Package ‘momentfit’

August 12, 2022

Version 0.3

Date 2022-07-30

Title Methods of Moments

Author Pierre Chausse <pchausse@uwaterloo.ca>

Maintainer Pierre Chausse <pchausse@uwaterloo.ca>

Description Several classes for moment-based models are defined. The classes are defined for moment conditions derived from a single equation or a system of equations. The conditions can also be expressed as functions or formulas. Several methods are also offered to facilitate the development of different estimation techniques. The methods that are currently provided are the Generalized method of moments (Hansen 1982; <doi:10.2307/1912775>), for single equations and systems of equation, and the Generalized Empirical Likelihood (Smith 1997; <doi:10.1111/j.0013-0133.1997.174.x>, Kitamura 1997; <doi:10.1214/aos/1069362388>, Newey and Smith 2004; <doi:10.1111/j.1468-0262.2004.00482.x>, and Anatolyev 2005 <doi:10.1111/j.1468-0262.2005.00601.x>).

Depends R (>= 3.0.0), sandwich

Imports stats, methods, parallel

Suggests lmtest, knitr, texreg

Collate 'allClasses.R' 'validity.R' 'momentData.R' 'momentModel-methods.R' 'momentModel.R' 'momentWeights-methods.R' 'gmmfit-methods.R' 'specTest-methods.R' 'summary-methods.R' 'rModel-methods.R' 'hypothesisTest-methods.R' 'sysMomentModel.R' 'sysMomentModel-methods.R' 'rsysMomentModel-methods.R' 'sgmmfit-methods.R' 'gmm4.R' 'gel.R' 'gelfit-methods.R' 'gel4.R'

NeedsCompilation yes

VignetteBuilder knitr

Repository CRAN

Repository/R-Forge/Project gmm

Repository/R-Forge/Revision 202

Repository/R-Forge/DateTimeStamp 2022-08-12 02:20:25

Date/Publication 2022-08-12 13:10:02 UTC

R topics documented:

allNLModel-class . . . 4
bread-methods . . . 4
CigarettesSW . . . 5
coef-methods . . . 7
confint-class . . . 8
confint-methods . . . 9
ConsumptionG . . . 10
Dresiduals-methods . . . 12
DWH-methods . . . 12
estfun-methods . . . 14
evalDMoment-methods . . . 14
evalGel-methods . . . 16
evalGelObj-methods . . . 18
evalGmm-methods . . . 19
evalGmmObj-methods . . . 20
evalMoment-methods . . . 21
evalWeights-methods . . . 22
formulaModel-class . . . 23
functionModel-class . . . 24
gel4 . . . 26
gelfit-class . . . 28
gelFit-methods . . . 29
getImpProb-methods . . . 30
getRestrict-methods . . . 30
gmm4 . . . 32
gmmfit-class . . . 35
gmmFit-methods . . . 36
Griliches . . . 39
HealthRWM . . . 40
hypothesisTest-class . . . 42
hypothesisTest-methods . . . 43
kernapply-methods . . . 46
Klein . . . 47
LabourCR . . . 48
lambdaAlgo . . . 50
linearModel-class . . . 51
ManufactCost . . . 53
mconfint-class . . . 54
meatGmm-methods . . . 54
merge-methods . . . 56
model.matrix-methods . . . 57
modelDims-methods . . . 59

modelResponse-methods . . . 60
momentModel . . . 60
momentModel-class . . . 62
momentStrength-methods . . . 63
momentWeights-class . . . 64
momFct-methods . . . 65
Mroz . . . 66
nonlinearModel-class . . . 67
plot-methods . . . 69
print-methods . . . 70
printRestrict-methods . . . 71
quadra-methods . . . 71
regModel-class . . . 73
residuals-methods . . . 73
restModel-methods . . . 74
rformulaModel-class . . . 76
rfunctionModel-class . . . 78
rhoFct . . . 79
rlinearModel-class . . . 80
rmomentModel-class . . . 81
rnonlinearModel-class . . . 82
rslinearModel-class . . . 83
rsnonlinearModel-class . . . 85
rsysModel-class . . . 86
setCoef-methods . . . 86
sfunctionModel-class . . . 88
sgmmfit-class . . . 89
show-methods . . . 90
simData . . . 90
slinearModel-class . . . 92
snonlinearModel-class . . . 93
solveGel-methods . . . 94
solveGmm-methods . . . 96
specTest-class . . . 97
specTest-methods . . . 98
sSpec-class . . . 99
stsls-class . . . 100
summary-methods . . . 101
summaryGel-class . . . 102
summaryGmm-class . . . 103
summarySysGmm-class . . . 104
sysModel-class . . . 105
sysMomentModel . . . 106
sysMomentWeights-class . . . 108
systemGmm-doc . . . 109
ThreeSLS-methods . . . 110
tsls-class . . . 111
tsls-methods . . . 112

update-methods 113
vcov-methods 114
vcovHAC-methods 117
[-methods 118

Index **120**

`allNLModel-class` *Class* `"allNLModel"`

Description

A union class for all nonlinear models. It includes `"nonlinearModel"`, `"formulaModel"`, and `"functionModel"`.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

solveGmm `signature(object = "allNLModel", wObj = "momentWeights")`: ...

Examples

```
showClass("allNLModel")
```

`bread-methods` *~~ Methods for Function* `bread` *in Package* **sandwich** *~~*

Description

It computes the bread in the sandwich representation of the covariance matrix of the GMM estimator.

Usage

```
## S4 method for signature 'gmmfit'
bread(x, ...)

## S4 method for signature 'sgmmfit'
bread(x, ...)

## S4 method for signature 'tsls'
bread(x, ...)
```

Arguments

x	GMM fit object
...	Arguments to pass to other methods

Methods

signature(x = "gmmfit")

signature(x = "sgmmfit")

signature(x = "tsls")

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

res <- gmmFit(model1)
m <- meatGmm(res)
b <- bread(res)

## Sandwich vcov
b

## TSLS
model2 <- momentModel(y~x1, ~z1+z2, data=simData, vcov="iid")
res <- tsls(model2)
bread(res)
```

CigarettesSW *Cigarette Consumption Panel Data*

Description

Panel data on cigarette consumption for the 48 continental US States from 1985–1995.

Usage

```
data("CigarettesSW")
```

Format

A data frame containing 48 observations on 7 variables for 2 periods.

state Factor indicating state.

year Factor indicating year.

cpi Consumer price index.

population State population.

packs Number of packs per capita.

income State personal income (total, nominal).

tax Average state, federal and average local excise taxes for fiscal year.

price Average price during fiscal year, including sales tax.

taxs Average excise taxes for fiscal year, including sales tax.

Source

Online complements to Stock and Watson (2007). The dataset and this help file comes from the AER package.

References

Stock, J.H. and Watson, M.W. (2007). *Introduction to Econometrics*, 2nd ed. Boston: Addison Wesley.

Christian Kleiber and Achim Zeileis (2008). Applied Econometrics with R. New York: Springer-Verlag. ISBN 978-0-387-77316-2. URL https://CRAN.R-project.org/package=AER

Examples

```
## Stock and Watson (2007)
## data and transformations
data(CigarettesSW)
CigarettesSW$rprice <- with(CigarettesSW, price/cpi)
CigarettesSW$rincome <- with(CigarettesSW, income/population/cpi)
CigarettesSW$tdiff <- with(CigarettesSW, (taxs - tax)/cpi)
c1985 <- subset(CigarettesSW, year == "1985")
c1995 <- subset(CigarettesSW, year == "1995")

## Equation 12.15
model1 <- momentModel(log(packs)~log(rprice)+log(rincome),
                    ~log(rincome)+tdiff, data = c1995, vcov="MDS")
res1 <- gmmFit(model1)

## HC0 robust se (different from the textbook)
summary(res1, sandwich=TRUE)

## HC1 robust se (like in the textbook)
## A little harder to get, but is it really worth it
## in the case of GMM?

summary(res1, sandwich=TRUE, df.adj=TRUE)@coef

## Equation 12.16
model2<- momentModel(log(packs)~log(rprice)+log(rincome),
                  ~log(rincome)+tdiff+I(tax/cpi), data = c1995,
                  centeredVcov=FALSE, vcov="MDS")
```

```
res2<- tsls(model2)
summary(res2, sandwich=TRUE, df.adj=TRUE)

## Table 12.1
data <- data.frame(dQ=log(c1995$pack/c1985$pack),
                   dP=log(c1995$rprice/c1985$rprice),
                   dTs=c1995$tdiff-c1985$tdiff,
                   dT=c1995$tax/c1995$cpi-c1985$tax/c1985$cpi,
                   dInc=log(c1995$rincome/c1985$rincome))
model1 <- momentModel(dQ~dP+dInc, ~dInc+dTs, vcov="MDS", data=data)
model2 <- momentModel(dQ~dP+dInc, ~dInc+dT, vcov="MDS", data=data)
model3 <- momentModel(dQ~dP+dInc, ~dInc+dTs+dT, vcov="MDS", data=data)

res1 <- tsls(model1)
summary(res1, TRUE, TRUE)
res2 <- tsls(model2)
summary(res2, TRUE, TRUE)
res3 <- tsls(model3)
summary(res3, TRUE, TRUE)
```

`coef-methods` ~~ *Methods for Function* `coef` *in Package* **stats** ~~

Description

It extract the coefficient estimates of some moment-based models.

Methods

`signature(object = "gmmfit")`

`signature(object = "gelfit")`

`signature(object = "sgmmfit")`

`signature(object = "momentModel")`

`signature(object = "rlinearModel")` It gives the unrestricted representation of a restricted model. See examples.

`signature(object = "rslinearModel")` It gives the unrestricted representation of a restricted model.

`signature(object = "rsnonlinearModel")` It gives the unrestricted representation of a restricted model.

`signature(object = "rfunctionModel")` It gives the unrestricted representation of a restricted model. See examples.

`signature(object = "rformulaModel")` It gives the unrestricted representation of a restricted model. See examples.

`signature(object = "rnonlinearModel")` It gives the unrestricted representation of a restricted nonlinear model.

Examples

```
data(simData)
model1 <- momentModel(y~x1+x2+x3+z1, ~x1+x2+z1+z2+z3+z4, data=simData)
res1 <- gmmFit(model1)
coef(res1)

### Restricted models
rmodel1 <- restModel(model1, R=c("x1=1", "x2=2*x3"))
res2 <- gmmFit(rmodel1)
res2
coef(rmodel1, coef(res2))
```

---

confint-class *Class* "confint"

Description

A class to store a confidence interval result.

Objects from the Class

Objects can be created by calls of the form `new("confint", ...)`. It is generated by the `"confint"` method (see `confint-methods`).

Slots

`interval`**:** Object of class `"matrix"` ~~

`type`**:** Object of class `"character"` ~~

`level`**:** Object of class `"numeric"` ~~

`theta`**:** Object of class `"numeric"` ~~

Methods

print `signature(x = "confint")`: ...

show `signature(object = "confint")`: ...

Examples

```
showClass("confint")
```

`confint-methods` *~~ Methods for Function* `confint` *in Package* **stats** *~~*

Description

Method to contruct confidence intervals for objects of class `"gmmfit"` and `"gelfit"`.

Usage

```
## S4 method for signature 'gmmfit'
confint(object, parm, level = 0.95, vcov=NULL,
                    area=FALSE, npoints=50, ...)

## S4 method for signature 'gelfit'
confint(object, parm, level = 0.95, lambda = FALSE,
                    type = c("Wald", "invLR", "invLM", "invJ"),
                    fact = 3, corr = NULL, vcov=NULL,
                    area = FALSE, npoints = 20, cores=4, ...)

## S4 method for signature 'numeric'
confint(object, parm, level = 0.95, gelType="EL",
                    type = c("Wald", "invLR", "invLM", "invJ"),
                    fact = 3, vcov="iid")

## S4 method for signature 'data.frame'
confint(object, parm, level = 0.95, gelType="EL",
                    type = c("Wald", "invLR", "invLM", "invJ"),
                    fact = 3, vcov="iid", npoints=10,
                    cores=4)

## S4 method for signature 'matrix'
confint(object, parm, level = 0.95, gelType="EL",
                    type = c("Wald", "invLR", "invLM", "invJ"),
                    fact = 3, vcov="iid", npoints=10,
                    cores=4)

## S4 method for signature 'ANY'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	Object of class `"gmmfit"`, `"gelfit"`, `"numeric"` or `"data.frame"`.
`parm`	Vector of integers or characters for selecting the elements for which the intervals should be computed.
`level`	The confidence level.
`lambda`	Should be compute intervals for the Lagrange multipliers?

`type` The type of confidence intervals. The default is the Wald interval, and the others are computed by inverting the LR, LM or J specification test.

`fact` For the inversion of the specification tests, `uniroot` searches within `fact` standard error of the coefficient estimates

`corr` Correction to apply to the specification tests

`vcov` For Wald intervals, an optional covariance matrix can be provided. For `"numeric"` or `"data.frame"`, it specifies the type of observations.

`cores` The number of cores for `mclapply`. It is set to 1 for Windows OS.

`gelType` Type of GEL confidence interval.

`npoints` Number of equally spaced points for the confidence region

`area` If TRUE, a cnnfidence region is computed. The length of `"parm"` must be 2 in that case.

`...` Other arguments to pass to `gmmFit` or `gelFit`.

Methods

`signature(object = "ANY")` The method from the **stats** in used in that case.

`signature(object = "gelfit")` Method for any GEL fit class.

`signature(object = "gmmfit")` Method for any GMM fit class.

`signature(object = "numeric")` It computes the GEL confidence interval for the mean.

`signature(object = "data.frame")` It computes the 2D GEL confidence region for the means of two variables.

`signature(object = "matrix")` It converts the object into a data.frame and call its method.

`ConsumptionG` *Consumption data from Greene (2012) applications.*

Description

Quarterly macroeconomic US data from 1950 to 2000.

Usage

```
data("ConsumptionG")
```

Format

A data frame with 204 observations on the following 14 variables.

`YEAR` Year

`QTR` Quarter

`REALGDP` Read GDP

`REALCONS` Real Consumption

`REALINVS` Real Investment

`REALGOVT` Real public expenditure

`REALDPI` ector

`CPI_U` CPI

`M1` Money stock

`TBILRATE` Interest rate

`UNEMP` Unemployment rate

`POP` Population

`INFL` Inflation

`REALINT` Real interest rate.

Source

Greene (2012) online resources: (http://pages.stern.nyu.edu/~wgreene/Text/Edition7/tablelist8new.htm)

References

Green, W.H.. (2012). *Econometric Analysis, 7th edition*, Prentice Hall.

Examples

```
data(ConsumptionG)
## Get the data ready for Table 8.2 of Greene (2012)
Y <- ConsumptionG$REALDPI
C <- ConsumptionG$REALCONS
n <- nrow(ConsumptionG)
Y1 <- Y[-c(1,n)]; Y2 <- Y[-c(n-1,n)]; Y <- Y[-c(1:2)]
C1 <- C[-c(1,n)]; C <- C[-(1:2)]
dat <- data.frame(Y=Y,Y1=Y1,Y2=Y2,C=C,C1=C1)

## Starting at the NLS estimates (from the table)
theta0=c(alpha=468, beta=0.0971, gamma=1.24)

## Greene (2012) seems to assume iid errors (probably wrong assumption here)
model <- momentModel(C~alpha+beta*Y^gamma, ~C1+Y1+Y2, data=dat, theta0=theta0, vcov="iid")

### Scaling the parameters increase the speed of convergence
res <- gmmFit(model, control=list(parscale=c(1000,.1,1)))

### It also seems that there is a degree of freedom adjustment for the
### estimate of the variance of the error term.
summary(res, df.adj=TRUE)@coef
```

`Dresiduals-methods` *~~ Methods for Function* `Dresiduals` *in Package* **Gmm** *~~*

Description

It returns the matrix of derivatives of the residuals with respect to the coefficients.

Methods

`signature(object = "linearModel")`

`signature(object = "nonlinearModel")`

`signature(object = "rsnonlinearModel")`

`signature(object = "sysMomentModel")`

Examples

```
data(simData)

theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

Dresiduals(model1, theta)[1:3,]
```

`DWH-methods` *~~ Methods for Function* `DWH` *in Package* **momentfit** *~~*

Description

It performs the Durbin-Wu-Hausman test on GMM fit models.

Usage

```
## S4 method for signature 'gmmfit,missing'
DWH(object1, object2)

## S4 method for signature 'gmmfit,lm'
DWH(object1, object2,
tol=sqrt(.Machine$double.eps), v1=NULL, v2=NULL, ...)

## S4 method for signature 'gmmfit,gmmfit'
DWH(object1, object2,
tol=sqrt(.Machine$double.eps), v1=NULL, v2=NULL, ...)
```

Arguments

`object1`	Object of class `"gmmfit"`.
`object2`	Object of class `"gmmfit"` or `"lm"`. If missing, the DWH test is a two step test in which the fitted endogenous variables from the first step are added to the regression. In that case, the test a a test of significance of the coefficients of the fitted endogenous variables.
`v1`	Alternatively, we can provide a different covariance matrix for object1
`v2`	Alternatively, we can provide a different covariance matrix for object2
`tol`	Tolerance for the Moore-Penrose generalized inverse
`...`	Argument to pass to `vcov`

Methods

`signature(object1 = "gmmfit", object2 = "lm")`

`signature(object1 = "gmmfit", object2 = "gmmfit")`

`signature(object1 = "gmmfit", object2 = "missing")`

References

Green, W.H.. (2012). *Econometric Analysis, 7th edition*, Prentice Hall.

Examples

```
### Exampe 8.7 of Greene (2012)
data(ConsumptionG)
Y <- ConsumptionG$REALDPI
C <- ConsumptionG$REALCONS
n <- nrow(ConsumptionG)
Y1 <- Y[-n]; Y <- Y[-1]
C1 <- C[-n]; C <- C[-1]
dat <- data.frame(Y=Y,Y1=Y1,C=C,C1=C1)



model1 <- momentModel(C~Y, ~Y, data=dat, vcov="iid")
model2 <- momentModel(C~Y, ~Y1+C1, data=dat, vcov="iid")
res1 <- tsls(model1)
res2 <- tsls(model2)
res <- lm(C~Y)

## Exampke 8.7-2. The difference is explained by the rounding
## error in Greene. Only the first the 3 digits of the t-test are used.
DWH(res2)

## Example 8.7-1. Not quite the same.
DWH(res2, res1)

## using lm object to compare OLS and 2SLS:
```

```
## The same adjustment on the vcov must be done (it is by default in lm)
## otherwise the different in the covariance matrices is mostly caused by the
## different ways to compute them.
DWH(res2, res, df.adj=TRUE)

## To reproduce the same results as Exampke 8.7-1,
## we need to specify the variance.
## But it is not necessary as the above way is
## asymptotically equivalent
X <- model.matrix(model1)
Xhat <- qr.fitted(res2@wObj@w, X)
s2 <- sum(residuals(res)^2)/(res$df.residual)
v1 <-  solve(crossprod(Xhat))*s2
v2 <- solve(crossprod(X))*s2
DWH(res2, res, v1=v1, v2=v2)
```

---

estfun-methods ~~ *Methods for Function* `estfun` *in Package* **sandwich** ~~

Description

Estimating equations for moment models.

Methods

`signature(x = "momentModel")`

---

evalDMoment-methods ~~ *Methods for Function* `evalDMoment` *in Package* **momentfit** ~~

Description

It computes the matrix of derivatives of the sample moments with respect to the coefficients.

Usage

```
## S4 method for signature 'functionModel'
evalDMoment(object, theta, impProb=NULL,
lambda=NULL)

## S4 method for signature 'rfunctionModel'
evalDMoment(object, theta, impProb=NULL,
lambda=NULL)

## S4 method for signature 'rnonlinearModel'
```

```
evalDMoment(object, theta, impProb=NULL,
lambda=NULL)

## S4 method for signature 'formulaModel'
evalDMoment(object, theta, impProb=NULL,
lambda=NULL)

## S4 method for signature 'rformulaModel'
evalDMoment(object, theta, impProb=NULL,
lambda=NULL)

## S4 method for signature 'regModel'
evalDMoment(object, theta, impProb=NULL,
lambda=NULL)

## S4 method for signature 'sysModel'
evalDMoment(object, theta)

## S4 method for signature 'rslinearModel'
evalDMoment(object, theta)

## S4 method for signature 'rsnonlinearModel'
evalDMoment(object, theta, impProb=NULL)
```

Arguments

`object`	An model object
`theta`	A numerical vector of coefficients
`impProb`	If a vector of implied probablities is provided, the sample means are computed using them. If not provided, the means are computed using the uniform weight
`lambda`	A vector of Lagrange multipliers associated with the moment conditions. Its length must therefore match the number of conditions. See details below.

Details

Without the argument `lambda`, the method returns a $q \times k$ matrix, where k is the number of coefficients, and q is the number of moment conditions. That matrix is the derivative of the sample mean of the moments with respect to the coefficient.

If `lambda` is provided, the method returns an $n \times k$ matrix, where n is the sample size. The ith row is $G_i'\lambda$, where G_i is the derivative of the moment function evaluated at the ith observation. For now, this option is used to compute robust-to-misspecified standard errors of GEL estimators.

Methods

`signature(object = "functionModel")`

`signature(object = "rfunctionModel")` The theta vector must match the number of coefficients in the restricted model.

`signature(object = "formulaModel")`

`signature(object = "rformulaModel")` The theta vector must match the number of coefficients in the restricted model.

`signature(object = "regModel")`

`signature(object = "sysModel")`

`signature(object = "rslinearModel")`

Examples

```
data(simData)
theta <- c(1,1)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
G <- evalDMoment(model1, theta)

## A nonlinearModel
g <- y~beta0+x1^beta1
h <- ~z1+z2
model2 <- momentModel(g, h, c(beta0=1, beta1=2), data=simData)
G <- evalDMoment(model2, c(beta0=1, beta1=2))

## A functionModel
fct <- function(tet, x)
    {
        m1 <- (tet[1] - x)
        m2 <- (tet[2]^2 - (x - tet[1])^2)
        m3 <- x^3 - tet[1]*(tet[1]^2 + 3*tet[2]^2)
        f <- cbind(m1, m2, m3)
        return(f)
    }
dfct <- function(tet, x)
        {
       jacobian <- matrix(c( 1, 2*(-tet[1]+mean(x)), -3*tet[1]^2-3*tet[2]^2,0, 2*tet[2],
   -6*tet[1]*tet[2]), nrow=3,ncol=2)
        return(jacobian)
        }
X <- rnorm(200)
model3 <- momentModel(fct, X, theta0=c(beta0=1, beta1=2), grad=dfct)
G <- evalDMoment(model3, c(beta0=1, beta1=2))
```

`evalGel-methods` *~~ Methods for Function* `evalGel` *in Package* **modelfit** *~~*

Description

Method to simply evaluate a GEL model at a fixed coefficient vector. It creates a `"gelfit"` object using that fixed vector.

Usage

```
## S4 method for signature 'momentModel'
evalGel(model, theta, lambda=NULL,
                                    gelType="EL", rhoFct=NULL,
                                    lamSlv=NULL, lControl=list(), ...)
```

Arguments

`model`	An object of class "momentModel".
`theta`	A vector of coefficients at which the model is estimated
`lambda`	The Lagrange multiplier vector. If not provided, the optimal vector is obtained for the given theta
`gelType`	The type of GEL. It is either "EL", "ET", "EEL", "HD", "ETEL" or "ETHD".
`rhoFct`	An alternative objective function for GEL. This argument is only used if we want to fit the model with a different GEL method. see `rhoFct`.
`lamSlv`	An alternative solver for the Lagrange multiplier. By default, either `Wu_lam`, `EEL_lam`, `REEL_lam` or `getLambda` is used.
`lControl`	A list of controls for the Lagrange multiplier algorithm.
`...`	Other arguments to pass. Not used for the moment.

Methods

`signature(model = "momentModel")`

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)

## A linear model with optimal lambda
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
evalGel(model1, c(1,1))

## A nonlinear model with fixed lambda
g <- y~beta0+x1^beta1
h <- ~z1+z2
model2 <- momentModel(g, h, c(beta0=1, beta1=2), data=simData)
evalGel(model2, theta=c(beta1=2, beta0=0.5), lambda=c(.1,.2,.3), gelType="ET")
```

evalGelObj-methods ~~ *Methods for Function* `evalGelObj` *in Package* **Gmm** ~~

Description

~~ Methods to compute the GEL objective function. ~~

Usage

```
## S4 method for signature 'momentModel,numeric,numeric'
evalGelObj(object, theta,
                                                        lambda, gelType,
                                                        rhoFct=NULL, ...)
```

Arguments

`object`	An object of class "`momentModel`"
`theta`	The vector for coefficients.
`lambda`	Vector of Lagrange multiplier.
`gelType`	The type of GEL. It is either "`EL`", "`ET`", "`EEL`", "`HD`", "`ETEL`" or "`ETHD`".
`rhoFct`	An alternative objective function for GEL. This argument is only used if we want to fit the model with a different GEL method. see `rhoFct`.
`...`	Arguments to pass to other methods

Methods

`signature(object = "momentModel", theta = "numeric", lambda = "numeric")`

Examples

```
data(simData)

theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
evalGelObj(model1, theta, c(.2,.3,.4), gelType="EL")
```

`evalGmm-methods` *~~ Methods for Function* `evalGmm` *in Package* **modelfit** *~~*

Description

Method to simply evaluate a GMM model at a fixed coefficient vector. It creates a `"gmmfit"` object using that fixed vector.

Usage

```
## S4 method for signature 'momentModel'
evalGmm(model, theta, wObj=NULL, ...)
## S4 method for signature 'sysModel'
evalGmm(model, theta, wObj=NULL, ...)
```

Arguments

`model`	An object of class `"momentModel"`.
`theta`	A vector of coefficients at which the model is estimated
`wObj`	An object of class `"momentWeights"`. If not provided, the optimal weights based on the specification of the model evaluated at `theta` will be computed.
`...`	Other arguments to pass. Not used for the moment.

Methods

`signature(model = "momentModel")`

`signature(model = "sysModel")`

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)

## A linear model
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
evalGmm(model1, c(1,1))

## A nonlinear model
g <- y~beta0+x1^beta1
h <- ~z1+z2
model2 <- momentModel(g, h, c(beta0=1, beta1=2), data=simData)
evalGmm(model2, theta=c(beta1=2, beta0=0.5))

## A function model
fct <- function(tet, x)
    {
        m1 <- (tet[1] - x)
        m2 <- (tet[2]^2 - (x - tet[1])^2)
```

```
        m3 <- x^3 - tet[1]*(tet[1]^2 + 3*tet[2]^2)
        f <- cbind(m1, m2, m3)
        return(f)
    }
dfct <- function(tet, x)
        {
       jacobian <- matrix(c( 1, 2*(-tet[1]+mean(x)), -3*tet[1]^2-3*tet[2]^2,0, 2*tet[2],
   -6*tet[1]*tet[2]), nrow=3,ncol=2)
        return(jacobian)
        }
model3 <- momentModel(fct, simData$x3, theta0=c(beta0=1, beta1=2), grad=dfct)
evalGmm(model3, theta=c(beta1=.1, beta0=0.3))
```

---

evalGmmObj-methods *~~ Methods for Function* `evalGmmObj` *in Package* **momentfit** ~~

Description

~~ Methods to compute the GMM objective function. ~~

Usage

```
## S4 method for signature 'momentModel,numeric,momentWeights'
evalGmmObj(object, theta,
wObj, ...)

## S4 method for signature 'sysModel,list,sysMomentWeights'
evalGmmObj(object, theta,
wObj, ...)
```

Arguments

`object`	An object of class "`momentModel`", or "`sysMomentModels`".
`theta`	The vector for coefficients for single equation, or a list of vector for system of equations.
`wObj`	An object of class "`momentWeights`" or "`sysMomentWeights`".
`...`	Arguments to pass to other methods

Methods

`signature(object = "momentModel", theta = "numeric", wObj = "momentWeights")`

`signature(object = "sysModel", theta = "list", wObj = "sysMomentWeights")`

Examples

```
data(simData)

theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
w <- evalWeights(model1, theta)
evalGmmObj(model1, theta, w)
```

---

evalMoment-methods ~~ *Methods for Function* `evalMoment` *in Package* **momentfit** ~~

Description

Method to evaluate the moment matrix at a given coefficient vector.

Methods

`signature(object = "functionModel")`

`signature(object = "formulaModel")`

`signature(object = "regModel")`

`signature(object = "sysModel")`

`signature(object = "rsysModel")`

Examples

```
data(simData)
theta <- c(1,1)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
gt <- evalMoment(model1, theta)

## A nonlinearGmm
g <- y~beta0+x1^beta1
h <- ~z1+z2
model2 <- momentModel(g, h, c(beta0=1, beta1=2), data=simData)
gt <- evalMoment(model2, c(beta0=1, beta1=2))

## A functionGmm
fct <- function(tet, x)
    {
        m1 <- (tet[1] - x)
        m2 <- (tet[2]^2 - (x - tet[1])^2)
        m3 <- x^3 - tet[1]*(tet[1]^2 + 3*tet[2]^2)
        f <- cbind(m1, m2, m3)
        return(f)
    }
dfct <- function(tet, x)
        {
```

```
        jacobian <- matrix(c( 1, 2*(-tet[1]+mean(x)), -3*tet[1]^2-3*tet[2]^2,0, 2*tet[2],
    -6*tet[1]*tet[2]), nrow=3,ncol=2)
         return(jacobian)
         }
X <- rnorm(200)
model3 <- momentModel(fct, X, theta0=c(beta0=1, beta1=2), grad=dfct)
gt <- evalMoment(model3, c(beta0=1, beta1=2))
```

---

evalWeights-methods *Methods for Function* evalWeights *in Package* **Gmm**

Description

This is a constructor for objects of class momentWeights

Usage

```
## S4 method for signature 'momentModel'
evalWeights(object, theta=NULL, w="optimal",
...)

## S4 method for signature 'sysModel'
evalWeights(object, theta = NULL, w="optimal",
wObj=NULL)

## S4 method for signature 'rslinearModel'
evalWeights(object, theta = NULL, w="optimal",
wObj=NULL)
```

Arguments

object	Object of class momentModel
theta	The vector of coefficients to compute the optimal weights. If NULL, theta0 for the object is used.
w	A matrix for fixed weights, one of "optimal" or "ident"
wObj	An object of class "sysMomentWeights". Providing it avoid having to recompute Z'Z.
...	Arguments to pass to other methods

Methods

signature(object = "momentModel")

signature(object = "sysModel")

signature(object = "rslinearModel") System of equations with restrictions on the coefficients. It only affects the computation of the weights when there are cross-equation restrictions.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

## Identity weights object
wObj1 <- evalWeights(model1, w="ident")

## Identity weights object (an alternative way less efficient)
wObj1 <- evalWeights(model1, w=diag(3))

## Optimal weights
wObj1 <- evalWeights(model1, theta, w="optimal")
```

---

formulaModel-class   *Class* "formulaModel"

Description

Class for moment-based models for which moments are expressed using formulas.

Objects from the Class

Objects can be created by calls of the form `new("formulaModel", ...)`. It is generated my `momentModel`.

Slots

`modelF`: Object of class `"data.frame"` ~~

`vcov`: Object of class `"character"` ~~

`theta0`: Object of class `"numeric"` ~~

`n`: Object of class `"integer"` ~~

`q`: Object of class `"integer"` ~~

`k`: Object of class `"integer"` ~~

`parNames`: Object of class `"character"` ~~

`momNames`: Object of class `"character"` ~~

`fRHS`: Object of class `"list"` ~~

`fLHS`: Object of class `"list"` ~~

`vcovOptions`: Object of class `"list"` ~~

`centeredVcov`: Object of class `"logical"` ~~

`varNames`: Object of class `"character"` ~~

`isEndo`: Object of class `"logical"` ~~

`isMDE`: Object of class "logical" ~~

`omit`: Object of class "integer" ~~

`survOptions`: Object of class "list" ~~

`sSpec`: Object of class "sSpec" ~~

`smooth`: Object of class "logical" ~~

Extends

Class "allNLModel", directly. Class "momentModel", directly.

Methods

[`signature(x = "formulaModel", i = "numeric", j = "missing")`: ...

evalDMoment `signature(object = "formulaModel")`: ...

evalMoment `signature(object = "formulaModel")`: ...

gmmFit `signature(model = "formulaModel")`: ...

modelDims `signature(object = "formulaModel")`: ...

momentStrength `signature(object = "formulaModel")`: ...

restModel `signature(object = "formulaModel")`: ...

subset `signature(x = "formulaModel")`: ...

Examples

```
showClass("formulaModel")
```

`functionModel-class` *Class* "functionModel"

Description

Class for moment-based models for which moment conditions are defined using a function.

Objects from the Class

Objects can be created by calls of the form `new("functionModel", ...)`. It is generated my `momentModel`.

Slots

X: Object of class "ANY" ~~

fct: Object of class "function" ~~

dfct: Object of class "functionORNULL" ~~

vcov: Object of class "character" ~~

theta0: Object of class "numeric" ~~

n: Object of class "integer" ~~

q: Object of class "integer" ~~

k: Object of class "integer" ~~

parNames: Object of class "character" ~~

momNames: Object of class "character" ~~

vcovOptions: Object of class "list" ~~

centeredVcov: Object of class "logical" ~~

varNames: Object of class "character" ~~

isEndo: Object of class "logical" ~~

omit: Object of class "integer" ~~

survOptions: Object of class "list" ~~

sSpec: Object of class "sSpec" ~~

smooth: Object of class "logical" ~~

Extends

Class "allNLModel", directly. Class "momentModel", directly.

Methods

[signature(x = "functionModel", i = "numeric", j = "missing"): ...

evalDMoment signature(object = "functionModel"): ...

evalMoment signature(object = "functionModel"): ...

modelDims signature(object = "functionModel"): ...

momentStrength signature(object = "functionModel"): ...

restModel signature(object = "functionModel"): ...

subset signature(x = "functionModel"): ...

Examples

```
showClass("functionModel")
```

gel4 *GEL estimation*

Description

The main functions and methods to fit any model with GEL. As opposed to gelFit, models don't need to be created. It is all done by the functions. It is meant to be more user friendly.

Usage

```
gel4(g, x=NULL, theta0=NULL,lambda0=NULL, getVcov=FALSE,
     gelType = c("EL","ET","EEL","HD", "REEL","ETEL","ETHD"),
     vcov = c("MDS","iid","HAC"), grad=NULL,
     vcovOptions=list(), centeredVcov = TRUE,
     cstLHS=NULL, cstRHS=NULL, lamSlv=NULL,
     rhoFct=NULL, initTheta=c("gmm", "theta0"),
     data = parent.frame(),
     coefSlv=c("optim","nlminb","constrOptim"),
     smooth=FALSE,
     lControl=list(), tControl=list())
```

Arguments

g	A function of the form $g(\theta, x)$ and which returns a $n \times q$ matrix with typical element $g_i(\theta, x_t)$ for $i = 1, ...q$ and $t = 1, ..., n$. This matrix is then used to build the q sample moment conditions. It can also be a formula if the model is linear (see detailsbelow).
x	The matrix or vector of data from which the function $g(\theta, x)$ is computed. If "g" is a formula, it is an $n \times Nh$ matrix of instruments or a formula (see details below).
theta0	A $k \times 1$ vector of starting values. It is required only when "g" is a function, a formula or a list of formulas. For these cases, they are needed to create the "momentModel" object.
lambda0	The $q \times 1$ vector of starting values for the Lagrange multipliers. By default a zero vector is used.
getVcov	Should the method computes the covariance matrices of the coefficients and Lagrange multipliers.
gelType	A character string specifying the type of GEL.
vcov	Assumption on the properties of the moment conditions.
grad	A function of the form $G(\theta, x)$ which returns a $q \times k$ matrix of derivatives of $\bar{g}(\theta)$ with respect to θ.
vcovOptions	A list of options for the covariance matrix of the moment conditions. See vcovHAC for the default values.

centeredVcov	Should the moment function be centered when computing its covariance matrix. Doing so may improve inference.
cstLHS	The left hand side of the constraints to impose on the coefficients. See restModel for more details.
cstRHS	The right hand side of the constraints to impose on the coefficients. See restModel for more details.
lamSlv	An alternative solver for the Lagrange multiplier. By default, either Wu_lam, EEL_lam, REEL_lam or getLambda is used. See the vignette for the required format.
rhoFct	An optional function that return $\rho(v)$. This is for users who want a GEL model that is not built in the package. The four arguments of the function must be "gmat", the matrix of moments, "lambda", the vector of Lagrange multipliers, "derive", which specify the order of derivative to return, and k a numeric scale factor required for time series and kernel smoothed moments.
initTheta	Method to obtain the starting values for the coefficient vector. By default the GMM estimate with identity matrix is used. The second argument means that "theta0" is used instead.
data	A required data.frame, in which all variables in g and x can be found.
smooth	If TRUE, "vcov" is set to "MDS" and the moment conditions are smoothed using a kernel. See the vignette for more details.
coefSlv	Minimization solver for the coefficient vector.
lControl	A list of controls for the Lagrange multiplier algorithm.
tControl	A list of controls for the coefficient algorithm.

Value

It returns an object of class "gelfit"

References

Anatolyev, S. (2005), GMM, GEL, Serial Correlation, and Asymptotic Bias. *Econometrica*, **73**, 983-1002.

Andrews DWK (1991), Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation. *Econometrica*, **59**, 817–858.

Kitamura, Yuichi (1997), Empirical Likelihood Methods With Weakly Dependent Processes. *The Annals of Statistics*, **25**, 2084-2102.

Kitamura, Y. and Otsu, T. and Evdokimov, K. (2013), Robustness, Infinitesimal Neighborhoods and Moment Restrictions. *Econometrica*, **81**, 1185-1201.

Newey, W.K. and Smith, R.J. (2004), Higher Order Properties of GMM and Generalized Empirical Likelihood Estimators. *Econometrica*, **72**, 219-255.

Smith, R.J. (2004), GEL Criteria for Moment Condition Models. *Working paper, CEMMAP*.

See Also

gelFit, momentModel

Examples

```
data(simData)
res <- gel4(y~x1, ~z1+z2, vcov="MDS", gelType="ET", data=simData)
res
```

---

gelfit-class *Class* "gelfit"

Description

A class to store fitted models obtained using a GEL method.

Objects from the Class

Objects can be created by calls of the form `new("gelfit", ...)`. It is created by `gelFit`.

Slots

`theta`**:** Object of class "numeric" ~~

`convergence`**:** Object of class "numeric" ~~

`lambda`**:** Object of class "numeric" ~~

`lconvergence`**:** Object of class "numeric" ~~

`call`**:** Object of class "callORNULL" ~~

`gelType`**:** Object of class "list" ~~

`vcov`**:** Object of class "list" ~~

`model`**:** Object of class "momentModel" ~~

`restrictedLam`**:** Object of class "integer" ~~

Methods

coef `signature(object = "gelfit")`: ...

confint `signature(object = "gelfit")`: ...

getImpProb `signature(object = "gelfit")`: ...

momFct `signature(eta = "numeric", object = "gelfit")`: ...

print `signature(x = "gelfit")`: ...

residuals `signature(object = "gelfit")`: ...

show `signature(object = "gelfit")`: ...

specTest `signature(object = "gelfit", which = "missing")`: ...

summary `signature(object = "gelfit")`: ...

update `signature(object = "gelfit")`: ...

vcov `signature(object = "gelfit")`: ...

Examples

```
showClass("gelfit")
```

gelFit-methods *~~ Methods for Function* `gelFit` *in Package* **momentfit** *~~*

Description

Method to fit a model using GEL, from an object of class "momentModel" or its restricted counterpart.

Usage

```
## S4 method for signature 'momentModel'
gelFit(model, gelType="EL", rhoFct=NULL,
              initTheta=c("gmm", "modelTheta0"), theta0=NULL,
              lambda0=NULL, vcov=FALSE, ...)

## S4 method for signature 'rmomentModel'
gelFit(model, gelType="EL", rhoFct=NULL,
              initTheta=c("gmm", "modelTheta0"), theta0=NULL,
              lambda0=NULL, vcov=FALSE, ...)
```

Arguments

`model`	A model class object
`gelType`	The type of GEL. It is either "EL", "ET", "EEL", "HD", "ETEL" or "ETHD".
`rhoFct`	An alternative objective function for GEL. This argument is only used if we want to fit the model with a different GEL method. see `rhoFct`.
`initTheta`	Method to obtain the starting values for the coefficient vector. By default the GMM estimate with identity matrix is used. The second argument means that the theta0 of the object, if any, should be used.
`theta0`	An optional initial vector for `optim` when the model is nonlinear. If provided, the argument "initTheta" is ignored.
`lambda0`	Manual starting values for the Lagrange multiplier. By default, it is a vector of zeros.
`vcov`	Should the method computes the covariance matrices of the coefficients and Lagrange multipliers.
`...`	Arguments to pass to other methods (mostly the optimization algorithm)

Methods

`signature(model = "momentModel")` The main method for all moment-based models.

`signature(model = "rmomentModel")` The main method for all restricted moment-based models.

Examples

```
data(simData)

theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

## EL estimate
res1 <- gelFit(model1)
res1

## ET estimate
res2 <- gelFit(model1, gelType="ET")
res2

## Restricted models by EL
## using the Brent method
R <- matrix(c(0,1), ncol=2)
q <- 2
rmodel1 <- restModel(model1, R, q)
gelFit(rmodel1, tControl=list(method="Brent", lower=-10, upper=10))
```

`getImpProb-methods` ~~ *Methods for Function* `getImpProb` *in Package* **momentfit** ~~

Description

Method to evaluate the implied probabilities of GEL.

Methods

`signature(object = "gelfit")`

`getRestrict-methods` ~~ *Methods for Function* `getRestrict` *in Package* **momentfit** ~~

Description

It computes the matrices related to linear and nonlinear contraints. Those matrices are used to perform hypothesis tests.

Usage

```
## S4 method for signature 'rlinearModel'
getRestrict(object, theta)

## S4 method for signature 'rslinearModel'
getRestrict(object, theta)

## S4 method for signature 'rsnonlinearModel'
getRestrict(object, theta)

## S4 method for signature 'rnonlinearModel'
getRestrict(object, theta)

## S4 method for signature 'rformulaModel'
getRestrict(object, theta)

## S4 method for signature 'momentModel'
getRestrict(object, theta, R, rhs=NULL)

## S4 method for signature 'sysModel'
getRestrict(object, theta, R, rhs=NULL)

## S4 method for signature 'rfunctionModel'
getRestrict(object, theta)
```

Arguments

`object`	Object of class included in `momentModel`, `rmomentModel`, and `rsysModel`.
`theta`	A vector of coefficients for the unrestricted model (see examples).
`R`	A matrix, character or list of formulas that specifies the contraints to impose on the coefficients. See `restModel` for more details.
`rhs`	The right hand side for the restriction on the coefficients. See `restModel` for more details. It is ignored for objects of class `"nonlinearModel"`.

Methods

`signature(object = "momentModel")` A restricted model is created from the constraints, and the restriction matrices are returned. The methods is applied to linear and nonlinear models in a regression form.

`signature(object = "sysModel")` A restricted model is created from the constraints, and the restriction matrices are returned. The methods is applied to systems of linear and nonlinear models.

`signature(object = "rlinearModel")` The restriction matrices are evaluated at the coefficient vector `theta` of the unrestricted representation.

`signature(object = "rslinearModel")` The restriction matrices are evaluated at the coefficient vector `theta` of the unrestricted representation.

signature(object = "rsnonlinearModel") The restriction matrices are evaluated at the coefficient vector theta of the unrestricted representation.

signature(object = "rnonlinearModel") The restriction matrices are evaluated at the coefficient vector theta of the unrestricted representation.

signature(object = "rfunctionModel") The restriction matrices are evaluated at the coefficient vector theta of the unrestricted representation.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)

## Unrestricted model
model1 <- momentModel(y~x1+x2+x3+z1, ~x1+x2+z1+z2+z3+z4, data=simData)

## The restricted model
R1 <- c("x1","2*x2+z1=2", "4+x3*5=3")
res <- gmmFit(model1)
rest <- getRestrict(model1, coef(res), R1)

## it allows to test the restriction
g <- rest$R-rest$q
v <- rest$dR%*%vcov(res)%*%t(rest$dR)
(test <- crossprod(g, solve(v, g)))
(pv <- 1-pchisq(test, length(rest$R)))


## Delta Method:
## To impose nonlinear restrictions, we need to convert
## the linear model into a nonlinear one
NLmodel <- as(model1, "nonlinearModel")
R1 <- c("theta2=2", "theta3=theta4^2")
res <- gmmFit(NLmodel)
rest <- getRestrict(NLmodel, coef(res), R1)

g <- rest$R-rest$q
v <- rest$dR%*%vcov(res)%*%t(rest$dR)
(test <- crossprod(g, solve(v, g)))
(pv <- 1-pchisq(test, length(rest$R)))

## See hypothesisTest method for an easier approach.
```

---

gmm4 *GMM estimation*

Description

The main functions and methods to fit any model with GMM. As opposed to gmmFit, models don't need to be created. It is all done by the functions. It is meant to be more user friendly. This document needs to changed. It is just a copy and paste from the gmm package

Usage

```
gmm4(g, x, theta0 = NULL, grad = NULL,
     type = c("twostep", "iter", "cue", "onestep"),
     vcov = c("iid", "HAC", "MDS", "TrueFixed", "CL"),
     initW = c("ident", "tsls", "EbyE"), weights = "optimal",
     itermaxit = 50, cstLHS=NULL, cstRHS=NULL,
     vcovOptions=list(), survOptions=list(),
     itertol = 1e-07, centeredVcov = TRUE,
     data = parent.frame(), ...)

## S4 method for signature 'formula'
tsls(model, x, vcov = c("iid", "HAC", "MDS", "CL"),
         vcovOptions=list(), survOptions=list(), centeredVcov = TRUE,
         data = parent.frame())

## S4 method for signature 'list'
tsls(model, x=NULL, vcov = c("iid", "HAC", "MDS",
          "CL"), vcovOptions=list(), survOptions=list(),
          centeredVcov = TRUE, data = parent.frame())

## S4 method for signature 'list'
ThreeSLS(model, x=NULL, vcov = c("iid", "HAC", "MDS",
          "CL"), vcovOptions=list(), survOptions=list(),
          centeredVcov = TRUE, data = parent.frame())
```

Arguments

`model` A formula or a list of formulas.

`g` A function of the form $g(\theta, x)$ and which returns a $n \times q$ matrix with typical element $g_i(\theta, x_t)$ for $i = 1, ...q$ and $t = 1, ..., n$. This matrix is then used to build the q sample moment conditions. It can also be a formula if the model is linear or nonlinear, or a list of formulas for systems of equations.

`x` The matrix or vector of data from which the function $g(\theta, x)$ is computed. If "g" is a formula, it is an $n \times Nh$ matrix of instruments or a formula (see details below).

`theta0` A $k \times 1$ vector of starting values. It is required only when "g" is a function or a nonlinear equation defined by a formula, in which case, it must be a named vector

`grad` A function of the form $G(\theta, x)$ which returns a $q \times k$ matrix of derivatives of $\bar{g}(\theta)$ with respect to θ. By default, the numerical algorithm `numericDeriv` is used. It is of course strongly suggested to provide this function when it is possible. This gradient is used to compute the asymptotic covariance matrix of $\hat{\theta}$ and to obtain the analytical gradient of the objective function if the method is set to "CG" or "BFGS" in `optim` and if "type" is not set to "cue". If "g" is a formula, the gradiant is not required (see the details below).

`type` What GMM methods should we use? for `type=="onestep"`, if `"weights"` is not a matrix, the model will be estimated with the weights equals to the identity

	matrix
vcov	Assumption on the properties of the random vector x. By default, x is a weakly dependant process. The "iid" option will avoid using the HAC matrix which will accelerate the estimation if one is ready to make that assumption. The option "TrueFixed" is used only when the matrix of weights is provided and it is the optimal one. For type `CL`, clustered covariance matrix is computed. The options are then included in `vcovOptions` (see `meatCL`).
initW	How should be compute the initial coefficient vector in the first. It only makes a difference for linear models for which the choice is GMM with identity matrix or two-stage least quares.
weights	What weighting matrix to use? The choices are `"optimal"`, in which case it is the inverse of the moment vovariance matrix, `"ident"` for the identity matrix, or a fixed matrix.
itermaxit	Maximum iterations for iterative GMM
itertol	Tolance for the stopping rule in iterative GMM
centeredVcov	Should the moment function be centered when computing its covariance matrix. Doing so may improve inference.
data	A data.frame or a matrix with column names (Optional).
cstLHS	The left hand side of the constraints to impose on the coefficients. See `restModel` for more details.
cstRHS	The right hand side of the constraints to impose on the coefficients. See `restModel` for more details.
vcovOptions	A list of options for the covariance matrix of the moment conditions. See `vcovHAC` for the default values.
survOptions	If needed, a list with the type of survey weights and the weights as a numeric vector, data.frame or formula. The type is either `"sampling"` or `"fequency"`.
...	Arguments to pass to `optim` when the model is nonlinear.

Value

It returns an object of class `"gmmfit"`

References

Zeileis A (2006), Object-oriented Computation of Sandwich Estimators. *Journal of Statistical Software*, **16**(9), 1–16. URL doi:10.18637/jss.v016.i09.

Andrews DWK (1991), Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation. *Econometrica*, **59**, 817–858.

Newey WK & West KD (1987), A Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix. *Econometrica*, **55**, 703–708.

Newey WK & West KD (1994), Automatic Lag Selection in Covariance Matrix Estimation. *Review of Economic Studies*, **61**, 631-653.

Hansen, L.P. (1982), Large Sample Properties of Generalized Method of Moments Estimators. *Econometrica*, **50**, 1029-1054,

Hansen, L.P. and Heaton, J. and Yaron, A.(1996), Finite-Sample Properties of Some Alternative GMM Estimators. *Journal of Business and Economic Statistics*, **14** 262-280.

See Also

gmmFit, momentModel

Examples

```
data(simData)

res <- gmm4(y~x1, ~z1+z2, vcov="MDS", type="iter", data=simData)
res
```

---

gmmfit-class *Class* "gmmfit"

Description

A class to store a fitted model obtained using GMM.

Objects from the Class

Objects can be created by calls of the form `new("gmmfit", ...)`. Generated by gmmFit.

Slots

`theta`**:** Object of class "numeric" ~~

`convergence`**:** Object of class "numericORNULL" ~~

`convIter`**:** Object of class "numericORNULL" ~~

`call`**:** Object of class "callORNULL" ~~

`type`**:** Object of class "character" ~~

`wObj`**:** Object of class "momentWeights" ~~

`niter`**:** Object of class "integer" ~~

`efficientGmm`**:** Object of class "logical" ~~

`model`**:** Object of class "momentModel" ~~

Methods

bread `signature(x = "gmmfit")`: ...

coef `signature(object = "gmmfit")`: ...

confint `signature(object = "gmmfit")`: ...

DWH `signature(object1 = "gmmfit", object2 = "gmmfit")`: ...

DWH `signature(object1 = "gmmfit", object2 = "lm")`: ...

DWH `signature(object1 = "gmmfit", object2 = "missing")`: ...

hypothesisTest `signature(object.u = "gmmfit", object.r = "gmmfit")`: ...

hypothesisTest `signature(object.u = "gmmfit", object.r = "missing")`: ...

hypothesisTest `signature(object.u = "missing", object.r = "gmmfit")`: ...

meatGmm `signature(object = "gmmfit")`: ...

print `signature(x = "gmmfit")`: ...

residuals `signature(object = "gmmfit")`: ...

show `signature(object = "gmmfit")`: ...

specTest `signature(object = "gmmfit", which = "missing")`: ...

specTest `signature(object = "gmmfit", which = "numeric")`: ...

summary `signature(object = "gmmfit")`: ...

update `signature(object = "gmmfit")`: ...

vcov `signature(object = "gmmfit")`: ...

Examples

```
showClass("gmmfit")
```

`gmmFit-methods` *~~ Methods for Function* `gmmFit` *in Package* **momentfit** *~~*

Description

Method to fit a model using GMM, from an object of class "`momentModel`" or "`sysModel`".

Usage

```
## S4 method for signature 'momentModel'
gmmFit(model, type=c("twostep", "iter","cue",
              "onestep"), itertol=1e-7, initW=c("ident", "tsls"),
              weights="optimal", itermaxit=100,
              efficientWeights=FALSE, theta0=NULL, ...)

## S4 method for signature 'formulaModel'
gmmFit(model, type=c("twostep", "iter","cue",
              "onestep"), itertol=1e-7, initW=c("ident", "tsls"),
              weights="optimal", itermaxit=100,
              efficientWeights=FALSE, theta0=NULL, ...)

## S4 method for signature 'sysModel'
gmmFit(model, type=c("twostep", "iter","cue",
              "onestep"), itertol=1e-7, initW=c("ident", "tsls", "EbyE"),
              weights="optimal", itermaxit=100,
              efficientWeights=FALSE, theta0=NULL, EbyE=FALSE, ...)

## S4 method for signature 'rnonlinearModel'
gmmFit(model, type=c("twostep", "iter","cue",
```

```
                "onestep"), itertol=1e-7, initW=c("ident", "tsls"),
                weights="optimal", itermaxit=100,
                efficientWeights=FALSE, theta0=NULL, ...)

## S4 method for signature 'rlinearModel'
gmmFit(model, type=c("twostep", "iter","cue",
                "onestep"), itertol=1e-7, initW=c("ident", "tsls"),
                weights="optimal", itermaxit=100,
                efficientWeights=FALSE, ...)

## S4 method for signature 'rformulaModel'
gmmFit(model, type=c("twostep", "iter","cue",
                "onestep"), itertol=1e-7, initW=c("ident", "tsls"),
                weights="optimal", itermaxit=100,
                efficientWeights=FALSE, theta0=NULL, ...)

## S4 method for signature 'rslinearModel'
gmmFit(model, type=c("twostep", "iter","cue",
                "onestep"), itertol=1e-7, initW=c("ident", "tsls", "EbyE"),
                weights="optimal", itermaxit=100,
                efficientWeights=FALSE, theta0=NULL, EbyE=FALSE, ...)
```

Arguments

`model` A model class object.

`type` What GMM methods should we use? for `type=="onestep"`, if `"weights"` is not a matrix, the model will be estimated with the weights equals to the identity matrix. For restricted

`itertol` Tolance for the stopping rule in iterative GMM

`initW` How should be compute the initial coefficient vector in the first. For single equation GMM, it only makes a difference for linear models for which the choice is GMM with identity matrix or two-stage least quares. For system of equations, `"tsls"`, refers to equation by equation two-stage least squares. It is also possible to start at the equation by equation estimate using the same GMM type as specified by `"type"`.

`weights` What weighting matrix to use? The choices are `"optimal"`, in which case it is the inverse of the moment vovariance matrix, `"ident"` for the identity matrix, or a fixed matrix. It is also possible for weights to be an object of class `gmmWeights`.

`itermaxit` Maximum iterations for iterative GMM

`EbyE` Should the system be estimated equation by equation?

`efficientWeights` If `weights` is a matrix or a `gmmWeights` class object, setting `efficientWeights` to `TRUE` implies that the resulting one-step GMM is efficient. As a result, the default covariance matrix for the coefficient estimates will not be a sandwich type.

`theta0` An optional initial vector for `optim` when the model is nonlinear. By default, the theta0 argument of the model is used

`...` Arguments to pass to other methods (mostly the optimization algorithm)

Methods

`signature(model = "momentModel")` The main method for all moment-based models.

`signature(model = "rnonlinearModel")` It makes a difference only if the number of contraints is equal to the number of coefficients, in which case, the method `evalGmm` is called at the contrained vector. If not, the next method is called.

`signature(model = "rformulaModel")` It makes a difference only if the number of contraints is equal to the number of coefficients, in which case, the method `evalGmm` is called at the contrained vector. If not, the next method is called.

`signature(model = "rlinearModel")` It makes a difference only if the number of contraints is equal to the number of coefficients, in which case, the method `evalGmm` is called at the contrained vector. If not, the next method is called.

`signature(model = "sysModel")` Method to estimate system of equations using GMM methods.

Examples

```
data(simData)

theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

## Efficient GMM with HAC vcov and tsls as first step.
res1 <- gmmFit(model1, init="tsls")

## GMM with identity. Two ways.
res2 <- gmmFit(model1, type="onestep")
res3 <- gmmFit(model1, weights=diag(3))

## nonlinear regression with iterative GMM.
g <- y~beta0+x1^beta1
h <- ~z1+z2
model2 <- momentModel(g, h, c(beta0=1, beta1=2), data=simData)
res4 <- gmmFit(model2, type="iter")

## GMM for with no endogenous vaiables is
## OLS with Robust standard error

library(lmtest)
model3 <- momentModel(y~x1, ~x1, data=simData, vcov="MDS")
resGmm <- gmmFit(model3)
resLm <- lm(y~x1, simData)
summary(resGmm)
coeftest(resLm, vcov=vcovHC(resLm, "HC0"))
summary(resGmm, df.adj=TRUE)
coeftest(resLm, vcov=vcovHC(resLm, "HC1"))
```

```
### All constrained
R <- diag(2)
q <- c(1,2)
rmodel1 <- restModel(model1, R, q)
gmmFit(rmodel1)

## Only one constraint
R <- matrix(c(0,1), ncol=2)
q <- 2
rmodel1 <- restModel(model1, R, q)
gmmFit(rmodel1)
```

Griliches *Return to Education Data*

Description

Labour data on 758 young workers between 16 and 30 years hold. Each observation provides information on one individual at two points in time: in 1980 (variable with 80) and in the year given be the YEAR (variable without 80).

Usage

```
data("Griliches")
```

Format

A data.frame with 758 observations and 20 variables.

RNS, RNS80 Dummy for residency in the southern states

MRT, MRT80 Dummy for marital status (1 if married)

SMSA, SMSA80 Dummy for residency in metropolitan areas

MED Mother's education in years

IQ IQ score

KWW "Knowledge of the World of Work" test score

Year The year of the first observation

AGE, AGE80 Age in years

S, S80 Completed years of schooling

EXPR, EXPR80 Experience in years

TENURE, TENURE80 Tenure im years

LW, LW80 log wage

Source

Online complements of Fumio Hayashi (2000)

References

Griliches, Z. (1976). Wages of Very Young Men. *Journal of Political Economy*, **84**, S69–S85.

Blackburn, M. and Neumark, D. (1992). Unobserved Ability, Efficiency Wages, and Interindustry Wage Differentials. *Quarterly Journal of Economics*, **107**, 1421–1436.

Hayashi, F. (2000). *Econometrics*, New Jersey: Princeton University Press.

`HealthRWM` *Health data from Greene (2012) applications.*

Description

The dataset is used in Greene (2012) and is taken from Riphahn, Wambach, Million (2003).

Usage

```
data("HealthRWM")
```

Format

A data frame with 27326 observations on the following 25 variables.

`ID` Person-identification number

`female` Female=1; male=0

`year` Calendar year of the observation

`age` Age in years

`hsat` Health satisfaction, coded 0 (low) to 10 (high)

`handdum` Handicapped=1; otherwise=0

`handper` Degree of handicap in percent (0 to 100)

`hhninc` Household nominal monthly net income in German marks/10,000

`hhkids` Children under age 16 in the household=1; otherwise=0

`educ` Years of schooling

`married` Married=1; otherwise=0

`haupts` Highest schooling degree is Hauptschul degree=1; otherwise=0

`reals` Highest schooling degree is Realschul degree=1; otherwise=0

`fachhs` Highest schooling degree is Polytechnical degree=1; otherwise=0

`abitur` Highest schooling degree is Abitur=1; otherwise=0

`univ` Highest schooling degree is university degree=1; otherwise=0

`working` Employed=1; otherwise=0

`bluec` Blue-collar employee=1; otherwise=0

`whitec` White-collar employee=1; otherwise=0 `self` Self-employed=1; otherwise=0

`beamt` Civil servant=1; otherwise=0

`docvis` Number of doctor visits in last three months,

`hospvis` Number of hospital visits in last calendar year,

`public` Insured in public health insurance=1; otherwise=0

`addon` Insured by add-on insurance=1; otherwise=0

Source

On Greene (2012) online resources, and on the Journal of Applied Econometrics website (http://qed.econ.queensu.ca/jae/2003-v18.4/riphahn-wambach-million/).

References

Riphahn, R.T. and Wambach, A. and Million, A. (2003), *Incentive Effects in the Demand for Health Care: A Bivariate Panel Count Data Estimation*, Journal of Applied Econometrics, Vol. 18, No. 4, 387–405.

Green, W.H.. (2012). *Econometric Analysis, 7th edition*, Prentice Hall.

Examples

```
###### Example 13.7 of Greene (2012)
####################################

## Selecting the same data point and scaling income
##########
data(HealthRWM)
dat88 <- subset(HealthRWM, year==1988 & hhninc>0)
dat88$hhninc <- dat88$hhninc/10000

### A guess start
thet0 <- c(b0=log(mean(dat88$hhninc)),b1=0,b2=0,b3=0)

## Table 13.2 First column
g <- hhninc~exp(b0+b1*age+b2*educ+b3*female)
res0 <- nls(g, dat88, start=thet0, control=list(maxiter=100))
summary(res0)$coef

## Table 13.2 Second column
## Trying very hard to reproduce the results,
## Who is right?
h1 <- ~age+educ+female
model1 <- momentModel(g, h1, thet0, vcov="MDS", data=dat88)
res1 <- gmmFit(model1, control=list(reltol=1e-10, abstol=1e-10))
summary(res1)@coef

## Table 13.2 third column (close enough)
## Here a sandwich vcov is required because it is not
## efficient GMM
h2 <- ~age+educ+female+hsat+married
```

```
model2 <- momentModel(g, h2, thet0, vcov="MDS", data=dat88)
res2 <- gmmFit(model2, type="onestep")
summary(res2, sandwich=TRUE)@coef

## Table 13.2 fourth column (Can't get closer than that)
res3 <- gmmFit(model2)
summary(res3)@coef

# Lets see what happens if we start on Greene solution

update(res3, theta0=c(b0=-1.61192, b1=.00092, b2=.04647, b3=-.01517))

## No...
```

---

hypothesisTest-class *Class* "hypothesisTest"

Description

A class to store results form an hypothesis test.

Objects from the Class

Objects can be created by calls of the form `new("hypothesisTest", ...)`. It is created by `hypothesisTest`.

Slots

`test`: Object of class "numeric" ~~

`hypothesis`: Object of class "character" ~~

`dist`: Object of class "character" ~~

`df`: Object of class "integer" ~~

`pvalue`: Object of class "numeric" ~~

`type`: Object of class "character" ~~

Methods

print `signature(x = "hypothesisTest")`: ...

show `signature(object = "hypothesisTest")`: ...

Examples

```
showClass("hypothesisTest")
```

hypothesisTest-methods   *~~ Methods for Function* `hypothesisTest` *in Package* **momentfit** *~~*

Description

Performs hypothesis tests on the coefficients estimated by any GMM fit method.

Usage

```
## S4 method for signature 'gmmfit,missing'
hypothesisTest(object.u, object.r, R,
rhs=NULL, vcov=NULL, ...)

## S4 method for signature 'sgmmfit,missing'
hypothesisTest(object.u, object.r, R,
rhs=NULL, vcov=NULL, ...)

## S4 method for signature 'gmmfit,gmmfit'
hypothesisTest(object.u, object.r,
type=c("Wald", "LR", "LM"), sameVcov=TRUE, vcov=NULL,
firstStepWeight=FALSE, wObj=NULL, ...)

## S4 method for signature 'sgmmfit,sgmmfit'
hypothesisTest(object.u, object.r,
type=c("Wald", "LR", "LM"), sameVcov=TRUE, vcov=NULL,
firstStepWeight=FALSE, wObj=NULL, ...)

## S4 method for signature 'missing,gmmfit'
hypothesisTest(object.u, object.r, wObj=NULL)

## S4 method for signature 'missing,sgmmfit'
hypothesisTest(object.u, object.r, wObj=NULL)
```

Arguments

`object.u` An object of class `gmmfit` or `sgmmfit` obtained using an unrestricted `"momentModel"` or `"sysModel"`.

`object.r` An object of class `gmmfit` obtained using a restricted `"momentModel"` or `"sysModel"`.

`R` If it is an object of class `gmmfit`, one of the model fit must be the restricted version of the other. The restrictions are then tested. If `R` is a `character` type, it expresses the restrictions using the coefficient names. If it `numeric`, it must be a matrix and the restrictions are $R\theta = 0$ for `NULL` rhs, or $R\theta = rhs$ otherwise. If missing, the `gmmfit` must be a fitted restricted model, in which case, a LM test is performed.

`rhs` A vector of right hand sides if `R` is `numeric` `type` Should we perform a Wald, LR or LM test?

`sameVcov` For the LR test, should we use the same estimate of the covariance matrix of the moment conditions? See details below.

`vcov` For the Wald test, it is possible to provide the method with the covariance matrix of the coefficients.

`wObj` For the LR test, it is possible to provide the `gmmWeights` object. In that case, the provided gmm weights object if used for the restricted and unrestricted models.

`...` Other argument to pass to `specTest`.

`firstStepWeight` Should we use the first step weighting matrix to compute the test (By default, the optimal weighting matrix is recomputed using the final vector of coefficient estimates). See details below.

Details

The LR test is the difference between the J-tests of the restricted and unrestricted models. It is therefore $n\bar{g}_r'W_r\bar{g}_r - n\bar{g}_u'W_u\bar{g}_u$, where $\bar{g}_r$ and $\bar{g}_u$ are respectively the restricted and unrestricted sample mean of the moment conditions, and W_r and W_u their respective optimal weigthing matrix. The test is therefore invalid if either of the weighting matrices does not converge to the inverse of the covariance matrix of the moment conditions. The restricted and unrestricted models must therefore be estimated by efficient GMM. This is not required for the Wald test.

Asymptotically, it makes no difference which consistent estimate of W_u or W_r is used. However, it will make a difference in finite samples.

If `sameVcov=TRUE`, both W_r and W_u are equal to the the optimal weighting matrix from the unrestricted model if `firstStepWeight=FALSE`, and they are equal to the first step weighting matrix (or the last step for iteratice GMM) of the unrestricted model if it is `TRUE`. For CUE, the value of `firstStepWeight` makes no difference since the weighting matrix and coefficients are computed simultaneously. Having $W_r = W_u$ prevents the test to be negative in small samples.

If `wObj` is provided, both W_r and W_u are equal to it. Of cource, `wObj` must be a consistent estimate of the optimal weighting matrix for the test to be valid.

Methods

`signature(object.u = "gmmfit", object.r = "gmmfit")` Used to test a restricted model against an unrestricted one.

`signature(object.u = "sgmmfit", object.r = "sgmmfit")` Used to test a restricted model against an unrestricted one (for systems of equations).

`signature(object.u = "missing", object.r= "gmmfit")` Used to test a restricted model using the LM test.

`signature(object.u = "missing", object.r= "sgmmfit")` Used to test a restricted model using the LM test (for systems of equations).

`signature(object.u = "gmmfit", object.r = "missing")` Perform a Wald test using an unrestricted model and a restriction matrix or vector.

`signature(object.u = "sgmmfit", object.r = "missing")` Perform a Wald test using an unrestricted model and a restriction matrix or vector in systems of linear equations.

Examples

```
data(simData)

## Unrestricted model
model1 <- momentModel(y~x1+x2+x3, ~x2+x3+z1+z2+z3, data=simData, vcov="MDS")
res1 <- gmmFit(model1)

## Wald test
R <- c("x1=0.5","x2=x3")
hypothesisTest(object.u=res1, R=R)

## LR tests

rmodel1 <- restModel(model1, R)
res2 <- gmmFit(rmodel1)
hypothesisTest(object.u=res1, object.r=res2, type="LR")

### LR and Wald should be the same as long as the same weighting
### matrix if used for both GMM fits, for the LR and Wald as well

# Unrestricted model and save the weights
res1 <- gmmFit(model1)
w <- res1@wObj
# estimate models with the same weights
res2 <- gmmFit(rmodel1, weights=w)

# LR test with the same weights
hypothesisTest(res1, res2, type="LR", wObj=w)

# Wald test with vcov based on the same weights (or the bread)
hypothesisTest(object.u=res1, R=R, breadOnly=TRUE)

### Another example with real data
data(Mroz)
model <- momentModel(log(wage)~educ+exper+I(exper^2),
                  ~exper+I(exper^2)+fatheduc+motheduc, vcov="MDS",
                  data=Mroz, centeredVcov=FALSE)
R <- c("educ=0","I(exper^2)=0")
rmodel <- restModel(model, R)

res1 <- gmmFit(model)
res2 <- gmmFit(rmodel, weights=res1@wObj)

hypothesisTest(object.u=res1, object.r=res2, type="LR", wObj=res1@wObj)
hypothesisTest(object.u=res1, object.r=res2, type="Wald",
vcov=vcov(res1, breadOnly=TRUE))

## LM test (identical to the other two tests as well)

hypothesisTest(object.r=res2)
# or
hypothesisTest(object.u=res1, object.r=res2, type="LM")
```

```
## Wald with the Delta Method:
## To impose nonlinear restrictions, we need to convert
## the linear model into a nonlinear one
NLmodel <- as(model1, "nonlinearModel")
R1 <- c("theta2=2", "theta3=theta4^2")
rNLmodel <- restModel(NLmodel, R1)
res.u <- gmmFit(NLmodel)
res.r <- gmmFit(rNLmodel)
hypothesisTest(object.u=res.u, R=R1)

## LM

hypothesisTest(object.r=res.r)

## LR

hypothesisTest(object.r=res.r, object.u=res.u, type="LR")
```

`kernapply-methods` *A kernel smoothing utility for* "`momentModel`" *classes*

Description

It either generates the optimal bandwidth and kernel weights or the smoothed moments of moment based models.

Usage

```
## S4 method for signature 'momentModel'
kernapply(x, theta=NULL, smooth=TRUE, ...)
```

Arguments

`x`	An object of class "`momentModel`".
`theta`	An optional vector of coefficients. For `smooth=FALSE`, it is used to obtain the optimal bandwidth. If `NULL`, the bandwidth is obtained using one step GMM with the identity matrix as weights. For `smooth=TRUE`, the coefficient is required since the function returns the smoothed moments at a given vector of coefficients.
`smooth`	By default, it returns the smoothed moment matrix. If `FALSE`, it computes the optimal bandwidth and kernel weights.
`...`	Other arguments to pass. Currently not used

Value

It return an object of class "sSpec".

References

Anatolyev, S. (2005), GMM, GEL, Serial Correlation, and Asymptotic Bias. *Econometrica*, **73**, 983-1002.

Kitamura, Yuichi (1997), Empirical Likelihood Methods With Weakly Dependent Processes. *The Annals of Statistics*, **25**, 2084-2102.

Smith, R.J. (2011), GEL Criteria for Moment Condition Models. *Econometric Theory*, **27**(6), 1192–1235.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)

## A linearModel
model1 <- momentModel(y~x1, ~z1+z2, data=simData,vcov="HAC",vcovOptions=list(kernel="Bartlett"))

### get the bandwidth
### Notice that the kernel name is the not the same
### That's because a Truncated kernel for smoothing
### lead to a Bartlett kernel for the HAC of the moments
### See Smith (2011)
kernapply(model1, smooth=FALSE)


### Adding the kernel option to the model

model2 <- momentModel(y~x1, ~z1+z2,
data=simData,vcov="HAC",vcovOptions=list(kernel="Bartlett"), smooth=TRUE)

kernapply(model2, theta)$smoothx[1:5,]
```

Klein *Klein (1950) macro data.*

Description

The data is used to reproduce examples of Greene (2012)

Usage

```
data("Klein")
```

Format

A data frame with 22 observations on the following 10 variables.

`YEAR` a numeric vector

`C` a numeric vector

`P` a numeric vector

`WP` a numeric vector

`I` a numeric vector

`K1` a numeric vector

`X` a numeric vector

`WG` a numeric vector

`G` a numeric vector

`T` a numeric vector

Source

On Greene (2012) online resources.

References

Klein, L. (1950), *Economic Fluctuations in the United-States 1921-1941*, New York: John Wiley and Sons.

Green, W.H.. (2012). *Econometric Analysis, 7th edition*, Prentice Hall.

Examples

```
data(Klein)
```

`LabourCR` *Labour data from Greene (2012) applications,*

Description

A panel data set of 565 individuals from 1976 to 1982 used by Cornwell and Rupert (1988)

Usage

```
data("LabourCR")
```

Format

A data frame with 4165 observations on the following 12 variables.

`EXP` Year of full time experience.

`WKS` Weeks worked.

`OCC` 1 if blue-collar occupation, 0 otherwise.

`IND` 1 if works in a manufacture industry, 0 otherwise.

`SOUTH` 1 if resides in the south, 0 otherwise.

`SMSA` 1 if resides in an SMSA, 0 otherwise.

`MS` 1 if married, 0 otherwise.

`FEM` 1 if the individual is a female and 0 otherwise.

`UNION` 1 if wage is set by a union contract and 0 otherwise.

`ED` Years of education.

`BLK` 1 if the individual is black and 0 otherwise.

`LWAGE` Log wage.

Source

Greene (2012) online resources: (http://pages.stern.nyu.edu/~wgreene/Text/Edition7/tablelist8new.htm)

References

Green, W.H.. (2012). *Econometric Analysis, 7th edition*, Prentice Hall.

Cornwell, C. and Rupert, P. (1988), *Efficient Estimation with Panel Data: An Empirical Comparison of Instrumental Variable Estimators*, Journal of Applied Econometrics, No.3, 149–155.

Examples

```
data(LabourCR)
## Table 8.1 of Greene (2012)
## Model with Z2 (iid is assumed in Table 8.1 given the s.e.)
model2 <- momentModel(WKS~LWAGE+ED+UNION+FEM, ~IND+ED+UNION+FEM+SMSA, vcov="iid",
                   data=LabourCR)
## Model with Z1 using the subsetting method '['
model1 <- model2[-6L]

# Second column
res1 <- tsls(model1)
summary(res1)@coef

# Third column
res2 <- tsls(model2)
summary(res2)@coef
```

lambdaAlgo *Algorithms to solve for the Lagrange multiplier*

Description

The algorithms finds the vector or Lagrange multipliers that maximizes the GEL objective function for a given vector of coefficient θ.

Usage

```
Wu_lam(gmat, tol=1e-8, maxiter=50, k=1)

EEL_lam(gmat, k=1)

REEL_lam(gmat, tol=NULL, maxiter=50, k=1)

ETXX_lam(gmat, lambda0, k, gelType, algo, method, control)

getLambda(gmat, lambda0=NULL, gelType=NULL, rhoFct=NULL,
          tol = 1e-07, maxiter = 100, k = 1, method="BFGS",
          algo = c("nlminb", "optim", "Wu"), control = list(),
          restrictedLam=integer())
```

Arguments

gmat	The $n \times q$ matrix of moments
lambda0	The $q \times 1$ vector of starting values for the Lagrange multipliers.
tol	A tolerance level for the stopping rule in the Wu algorithm
maxiter	The maximum number of iteration in the Wu algorithm
gelType	A character string specifying the type of GEL. The available types are "EL", "ET", "EEL", "HD" and "REEL". For the latter, the algorithm restricts the implied probabilities to be non negative.
rhoFct	An optional function that return $\rho(v)$. This is for users who want a GEL model that is not built in the package. The four arguments of the function must be "gmat", the matrix of moments, "lambda", the vector of Lagrange multipliers, "derive", which specify the order of derivative to return, and k a numeric scale factor required for time series and kernel smoothed moments.
k	A numeric scaling factor that is required when "gmat" is a matrix of time series which require smoothing. The value depends on the kernel and is automatically set when the "gelModels" is created.
method	This is the method for optim.
algo	Which algorithm should be used to maximize the GEL objective function. If set to "Wu", which is only for "EL", the Wu (2005) algorithm is used.
control	A list of control to pass to either optim or nlminb.
restrictedLam	A vector of integers indicating which "lambda" are restricted to be equal to 0.

Details

The ETXX_lam is used for ETEL and ETHD. In general, it computes lambda using ET, and returns the value of the objective function determined by the gelType.

Value

It returns the vector $\rho(gmat\lambda)$ when derive=0, $\rho'(gmat\lambda)$ when derive=1 and $\rho''(gmat\lambda)$ when derive=2.

References

Anatolyev, S. (2005), GMM, GEL, Serial Correlation, and Asymptotic Bias. *Econometrica*, **73**, 983-1002.

Kitamura, Yuichi (1997), Empirical Likelihood Methods With Weakly Dependent Processes. *The Annals of Statistics*, **25**, 2084-2102.

Kitamura, Y. and Otsu, T. and Evdokimov, K. (2013), Robustness, Infinitesimal Neighborhoods and Moment Restrictions. *Econometrica*, **81**, 1185-1201.

Newey, W.K. and Smith, R.J. (2004), Higher Order Properties of GMM and Generalized Empirical Likelihood Estimators. *Econometrica*, **72**, 219-255.

Smith, R.J. (2011), GEL Criteria for Moment Condition Models. *Econometric Theory*, **27**(6), 1192–1235.

Wu, C. (2005), Algorithms and R codes for the pseudo empirical likelihood method in survey sampling. *Survey Methodology*, **31**(2), page 239.

---

linearModel-class *Class* "linearModel"

Description

Class for moment-based models for which moment conditions are linear and expressed by a formula.

Objects from the Class

Objects can be created by calls of the form new("linearModel", ...). It is generated my momentModel.

Slots

modelF: Object of class "data.frame" ~~

instF: Object of class "data.frame" ~~

vcov: Object of class "character" ~~

n: Object of class "integer" ~~

q: Object of class "integer" ~~

k: Object of class "integer" ~~

`parNames`: Object of class "`character`" ~~

`momNames`: Object of class "`character`" ~~

`vcovOptions`: Object of class "`list`" ~~

`centeredVcov`: Object of class "`logical`" ~~

`varNames`: Object of class "`character`" ~~

`isEndo`: Object of class "`logical`" ~~

`omit`: Object of class "`integer`" ~~

`survOptions`: Object of class "`list`" ~~

`sSpec`: Object of class "`sSpec`" ~~

`smooth`: Object of class "`logical`" ~~

Extends

Class "`regModel`", directly. Class "`momentModel`", directly.

Methods

Dresiduals `signature(object = "linearModel")`: ...

merge `signature(x = "linearModel", y = "linearModel")`: ...

merge `signature(x = "slinearModel", y = "linearModel")`: ...

model.matrix `signature(object = "linearModel")`: ...

modelDims `signature(object = "linearModel")`: ...

modelResponse `signature(object = "linearModel")`: ...

momentStrength `signature(object = "linearModel")`: ...

residuals `signature(object = "linearModel")`: ...

restModel `signature(object = "linearModel")`: ...

solveGmm `signature(object = "linearModel", wObj = "momentWeights")`: ...

tsls `signature(model = "linearModel")`: ...

Examples

```
showClass("linearModel")
```

ManufactCost *Manufacturing Costs data from Bernt and Wood (1975)*

Description

The data is used to reproduce examples of Greene (2012)

Usage

```
data("ManufactCost")
```

Format

A data frame with 25 observations on the following 10 variables.

`Year` a numeric vector

`Cost` a numeric vector

`K` a numeric vector

`L` a numeric vector

`E` a numeric vector

`M` a numeric vector

`Pk` a numeric vector

`Pl` a numeric vector

`Pe` a numeric vector

`Pm` a numeric vector

Source

On Greene (2012) online resources.

References

Berndt, E. and Wood, D. (1975), *Technology, Prices, and the Derived Demand for Energy*, Review of Economics and Statistics, Vol. 57, 376–384.

Green, W.H.. (2012). *Econometric Analysis, 7th edition*, Prentice Hall.

Examples

```
data(ManufactCost)
```

mconfint-class *Class* "mconfint"

Description

A class to store confidence region.

Objects from the Class

Objects can be created by calls of the form `new("mconfint", ...)`. It is created by the "confint" method with the option `area=TRUE` (see `confint-methods`).

Slots

`areaPoints`: Object of class "matrix" ~~

`type`: Object of class "character" ~~

`level`: Object of class "numeric" ~~

`theta`: Object of class "numeric" ~~

Methods

plot `signature(x = "mconfint")`: ...

print `signature(x = "mconfint")`: ...

show `signature(object = "mconfint")`: ...

Examples

```
showClass("mconfint")
```

meatGmm-methods ~~ *Methods for Function* `meatGmm` *in Package* **momentfit** ~~

Description

It computes the meat in the sandwich representation of the covariance matrix of the GMM estimator.

Usage

```
## S4 method for signature 'gmmfit'
meatGmm(object, robust=FALSE)

## S4 method for signature 'sgmmfit'
meatGmm(object, robust=FALSE)

## S4 method for signature 'tsls'
meatGmm(object, robust=FALSE)
```

Arguments

`object`	GMM fit object
`robust`	If `TRUE`, the meat is robust to the failure of the assumption that the weighting matrix is the inverse of the covariance matrix of the moment conditions. (see details)

Details

If `robust=FALSE`, then the meat is $G'V^{-1}G$, where G and V are respectively the sample mean of the derivatives and the covariance matrix of the moment conditions. If it is `TRUE`, the meat is $G'WVWG$, where W is the weighting matrix.

For `tsls` objects, the function makes use of the QR representation of the weighting matrix. It is simply possible to get the meat in a more stable way. In that case, $W = (\sigma^2 Z'Z/n)^{-1}$. If `robust` is FALSE, `V` is assumed to be $\sigma^2 Z'Z/n$ which is the inverse of the `bread`. Therefore, a sandwich covariance matrix with `robust=FALSE` will result in a non-sandwich matrix.

For `sgmmfit`, the covariance is for the vectorized coefficient vector of all equations.

Methods

`signature(object = "gmmfit")` General GMM fit.

`signature(object = "tsls")` For model estimated by two-stage least squares.

`signature(object = "sgmmfit")` For system of equations.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

res <- gmmFit(model1)
meatGmm(res)

## It is a slightly different because the weighting matrix
## is computed using the first step estimate and the covariance
## matrix of the moment conditions is based on the final estimate.
## They should, however, be asymptotically equivalent.

meatGmm(res, robust=TRUE)

## TSLS
res2 <- tsls(model1)

## Robust meat
meatGmm(res2, TRUE)

## It makes no difference is the model is assumed iid
model2 <- momentModel(y~x1, ~z1+z2, data=simData, vcov="iid")
res2 <- tsls(model2)
meatGmm(res2, FALSE)
```

```
meatGmm(res2, TRUE)
```

merge-methods ~~ *Methods for Function* `merge` *in Package* **base** ~~

Description

It allows to merge `momentModel` classes into system objects.

Usage

```
## S4 method for signature 'linearModel,linearModel'
merge(x, y, ...)

## S4 method for signature 'nonlinearModel,nonlinearModel'
merge(x, y, ...)

## S4 method for signature 'slinearModel,linearModel'
merge(x, y, ...)

## S4 method for signature 'snonlinearModel,nonlinearModel'
merge(x, y, ...)
```

Arguments

`x`	An object on which the other objects are merged to.
`y`	An object to be merged to `x`.
`...`	Other objects of the same class as `y` to be merged to `x`.

Methods

`signature(x = "linearModel", y = "linearModel")` Merging linear models into a system of equations.

`signature(x = "nonlinearModel", y = "nonlinearModel")` Merging nonlinear models into a system of equations.

`signature(x = "slinearModel", y = "linearModel")` Adding linear equations to a system of linear equations.

`signature(x = "snonlinearModel", y = "nonlinearModel")` Adding nonlinear equations to a system of nonlinear equations.

Examples

```
data(simData)
g1 <- y1~x1+x4; h1 <- ~z1+z2+z3+z4+x4
g2 <- y2~x1+x2+x3; h2 <- ~z1+z2+z3+z4+x3
g3 <- y3~x2+x3+x4; h3 <- ~z2+z3+z4+x3+x4
## Linear models
m1 <- momentModel(g1, h1, data=simData)
m2 <- momentModel(g2, h2, data=simData)
m3 <- momentModel(g3, h3, data=simData)

##
(sys1 <- merge(m1, m2))

## add an equation to the model

(sys2 <- merge(sys1, m3))

## want to get back the first?

sys2[1:2]

## Nonlinear (not really, just written as nonlinear)

nlg <- list(y1~theta0+theta1*x1+theta2*x4,
            y2~alpha0+alpha1*x1+alpha2*x2+alpha3*x3,
            y3~beta0+beta1*x2+beta2*x3+beta3*x4)
theta0 <- list(c(theta0=1,theta1=2,theta2=3),
              c(alpha0=1,alpha1=2,alpha2=3, alpha3=4),
              c(beta0=1,beta1=2,beta2=3,beta3=4))

nm1 <- momentModel(nlg[[1]], h1, theta0[[1]], data=simData)
nm2 <- momentModel(nlg[[2]], h2, theta0[[2]], data=simData)
nm3 <- momentModel(nlg[[3]], h3, theta0[[3]], data=simData)

merge(nm1, nm2, nm3)
```

`model.matrix-methods` *~~ Methods for Function* `model.matrix` *in Package* **stats** *~~*

Description

Model matrix form `momentModel`. It returns the matrix of regressors or the instruments. In restricted models, it returns the reduced matrix of regressors.

Usage

```
## S4 method for signature 'linearModel'
```

```
model.matrix(object,
type=c("regressors","instruments"))
## S4 method for signature 'rlinearModel'
model.matrix(object,
type=c("regressors","instruments"))
## S4 method for signature 'nonlinearModel'
model.matrix(object,
type=c("regressors","instruments"))
## S4 method for signature 'slinearModel'
model.matrix(object,
type=c("regressors","instruments"))
## S4 method for signature 'rslinearModel'
model.matrix(object,
type=c("regressors","instruments"))
## S4 method for signature 'rsnonlinearModel'
model.matrix(object,
type=c("regressors","instruments"))
## S4 method for signature 'snonlinearModel'
model.matrix(object,
type=c("regressors","instruments"))
```

Arguments

object	Object of class `linearModel`, `rlinearModel` or any system of equations class.
type	Should the function returns the matrix of instruments or the matrix of regressors. For `nonlinearModel` classes, type='regressors' will produce an error message, because there is no such model matrix in this case, at least not for now.

Methods

`signature(object = "linearModel")` Linear models with not restrictions.

`signature(object = "nonlinearModel")` Nonlinear models with not restrictions.

`signature(object = "rlinearModel")` linear models with restrictions.

`signature(object = "slinearModel")` System of linear equations with no restrictions.

`signature(object = "rslinearModel")` System of linear equations with restrictions.

`signature(object = "rsnonlinearModel")` System of nonlinear equations with restrictions.

`signature(object = "snonlinearModel")` System of nonlinear equations with no restrictions.

Examples

```
data(simData)

## Unrestricted model
model1 <- momentModel(y~x1+x2+x3, ~x2+x3+z1+z2, data=simData)
model.matrix(model1)[1:3,]

## Restrictions change the response
R <- c("x2=2","x3+x1=3")
```

```
rmodel1 <- restModel(model1, R)
rmodel1
model.matrix(rmodel1)[1:3,]
```

---

modelDims-methods   *Methods for Function* `modelDims`

Description

It extracts important information from the model. It is mostly used by other methods when a `modelModel` has been modifed. An example is when restrictions have been imposed on coefficients.

Methods

`signature(object = "rlinearModel")`

`signature(object = "rnonlinearModel")`

`signature(object = "rfunctionModel")`

`signature(object = "linearModel")`

`signature(object = "nonlinearModel")`

`signature(object = "functionModel")`

`signature(object = "formulaModel")`

`signature(object = "rformulaModel")`

`signature(object = "slinearModel")`

`signature(object = "rslinearModel")`

`signature(object = "rsnonlinearModel")`

`signature(object = "snonlinearModel")`

`signature(object = "sfunctionModel")`

Examples

```
data(simData)

model1 <- momentModel(y~x1+x2, ~x2+z1+z2, data=simData)
modelDims(model1)

## Unrestricted model

rmodel1 <- restModel(model1, R=c("x1+x2=4"))
modelDims(rmodel1)
```

modelResponse-methods *~~ Methods for Function* modelResponse *in Package* **momentfit** *~~*

Description

Return the response vector in models with and without restrictions

Methods

signature(object = "linearModel") For linear models without restrictions on the coefficients.

signature(object = "slinearModel") For system of linear models without restrictions on the coefficients.

signature(object = "rslinearModel") For system of linear models with restrictions on the coefficients.

signature(object = "rlinearModel") For linear models with restrictions on the coefficients.

Examples

```
data(simData)

## Unrestricted model
model1 <- momentModel(y~x1+x2+x3, ~x2+x3+z1+z2, data=simData)
y <- modelResponse(model1)

## Restrictions change the response
R <- c("x2=2","x3=3")
rmodel1 <- restModel(model1, R)
rmodel1
restY <- modelResponse(rmodel1)
```

momentModel *Constructor for* "momentModel" *classes*

Description

It builds an object class "momentModel", which is a union class for "linearModel", "nonlinearModel", "formulaModel" and "functionModel" classes. These are classes for moment based models. This is the first step before running any estimation algorithm.

Usage

```
momentModel(g, x=NULL, theta0=NULL,grad=NULL,
            vcov = c("iid", "HAC", "MDS", "CL"),
            vcovOptions=list(), centeredVcov = TRUE, data=parent.frame(),
            na.action="na.omit", survOptions=list(), smooth=FALSE)
```

Arguments

g
: A function of the form $g(\theta, x)$ and which returns a $n \times q$ matrix with typical element $g_i(\theta, x_t)$ for $i = 1, ...q$ and $t = 1, ..., n$. This matrix is then used to build the q sample moment conditions. It can also be a formula if the model is linear (see detailsbelow).

x
: The matrix or vector of data from which the function $g(\theta, x)$ is computed. If "g" is a formula, it is an $n \times Nh$ matrix of instruments or a formula (see details below).

theta0
: A $k \times 1$ vector of starting values. It is required only when "g" is a function because only then a numerical algorithm is used to minimize the objective function. If the dimension of θ is one, see the argument "optfct".

grad
: A function of the form $G(\theta, x)$ which returns a $q \times k$ matrix of derivatives of $\bar{g}(\theta)$ with respect to θ. By default, the numerical algorithm `numericDeriv` is used. It is of course strongly suggested to provide this function when it is possible. This gradient is used to compute the asymptotic covariance matrix of $\hat{\theta}$ and to obtain the analytical gradient of the objective function if the method is set to "CG" or "BFGS" in `optim` and if "type" is not set to "cue". If "g" is a formula, the gradiant is not required (see the details below).

vcov
: Assumption on the properties of the moment conditions. By default, they are weakly dependant processes. For `MDS`, we assume that the conditions are martingale difference sequences, which implies they are serially uncorrelated, but may be heteroscedastic. There is a difference between `iid` and `MDS` only when `g` is a formula. In that case, residuals are assumed homoscedastic as well as serially uncorrelated. For type `CL`, clustered covariance matrix is computed. The options are then included in `vcovOptions` (see `meatCL`).

vcovOptions
: A list of options for the covariance matrix of the moment conditions. See `vcovHAC` for the default values.

centeredVcov
: Should the moment function be centered when computing its covariance matrix. Doing so may improve inference.

data
: A data.frame or a matrix with column names (Optional).

na.action
: Action to take for missing values. If missing values are present and the option is set to `"na.pass"`, the model won't be estimable.

survOptions
: If needed, a list with the type of survey weights and the weights as a numeric vector, data.frame or formula. The type is either `"sampling"` or `"fequency"`.

smooth
: If `TRUE`, the moment function is smoothed using a kernel method.

Value

'momentModel' returns an object of one of the subclasses of `"momentModel"`.

References

Andrews DWK (1991), Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation. *Econometrica*, **59**, 817–858.

Newey WK & West KD (1987), A Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix. *Econometrica*, **55**, 703–708.

Newey WK & West KD (1994), Automatic Lag Selection in Covariance Matrix Estimation. *Review of Economic Studies*, **61**, 631-653.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)

## A linearModel
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

## A nonlinearModel
g <- y~beta0+x1^beta1
h <- ~z1+z2
model2 <- momentModel(g, h, c(beta0=1, beta1=2), data=simData)

## A functionModel
fct <- function(tet, x)
    {
        m1 <- (tet[1] - x)
        m2 <- (tet[2]^2 - (x - tet[1])^2)
        m3 <- x^3 - tet[1]*(tet[1]^2 + 3*tet[2]^2)
        f <- cbind(m1, m2, m3)
        return(f)
    }
dfct <- function(tet, x)
        {
        jacobian <- matrix(c( 1, 2*(-tet[1]+mean(x)), -3*tet[1]^2-3*tet[2]^2,0, 2*tet[2],
   -6*tet[1]*tet[2]), nrow=3,ncol=2)
        return(jacobian)
        }
model3 <- momentModel(fct, simData$x3, theta0=c(beta0=1, beta1=2), grad=dfct)
```

`momentModel-class` *Class* `"momentModel"`

Description

A union class for all moment based models. It is created by `momentModel`.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

[`signature(x = "momentModel", i = "missing", j = "missing")`: ...

coef `signature(object = "momentModel")`: ...

evalGel `signature(model = "momentModel")`: ...

evalGelObj `signature(object = "momentModel", theta = "numeric", lambda = "numeric")`: ...

evalGmm `signature(model = "momentModel")`: ...

evalGmmObj `signature(object = "momentModel", theta = "numeric", wObj = "momentWeights")`: ...

evalWeights `signature(object = "momentModel")`: ...

gelFit `signature(model = "momentModel")`: ...

getRestrict `signature(object = "momentModel")`: ...

gmmFit `signature(model = "momentModel")`: ...

kernapply `signature(x = "momentModel")`: ...

print `signature(x = "momentModel")`: ...

show `signature(object = "momentModel")`: ...

solveGel `signature(object = "momentModel")`: ...

update `signature(object = "momentModel")`: ...

vcov `signature(object = "momentModel")`: ...

vcovHAC `signature(x = "momentModel")`: ...

Examples

```
showClass("momentModel")
```

`momentStrength-methods`

~~ *Methods for Function* `momentStrength` *in Package* **momentfit** ~~

Description

It produces measures of the strength of the moment conditons.

Usage

```
## S4 method for signature 'linearModel'
momentStrength(object, theta,
vcovType=c("OLS","HC","HAC","CL"))
```

Arguments

object
: An object of class "linearModel"

theta
: Coefficient vector at which the strength must be measured. It does not impact the measure for objects of class linearModel.

vcovType
: Type of covariance matrix used to compute the F-test of the first-stage regression. For HC, the function vcovHC is used with "HC1", and vcovHAC is used with the default setup is "HAC" is chosen. In summary for gmmfit objects, it is adjusted to the type of covariance that is set in the object. For type CL, clustered covariance matrix is computed. The options are the one included in the vcovOptions slot of the object (see meatCL). The object must have be defined with clusters for that to work. See momentModel.

Methods

signature(object = "functionModel") Not implemented yet. In that case, we want some measure of the rank of the matrix of derivatives.

signature(object = "formulaModel") Not implemented yet. In that case, we want some measure of the rank of the matrix of derivatives.

signature(object = "linearModel") It returns the F-test of the first stage regression. It is a measure of the strength of the instruments.

signature(object = "rlinearModel") Returns nothing for now.

signature(object = "nonlinearModel") Not implemented yet.

Examples

```
data(simData)

theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
momentStrength(model1)
```

---

momentWeights-class *Class* "momentWeights"

Description

A class to store the weighting matrix of a set of moment conditions.

Objects from the Class

Objects can be created by calls of the form new("momentWeights", ...). It is created my evalWeights.

Slots

w: Object of class "ANY" ~~

type: Object of class "character" ~~

wSpec: Object of class "list" ~~

Methods

[`signature(x = "momentWeights", i = "missing", j = "missing")`: ...

[`signature(x = "momentWeights", i = "numeric", j = "missing")`: ...

evalGmmObj `signature(object = "momentModel", theta = "numeric", wObj = "momentWeights")`: ...

print `signature(x = "momentWeights")`: ...

quadra `signature(w = "momentWeights", x = "matrixORnumeric", y = "matrixORnumeric")`: ...

quadra `signature(w = "momentWeights", x = "matrixORnumeric", y = "missing")`: ...

quadra `signature(w = "momentWeights", x = "missing", y = "missing")`: ...

show `signature(object = "momentWeights")`: ...

solveGmm `signature(object = "allNLModel", wObj = "momentWeights")`: ...

solveGmm `signature(object = "linearModel", wObj = "momentWeights")`: ...

Examples

```
showClass("momentWeights")
```

---

momFct-methods *Methods for Function* `momFct` *in Package* **momentfit**

Description

The methods computes the moment matrix. It is use to create special moment functions

Usage

```
## S4 method for signature 'numeric,gelfit'
momFct(eta, object)
```

Arguments

`eta`	A vector that includes the coefficient and the Lagrange multipliers
`object`	An object of class "gmmfit"

Methods

`signature(eta = "numeric", object = "gelfit")`

Mroz — *Labour data on married women*

Description

The dataset was used by Mroz (1987) and in examples in Wooldridge (2016)

Usage

```
data("Mroz")
```

Format

A data frame with 753 observations on the following 22 variables.

- `inlf` =1 if in lab frce, 1975
- `hours` hours worked, 1975
- `kidslt6` number of kids < 6 years
- `kidsge6` number of kids 6-18
- `age` woman's age in years
- `educ` years of schooling
- `wage` Estimated wage from earnings and hours
- `repwage` reported wage at interview in 1976
- `hushrs` hours worked by husband, 1975
- `husage` husband's age
- `huseduc` husband's years of schooling
- `huswage` husband's hourly wage, 1975
- `faminc` family income, 1975
- `mtr` federal marginal tax rate facing woman
- `motheduc` mother's years of schooling
- `fatheduc` father's years of schooling
- `unem` unemployment rate in county of residence
- `city` =1 if live in SMSA
- `exper` actual labor market experience
- `nwifeinc` $(faminc - wage * hours)/1000$

Source

From Wooldridge (2016) online resources.

References

Mroz, T.A. (1987), *The Sensitivity of an Empirical Model of Married Women's Hours of Work to Economic and Statistical Assumptions*, Econometrica, **55**, 657–678. 387–405.

Wooldridge, J.M. (2016). *Introductory Econometrics, A Modern Approach, 6th edition*, Cengage Learning.

Examples

```
## Example 15.1 of Wooldridge (2016)

data(Mroz)
Mroz <- subset(Mroz, hours>0)
## I guess IID is assumed (That's how we get the same s.e.)
## By default a sandwich vcov is computed because it is
## a just-identified model.
res4 <- gmm4(log(wage)~educ, ~fatheduc, vcov="iid", data=Mroz)
summary(res4)

## If we adjust the variance of the residuals, however,
## we are a little off (very little)

summary(res4, df.adj=TRUE)


## Example 15.5 of Wooldridge (2016)
## Need to adjust for degrees of freedom in order
## to get the same s.e.
## The first stage F-test is very different though
## Cannot get the same even if do it manually
## with the linearHypothesis from the car package
model <- momentModel(log(wage)~educ+exper+I(exper^2),
~exper+I(exper^2)+fatheduc+motheduc, vcov="iid", data=Mroz)
res <- tsls(model)
summary(res, df.adj=TRUE)
```

`nonlinearModel-class` *Class* "nonlinearModel"

Description

Class for moment-based models for which moment conditions are orthogonality conditions between instruments and the residuals from a nonlinear regression.

Objects from the Class

Objects can be created by calls of the form `new("nonlinearModel", ...)`. It is generated my `momentModel`.

Slots

`modelF`: Object of class `"data.frame"` ~~

`instF`: Object of class `"data.frame"` ~~

`vcov`: Object of class `"character"` ~~

`theta0`: Object of class `"numeric"` ~~

`n`: Object of class `"integer"` ~~

`q`: Object of class `"integer"` ~~

`k`: Object of class `"integer"` ~~

`parNames`: Object of class `"character"` ~~

`momNames`: Object of class `"character"` ~~

`fRHS`: Object of class `"expression"` ~~

`fLHS`: Object of class `"expressionORNULL"` ~~

`vcovOptions`: Object of class `"list"` ~~

`centeredVcov`: Object of class `"logical"` ~~

`varNames`: Object of class `"character"` ~~

`isEndo`: Object of class `"logical"` ~~

`omit`: Object of class `"integer"` ~~

`survOptions`: Object of class `"list"` ~~

`sSpec`: Object of class `"sSpec"` ~~

`smooth`: Object of class `"logical"` ~~

Extends

Class `"regModel"`, directly. Class `"allNLModel"`, directly. Class `"momentModel"`, directly.

Methods

Dresiduals `signature(object = "nonlinearModel")`: ...

merge `signature(x = "nonlinearModel", y = "nonlinearModel")`: ...

merge `signature(x = "snonlinearModel", y = "nonlinearModel")`: ...

model.matrix `signature(object = "nonlinearModel")`: ...

modelDims `signature(object = "nonlinearModel")`: ...

momentStrength `signature(object = "nonlinearModel")`: ...

residuals `signature(object = "nonlinearModel")`: ...

restModel `signature(object = "nonlinearModel")`: ...

Examples

```
showClass("nonlinearModel")
```

plot-methods *~~ Methods for Function* `plot` *from package* **graphics** *~~*

Description

It plots the confidence region.

Usage

```
## S4 method for signature 'ANY'
plot(x, y, ...)

## S4 method for signature 'mconfint'
plot(x, y, main=NULL, xlab=NULL, ylab=NULL,
                          pch=21, bg=1, Pcol=1, ylim=NULL, xlim=NULL,
                          add=FALSE, addEstimates=TRUE, ...)
```

Arguments

`x`	An object to plot
`y`	On used for `"ANY"`.
`main`	Optional title
`xlab`	Optional label for the x-axis.
`ylab`	Optional label for the y-axis.
`pch`	Type of points (see `points`).
`bg`	Background color for points.
`Pcol`	The color for the points. If col is used, it is passed to `polygon`
`xlim`	Optional range for the x-axis.
`ylim`	Optional range for the y-axis.
`add`	If TRUE, the region is added to an existing plot.
`addEstimates`	Should we add the point estimate to the confidence region? This option is only used when add is FALSE.
`...`	Arguments to pass to `polygon`

Methods

`signature(object = "ANY")` It uses the `plot` from package **graphics**

`signature(object = "mconfint")` Plot the 2D confidence region.

print-methods *Methods for Function* `print` *in Package* **base**

Description

Print method for all "momentModel", "gmmfit", "summaryGmm" "hypothesisTest" and "specTest" objects.

Methods

signature(x = "ANY")

signature(x = "momentModel")

signature(x = "sSpec")

signature(x = "confint")

signature(x = "mconfint")

signature(x = "sysModel")

signature(x = "sysMomentWeights")

signature(x = "gmmfit")

signature(x = "gelfit")

signature(x = "sgmmfit")

signature(x = "summaryGmm")

signature(x = "summaryGel")

signature(x = "summarySysGmm")

signature(x = "specTest")

signature(x = "rlinearModel")

signature(x = "rformulaModel")

signature(x = "rslinearModel")

signature(x = "rsnonlinearModel")

signature(x = "rnonlinearModel")

signature(x = "rfunctionModel")

signature(x = "hypothesisTest")

signature(x = "momentWeights")

printRestrict-methods ~~ *Methods for Function* `printRestrict` *in Package* **momentfit** ~~

Description

It prints the detailed restrictions imposed on "momentModel" classes.

Methods

signature(object = "rgelModels")

signature(object = "rlinearModel")

signature(object = "rnonlinearModel")

signature(object = "rfunctionModel")

signature(object = "rformulaModel")

signature(object = "rslinearModel")

signature(object = "rsnonlinearModel")

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)

## Unrestricted model
model1 <- momentModel(y~x1+x2+x3+z1, ~x1+x2+z1+z2+z3+z4, data=simData)

## restricted model
R <- matrix(c(1,1,0,0,0,0,0,2,0,0,0,0,0,1,-1),3,5, byrow=TRUE)
q <- c(0,1,3)
rmodel1 <- restModel(model1, R, q)
printRestrict(rmodel1)
```

quadra-methods ~~ *Methods for Function* `quadra` *in Package* **momentfit** ~~

Description

~~ Computes the quadratic form, where the center matrix is a class `momentWeights` object ~~

Usage

```
## S4 method for signature 'momentWeights,missing,missing'
quadra(w, x, y)

## S4 method for signature 'momentWeights,matrixORnumeric,missing'
quadra(w, x, y)

## S4 method for signature 'momentWeights,matrixORnumeric,matrixORnumeric'
quadra(w, x,
y)

## S4 method for signature 'sysMomentWeights,matrixORnumeric,matrixORnumeric'
quadra(w, x,
y)

## S4 method for signature 'sysMomentWeights,matrixORnumeric,missing'
quadra(w, x, y)

## S4 method for signature 'sysMomentWeights,missing,missing'
quadra(w, x, y)
```

Arguments

w	An object of class "momentWeights"
x	A matrix or numeric vector
y	A matrix or numeric vector

Methods

`signature(w = "momentWeights", x = "matrixORnumeric", y = "matrixORnumeric")` It computes $x'Wy$, where W is the weighting matrix.

`signature(w = "momentWeights", x = "matrixORnumeric", y = "missing")` It computes $x'Wx$, where W is the weighting matrix.

`signature(w = "momentWeights", x = "missing", y = "missing")` It computes W, where W is the weighting matrix. When W is the inverse of the covariance matrix of the moment conditions, it is saved as either a QR decompisition, a Cholesky decomposition or a covariance matrix into the `momentWeights` object. The `quadra` method with no `y` and `x` is therefore a way to invert it. The same applies to system of equations

Examples

```
data(simData)

theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
```

```
gbar <- evalMoment(model1, theta)
gbar <- colMeans(gbar)

### Onjective function of GMM with identity matrix
wObj <- evalWeights(model1, w="ident")
quadra(wObj, gbar)

### Onjective function of GMM with efficient weights
wObj <- evalWeights(model1, theta)
quadra(wObj, gbar)
```

`regModel-class` *Class* `"regModel"`

Description

A union class for `"linearModel"` and `"nonlinearModel"` classes.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

[`signature(x = "regModel", i = "numeric", j = "missing")`: ...

evalDMoment `signature(object = "regModel")`: ...

evalMoment `signature(object = "regModel")`: ...

subset `signature(x = "regModel")`: ...

Examples

```
showClass("regModel")
```

`residuals-methods` *~~ Methods for Function* `residuals` *in Package* **stats** *~~*

Description

It computes the residual for a given coefficient vector, when the model is a linear of nonlinear regression with instruments. The method can be called on a `momentModel` class for a given coefficient theta or on a `gmmfit` object.

Methods

signature(object = "rsysModel")

signature(object = "linearModel")

signature(object = "nonlinearModel")

signature(object = "gmmfit")

signature(object = "gelfit")

signature(object = "sgmmfit")

signature(object = "sysModel")

Examples

```
x <- rchisq(200,5)
z1 <- rnorm(200)
z2 <- .2*x+rnorm(200)
y <- x+rnorm(200)
dat <- data.frame(y=y,z1=z1,x=x,z2=z2)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x, ~z1+z2, data=dat)

## residuals for a given theta
e <- residuals(model1, theta)

## residuals of the fit
res <- gmmFit(model1)
e <- residuals(res)
```

---

restModel-methods *~~ Methods for Function* restModel *in Package* **momentfit** ~~

Description

It converts momentModel objects into its restricted counterpart.

Usage

```
## S4 method for signature 'linearModel'
restModel(object, R, rhs=NULL)

## S4 method for signature 'slinearModel'
restModel(object, R, rhs=NULL)

## S4 method for signature 'snonlinearModel'
restModel(object, R, rhs=NULL)
```

```
## S4 method for signature 'nonlinearModel'
restModel(object, R, rhs=NULL)

## S4 method for signature 'formulaModel'
restModel(object, R, rhs=NULL)

## S4 method for signature 'functionModel'
restModel(object, R, rhs=NULL)
```

Arguments

`object`	An object of class "`momentModel`" or "`sysModel`".
`R`	Either a matrix or a vector of characters for linear models and a list of formulas for nonlinear models
`rhs`	The right hand side of the linear restrictions. It is ignored for nonlinear models.

Methods

`signature(object = "linearModel")` Method for object of class `linearModel`.

`signature(object = "linearGel")` Method for all classes related to `linearGel`.

`signature(object = "slinearModel")` Method for object of class `slinearModel`.

`signature(object = "snonlinearModel")` Method for object of class `snonlinearModel`.

`signature(object = "nonlinearModel")` Method for object of class `nonlinearModel`.

`signature(object = "nonlinearGel")` Method for object of class `nonlinearGel`.

`signature(object = "functionModel")` Method for object of class `functionModel`.

`signature(object = "functionGel")` Method for object of class `functionGel`.

`signature(object = "formulaModel")` Method for object of class `formulaModel`.

`signature(object = "formulaGel")` Method for object of class `formulaGel`.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)

## Unrestricted model
model1 <- momentModel(y~x1+x2+x3+z1, ~x1+x2+z1+z2+z3+z4, data=simData)

## Using matrix R
R <- matrix(c(1,1,0,0,0,0,0,2,0,0,0,0,0,1,-1),3,5, byrow=TRUE)
q <- c(0,1,3)

rmodel1 <- restModel(model1, R, q)
rmodel1

## Using character
## Many ways to write the constraints
```

```
R1 <- c("x1","2*x2+z1=2", "4+x3*5=3")
rmodel1 <- restModel(model1, R1)
rmodel1

## Works with interaction and identity function I()

model1 <- momentModel(y~x1*x2+exp(x3)+I(z1^2), ~x1+x2+z1+z2+z3+z4, data=simData)
R1 <- c("x1","exp(x3)+2*x1:x2", "I(z1^2)=3")
rmodel1 <- restModel(model1, R1)
rmodel1

## nonlinear constraints on a linear model
## we need to convert the linear model into a nonlinear one

model <- momentModel(y~x1+x2+x3+z1, ~x1+x2+z1+z2+z3+z4, data=simData)
NLmodel <- as(model, "nonlinearModel")

## To avoid having unconventional parameter names, which happens
## when I() is used or with interaction, the X's and coefficients are
## renamed

NLmodel@parNames

## Restriction can be a list of formula or vector of characters
## For the latter, it will be converted into a list of formulas

R1 <- c("theta2=2", "theta3=theta4^2")
rmod1 <- restModel(NLmodel, R1)
res1 <- gmmFit(rmod1)
res1
## recover the orignial form
coef(rmod1, coef(res1))

## with formulas

R2 <- list(theta2~2, theta3~1/theta4)
rmod2 <- restModel(NLmodel, R2)
res2 <- gmmFit(rmod2)
res2
coef(rmod2, coef(res2))

## The same can be done with function based models
```

---

rformulaModel-class *Class* "rformulaModel"

Description

A class for restricted moment-based models for which moment conditions are expressed using a list of formulas.

Objects from the Class

Objects can be created by calls of the form `new("rformulaModel", ...)`. It is created by `restModel-methods`.

Slots

`R`: Object of class `"list"` ~~

`cstSpec`: Object of class `"list"` ~~

`modelF`: Object of class `"data.frame"` ~~

`vcov`: Object of class `"character"` ~~

`theta0`: Object of class `"numeric"` ~~

`n`: Object of class `"integer"` ~~

`q`: Object of class `"integer"` ~~

`k`: Object of class `"integer"` ~~

`parNames`: Object of class `"character"` ~~

`momNames`: Object of class `"character"` ~~

`fRHS`: Object of class `"list"` ~~

`fLHS`: Object of class `"list"` ~~

`vcovOptions`: Object of class `"list"` ~~

`centeredVcov`: Object of class `"logical"` ~~

`varNames`: Object of class `"character"` ~~

`isEndo`: Object of class `"logical"` ~~

`isMDE`: Object of class `"logical"` ~~

`omit`: Object of class `"integer"` ~~

`survOptions`: Object of class `"list"` ~~

`sSpec`: Object of class `"sSpec"` ~~

`smooth`: Object of class `"logical"` ~~

Extends

Class `"formulaModel"`, directly. Class `"rmomentModel"`, directly. Class `"allNLModel"`, by class "formulaModel", distance 2. Class `"momentModel"`, by class "formulaModel", distance 2.

Methods

coef `signature(object = "rformulaModel")`: ...

evalDMoment `signature(object = "rformulaModel")`: ...

getRestrict `signature(object = "rformulaModel")`: ...

gmmFit `signature(model = "rformulaModel")`: ...

modelDims `signature(object = "rformulaModel")`: ...

print `signature(x = "rformulaModel")`: ...

printRestrict `signature(object = "rformulaModel")`: ...

Examples

```
showClass("rformulaModel")
```

---

rfunctionModel-class   *Class* "rfunctionModel"

Description

A restricted moment-based model for which moment conditions are defined by a function.

Objects from the Class

Objects can be created by calls of the form `new("rfunctionModel", ...)`. It is created by `restModel-methods`.

Slots

`R`: Object of class `"list"` ~~

`cstSpec`: Object of class `"list"` ~~

`X`: Object of class `"ANY"` ~~

`fct`: Object of class `"function"` ~~

`dfct`: Object of class `"functionORNULL"` ~~

`vcov`: Object of class `"character"` ~~

`theta0`: Object of class `"numeric"` ~~

`n`: Object of class `"integer"` ~~

`q`: Object of class `"integer"` ~~

`k`: Object of class `"integer"` ~~

`parNames`: Object of class `"character"` ~~

`momNames`: Object of class `"character"` ~~

`vcovOptions`: Object of class `"list"` ~~

`centeredVcov`: Object of class `"logical"` ~~

`varNames`: Object of class `"character"` ~~

`isEndo`: Object of class `"logical"` ~~

`omit`: Object of class `"integer"` ~~

`survOptions`: Object of class `"list"` ~~

`sSpec`: Object of class `"sSpec"` ~~

`smooth`: Object of class `"logical"` ~~

Extends

Class `"functionModel"`, directly. Class `"rmomentModel"`, directly. Class `"allNLModel"`, by class "functionModel", distance 2. Class `"momentModel"`, by class "functionModel", distance 2.

Methods

[`signature(x = "rfunctionModel", i = "numeric", j = "missing")`: ...

coef `signature(object = "rfunctionModel")`: ...

evalDMoment `signature(object = "rfunctionModel")`: ...

getRestrict `signature(object = "rfunctionModel")`: ...

modelDims `signature(object = "rfunctionModel")`: ...

print `signature(x = "rfunctionModel")`: ...

printRestrict `signature(object = "rfunctionModel")`: ...

Examples

```
showClass("rfunctionModel")
```

`rhoFct` *GEL objective functions*

Description

Functions that returns the GEL function $\rho(g(\theta, x)'\lambda)$ and its derivatives.

Usage

```
rhoET(gmat, lambda, derive = 0, k = 1)

rhoETEL(gmat, lambda, derive = 0, k = 1)

rhoEL(gmat, lambda, derive = 0, k = 1)

rhoEEL(gmat, lambda, derive = 0, k = 1)

rhoREEL(gmat, lambda, derive = 0, k = 1)

rhoHD(gmat, lambda, derive = 0, k = 1)

rhoETHD(gmat, lambda, derive = 0, k = 1)
```

Arguments

`gmat`	The $n \times q$ matrix of moments
`lambda`	The $q \times 1$ vector of Lagrange multipliers.
`derive`	An integer which indicates which derivative to return
`k`	A numeric scaling factor that is required when `"gmat"` is a matrix of time series which require smoothing. The value depends on the kernel and is automatically set when the `"gelModels"` is created.

Value

It returns the vector $\rho(gmat\lambda)$ when `derive=0`, $\rho'(gmat\lambda)$ when `derive=1` and $\rho''(gmat\lambda)$ when `derive=2`.

References

Anatolyev, S. (2005), GMM, GEL, Serial Correlation, and Asymptotic Bias. *Econometrica*, **73**, 983-1002.

Kitamura, Yuichi (1997), Empirical Likelihood Methods With Weakly Dependent Processes. *The Annals of Statistics*, **25**, 2084-2102.

Kitamura, Y. and Otsu, T. and Evdokimov, K. (2013), Robustness, Infinitesimal Neighborhoods and Moment Restrictions. *Econometrica*, **81**, 1185-1201.

Newey, W.K. and Smith, R.J. (2004), Higher Order Properties of GMM and Generalized Empirical Likelihood Estimators. *Econometrica*, **72**, 219-255.

Smith, R.J. (2011), GEL Criteria for Moment Condition Models. *Econometric Theory*, **27**(6), 1192–1235.

`rlinearModel-class` *Class* `"rlinearModel"`

Description

A class for restricted moment-based models for which moment conditions are orthogonality conditions between instruments and the residuals from a linear regression.

Objects from the Class

Objects can be created by calls of the form `new("rlinearModel", ...)`. It is created by `restModel-methods`.

Slots

`cstLHS`**:** Object of class `"matrix"` ~~

`cstRHS`**:** Object of class `"numeric"` ~~

`cstSpec`**:** Object of class `"list"` ~~

`modelF`**:** Object of class `"data.frame"` ~~

`instF`**:** Object of class `"data.frame"` ~~

`vcov`**:** Object of class `"character"` ~~

`n`**:** Object of class `"integer"` ~~

`q`**:** Object of class `"integer"` ~~

`k`**:** Object of class `"integer"` ~~

`parNames`**:** Object of class `"character"` ~~

`momNames`**:** Object of class `"character"` ~~

`vcovOptions`: Object of class "list" ~~

`centeredVcov`: Object of class "logical" ~~

`varNames`: Object of class "character" ~~

`isEndo`: Object of class "logical" ~~

`omit`: Object of class "integer" ~~

`survOptions`: Object of class "list" ~~

`sSpec`: Object of class "sSpec" ~~

`smooth`: Object of class "logical" ~~

Extends

Class "linearModel", directly. Class "rmomentModel", directly. Class "regModel", by class "linearModel", distance 2. Class "momentModel", by class "linearModel", distance 2.

Methods

coef `signature(object = "rlinearModel")`: ...

getRestrict `signature(object = "rlinearModel")`: ...

gmmFit `signature(model = "rlinearModel")`: ...

model.matrix `signature(object = "rlinearModel")`: ...

modelDims `signature(object = "rlinearModel")`: ...

modelResponse `signature(object = "rlinearModel")`: ...

momentStrength `signature(object = "rlinearModel")`: ...

print `signature(x = "rlinearModel")`: ...

printRestrict `signature(object = "rlinearModel")`: ...

Examples

```
showClass("rlinearModel")
```

---

rmomentModel-class *Class* "rmomentModel"

Description

A union class for all restricted moment-based models.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

gelFit `signature(model = "rmomentModel")`: ...

Examples

```
showClass("rmomentModel")
```

---

rnonlinearModel-class *Class* "rnonlinearModel"

Description

A class for restricted moment-based models for which moment conditions are orthogonality conditions between instruments and the residuals from a nonlinear regression.

Objects from the Class

Objects can be created by calls of the form `new("rnonlinearModel", ...)`. It is created by `restModel-methods`.

Slots

`R`: Object of class `"list"` ~~

`cstSpec`: Object of class `"list"` ~~

`modelF`: Object of class `"data.frame"` ~~

`instF`: Object of class `"data.frame"` ~~

`vcov`: Object of class `"character"` ~~

`theta0`: Object of class `"numeric"` ~~

`n`: Object of class `"integer"` ~~

`q`: Object of class `"integer"` ~~

`k`: Object of class `"integer"` ~~

`parNames`: Object of class `"character"` ~~

`momNames`: Object of class `"character"` ~~

`fRHS`: Object of class `"expression"` ~~

`fLHS`: Object of class `"expressionORNULL"` ~~

`vcovOptions`: Object of class `"list"` ~~

`centeredVcov`: Object of class `"logical"` ~~

`varNames`: Object of class `"character"` ~~

`isEndo`: Object of class `"logical"` ~~

`omit`: Object of class `"integer"` ~~

`survOptions`: Object of class `"list"` ~~

`sSpec`: Object of class `"sSpec"` ~~

`smooth`: Object of class `"logical"` ~~

Extends

Class "nonlinearModel", directly. Class "rmomentModel", directly. Class "regModel", by class "nonlinearModel", distance 2. Class "allNLModel", by class "nonlinearModel", distance 2. Class "momentModel", by class "nonlinearModel", distance 2.

Methods

coef `signature(object = "rnonlinearModel")`: ...

evalDMoment `signature(object = "rnonlinearModel")`: ...

getRestrict `signature(object = "rnonlinearModel")`: ...

gmmFit `signature(model = "rnonlinearModel")`: ...

modelDims `signature(object = "rnonlinearModel")`: ...

print `signature(x = "rnonlinearModel")`: ...

printRestrict `signature(object = "rnonlinearModel")`: ...

Examples

```
showClass("rnonlinearModel")
```

---

rslinearModel-class   *Class* "rslinearModel"

Description

A class for restricted system of linear equations.

Objects from the Class

Objects can be created by calls of the form `new("rslinearModel", ...)`. It is created by restModel-methods.

Slots

`cstLHS`: Object of class "matrix" ~~

`cstRHS`: Object of class "numeric" ~~

`cstSpec`: Object of class "list" ~~

`modelT`: Object of class "list" ~~

`instT`: Object of class "list" ~~

`data`: Object of class "data.frame" ~~

`vcov`: Object of class "character" ~~

`n`: Object of class "integer" ~~

`q`: Object of class "integer" ~~

`k`: Object of class "integer" ~~

`parNames`: Object of class "list" ~~

`momNames`: Object of class "list" ~~

`eqnNames`: Object of class "character" ~~

`vcovOptions`: Object of class "list" ~~

`centeredVcov`: Object of class "logical" ~~

`sameMom`: Object of class "logical" ~~

`SUR`: Object of class "logical" ~~

`varNames`: Object of class "list" ~~

`isEndo`: Object of class "list" ~~

`omit`: Object of class "integer" ~~

`survOptions`: Object of class "list" ~~

`sSpec`: Object of class "sSpec" ~~

`smooth`: Object of class "logical" ~~

Extends

Class "slinearModel", directly. Class "rsysModel", directly. Class "sysModel", by class "slinearModel", distance 2.

Methods

[`signature(x = "rslinearModel", i = "numeric", j = "missing")`: ...

coef `signature(object = "rslinearModel")`: ...

evalDMoment `signature(object = "rslinearModel")`: ...

evalMoment `signature(object = "rslinearModel")`: ...

evalWeights `signature(object = "rslinearModel")`: ...

getRestrict `signature(object = "rslinearModel")`: ...

gmmFit `signature(model = "rslinearModel")`: ...

model.matrix `signature(object = "rslinearModel")`: ...

modelDims `signature(object = "rslinearModel")`: ...

modelResponse `signature(object = "rslinearModel")`: ...

print `signature(x = "rslinearModel")`: ...

printRestrict `signature(object = "rslinearModel")`: ...

residuals `signature(object = "rslinearModel")`: ...

solveGmm `signature(object = "rslinearModel", wObj = "sysMomentWeights")`: ...

ThreeSLS `signature(model = "rslinearModel")`: ...

Examples

```
showClass("rslinearModel")
```

rsnonlinearModel-class
Class "rsnonlinearModel"

Description

A class for restricted systems of nonlinear equations.

Objects from the Class

Objects can be created by calls of the form `new("rsnonlinearModel", ...)`. It is created by `restModel-methods`.

Slots

`R`: Object of class "list" ~~

`cstSpec`: Object of class "list" ~~

`data`: Object of class "data.frame" ~~

`instT`: Object of class "list" ~~

`vcov`: Object of class "character" ~~

`theta0`: Object of class "list" ~~

`n`: Object of class "integer" ~~

`q`: Object of class "integer" ~~

`k`: Object of class "integer" ~~

`parNames`: Object of class "list" ~~

`momNames`: Object of class "list" ~~

`fRHS`: Object of class "list" ~~

`fLHS`: Object of class "list" ~~

`eqnNames`: Object of class "character" ~~

`vcovOptions`: Object of class "list" ~~

`centeredVcov`: Object of class "logical" ~~

`sameMom`: Object of class "logical" ~~

`SUR`: Object of class "logical" ~~

`varNames`: Object of class "list" ~~

`isEndo`: Object of class "list" ~~

`omit`: Object of class "integer" ~~

`survOptions`: Object of class "list" ~~

`sSpec`: Object of class "sSpec" ~~

`smooth`: Object of class "logical" ~~

Extends

Class "snonlinearModel", directly. Class "rsysModel", directly. Class "sysModel", by class "snonlinearModel", distance 2.

Methods

No methods defined with class "rsnonlinearModel" in the signature.

Examples

```
showClass("rsnonlinearModel")
```

---

rsysModel-class *Class* "rsysModel"

Description

A union class for all systems of equations. (see `link{systemGmm}`)

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

No methods defined with class "rsysModel" in the signature.

Examples

```
showClass("rsysModel")
```

---

setCoef-methods *Methods for Function* `setCoef` *in Package* **momentfit** ~~

Description

The method validates the coefficient theta and returns a coefficient object in a format that satisfies the moment model.

Usage

```
## S4 method for signature 'momentModel'
setCoef(model, theta)
## S4 method for signature 'sysModel'
setCoef(model, theta)
```

Arguments

`model` A moment model object.

`theta` A coefficient object. The type depends on the model object. See the examples below.

Methods

`signature(object = "momentModel")` Methods for all single equation models including the restricted ones.

`signature(object = "sysModel")` Methods for all system of equations models including the restricted ones.

Examples

```
### A few system of equation models:
data(simData)
h <- list(~z1+z2+z3, ~x3+z1+z2+z3+z4, ~x3+x4+z1+z2+z3)
nlg <- list(Supply=y1~theta0+theta1*x1+theta2*z2,
            Demand1=y2~alpha0+alpha1*x1+alpha2*x2+alpha3*x3,
            Demand2=y3~beta0+beta1*x3+beta2*x4+beta3*z1)
g <- list(Supply=y1~x1+z2, Demand1=y2~x1+x2+x3, Demand2=y3~x3+x4+z1)
theta0 <- list(c(theta0=1,theta1=2,theta2=3),
               c(alpha0=1,alpha1=2,alpha2=3, alpha3=4),
               c(beta0=1,beta1=2,beta2=3,beta3=4))
nlin <- sysMomentModel(nlg, h, theta0, data=simData)
lin <- sysMomentModel(g, h, data=simData)

### from numeric vector to the proper format with names:
setCoef(nlin, 1:11)

### reorder the equation and name the coefficients
setCoef(nlin, list(Demand1=1:4, Supply=1:3, Demand2=1:4))

### reorder the coefficint to match the order in the model
tet <- do.call("c", theta0)
set.seed(112233)
setCoef(nlin, tet[sample(11)])

### It validates length and names and provide source of errors
## Not run:
setCoef(nlin, list(Demand1=1:4, Supply=1:2, Demand2=1:4))
names(tet)[4] <- "gamma3"
setCoef(nlin, tet)
setCoef(nlin, list(Demand1=1:4, Supply=1:3, Demand4=1:4))

## End(Not run)

### a single equation model
single <- momentModel(nlg[[1]], h[[1]], theta0[[1]], data=simData)
setCoef(single, c(theta1=4, theta0=6, theta2=8))
setCoef(single, 1:3)
```

sfunctionModel-class *Class* "sfunctionModel"

Description

A class for systems of nonlinear equations.

Objects from the Class

Objects can be created by calls of the form new("sfunctionModel", ...). It is created by momentModel.

Slots

X: Object of class "ANY" ~~

fct: Object of class "list" ~~

dfct: Object of class "list" ~~

vcov: Object of class "character" ~~

theta0: Object of class "list" ~~

n: Object of class "integer" ~~

q: Object of class "integer" ~~

k: Object of class "integer" ~~

parNames: Object of class "list" ~~

momNames: Object of class "list" ~~

eqnNames: Object of class "character" ~~

vcovOptions: Object of class "list" ~~

centeredVcov: Object of class "logical" ~~

sameMom: Object of class "logical" ~~

SUR: Object of class "logical" ~~

varNames: Object of class "list" ~~

omit: Object of class "integer" ~~

survOptions: Object of class "list" ~~

sSpec: Object of class "sSpec" ~~

smooth: Object of class "logical" ~~

Extends

Class "sysModel", directly.

Examples

```
showClass("sfunctionModel")
```

sgmmfit-class *Class* "sgmmfit"

Description

Class to store fitted system of equations obtained using the GMM method.

Objects from the Class

Objects can be created by calls of the form `new("sgmmfit", ...)`. It is created by `gmmFit`.

Slots

`theta`: Object of class "list" ~~

`convergence`: Object of class "numericORNULL" ~~

`convIter`: Object of class "numericORNULL" ~~

`call`: Object of class "callORNULL" ~~

`type`: Object of class "character" ~~

`wObj`: Object of class "sysMomentWeights" ~~

`niter`: Object of class "integer" ~~

`efficientGmm`: Object of class "logical" ~~

`model`: Object of class "sysModel" ~~

Methods

bread `signature(x = "sgmmfit")`: ...

coef `signature(object = "sgmmfit")`: ...

hypothesisTest `signature(object.u = "missing", object.r = "sgmmfit")`: ...

hypothesisTest `signature(object.u = "sgmmfit", object.r = "missing")`: ...

hypothesisTest `signature(object.u = "sgmmfit", object.r = "sgmmfit")`: ...

meatGmm `signature(object = "sgmmfit")`: ...

print `signature(x = "sgmmfit")`: ...

residuals `signature(object = "sgmmfit")`: ...

show `signature(object = "sgmmfit")`: ...

specTest `signature(object = "sgmmfit", which = "missing")`: ...

summary `signature(object = "sgmmfit")`: ...

vcov `signature(object = "sgmmfit")`: ...

Examples

```
showClass("sgmmfit")
```

`show-methods` *~~ Methods for Function* `show` *in Package* **methods** *~~*

Description

Display method for all objects.

Methods

`signature(object = "ANY")`

`signature(object = "confint")`

`signature(object = "mconfint")`

`signature(object = "sSpec")`

`signature(object = "momentModel")`

`signature(object = "sysModel")`

`signature(object = "gmmfit")`

`signature(object = "gelfit")`

`signature(object = "sgmmfit")`

`signature(object = "specTest")`

`signature(object = "summarySysGmm")`

`signature(object = "summaryGmm")`

`signature(object = "summaryGel")`

`signature(object = "hypothesisTest")`

`signature(object = "momentWeights")`

`signature(object = "sysMomentWeights")`

`simData` *Simulated data.*

Description

This dataset is used in several documentation files to illustrate the different functionality of the package.

Usage

```
data("simData")
```

Format

A data frame with 50 observations on the following 12 variables. See the examples for the method used to generate them.

y a numeric vector

y1 a numeric vector

y3 a numeric vector

y2 a numeric vector

z1 a numeric vector

x1 a numeric vector

z2 a numeric vector

x2 a numeric vector

z3 a numeric vector

x3 a numeric vector

x4 a numeric vector

z4 a numeric vector

z5 a numeric vector

Examples

```
# Here is how the data was simulated
set.seed(1122)
n <- 50
x1 <- rchisq(n,5)
x2 <- rchisq(n,5)
x3 <- rnorm(n)
x4 <- rnorm(n)
z1 <- .2*x1+rnorm(n)
z2 <- .2*x2+rnorm(n)
z3 <- rnorm(n)
z4 <- rnorm(n)
z5 <- rnorm(n)
y <- y1 <- x1+rnorm(n)
y2 <- 2*x1+rnorm(n)
y3 <- 0.5*x2+rnorm(n)
simData <- data.frame(y=y, y1=y1,y3=y3,y2=y2, z1=z1,x1=x1,z2=z2,x2=x2,z3=z3,x3=x3,
                  x4=x4,z4=z4,z5=z5)
```

slinearModel-class *Class* "slinearModel"

Description

A class for systems of linear equations.

Objects from the Class

Objects can be created by calls of the form `new("slinearModel", ...)`. It is created by `momentModel`.

Slots

`modelT`: Object of class `"list"` ~~

`instT`: Object of class `"list"` ~~

`data`: Object of class `"data.frame"` ~~

`vcov`: Object of class `"character"` ~~

`n`: Object of class `"integer"` ~~

`q`: Object of class `"integer"` ~~

`k`: Object of class `"integer"` ~~

`parNames`: Object of class `"list"` ~~

`momNames`: Object of class `"list"` ~~

`eqnNames`: Object of class `"character"` ~~

`vcovOptions`: Object of class `"list"` ~~

`centeredVcov`: Object of class `"logical"` ~~

`sameMom`: Object of class `"logical"` ~~

`SUR`: Object of class `"logical"` ~~

`varNames`: Object of class `"list"` ~~

`isEndo`: Object of class `"list"` ~~

`omit`: Object of class `"integer"` ~~

`survOptions`: Object of class `"list"` ~~

`sSpec`: Object of class `"sSpec"` ~~

`smooth`: Object of class `"logical"` ~~

Extends

Class `"sysModel"`, directly.

Methods

[signature(x = "slinearModel", i = "numeric", j = "missing"): ...

merge signature(x = "slinearModel", y = "linearModel"): ...

model.matrix signature(object = "slinearModel"): ...

modelDims signature(object = "slinearModel"): ...

modelResponse signature(object = "slinearModel"): ...

restModel signature(object = "slinearModel"): ...

solveGmm signature(object = "slinearModel", wObj = "sysMomentWeights"): ...

ThreeSLS signature(model = "slinearModel"): ...

tsls signature(model = "slinearModel"): ...

Examples

```
showClass("slinearModel")
```

---

snonlinearModel-class *Class* "snonlinearModel"

Description

A class for systems of nonlinear equations.

Objects from the Class

Objects can be created by calls of the form new("snonlinearModel", ...). It is created by momentModel.

Slots

data**:** Object of class "data.frame" ~~

instT**:** Object of class "list" ~~

vcov**:** Object of class "character" ~~

theta0**:** Object of class "list" ~~

n**:** Object of class "integer" ~~

q**:** Object of class "integer" ~~

k**:** Object of class "integer" ~~

parNames**:** Object of class "list" ~~

momNames**:** Object of class "list" ~~

fRHS**:** Object of class "list" ~~

fLHS**:** Object of class "list" ~~

eqnNames**:** Object of class "character" ~~

`vcovOptions`: Object of class "list" ~~

`centeredVcov`: Object of class "logical" ~~

`sameMom`: Object of class "logical" ~~

`SUR`: Object of class "logical" ~~

`varNames`: Object of class "list" ~~

`isEndo`: Object of class "list" ~~

`omit`: Object of class "integer" ~~

`survOptions`: Object of class "list" ~~

`sSpec`: Object of class "sSpec" ~~

`smooth`: Object of class "logical" ~~

Extends

Class "sysModel", directly.

Methods

[`signature(x = "snonlinearModel", i = "numeric", j = "missing")`: ...

merge `signature(x = "snonlinearModel", y = "nonlinearModel")`: ...

model.matrix `signature(object = "snonlinearModel")`: ...

modelDims `signature(object = "snonlinearModel")`: ...

solveGmm `signature(object = "snonlinearModel", wObj = "sysMomentWeights")`: ...

Examples

```
showClass("snonlinearModel")
```

---

solveGel-methods ~~ *Methods for Function* solveGel *in Package* **momentfit** ~~

Description

It fits a moment-based model using GEL methods.

Usage

```
## S4 method for signature 'momentModel'
solveGel(object, gelType="EL", theta0=NULL,
                                    lambda0=NULL, lamSlv=NULL,
                                    coefSlv=c("optim","nlminb","constrOptim"),
                                    rhoFct=NULL,
                                    lControl=list(), tControl=list())
```

Arguments

object	An object of class "gelModels"
gelType	The type of GEL. It is either "EL", "ET", "EEL", "HD", "ETEL" or "ETHD".
theta0	The vector of coefficients for the starting values used in minimization algorithm. If NULL, the starting values in the object is used. For linear models, it must be provided because "linearGel" objects do not have a theta0 slot.
lambda0	The $q \times 1$ vector of starting values for the Lagrange multipliers. By default a zero vector is used.
lamSlv	An alternative solver for the Lagrange multiplier. By default, either Wu_lam, EEL_lam, REEL_lam or getLambda is used.
coefSlv	Minimization solver for the coefficient vector.
rhoFct	An alternative objective function for GEL. This argument is only used if we want to fit the model with a different GEL method. see rhoFct.
lControl	A list of controls for the Lagrange multiplier algorithm.
tControl	A list of controls for the coefficient algorithm.

Value

A list with the following:

theta	The vector of solution
lambda	The vector of Lagrange multiplier
lconvergence	convergence code for the Lagrange multiplier. 0 means normal convergence.
convergence	convergence code for the coefficients. 0 means normal convergence. For higher numbers, see optim, constrOptim or nlminb

Methods

signature(object = "momentModel") The method applies to all GEL classes.

Examples

```
data(simData)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

## Get a good starting value
theta0 <- gmmFit(model1)@theta

## EL by default, with Wu algorithm
res2 <- solveGel(model1, theta0=theta0)

## Change solver parameters
res3 <- solveGel(model1, theta0=theta0,
                 tControl=list(method="Nelder", control=list(maxit=2000)))
```

solveGmm-methods ~~ *Methods for Function* solveGmm *in Package* **momentfit** ~~

Description

The main function to get the GMM solution for a given weighting matrix.

Usage

```
## S4 method for signature 'linearModel,momentWeights'
solveGmm(object, wObj, theta0=NULL,
...)

## S4 method for signature 'allNLModel,momentWeights'
solveGmm(object, wObj, theta0=NULL,
algo=c("optim","nlminb"), ...)

## S4 method for signature 'rnonlinearModel,momentWeights'
solveGmm(object, wObj, theta0=NULL,
...)

## S4 method for signature 'slinearModel,sysMomentWeights'
solveGmm(object, wObj, theta0=NULL)

## S4 method for signature 'rslinearModel,sysMomentWeights'
solveGmm(object, wObj, theta0=NULL)

## S4 method for signature 'snonlinearModel,sysMomentWeights'
solveGmm(object, wObj,
theta0=NULL, ...)

## S4 method for signature 'sfunctionModel,sysMomentWeights'
solveGmm(object, wObj,
theta0=NULL, ...)
```

Arguments

object	A moment-based model
theta0	The vector of coefficients for the starting values used in optim. If NULL, the starting values in the object if used. For system of equations, it is a list of vectors.
wObj	An object of class "momentWeights" or "sysMomentWeights".
algo	The numerical algorithm to minimize the objective function.
...	Arguments to pass to optim.

Value

A list with the following:

`theta` The vector of solution

`convergence` convergence code. 0 means normal convergence. For higher numbers, see `optim`

Methods

`signature(object = "allNLMoment", wObj = "momentWeights")` Method to solve either nonlinear regressions or models in which moments are computed with a function. The objective is minimized using optim.

`signature(object = "rnonlinearModel", wObj = "momentWeights")` Method to solve restricted nonlinear models. It computes the analytical solution.

`signature(object = "linearModel", wObj = "momentWeights")` Method to solve linear models. It computes the analytical solution.

`signature(object = "slinearModel", wObj = "sysMomentWeights")` Method to solve system of linear models. It computes the analytical solution.

`signature(object = "rslinearModel", wObj = "sysMomentWeights")` Method to solve system of linear models in which restrictions have been imposed on the coefficients. It computes the analytical solution.

`signature(object = "slinearModel", wObj = "sysMomentWeights")` Method to solve system of nonlinear models. The solution is obtained with optim using the analytical derivatives.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

## A manual two-step GMM
w0 <- evalWeights(model1, w="ident")
theta0 <- solveGmm(model1, w0)$theta
w <- evalWeights(model1, theta0)
theta1 <- solveGmm(model1, w)$theta
```

`specTest-class` *Class* `"specTest"`

Description

A class to store results from a specification test.

Objects from the Class

Objects can be created by calls of the form `new("specTest", ...)`. It is created my `specTest-methods`.

Slots

test: Object of class "matrix" ~~

testname: Object of class "character" ~~

Methods

print signature(x = "specTest"): ...

show signature(object = "specTest"): ...

Examples

```
showClass("specTest")
```

---

specTest-methods ~~ *Methods for Function* specTest *in Package* **momentfit** ~~

Description

It computes tests of specification for GMM fit.

Usage

```
## S4 method for signature 'gmmfit,missing'
specTest(object, which, df.adj=FALSE, wObj=NULL)

## S4 method for signature 'sgmmfit,missing'
specTest(object, which, df.adj=FALSE, wObj=NULL)

## S4 method for signature 'gmmfit,numeric'
specTest(object, which)

## S4 method for signature 'gelfit,missing'
specTest(object, which,
                    type = c("All", "LR", "LM", "J"))
```

Arguments

object	GMM or GEL fit object
which	Which sub-moment conditions to test.
df.adj	Should we adjust the covariance matrix of the moment conditions for degrees of freedom. If TRUE the covariance matrix is multiplied by n/(n-k), where n is the sample size and k is the number of coefficients. For heteroscedastic robust covariance matrix, adjusting is equivalent to computing HC1 while not adjusting is HC0.

`wObj` An object of class `gmmWeights`. If NULL (the recommended value), the optimal weights is computed at the fitted coefficient estimates. It is used by `hypothesisTest` if one wants the LR statistics to be computed using the same weights for the restricted and unrestricted model.

`type` For GEL, three specification tests are available

Methods

`signature(object = "gmmfit", which="missing")`

`signature(object = "sgmmfit", which="missing")`

`signature(object = "gmmfit", which="numeric")`

References

Eichenbaum, M. and Hansen L. and Singleton, K. (1985). A time Series Analysis of Representative Agent Models of Consumption and Leisure Choise under Uncertainty. *Quarterly Journal of Economics*, **103**, 51–78.

Hayashi, F. (2000). *Econometrics*, New Jersey: Princeton University Press.

Examples

```
data(simData)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

res <- gmmFit(model1)
specTest(res)

## Hayashi Example 3.3 (there is not result in the book but
## that's how we would do it for YEAR=1967
data(Griliches)
dat <- subset(Griliches, YEAR==67)
model <- momentModel(LW~S+EXPR+IQ, ~S+EXPR+AGE+MED, data=dat, vcov="MDS")
res <- gmmFit(model)
## testing the orthogonality conditions of S
specTest(res, 2)
```

`sSpec-class` *Class* `"sSpec"`

Description

A class to store the specifications of the kernel used to smooth moment conditions.

Objects from the Class

Objects can be created by calls of the form `new("sSpec", ...)`. It is created by `kernapply-methods`.

Slots

`k`: Object of class `"numeric"` ~~

`kernel`: Object of class `"character"` ~~

`bw`: Object of class `"numeric"` ~~

`w`: Object of class `"tskernel"` ~~

`bwMet`: Object of class `"character"` ~~

Methods

print `signature(x = "sSpec")`: ...

show `signature(object = "sSpec")`: ...

Examples

```
showClass("sSpec")
```

`stsls-class` *Class* `"stsls"`

Description

A class to store a fitted system of equations obtained using the two-stage least squares method.

Objects from the Class

Objects can be created by calls of the form `new("stsls", ...)`. It is created my `tsls-methods`.

Slots

`theta`: Object of class `"list"` ~~

`convergence`: Object of class `"numericORNULL"` ~~

`convIter`: Object of class `"numericORNULL"` ~~

`call`: Object of class `"callORNULL"` ~~

`type`: Object of class `"character"` ~~

`wObj`: Object of class `"sysMomentWeights"` ~~

`niter`: Object of class `"integer"` ~~

`efficientGmm`: Object of class `"logical"` ~~

`model`: Object of class `"sysModel"` ~~

Extends

Class `"sgmmfit"`, directly.

Methods

No methods defined with class "stsls" in the signature.

Examples

```
showClass("stsls")
```

---

summary-methods ~~ *Methods for Function* `summary` *in Package* **base** ~~

Description

Compute several results from a moment based model fit.

Usage

```
## S4 method for signature 'gmmfit'
summary(object, ...)

## S4 method for signature 'gelfit'
summary(object, ...)

## S4 method for signature 'sgmmfit'
summary(object, ...)
```

Arguments

`object`	A fit object from the package (GMM and GEL are the only methods for now)
`...`	Other arguments to pass to `vcov-methods`

Methods

`signature(object = "gmmfit")`

`signature(object = "gmmfit")`

`signature(object = "sgmmfit")`

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

res <- gmmFit(model1)
summary(res)

## Fixed and True Weights matrix
## Consider the moment of a normal distribution:
```

```
## Using the first three non centered moments

g <- function(theta, x)
{
mu <- theta[1]
sig2 <- theta[2]
m1 <- x-mu
m2 <- x^2-mu^2-sig2
m3 <- x^3-mu^3-3*mu*sig2
cbind(m1,m2,m3)
}

dg <- function(theta, x)
{
mu <- theta[1]
sig2 <- theta[2]
G <- matrix(c(-1,-2*mu,-3*mu^2-3*sig2, 0, -1, -3*mu),3,2)
}

x <- simData$x3
model <- momentModel(g, x, c(mu=.1, sig2=1.5), vcov="iid")
res1 <- gmmFit(model)
summary(res1)
## Same results (that's because the moment vcov is centered by default)
W <- solve(var(cbind(x,x^2,x^3)))
res2 <- gmmFit(model, weights=W)
res2
## If is therefore more efficient in this case to do the following:
summary(res2, breadOnly=TRUE)
```

`summaryGel-class` *Class* `"summaryGel"`

Description

Class to store the summary of a model fitted by GEL.

Objects from the Class

Objects can be created by calls of the form `new("summaryGel", ...)`. It is created by `link{summary-methods}`.

Slots

`coef`: Object of class `"matrix"` ~~

`specTest`: Object of class `"specTest"` ~~

`model`: Object of class `"momentModel"` ~~

`lambda`: Object of class `"matrix"` ~~

`convergence`: Object of class "`numeric`" ~~

`lconvergence`: Object of class "`numeric`" ~~

`impProb`: Object of class "`list`" ~~

`gelType`: Object of class "`list`" ~~

`restrictedLam`: Object of class "`integer`" ~~

Methods

print `signature(x = "summaryGel")`: ...

show `signature(object = "summaryGel")`: ...

Examples

```
showClass("summaryGel")
```

`summaryGmm-class` *Class* "`summaryGmm`"

Description

A class to store the summary of a model fitted by GMM.

Objects from the Class

Objects can be created by calls of the form `new("summaryGmm", ...)`. It is created by `link{summary-methods}`.

Slots

`coef`: Object of class "`matrix`" ~~

`specTest`: Object of class "`specTest`" ~~

`strength`: Object of class "`list`" ~~

`model`: Object of class "`momentModel`" ~~

`sandwich`: Object of class "`logical`" ~~

`type`: Object of class "`character`" ~~

`convergence`: Object of class "`numericORNULL`" ~~

`convIter`: Object of class "`numericORNULL`" ~~

`wSpec`: Object of class "`list`" ~~

`niter`: Object of class "`integer`" ~~

`df.adj`: Object of class "`logical`" ~~

`breadOnly`: Object of class "`logical`" ~~

Methods

print `signature(x = "summaryGmm")`: ...

show `signature(object = "summaryGmm")`: ...

Examples

```
showClass("summaryGmm")
```

`summarySysGmm-class` *Class* `"summarySysGmm"`

Description

A class to store the summary of a system of equations fitted by GMM.

Objects from the Class

Objects can be created by calls of the form `new("summarySysGmm", ...)`. It is created by `summary-methods`.

Slots

`coef`: Object of class `"list"` ~~

`specTest`: Object of class `"specTest"` ~~

`strength`: Object of class `"list"` ~~

`model`: Object of class `"sysModel"` ~~

`sandwich`: Object of class `"logical"` ~~

`type`: Object of class `"character"` ~~

`convergence`: Object of class `"numericORNULL"` ~~

`convIter`: Object of class `"numericORNULL"` ~~

`wSpec`: Object of class `"list"` ~~

`niter`: Object of class `"integer"` ~~

`df.adj`: Object of class `"logical"` ~~

`breadOnly`: Object of class `"logical"` ~~

Methods

print `signature(x = "summarySysGmm")`: ...

show `signature(object = "summarySysGmm")`: ...

Examples

```
showClass("summarySysGmm")
```

sysModel-class *Class* "sysModel"

Description

A union class for all systems of equations.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

[signature(x = "sysModel", i = "missing", j = "list"): ...

[signature(x = "sysModel", i = "missing", j = "missing"): ...

[signature(x = "sysModel", i = "numeric", j = "list"): ...

Dresiduals signature(object = "sysModel"): ...

evalDMoment signature(object = "sysModel"): ...

evalGmmObj signature(object = "sysModel", theta = "list", wObj = "sysMomentWeights"): ...

evalMoment signature(object = "sysModel"): ...

evalWeights signature(object = "sysModel"): ...

getRestrict signature(object = "sysModel"): ...

gmmFit signature(model = "sysModel"): ...

print signature(x = "sysModel"): ...

residuals signature(object = "sysModel"): ...

show signature(object = "sysModel"): ...

subset signature(x = "sysModel"): ...

vcov signature(object = "sysModel"): ...

Examples

```
showClass("sysModel")
```

`sysMomentModel` *Constructor for* `"sysMomentModel"` *classes*

Description

It builds the object of either class `"slinearModel"` or `"snonlinearModel"`, which are system of equations based on moment conditions.

Usage

```
sysMomentModel(g, h=NULL, theta0=NULL, grad=NULL,
            vcov = c("iid", "HAC", "MDS", "CL"),
            vcovOptions=list(), centeredVcov = TRUE,
            data=parent.frame(),na.action="na.omit",
            survOptions=list())
```

Arguments

`g`	A list of linear or nonlinear regression formulas for each equation in the system.
`h`	A list of linear formulas for the instruments in each equation in the system.
`theta0`	A list of vectors of starting values. It is required only when the equations are nonlinear, in which case, it must be a list of named vector, with the names corresponding to the coefficient names in the regression formulas.
`grad`	A list of functions that returns the derivative of the moment functions. Only used if `g` is a list of functions.
`vcov`	Assumption on the properties of the moment conditions. By default, they are weakly dependant processes. For `MDS`, we assume that the conditions are martingale difference sequences, which implies they are serially uncorrelated, but may be heteroscedastic. There is a difference between `iid` and `MDS` only when `g` is a formula. In that case, residuals are assumed homoscedastic as well as serially uncorrelated. For type `CL`, clustered covariance matrix is computed. The options are then included in `vcovOptions` (see `meatCL`).
`vcovOptions`	A list of options for the covariance matrix of the moment conditions. See `vcovHAC` for the default values.
`centeredVcov`	Should the moment function be centered when computing its covariance matrix. Doing so may improve inference.
`data`	A data.frame or a matrix with column names (Optional).
`na.action`	Action to take for missing values. If missing values are present and the option is set to `"na.pass"`, the model won't be estimable.
`survOptions`	If needed, a list with the type of survey weights and the weights as a numeric vector, data.frame or formula. The type is either `"sampling"` or `"fequency"`.

Value

'sysMomentModel' returns an object of one of the subclasses of `"sysMomentModel"`.

References

Hayashi, F. (2000). *Econometrics*, New Jersey: Princeton University Press.

Andrews DWK (1991), Heteroskedasticity and Autocorrelation Consistent Covariance Matrix Estimation. *Econometrica*, **59**, 817–858.

Newey WK & West KD (1987), A Simple, Positive Semi-Definite, Heteroskedasticity and Autocorrelation Consistent Covariance Matrix. *Econometrica*, **55**, 703–708.

Newey WK & West KD (1994), Automatic Lag Selection in Covariance Matrix Estimation. *Review of Economic Studies*, **61**, 631-653.

Examples

```
set.seed(1122)
x1 <- rchisq(50,5)
x2 <- rchisq(50,5)
x3 <- rnorm(50)
x4 <- rnorm(50)
z1 <- .2*x1+rnorm(50)
z2 <- .2*x2+rnorm(50)
z3 <- rnorm(50)
z4 <- rnorm(50)
z5 <- rnorm(50)
y1 <- x1+rnorm(50)
y2 <- 2*x1+rnorm(50)
y3 <- 0.5*x2+rnorm(50)
dat <- data.frame(y1=y1,y3=y3,y2=y2, z1=z1,x1=x1,z2=z2,x2=x2,z3=z3,x3=x3,
                  x4=x4,z4=z4,z5=z5)

g1 <- y1~x1+x4; h1 <- ~z1+z2+z3+z4+x4
g2 <- y2~x1+x2+x3; h2 <- ~z1+z2+z3+z4+x3
g3 <- y3~x2+x3+x4; h3 <- ~z2+z3+z4+x3+x4
g <- list(g1,g2,g3)
h <- list(h1,h2,h3)

smodel <- sysMomentModel(g, h, data=dat)

## not really nonlinear
nlg <- list(y1~theta0+theta1*x1+theta2*x4,
            y2~alpha0+alpha1*x1+alpha2*x2+alpha3*x3,
            y3~beta0+beta1*x2+beta2*x3+beta3*x4)
theta0 <- list(c(theta0=1,theta1=2,theta2=3),
              c(alpha0=1,alpha1=2,alpha2=3, alpha3=4),
              c(beta0=1,beta1=2,beta2=3,beta3=4))
snmodel <- sysMomentModel(nlg, h, theta0, data=dat)
```

sysMomentWeights-class

Class "sysMomentWeights"

Description

A class to store the weighting matrix of the moment conditions from a system of equations.

Objects from the Class

Objects can be created by calls of the form `new("sysMomentWeights", ...)`. It is created by the `evalWeights` method.

Slots

`w`**:** Object of class "ANY" ~~

`type`**:** Object of class "character" ~~

`wSpec`**:** Object of class "list" ~~

`Sigma`**:** Object of class "ANY" ~~

`momNames`**:** Object of class "list" ~~

`eqnNames`**:** Object of class "character" ~~

`sameMom`**:** Object of class "logical" ~~

Methods

[`signature(x = "sysMomentWeights", i = "missing", j = "list")`: ...

[`signature(x = "sysMomentWeights", i = "numeric", j = "list")`: ...

[`signature(x = "sysMomentWeights", i = "numeric", j = "missing")`: ...

evalGmmObj `signature(object = "sysModel", theta = "list", wObj = "sysMomentWeights")`: ...

print `signature(x = "sysMomentWeights")`: ...

quadra `signature(w = "sysMomentWeights", x = "matrixORnumeric", y = "matrixORnumeric")`: ...

quadra `signature(w = "sysMomentWeights", x = "matrixORnumeric", y = "missing")`: ...

quadra `signature(w = "sysMomentWeights", x = "missing", y = "missing")`: ...

show `signature(object = "sysMomentWeights")`: ...

solveGmm `signature(object = "rslinearModel", wObj = "sysMomentWeights")`: ...

solveGmm `signature(object = "slinearModel", wObj = "sysMomentWeights")`: ...

solveGmm `signature(object = "snonlinearModel", wObj = "sysMomentWeights")`: ...

Examples

```
showClass("sysMomentWeights")
```

`systemGmm-doc` *A guide to estimating systems of equations*

Description

This document is meant to describe how to create system of equations objects, estimating them and peforming hypothesis tests.

Details

Instread of repeating the same example for each method, we are going through all methods and classes for systems of equations.

Examples

```
data(simData)

## first, we create an sysGmm object
g1 <- y1~x1+x4; h1 <- ~x4+z1+z2+z3+z4
g2 <- y2~x1+x2+x3; h2 <- ~x3+z1+z2+z3+z4
g3 <- y3~x2+x3+x4; h3 <- ~x3+x4+z1+z2+z3+z4
g <- list(g1,g2,g3)
h <- list(h1,h2,h3)
smodel <- sysMomentModel(g, h, data=simData, vcov="MDS")

## The show or print method
smodel

## The ']' method
smodel[1:2]
smodel[1] ## becomes a one equation model

## equation by equation 2SLS
tsls(smodel)

## or manually
lapply(1:3, function(i) coef(tsls(smodel[i])))

## Fitting the model by two-step GMM
res <- gmmFit(smodel)

## testing Overidentifying restrictions
specTest(res)

## All info using the summary method
## which includes equation by equation measures of
## the instrument stengths
summary(res)

### When the error id iid (homoscedastic), we have a
```

```
### FIVE estimator with 2SLS  as the first step
smodel <- sysMomentModel(g, h, data=simData, vcov="iid")
gmmFit(smodel)

### When the error is iid (homoscedastic),
### all instruments are the same, and the first step is 2SLS,
### we have 3SLS
smodel <- sysMomentModel(g, ~x4+z1+z2+z3+z4, data=simData, vcov="iid")
gmmFit(smodel, initW='tsls')

### When the error is iid (homoscedastic),
### the instruments are the same and are the union of all regressors,
### we have SUR
smodel <- sysMomentModel(g, NULL, data=simData, vcov="iid")
gmmFit(smodel, initW='tsls')

############ Restricted models ##################

## unrestricted
smodel <- sysMomentModel(g, h, data=simData, vcov="MDS")
res <- gmmFit(smodel)

## no cross-equation restrictions
R1 <- list(c("x1=-12*x4"), character(), c("x2=0.8", "x4=0.3"))
rm1 <- restModel(smodel, R1)
(res1 <- gmmFit(rm1))

## Cross equation restrictions
R2<- c("Eqn1.x1=1", "Eqn2.x1=Eqn3.x2")
rm2 <- restModel(smodel, R2)
(es2 <- gmmFit(rm2))## no longer expressed as a system

## testing the restriction

hypothesisTest(res, res1, type="LR")
hypothesisTest(res, res1, type="LM")
hypothesisTest(res, res1, type="Wald")
```

ThreeSLS-methods ~~ *Methods for Function* ThreeSLS *in Package* **momentfit** ~~

Description

Method to estimate system of equations by Three-Stage least squares (3SLS) or, as a special case, by Seemingly Unrelatd Regressions (SUR).

Usage

```
## S4 method for signature 'slinearModel'
```

```
ThreeSLS(model, coefOnly=FALSE, qrZ=NULL,
Sigma=NULL)
## S4 method for signature 'rslinearModel'
ThreeSLS(model, coefOnly=FALSE, qrZ=NULL,
Sigma=NULL)
```

Arguments

model	An object of class "slinearModel" in which instruments are the same in each equation and the error terms are homoscedastic.
coefOnly	Should the method return the only the coefficients or create an object of class "sgmmfit".
qrZ	The qr decomposition of the common instruments. It is mostly used by gmmFit to avoid recomputing it in iterative GMM or CUE. It should not be used directly unless the user knows what he is doing.
Sigma	The covariance matrix of the residuals. If not provided, it is computed using the residuals of the equation by equation two-stage least squares. It should not be used directly unless the user knows what he is doing.

Methods

`signature(model = "slinearModel")` The method is specifically for system of linear models with the same instruments and homoscedastic errors. It becomes SUR as a special case when the instruments are the union of all regressors.

`signature(model = "rslinearModel")` This method is for restricted models that does not impose cross-equation restrictions. With such restrictions 3SLS is not possible as we can no longer write the model as a system of equations.

---

tsls-class *Class* "tsls"

Description

Class that contains a fitted model using two-stage least squares

Objects from the Class

Objects can be created by calls of the form `new("tsls", ...)`. It is created my the

Slots

`theta`: Object of class "numeric" ~~

`convergence`: Object of class "numericORNULL" ~~

`convIter`: Object of class "numericORNULL" ~~

`call`: Object of class "callORNULL" ~~

`type`: Object of class "character" ~~

`wObj`: Object of class "momentWeights" ~~

`niter`: Object of class "integer" ~~

`efficientGmm`: Object of class "logical" ~~

`model`: Object of class "momentModel" ~~

Extends

Class "gmmfit", directly.

Examples

```
showClass("tsls")
```

---

tsls-methods — *~~ Methods for Function* `tsls` *in Package* **momentfit** *~~*

Description

It estimates a linear model using two-stage least squares.

Usage

```
## S4 method for signature 'linearModel'
tsls(model)

## S4 method for signature 'slinearModel'
tsls(model)
```

Arguments

`model`	An object of class `linearModel` or `slinearModel`.

Methods

`signature(model = "linearModel")`

`signature(model = "slinearModel")` 2SLS for equation by equation estimation of a system of equations.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)
res <- tsls(model1)
summary(res)

## Econometrics, Fumio Hayashi (2000)
## Empirical exercises (b) and (c)
data(Griliches)
Griliches$YEAR <- as.factor(Griliches$YEAR)
model1  <- momentModel(LW~S+IQ+EXPR+TENURE+RNS+SMSA+YEAR-1,
                   ~S+EXPR+TENURE+RNS+SMSA+YEAR+MED+KWW+MRT+AGE-1,
                   data=Griliches, vcov="MDS")
res <- tsls(model1)
summary(res)
```

`update-methods` *~~ Methods for Function* `update` *in Package* **stats** *~~*

Description

The method is used to refit a model with either a different method or with modifications to the `momentModel`.

Usage

```
## S4 method for signature 'gmmfit'
update(object, ..., evaluate=TRUE)

## S4 method for signature 'momentModel'
update(object, ...)

## S4 method for signature 'gelfit'
update(object, newModel=NULL, ...,
evaluate=TRUE)

## S4 method for signature 'list'
update(object, ...)
```

Arguments

`object`	An object produced by `"gelFit"`, `"gmmFit"` or a model. It can also be a list, in which case, it is used to change elements of a list.
`...`	Arguments to modify the model or the GMM method

newModel
: When provided, the new model is estimated using the same specification. For example, it is particularly useful to estimate the restricted model using the same optim specification as the unrestricted model.

evaluate
: The modified `call` argument is only evaluated when evaluate is `TRUE`

Methods

`signature(object = "ANY")` That just calls `"update"` from the `"stats"` package.

`signature(object = "gmmfit")`

`signature(object = "momentModel")`

`signature(object = "list")`

Examples

```
x <- rchisq(200,5)
z1 <- rnorm(200)
z2 <- .2*x+rnorm(200)
y <- x+rnorm(200)
dat <- data.frame(y=y,z1=z1,x=x,z2=z2)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x, ~z1+z2, data=dat)

(res <- gmmFit(model1))

## lets change to iterative
update(res, type="iter")

## Let change the HAC specification in the model1 object
## to MDS
update(res, vcov="MDS")
```

---

vcov-methods *~~ Methods for Function* `vcov` *in Package* **stats** *~~*

Description

Computes the covariance matrix of the coefficient estimated by GMM or GEL.

Usage

```
## S4 method for signature 'gmmfit'
vcov(object, sandwich=NULL, df.adj=FALSE,
breadOnly=FALSE, modelVcov=NULL)

## S4 method for signature 'sgmmfit'
vcov(object, sandwich=NULL, df.adj=FALSE,
```

```
breadOnly=FALSE, modelVcov=NULL)

## S4 method for signature 'tsls'
vcov(object, sandwich=TRUE, df.adj=FALSE)

## S4 method for signature 'gelfit'
vcov(object, withImpProb=FALSE, tol=1e-10,
                        robToMiss=FALSE)

## S4 method for signature 'momentModel'
vcov(object, theta)

## S4 method for signature 'sysModel'
vcov(object, theta)
```

Arguments

`object`	A fitted model or a model, For fitted models, it computes the covariance matrix of the estimators. For models, it computes the covariance matrix of the moment conditions, in which case, the coefficient vector must be provided.
`theta`	Coefficient vector to compute the covariance matrix of the moment conditions.
`sandwich`	Should we compute the sandwich covariance matrix. This is only necessary if the weighting matrix is not the optimal one, or if we think it is a bad estimate of it. If `NULL`, it will be set to "TRUE" for One-Step GMM, which includes just-identified GMM like IV, and "FALSE" otherwise.
`df.adj`	Should we adjust for degrees of freedom. If `TRUE` the covariance matrix is multiplied by `n/(n-k)`, where `n` is the sample size and `k` is the number of coefficients. For heteroscedastic robust covariance matrix, adjusting is equivalent to computing HC1 while not adjusting is HC0.
`breadOnly`	If `TRUE`, the covariance matrix is set to the bread (see details below).
`modelVcov`	Should be one of "iid", "MDS" or "HAC". It is meant to change the way the variance of the moments is computed. If it is set to a different specification included in the model, `sandwich` is set to `TRUE`.
`withImpProb`	Should we compute the moments with the implied probabilities
`tol`	Any diagonal less than `"tol"` is set to tol
`robToMiss`	Should we compute a covariance matrix that is robust to misspecification?

Details

If `sandwich=FALSE`, then it returns $(G'V^{-1}G)^{-1}/n$, where G and V are respectively the matrix of average derivatives and the covariance matrix of the moment conditions. If it is `TRUE`, it returns $(G'WG)^{-1}G'WVWG(G'WG)^{-1}/n$, where W is the weighting matrix used to obtain the vector of estimates.

If `breadOnly=TRUE`, it returns $(G'WG)^{-1}/n$, where the value of W depends on the type of GMM. For two-step GMM, it is the first step weighting matrix, for one-step GMM, it is either the identity matrix or the fixed weighting matrix that was provided when `gmmFit` was called, for iterative GMM, it is the weighting matrix used in the last step. For CUE, the result is identical to `sandwich=FALSE` and `beadOnly=FALSE`, because the weighting and coefficient estimates are obtained simultaneously, which makes W identical to V.

`breadOnly=TRUE` should therefore be used with caution because it will produce valid standard errors only if the weighting matrix converges to the the inverse of the covariance matrix of the moment conditions.

For `"tsls"` objects, `sandwich` is TRUE by default. If we assume that the error term is iid, then setting it to FALSE to result in the usual $\sigma^2(\hat{X}'\hat{X})^{-1}$ covariance matrix. If `FALSE`, it returns a robust covariance matrix determined by the value of `vcov` in the `momentModel`.

Methods

`signature(object = "gmmfit")` For any model estimated by any GMM methods.

`signature(object = "gelfit")` For any model estimated by any GMM methods.

`signature(object = "sgmmfit")` For any system of equations estimated by any GMM methods.

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

## optimal matrix
res <- gmmFit(model1)
vcov(res)

## not the optimal matrix
res <- gmmFit(model1, weights=diag(3))
vcov(res, TRUE)

## Model with heteroscedasticity
## MDS is for models with no autocorrelation.
## No restrictions are imposed on the structure of the
## variance of the moment conditions
model2 <- momentModel(y~x1, ~z1+z2, data=simData, vcov="MDS")
res <- tsls(model2)

## HC0 type of robust variance
vcov(res, sandwich=TRUE)
## HC1 type of robust variance
vcov(res, sandwich=TRUE, df.adj=TRUE)

## Fixed and True Weights matrix
## Consider the moment of a normal distribution:
## Using the first three non centered moments

g <- function(theta, x)
{
mu <- theta[1]
sig2 <- theta[2]
```

```
m1 <- x-mu
m2 <- x^2-mu^2-sig2
m3 <- x^3-mu^3-3*mu*sig2
cbind(m1,m2,m3)
}

dg <- function(theta, x)
{
mu <- theta[1]
sig2 <- theta[2]
G <- matrix(c(-1,-2*mu,-3*mu^2-3*sig2, 0, -1, -3*mu),3,2)
}

x <- simData$x3

model <- momentModel(g, x, c(mu=.1, sig2=1.5), vcov="iid")
res1 <- gmmFit(model)
summary(res1)
## Same results (that's because the moment vcov is centered by default)
W <- solve(var(cbind(x,x^2,x^3)))
res2 <- gmmFit(model, weights=W)
res2
## If is therefore more efficient in this case to do the following:
## the option breadOnly of summary() is passed to vcov()
summary(res2, breadOnly=TRUE)
```

---

vcovHAC-methods ~~ *Methods for Function* vcovHAC *in Package* **sandwich** ~~

Description

Methods to compute the HAC covariance matrix of the moment matrix ~~

Methods

signature(x = "momentModel")

Examples

```
data(simData)
theta <- c(beta0=1,beta1=2)
model1 <- momentModel(y~x1, ~z1+z2, data=simData)

# a warning is given if the model is not set as being HAC
vcovHAC(model1, theta)

model1 <- momentModel(y~x1, ~z1+z2, data=simData, vcov="HAC",vcovOptions=list(kernel="B"))
vcovHAC(model1, theta)
```

`[-methods` *Subsetting methods*

Description

Different subsetting methods for S4 class objects of the package. The subset method returns an new object with observations selected by the second argument. See example.

Methods

`signature(x = "momentWeights", i = "integer", j = "missing")` It creates a partition from the weighting matrix.

`signature(x = "momentWeights", i = "missing", j = "missing")` It generates the whole weighting matrix.

`signature(x = "sysMomentWeights", i = "missing", j = "list")` It creates a partition from the weighting matrix. j has no effect here. It creates a partition from the weighting matrix in a systemof equations. i selects the equation and the list j the moments in each equation. Missing i means all equations.

`signature(x = "sysMomentWeights", i = "numeric", j = "missing")` It creates a partition from the weighting matrix. j has no effect here. It creates a partition from the weighting matrix in a systemof equations. i selects the equation and the list j the moments in each equation. Missing j means all moments.

`signature(x = "sysMomentWeights", i = "missing", j = "missing")` No effect. It returns x.

`signature(x = "snonlinearModel", i = "numeric", j="missing")` It generates a system of equations with a subset of equations selected by i. If the number of remaining equations is one, it returns an object of class `"nonlinearGmm"`.

`signature(x = "slinearModel", i = "numeric", j="missing")` It generates a system of equations with a subset of equations selected by i. If the number of remaining equations is one, it returns an object of class `"linearModel"`.

`signature(x = "rslinearModel", i = "numeric", j="missing")` It is only use to select one equation when no cross-equation restrictions are imposed. Only one equation can be selected.

`signature(x = "rsnonlinearModel", i = "numeric", j="missing")` It is only use to select one equation when no cross-equation restrictions are imposed. Only one equation can be selected.

`signature(x = "sysMomentModel", i = "numeric", j="list")` It generates a system of equations with a subset of equations selected by i and a subset of moment conditions selected by j. If the number of remaining equations is one, it returns an object of class `"linearGmm"`.

`signature(x = "sysMomentModel", i = "missing", j="missing")` No effect. It returns x.

`signature(x = "momentModel", i = "missing", j = "missing")` Returns the model without any change.

`signature(x = "functionModel", i = "numeric", j = "missing")` It generates the same model with a subset of the moment conditions.

`signature(x = "formulaModel", i = "numeric", j = "missing")` It generates the same model with a subset of the moment conditions.

`signature(x = "rfuncionModel", i = "numeric", j = "missing")` It generates the same model with a subset of the moment conditions. j has no effect here.

Examples

```
data(simData)
model1 <- momentModel(y~x1+x2, ~x2+x3+z1+z2+z3, data=simData, vcov="MDS")
w <- evalWeights(model1, theta=1:3)
w[]
w[1:3]

## A model with a subset of the instruments
model1[1:4]

## Selecting the observations:

subset(model1, simData[["x1"]]<3)
subset(model1, 1:25)
```

Index

∗ **3SLS**
systemGmm-doc, 109
∗ **Confidence Intervals**
confint-methods, 9
∗ **FIVE**
systemGmm-doc, 109
∗ **GEL**
gel4, 26
∗ **GMM**
gmm4, 32
systemGmm-doc, 109
∗ **Instruments**
gel4, 26
gmm4, 32
∗ **Moment**
gel4, 26
gmm4, 32
∗ **SUR**
systemGmm-doc, 109
∗ **System of Equations**
systemGmm-doc, 109
∗ **classes**
allNLModel-class, 4
confint-class, 8
formulaModel-class, 23
functionModel-class, 24
gelfit-class, 28
gmmfit-class, 35
hypothesisTest-class, 42
linearModel-class, 51
mconfint-class, 54
momentModel-class, 62
momentWeights-class, 64
nonlinearModel-class, 67
regModel-class, 73
rformulaModel-class, 76
rfunctionModel-class, 78
rlinearModel-class, 80
rmomentModel-class, 81
rnonlinearModel-class, 82
rslinearModel-class, 83
rsnonlinearModel-class, 85
rsysModel-class, 86
sfunctionModel-class, 88
sgmmfit-class, 89
slinearModel-class, 92
snonlinearModel-class, 93
specTest-class, 97
sSpec-class, 99
stsls-class, 100
summaryGel-class, 102
summaryGmm-class, 103
summarySysGmm-class, 104
sysModel-class, 105
sysMomentWeights-class, 108
tsls-class, 111
∗ **datasets**
CigarettesSW, 5
ConsumptionG, 10
Griliches, 39
HealthRWM, 40
Klein, 47
LabourCR, 48
ManufactCost, 53
Mroz, 66
simData, 90
∗ **methods**
[-methods, 118
bread-methods, 4
coef-methods, 7
confint-methods, 9
Dresiduals-methods, 12
DWH-methods, 12
estfun-methods, 14
evalDMoment-methods, 14
evalGel-methods, 16
evalGelObj-methods, 18
evalGmm-methods, 19

evalGmmObj-methods, 20
evalMoment-methods, 21
evalWeights-methods, 22
gelFit-methods, 29
getRestrict-methods, 30
gmmFit-methods, 36
hypothesisTest-methods, 43
meatGmm-methods, 54
merge-methods, 56
model.matrix-methods, 57
modelDims-methods, 59
modelResponse-methods, 60
momentStrength-methods, 63
momFct-methods, 65
plot-methods, 69
print-methods, 70
printRestrict-methods, 71
quadra-methods, 71
residuals-methods, 73
restModel-methods, 74
setCoef-methods, 86
show-methods, 90
solveGel-methods, 94
solveGmm-methods, 96
specTest-methods, 98
summary-methods, 101
ThreeSLS-methods, 110
tsls-methods, 112
update-methods, 113
vcov-methods, 114
vcovHAC-methods, 117
∗ **misspecified**
momFct-methods, 65
∗ **probabilities**
getImpProb-methods, 30
[,formulaModel,numeric,missing-method (*[-methods*), 118
[,functionModel,numeric,missing-method (*[-methods*), 118
[,momentModel,missing,missing-method (*[-methods*), 118
[,momentWeights,missing,missing-method (*[-methods*), 118
[,momentWeights,numeric,missing-method (*[-methods*), 118
[,regModel,numeric,missing-method (*[-methods*), 118
[,rfunctionModel,numeric,missing-method (*[-methods*), 118
[,rslinearModel,numeric,missing-method (*[-methods*), 118
[,rsnonlinearModel,numeric,missing-method (*[-methods*), 118
[,sfunctionModel,numeric,missing-method (*[-methods*), 118
[,slinearModel,numeric,missing-method (*[-methods*), 118
[,snonlinearModel,numeric,missing-method (*[-methods*), 118
[,sysModel,missing,list-method (*[-methods*), 118
[,sysModel,missing,missing-method (*[-methods*), 118
[,sysModel,numeric,list-method (*[-methods*), 118
[,sysMomentWeights,missing,list-method (*[-methods*), 118
[,sysMomentWeights,numeric,list-method (*[-methods*), 118
[,sysMomentWeights,numeric,missing-method (*[-methods*), 118
[-methods, 118

allNLModel, *24, 25*, *68*, *77, 78*, *83*
allNLModel-class, 4

bread (*bread-methods*), 4
bread,gmmfit-method (*bread-methods*), 4
bread,sgmmfit-method (*bread-methods*), 4
bread,tsls-method (*bread-methods*), 4
bread-methods, 4

CigarettesSW, 5
coef,gelfit-method (*coef-methods*), 7
coef,gmmfit-method (*coef-methods*), 7
coef,momentModel-method (*coef-methods*), 7
coef,rformulaModel-method (*coef-methods*), 7
coef,rfunctionModel-method (*coef-methods*), 7
coef,rlinearModel-method (*coef-methods*), 7
coef,rnonlinearModel-method (*coef-methods*), 7
coef,rslinearModel-method (*coef-methods*), 7

coef,rsnonlinearModel-method (coef-methods), 7
coef,sgmmfit-method (coef-methods), 7
coef-methods, 7
confint,ANY-method (confint-methods), 9
confint,data.frame-method (confint-methods), 9
confint,gelfit-method (confint-methods), 9
confint,gmmfit-method (confint-methods), 9
confint,matrix-method (confint-methods), 9
confint,numeric-method (confint-methods), 9
confint-class, 8
confint-methods, 9
constrOptim, *95*
ConsumptionG, 10

Dresiduals (Dresiduals-methods), 12
Dresiduals,linearModel-method (Dresiduals-methods), 12
Dresiduals,nonlinearModel-method (Dresiduals-methods), 12
Dresiduals,rsnonlinearModel-method (Dresiduals-methods), 12
Dresiduals,sysModel-method (Dresiduals-methods), 12
Dresiduals-methods, 12
DWH (DWH-methods), 12
DWH,gmmfit,gmmfit-method (DWH-methods), 12
DWH,gmmfit,lm-method (DWH-methods), 12
DWH,gmmfit,missing-method (DWH-methods), 12
DWH-methods, 12

EEL_lam, *17*, *27*, *95*
EEL_lam (lambdaAlgo), 50
estfun,momentModel-method (estfun-methods), 14
estfun-methods, 14
ETXX_lam (lambdaAlgo), 50
evalDMoment (evalDMoment-methods), 14
evalDMoment,formulaModel-method (evalDMoment-methods), 14
evalDMoment,functionModel-method (evalDMoment-methods), 14
evalDMoment,regModel-method (evalDMoment-methods), 14
evalDMoment,rformulaModel-method (evalDMoment-methods), 14
evalDMoment,rfunctionModel-method (evalDMoment-methods), 14
evalDMoment,rnonlinearModel-method (evalDMoment-methods), 14
evalDMoment,rslinearModel-method (evalDMoment-methods), 14
evalDMoment,rsnonlinearModel-method (evalDMoment-methods), 14
evalDMoment,sysModel-method (evalDMoment-methods), 14
evalDMoment-methods, 14
evalGel (evalGel-methods), 16
evalGel,momentModel-method (evalGel-methods), 16
evalGel-methods, 16
evalGelObj (evalGelObj-methods), 18
evalGelObj,momentModel,numeric,numeric-method (evalGelObj-methods), 18
evalGelObj-methods, 18
evalGmm, *38*
evalGmm (evalGmm-methods), 19
evalGmm,momentModel-method (evalGmm-methods), 19
evalGmm,sysModel-method (evalGmm-methods), 19
evalGmm-methods, 19
evalGmmObj (evalGmmObj-methods), 20
evalGmmObj,momentModel,numeric,momentWeights-method (evalGmmObj-methods), 20
evalGmmObj,sysModel,list,sysMomentWeights-method (evalGmmObj-methods), 20
evalGmmObj-methods, 20
evalMoment (evalMoment-methods), 21
evalMoment,formulaModel-method (evalMoment-methods), 21
evalMoment,functionModel-method (evalMoment-methods), 21
evalMoment,regModel-method (evalMoment-methods), 21
evalMoment,rsysModel-method (evalMoment-methods), 21
evalMoment,sysModel-method (evalMoment-methods), 21
evalMoment-methods, 21

evalWeights, *64*, *108*
evalWeights *(*evalWeights-methods*)*, 22
evalWeights,momentModel-method
  *(*evalWeights-methods*)*, 22
evalWeights,rslinearModel-method
  *(*evalWeights-methods*)*, 22
evalWeights,sysModel-method
  *(*evalWeights-methods*)*, 22
evalWeights-methods, 22

formulaModel, *77*
formulaModel-class, 23
functionModel, *78*
functionModel-class, 24

gel4, 26
gelFit, *10*, *26–28*
gelFit *(*gelFit-methods*)*, 29
gelFit,momentModel-method
  *(*gelFit-methods*)*, 29
gelFit,rmomentModel-method
  *(*gelFit-methods*)*, 29
gelfit-class, 28
gelFit-methods, 29
getImpProb *(*getImpProb-methods*)*, 30
getImpProb,gelfit-method
  *(*getImpProb-methods*)*, 30
getImpProb-methods, 30
getLambda, *17*, *27*, *95*
getLambda *(*lambdaAlgo*)*, 50
getRestrict *(*getRestrict-methods*)*, 30
getRestrict,momentModel-method
  *(*getRestrict-methods*)*, 30
getRestrict,rformulaModel-method
  *(*getRestrict-methods*)*, 30
getRestrict,rfunctionModel-method
  *(*getRestrict-methods*)*, 30
getRestrict,rlinearModel-method
  *(*getRestrict-methods*)*, 30
getRestrict,rnonlinearModel-method
  *(*getRestrict-methods*)*, 30
getRestrict,rslinearModel-method
  *(*getRestrict-methods*)*, 30
getRestrict,rsnonlinearModel-method
  *(*getRestrict-methods*)*, 30
getRestrict,sysModel-method
  *(*getRestrict-methods*)*, 30
getRestrict-methods, 30
gmm4, 32
gmmFit, *10*, *32*, *35*, *89*, *111*, *115*
gmmFit *(*gmmFit-methods*)*, 36
gmmfit, *112*
gmmFit,formulaModel-method
  *(*gmmFit-methods*)*, 36
gmmFit,momentModel-method
  *(*gmmFit-methods*)*, 36
gmmFit,rformulaModel-method
  *(*gmmFit-methods*)*, 36
gmmFit,rlinearModel-method
  *(*gmmFit-methods*)*, 36
gmmFit,rnonlinearModel-method
  *(*gmmFit-methods*)*, 36
gmmFit,rslinearModel-method
  *(*gmmFit-methods*)*, 36
gmmFit,sysModel-method
  *(*gmmFit-methods*)*, 36
gmmfit-class, 35
gmmFit-methods, 36
Griliches, 39

HealthRWM, 40
hypothesisTest, *42*, *99*
hypothesisTest
  *(*hypothesisTest-methods*)*, 43
hypothesisTest,gmmfit,gmmfit-method
  *(*hypothesisTest-methods*)*, 43
hypothesisTest,gmmfit,missing-method
  *(*hypothesisTest-methods*)*, 43
hypothesisTest,missing,gmmfit-method
  *(*hypothesisTest-methods*)*, 43
hypothesisTest,missing,sgmmfit-method
  *(*hypothesisTest-methods*)*, 43
hypothesisTest,sgmmfit,missing-method
  *(*hypothesisTest-methods*)*, 43
hypothesisTest,sgmmfit,sgmmfit-method
  *(*hypothesisTest-methods*)*, 43
hypothesisTest-class, 42
hypothesisTest-methods, 43

kernapply *(*kernapply-methods*)*, 46
kernapply,momentModel-method
  *(*kernapply-methods*)*, 46
kernapply-methods, 46
Klein, 47

LabourCR, 48
lambdaAlgo, 50
linearModel, *81*

linearModel-class, 51

ManufactCost, 53
mclapply, *10*
mconfint-class, 54
meatCL, *34*, *61*, *64*, *106*
meatGmm *(*meatGmm-methods*)*, 54
meatGmm,gmmfit-method *(*meatGmm-methods*)*, 54
meatGmm,sgmmfit-method *(*meatGmm-methods*)*, 54
meatGmm,tsls-method *(*meatGmm-methods*)*, 54
meatGmm-methods, 54
merge *(*merge-methods*)*, 56
merge,ANY,ANY-method *(*merge-methods*)*, 56
merge,linearModel,linearModel-method *(*merge-methods*)*, 56
merge,nonlinearModel,nonlinearModel-method *(*merge-methods*)*, 56
merge,slinearModel,linearModel-method *(*merge-methods*)*, 56
merge,snonlinearModel,nonlinearModel-method *(*merge-methods*)*, 56
merge-methods, 56
model.matrix *(*model.matrix-methods*)*, 57
model.matrix,linearModel-method *(*model.matrix-methods*)*, 57
model.matrix,nonlinearModel-method *(*model.matrix-methods*)*, 57
model.matrix,rlinearModel-method *(*model.matrix-methods*)*, 57
model.matrix,rslinearModel-method *(*model.matrix-methods*)*, 57
model.matrix,rsnonlinearModel-method *(*model.matrix-methods*)*, 57
model.matrix,slinearModel-method *(*model.matrix-methods*)*, 57
model.matrix,snonlinearModel-method *(*model.matrix-methods*)*, 57
model.matrix-methods, 57
modelDims *(*modelDims-methods*)*, 59
modelDims,formulaModel-method *(*modelDims-methods*)*, 59
modelDims,functionModel-method *(*modelDims-methods*)*, 59
modelDims,linearModel-method *(*modelDims-methods*)*, 59
modelDims,nonlinearModel-method *(*modelDims-methods*)*, 59
modelDims,rformulaModel-method *(*modelDims-methods*)*, 59
modelDims,rfunctionModel-method *(*modelDims-methods*)*, 59
modelDims,rlinearModel-method *(*modelDims-methods*)*, 59
modelDims,rnonlinearModel-method *(*modelDims-methods*)*, 59
modelDims,rslinearModel-method *(*modelDims-methods*)*, 59
modelDims,rsnonlinearModel-method *(*modelDims-methods*)*, 59
modelDims,sfunctionModel-method *(*modelDims-methods*)*, 59
modelDims,slinearModel-method *(*modelDims-methods*)*, 59
modelDims,snonlinearModel-method *(*modelDims-methods*)*, 59
modelDims,sysMomentModel-method *(*modelDims-methods*)*, 59
modelDims-methods, 59
modelResponse *(*modelResponse-methods*)*, 60
modelResponse,linearModel-method *(*modelResponse-methods*)*, 60
modelResponse,rlinearModel-method *(*modelResponse-methods*)*, 60
modelResponse,rslinearModel-method *(*modelResponse-methods*)*, 60
modelResponse,slinearModel-method *(*modelResponse-methods*)*, 60
modelResponse-methods, 60
momentModel, *23–25*, *27*, *35*, *51, 52*, 60, *62*, *64*, *67, 68*, *77, 78*, *81*, *83*, *88*, *92, 93*
momentModel-class, 62
momentStrength *(*momentStrength-methods*)*, 63
momentStrength,formulaModel-method *(*momentStrength-methods*)*, 63
momentStrength,functionModel-method *(*momentStrength-methods*)*, 63
momentStrength,linearModel-method *(*momentStrength-methods*)*, 63
momentStrength,nonlinearModel-method *(*momentStrength-methods*)*, 63
momentStrength,rlinearModel-method

(momentStrength-methods), 63
momentStrength-methods, 63
momentWeights, *72*
momentWeights (momentWeights-class), 64
momentWeights-class, 64
momFct (momFct-methods), 65
momFct,numeric,gelfit-method (momFct-methods), 65
momFct-methods, 65
Mroz, 66

nlminb, *50*, *95*
nonlinearModel, *83*
nonlinearModel-class, 67

optim, *29*, *33, 34*, *38*, *50*, *61*, *95–97*

plot,ANY-method (plot-methods), 69
plot,mconfint-method (plot-methods), 69
plot-methods, 69
points, *69*
polygon, *69*
print,ANY-method (print-methods), 70
print,confint-method (print-methods), 70
print,gelfit-method (print-methods), 70
print,gmmfit-method (print-methods), 70
print,hypothesisTest-method (print-methods), 70
print,mconfint-method (print-methods), 70
print,momentModel-method (print-methods), 70
print,momentWeights-method (print-methods), 70
print,rformulaModel-method (print-methods), 70
print,rfunctionModel-method (print-methods), 70
print,rlinearModel-method (print-methods), 70
print,rnonlinearModel-method (print-methods), 70
print,rslinearModel-method (print-methods), 70
print,rsnonlinearModel-method (print-methods), 70
print,sgmmfit-method (print-methods), 70
print,specTest-method (print-methods), 70
print,sSpec-method (print-methods), 70
print,summaryGel-method (print-methods), 70
print,summaryGmm-method (print-methods), 70
print,summarySysGmm-method (print-methods), 70
print,sysModel-method (print-methods), 70
print,sysMomentWeights-method (print-methods), 70
print-methods, 70
printRestrict (printRestrict-methods), 71
printRestrict,rformulaModel-method (printRestrict-methods), 71
printRestrict,rfunctionModel-method (printRestrict-methods), 71
printRestrict,rlinearModel-method (printRestrict-methods), 71
printRestrict,rnonlinearModel-method (printRestrict-methods), 71
printRestrict,rslinearModel-method (printRestrict-methods), 71
printRestrict,rsnonlinearModel-method (printRestrict-methods), 71
printRestrict-methods, 71

quadra (quadra-methods), 71
quadra,momentWeights,matrixORnumeric,matrixORnumeric-metho (quadra-methods), 71
quadra,momentWeights,matrixORnumeric,missing-method (quadra-methods), 71
quadra,momentWeights,missing,missing-method (quadra-methods), 71
quadra,sysMomentWeights,matrixORnumeric,matrixORnumeric-me (quadra-methods), 71
quadra,sysMomentWeights,matrixORnumeric,missing-method (quadra-methods), 71
quadra,sysMomentWeights,missing,missing-method (quadra-methods), 71
quadra-methods, 71

REEL_lam, *17*, *27*, *95*
REEL_lam (lambdaAlgo), 50
regModel, *52*, *68*, *81*, *83*
regModel-class, 73
residuals,ANY-method (residuals-methods), 73

residuals,gelfit-method (residuals-methods), 73
residuals,gmmfit-method (residuals-methods), 73
residuals,linearModel-method (residuals-methods), 73
residuals,nonlinearModel-method (residuals-methods), 73
residuals,rsysModel-method (residuals-methods), 73
residuals,sgmmfit-method (residuals-methods), 73
residuals,sysModel-method (residuals-methods), 73
residuals-methods, 73
restModel, *27*, *31*, *34*
restModel (restModel-methods), 74
restModel,formulaModel-method (restModel-methods), 74
restModel,functionModel-method (restModel-methods), 74
restModel,linearModel-method (restModel-methods), 74
restModel,nonlinearModel-method (restModel-methods), 74
restModel,slinearModel-method (restModel-methods), 74
restModel,snonlinearModel-method (restModel-methods), 74
restModel-methods, 74
rformulaModel-class, 76
rfunctionModel-class, 78
rhoEEL (rhoFct), 79
rhoEL (rhoFct), 79
rhoET (rhoFct), 79
rhoETEL (rhoFct), 79
rhoETHD (rhoFct), 79
rhoFct, *17, 18*, *29*, 79, *95*
rhoHD (rhoFct), 79
rhoREEL (rhoFct), 79
rlinearModel-class, 80
rmomentModel, *77, 78*, *81*, *83*
rmomentModel-class, 81
rnonlinearModel-class, 82
rslinearModel-class, 83
rsnonlinearModel-class, 85
rsysModel, *84*, *86*
rsysModel-class, 86
setCoef (setCoef-methods), 86
setCoef,momentModel-method (setCoef-methods), 86
setCoef,sysModel-method (setCoef-methods), 86
setCoef-methods, 86
sfunctionModel-class, 88
sgmmfit, *100*
sgmmfit-class, 89
show,ANY-method (show-methods), 90
show,confint-method (show-methods), 90
show,gelfit-method (show-methods), 90
show,gmmfit-method (show-methods), 90
show,hypothesisTest-method (show-methods), 90
show,mconfint-method (show-methods), 90
show,momentModel-method (show-methods), 90
show,momentWeights-method (show-methods), 90
show,sgmmfit-method (show-methods), 90
show,specTest-method (show-methods), 90
show,sSpec-method (show-methods), 90
show,summaryGel-method (show-methods), 90
show,summaryGmm-method (show-methods), 90
show,summarySysGmm-method (show-methods), 90
show,sysModel-method (show-methods), 90
show,sysMomentWeights-method (show-methods), 90
show-methods, 90
simData, 90
slinearModel, *84*
slinearModel-class, 92
snonlinearModel, *86*
snonlinearModel-class, 93
solveGel (solveGel-methods), 94
solveGel,momentModel-method (solveGel-methods), 94
solveGel-methods, 94
solveGmm (solveGmm-methods), 96
solveGmm,allNLModel,momentWeights-method (solveGmm-methods), 96
solveGmm,linearModel,momentWeights-method (solveGmm-methods), 96
solveGmm,rnonlinearModel,momentWeights-method

(solveGmm-methods), 96
solveGmm,rslinearModel,sysMomentWeights-method
(solveGmm-methods), 96
solveGmm,sfunctionModel,sysMomentWeights-method
(solveGmm-methods), 96
solveGmm,slinearModel,sysMomentWeights-method
(solveGmm-methods), 96
solveGmm,snonlinearModel,sysMomentWeights-method
(solveGmm-methods), 96
solveGmm-methods, 96
specTest, *44*
specTest (specTest-methods), 98
specTest,gelfit,missing-method
(specTest-methods), 98
specTest,gmmfit,missing-method
(specTest-methods), 98
specTest,gmmfit,numeric-method
(specTest-methods), 98
specTest,sgmmfit,missing-method
(specTest-methods), 98
specTest-class, 97
specTest-methods, 98
sSpec-class, 99
stsls-class, 100
subset ([-methods), 118
subset,formulaModel-method ([-methods), 118
subset,functionModel-method
([-methods), 118
subset,regModel-method ([-methods), 118
subset,sysModel-method ([-methods), 118
summary (summary-methods), 101
summary,gelfit-method
(summary-methods), 101
summary,gmmfit-method
(summary-methods), 101
summary,sgmmfit-method
(summary-methods), 101
summary-methods, 101
summaryGel-class, 102
summaryGmm-class, 103
summarySysGmm-class, 104
sysModel, *84*, *86*, *88*, *92*, *94*
sysModel-class, 105
sysMomentModel, 106
sysMomentWeights-class, 108
systemGmm (systemGmm-doc), 109
systemGmm-doc, 109
ThreeSLS (ThreeSLS-methods), 110
ThreeSLS,list-method (gmm4), 32
ThreeSLS,rslinearModel-method
(ThreeSLS-methods), 110
ThreeSLS,slinearModel-method
(ThreeSLS-methods), 110
ThreeSLS-methods, 110
tsls (tsls-methods), 112
tsls,formula-method (gmm4), 32
tsls,linearModel-method (tsls-methods), 112
tsls,list-method (gmm4), 32
tsls,slinearModel-method
(tsls-methods), 112
tsls-class, 111
tsls-methods, 112

uniroot, *10*
update,ANY-method (update-methods), 113
update,gelfit-method (update-methods), 113
update,gmmfit-method (update-methods), 113
update,list-method (update-methods), 113
update,momentModel-method
(update-methods), 113
update-methods, 113

vcov, *13*
vcov,gelfit-method (vcov-methods), 114
vcov,gmmfit-method (vcov-methods), 114
vcov,momentModel-method (vcov-methods), 114
vcov,sgmmfit-method (vcov-methods), 114
vcov,sysModel-method (vcov-methods), 114
vcov,tsls-method (vcov-methods), 114
vcov-methods, 114
vcovHAC, *26*, *34*, *61*, *64*, *106*
vcovHAC,momentModel-method
(vcovHAC-methods), 117
vcovHAC-methods, 117
vcovHC, *64*

Wu_lam, *17*, *27*, *95*
Wu_lam (lambdaAlgo), 50