is set to TRUE by default.

m1	one-sided formula for marginal treatment response function for treated group. Splines can also be incorporated using the expression "uSplines(degree, knots, intercept)". The 'intercept' argument may be omitted, and is set to TRUE by default.
uname	variable name for unobservable used in declaring MTRs.
target	target parameter to be estimated. Currently 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. 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.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 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 splines weights for the treated group. The arguments of the function should be 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.
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 which we condition on when defining the LATE.
eval.X	numeric vector of the values at which we condition variables in late.X on when estimating the LATE.
genlate.lb	lower bound value of unobservable u for estimating generalized LATE.
genlate.ub	upper bound value of unobservable u for estimating generalized LATE.
obseq.tol	threshold for violation of observational equivalence. The threshold enters in multiplicatively. Thus, a value of 0 corresponds to no violation of observational equivalence other than statistical noise, and the assumption that the model is correctly specified.
grid.nu	number of evenly spread points in the interval [0, 1] of the unobservable u used to form the grid for imposing shape restrictions on the MTRs.
grid.nx	number of evenly spread points of the covariates to use to form the grid for imposing shape restrictions on the MTRs.
audit.nx	number of points on the covariates space to audit in each iteration of the audit procedure.

audit.nu	number of points in the interval [0, 1], corresponding to the normalized value of the unobservable term, to audit in each iteration of the audit procedure.
audit.max	maximum number of iterations in the audit procedure.
audit.tol	tolerance for determining when to end the audit procedure.
m1.ub	numeric value for upper bound on MTR for 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 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 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 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 paramter of interest.
mte.lb	numeric value for lower bound on treatment effect paramter of interest.
m0.dec	logical, equal to TRUE if we want MTR for control group to be weakly monotone decreasing.
m0.inc	logical, equal to TRUE if we want MTR for control group to be weakly monotone increasing.
m1.dec	logical, equal to TRUE if we want MTR for treated group to be weakly monotone decreasing.
m1.inc	logical, equal to TRUE if we want MTR for treated group to be weakly monotone increasing.
mte.dec	logical, equal to TRUE if we want the MTE to be weakly monotone decreasing.
mte.inc	logical, equal to TRUE if we want the MTE to be weakly monotone decreasing.
lpsolver	name of the linear programming package in R used to obtain the bounds on the treatment effect.
point	boolean, default set to FALSE. Set to TRUE if it is believed that the treatment effects are point identified. If set to TRUE, then a two-step GMM procedure is implemented to estimate the treatment effects. Shape constraints on the MTRs will be ignored under point identification.
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.
seed	integer, the seed that determines the random grid in the audit procedure.

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 problem.

Examples

```
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,
      lpsolver = "lpSolveAPI")
```

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

Description

This function estimates bounds on treatment effect parameters, following the procedure described in Mogstad, Torgovitsky (2017). Of the target parameters, the user can choose from 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. By restricting the set of coefficients on each term of the MTRs to be consistent with the regression estimates, the function is able to restrict itself to a set of MTRs. The bounds on the treatment effect parameter correspond to finding coefficients on the MTRs that maximize their average difference.

Usage

```
ivmteEstimate(ivlike, data, subset, components, propensity,
  link = "logit", treat, m0, m1, vars_y, vars_mtr, terms_mtr0,
  terms_mtr1, splinesobj, uname = u, target, target.weight0,
  target.weight1, target.knots0 = NULL, target.knots1 = NULL, late.Z,
  late.from, late.to, late.X, eval.X, genlate.lb, genlate.ub,
  obseq.tol = 0.05, grid.nu = 20, grid.nx = 20, audit.nx = 20,
  audit.nu = 20, audit.max = 10, audit.tol = 1e-08, m1.ub, m0.ub,
  m1.lb, m0.lb, mte.ub, mte.lb, m0.dec, m0.inc, m1.dec, m1.inc, mte.dec,
  mte.inc, lpsolver = NULL, point = FALSE, noisy = TRUE,
  seed = 12345)
```

Arguments

ivlike	formula or vector of formulas used to specify the regressions for the IV-like estimands.
data	data.frame used to estimate the treatment effects.
subset	single subset condition or list of subset conditions corresponding to each IV-like estimand. 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.
components	a list of vectors of the terms/components from the regressions specifications we want to include in the set of IV-like estimands. To select the intercept term, include in the vector of variable names, 'intercept'. See l on how to input the argument. If no components for a IV specification are given, then all components from that IV specification will be included.
propensity	formula or variable name corresponding to propensity to take up treatment. If a formula is declared, then the function estimates propensity score according to the formula and link specified. If a variable name is declared, then the corresponding column in the data is taken as the vector of propensity scores.
link	name of link function to estimate propensity score. Can be chosen from linear, probit, or logit. Default is set to "logit".
treat	variable name for treatment indicator.
m0	one-sided formula for marginal treatment response function for control group. Splines can also be incorporated using the expression "uSplines(degree, knots, intercept)". The 'intercept' argument may be omitted, and is set to TRUE by default.
m1	one-sided formula for marginal treatment response function for treated group. Splines can also be incorporated using the expression "uSplines(degree, knots, intercept)". The 'intercept' argument may be omitted, and is set to TRUE by default.
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.
splinesobj	list of spline components in the MTRs for treated and control groups. Spline terms are extracted using removeSplines.
uname	variable name for unobservable used in declaring MTRs.
target	target parameter to be estimated. Currently 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. 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.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.

`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 which we condition on when defining the LATE.

`eval.X` numeric vector of the values at which we condition variables in `late.X` on when estimating the LATE.

`genlate.lb` lower bound value of unobservable u for estimating generalized LATE.

`genlate.ub` upper bound value of unobservable u for estimating generalized LATE.

`obseq.tol` threshold for violation of observational equivalence. The threshold enters in multiplicatively. Thus, a value of 0 corresponds to no violation of observational equivalence other than statistical noise, and the assumption that the model is correctly specified.

`grid.nu` number of evenly spread points in the interval [0, 1] of the unobservable u used to form the grid for imposing shape restrictions on the MTRs.

`grid.nx` number of evenly spread points of the covariates to use to form the grid for imposing shape restrictions on the MTRs.

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

`audit.nu` number of points in the interval [0, 1], corresponding to the normalized value of the unobservable term, to audit in each iteration of the audit procedure.

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

`audit.tol` tolerance for determining when to end the audit procedure.

`m1.ub` numeric value for upper bound on MTR for 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 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 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 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 paramter of interest.
mte.lb	numeric value for lower bound on treatment effect paramter of interest.
m0.dec	logical, equal to TRUE if we want MTR for control group to be weakly monotone decreasing.
m0.inc	logical, equal to TRUE if we want MTR for control group to be weakly monotone increasing.
m1.dec	logical, equal to TRUE if we want MTR for treated group to be weakly monotone decreasing.
m1.inc	logical, equal to TRUE if we want MTR for treated group to be weakly monotone increasing.
mte.dec	logical, equal to TRUE if we want the MTE to be weakly monotone decreasing.
mte.inc	logical, equal to TRUE if we want the MTE to be weakly monotone decreasing.
lpsolver	name of the linear programming package in R used to obtain the bounds on the treatment effect.
point	boolean, default set to FALSE. Set to TRUE if it is believed that the treatment effects are point identified. If set to TRUE, then a GMM procedure is implemented to estimate the treatment effects. Shape constraints on the MTRs will be ignored under point identification.
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.
seed	integer, the seed that determines the random grid in the audit procedure.

Details

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, also. Specifically, bounds and monotonicity restrictions are permitted. These constraints are only enforced over a subset of the data. However, an audit procedure randomly selects points outside of this subset to determine whether or not the constraints hold. The user can specify how stringent this audit procedure is using the function arguments.

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 problem.

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

This function takes in the IV estimates from the set of IV regressions declared by the user, as well as their corresponding moments of the terms in the MTR. These are then used to construct the components that make up the LP problem. Additional constraint matrix is added using `mbA` (`mb` stands for "monotonicity/boundedness"); extra model sense is added using `mbs`; extra RHS values added using `mbrhs`). Depending on the linear programming solver used, this function will return different output specific to the solver.

Usage

```
lpSetup(sset, mbA = NULL, mbs = NULL, mbrhs = NULL, lpsolver,
  shape = TRUE)
```

Arguments

sset	List of IV-like estimates and the corresponding gamma terms.
mbA	Matrix used to define the constraints in the LP problem.
mbs	Vector of model sense/inequalities signs used to define the constraints in the LP problem.
mbrhs	Vector of constants used to define the constraints in the LP problem.
lpsolver	string, name of the package used to solve the LP problem.
shape	boolean, default set to TRUE. Switch to determine whether or not to include shape restrictions in the LP problem.

Value

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

Examples

```
## 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 target gamma moments
ivEstimates <- ivEstimate(formula = ey ~ d | z,
                          data = dtm,
                          components = l(intercept, d),
                          treat = d,
                          list = FALSE)
```

```
## 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)

## Construct the LP problem to be solved using lpSolveAPI
lpSetup(sset = sSet$sset, lpsolver = "lpSolveAPI")
```

`matchrow` *Auxiliary function: matching rows in data to a vector*

Description

For a given vector `match`, this function returns a binary indicator for each row in `data`, telling you whether or not that row matches the entries in the vector `match`. Note this assumes that `data` and `match` have the same column order.

Usage

```
matchrow(data, match)
```

Arguments

`data`	`data.frame` whose rows will be compared against a given vector to determine whether or not they are identical in values.
`match`	the vector against which `data` will be compared against.

Value

binary vector, each element indicating whether the corresponding row in `data` matches the vector `match`.

maxminmatch *Auxiliary function: finding the max/min within a group in a data set*

Description

This function takes the a data set `data`. Assuming the data has a column with the name stored in the `group` argument indicating the groups (see `groupby`), and a column with the `count` variable that ranks the elements in the group (data is assumed to be ordered by `(group, rank)`); this function returns a dummy indicator for each row in `data` that equals to 1 if the `count` entry matches that of `type`, which can either be `max` or `min` (i.e. either the largest in the group, or the smallest).

Usage

```
maxminmatch(data, count = ".mst.monoc", group = ".mst.monog", type)
```

Arguments

`data`	`data.frame` that user wishes to determine the row with the largest or smallest ranking within each group.
`count`	string of column name indicating the ranking of each row within a group.
`group`	string of column name indicating the group of each row.
`type`	input `"max"` to tag the row in each group with the largest rank, and `"min"` to tag the row in each group with the smallest rank.

Value

A binary vector indicating whether or not each corresponding row in `data` is the max/min within its group.

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.

`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.

`obsEqMin` *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.

Usage

```
obsEqMin(sset, lpobj, lpsolver)
```

Arguments

sset	A list of IV-like estimates and the corresponding gamma terms.
lpobj	A list of matrices and vectors defining an LP problem.
lpsolver	string, name of the package used to solve the LP problem.

Value

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

Examples

```
## 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(intercept, d),
                          treat = d,
                          list = FALSE)

## Generate target gamma moments
targetGamma <- genTarget(treat = "d",
                         m0 = ~ 1 + u,
                         m1 = ~ 1 + u,
```

```
                          uname = u,
                          target = "atu",
                          data = dtm,
                          splinesobj = splinesList,
                          pmodobj = propensityObj,
                          pm0 = polynomials0,
                          pm1 = polynomials1,
                          point = FALSE)

## 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)

## 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
lpObject <- lpSetup(sset = sSet$sset,
                    mbA = A,
                    mbs = sense,
                    mbrhs = rhs,
                    lpsolver = "lpSolveAPI")

## Estimate the bounds
obsEqMin(sset = sSet$sset,
         lpobj = lpObject,
         lpsolver = "lpSolveAPI")
```

---

olsj *OLS weights*

Description

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

Usage

```
olsj(X, X0, X1, components, treat)
```

Arguments

X	Matrix of covariates, including the treatment indicator.
X0	Matrix of covariates, once fixing treatment to be 1.
X1	Matrix of covariates, once fixing treatment to be 0.
components	Vector of variable names of which user wants the S-weights for.
treat	Variable name for the treatment indicator.

Value

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

`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, weights = NULL)
```

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.
weights	vector of weights.

Value

vector of select coefficient estimates.

polylisteval *Evaluating polynomials*

Description

This function allows one to evaluate a list of polynomials at various points (the points are allowed to differ across polynomials). This function will instead be used to evaluate a list of monomials.

Usage

```
polylisteval(polynomials, points)
```

Arguments

polynomials	a list of polynomials.
points	a list/vector of points at which we want to evaluate each polynomial.

Value

A matrix of values, corresponding to the polynomials specified in `polynomials` evaluated at the points specified in `points`.

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, 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.
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

```
## 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.

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)
```

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.

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

```
## 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")
```

---

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)
```

Arguments

formula	the formula that is to be parsed.

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)
```

---

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)
```

`runCplexAPI` *Running cplexAPI LP 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.

Usage

```
runCplexAPI(lpobj, lpdir)
```

Arguments

`lpobj`	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.

Value

a list of the output from CPLEX. This includes the optimization status, the objective value, the solution vector, amongst other things.

`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.

Usage

```
runLpSolveAPI(lpobj, modelsense)
```

Arguments

lpobj	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.

Value

a list of the output from lpSolveAPI. This includes the optimization status, the objective value, the solution vector.

---

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.

---

stackA *Stacking monotonicity constraint matrices and vectors*

Description

This function generates the objects in the LP problem associated with the monotonicity constraints declared by the user. This function simply stacks the matrices corresponding to the monotonicity constraints declared by the user. It also stacks the RHS vector associated with the monotonicity constraints, and stacks the vector of inequalities. It is called by the wrapper function genmonoA.

Usage

```
stackA(A0, A1, sset, monogrid, gstar0, gstar1, m0.dec, m0.inc, m1.dec,
  m1.inc, mte.dec, mte.inc)
```

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.

monogrid	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, set to TRUE if MTR for D = 0 should be monotone decreasing in the unobservable.
m0.inc	boolean, set to TRUE if MTR for D = 0 should be monotone increasing in the unobservable.
m1.dec	boolean, set to TRUE if MTR for D = 1 should be monotone decreasing in the unobservable.
m1.inc	boolean, set to TRUE if MTR for D = 1 should be monotone increasing in the unobservable.
mte.dec	boolean, set to TRUE if MTE should be monotone decreasing in the unobservable.
mte.inc	boolean, set to TRUE if MTE should be monotone decreasing in the unobservable.

Value

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

---

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

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, components, treat, order = NULL)
```

Arguments

X	Matrix of covariates, including the treatment indicator.
Z	Matrix of instruments.
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.

tukeydist *Obtain Tukey half-space quantiles*

Description

This function calculates the Tukey half-space quantiles for multivariate random variables.

Usage

```
tukeydist(x, data)
```

Arguments

x	vector of values.
data	a matrix of data, characterizing the empirical distribution.

Value

scalar, representing the quantile of x under the empirical distribution characterized by data.

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.

`vignetteResult` *Vignette results*

Description

The results for the vignette example. This is to avoid having to re-run the code when compiling the vignette.

Usage

```
vignetteResult
```

Format

An object of class `list` of length 31.

wald *Wald weights*

Description

Function generating the S-weighs for Wald estimand. The function operates by using only a single dummy as the instrument. It is up to the user to construct this dummy and subset the data accordingly to get the correct Wald weights.

Usage

```
wald(D, Z)
```

Arguments

D	Matrix, one column being 1s, the other being treatment indicator D.
Z	Matrix, one column being 1s, the other being the instrument dummy.

Value

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

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