Generalized Method of Moments with R

Pierre Chaussé

June 28, 2018

Abstract

This vignette presents the gmm4 package, which is an attempt to rebuild the gmm package using S4 classes and methods. The goal is to facilitate the development of new functionalities.

Contents

1 Single Equation

1.1 An S4 class object for GMM models

In general, GMM models are based on the moment conditions:

$$\mathrm{E}[g_i(\theta)] = 0$$

The GMM estimator is defined as

$$\hat{\theta}(W) = \arg\min_{\theta} \bar{g}(\theta)' W \bar{g}(\theta)$$

Under some regularity conditions (see Hansen, 1982), we have the following result:

$$\sqrt{n}\left(\hat{\theta}(W) - \theta\right) \xrightarrow{d} N\left(0, (G'WG)^{-1}G'WVWG(G'WG)^{-1}\right),$$

where $G = \mathrm{E}[dg_i(\theta)/d\theta]$ and V is the asymptotic variance of $\sqrt{n}\bar{g}(\theta)$. We can therefore use the following approximation for inference:

$$\hat{\theta}(W) \approx N\left(\theta, (\hat{G}'W\hat{G})^{-1}\hat{G}'W\hat{V}W\hat{G}(\hat{G}'W\hat{G})^{-1}/n\right)$$

with $\hat{G} = \frac{1}{n}\sum_{i=1}^{n} dg_i(\hat{\theta}(W))/d\theta$ and $\hat{V}$ is some consistent estimate V. Therefore, the property depends on the method, which in this case is simply characterized by the choice of the weighting matrix W, and on the statistical properties of $g_i(\theta)$. The GMM model class will only include the definition of $g_i(\theta)$ and its assumed statistical properties, which is basically represented by its variance.

We want to distinguish three types of $g_i(\theta)$:

1. The linear model:
$$Y_i = X_i'\theta + \varepsilon_i,$$

   with the moment condition $\mathrm{E}[\varepsilon_i(\theta)Z_i] = 0$, where X_i is $k \times 1$ and Z_i is $q \times 1$ with $q \geq k$. We consider three possibilities for the asymptotic variance of $\sqrt{n}\bar{g}_i(\theta)$:

   a) "iid": Here we assume no autocorrelation and homoscedastic error with $\mathrm{Var}(\varepsilon_i|Z_i) = \sigma^2$, which implies that the asymptotic variance V is $\sigma^2 \mathrm{E}[Z_i Z_i']$ and can be estimated by:

   $$\hat{V} = \hat{\sigma}^2 \left(\frac{1}{n}\sum_{i=1}^{n} Z_i Z_i'\right),$$

   where $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}\hat{\varepsilon}_i^2$, and $\hat{\varepsilon}_i = Y_i - X_i'\hat{\theta}(W)$.

   b) "MDS": We assume that $g_i(\theta) \equiv (\varepsilon_i Z_i)$ is a martingale difference sequence with no additional assumption on the conditional variance of the error term. Heteroscedasticity is therefore allowed. The asymptotic variance is therefore $V = \mathrm{E}(\varepsilon_i^2 Z_i Z_i')$, and can be estimated by:

   $$\hat{V} = \frac{1}{n}\sum_{i=1}^{n} \hat{\varepsilon}_i^2 Z_i Z_i',$$

   which represents the HC0 version of the heteroscedasticity consistent covariance matrix (HCCM) estimator.

   c) "HAC": If we assume that $g_t(\theta)$ (t is used when we have time series) is weakly dependent, the asymptotic covariance matrix is $V = \Gamma_0 + \sum_{i=1}^{\infty}(\Gamma_i + \Gamma_i')$, with $\Gamma_i = \mathrm{E}(\varepsilon_t \varepsilon_{t-i} Z_t Z_{t-i}')$. It can be estimated using a kernel estimator:

   $$\hat{V} = \sum_{i=-M}^{M} K_h(i)\hat{\Gamma}_i,$$

   where $K_h(i)$ is a kernel that depends on the bandwidth h, and $\hat{\Gamma}_i$ is an estimator of Γ_i.

2. The nonlinear model:
$$y_i(\theta) = x_i(\theta) + \varepsilon_i,$$
with the moment condition $\mathrm{E}[\varepsilon_i(\theta)Z_i] = 0.$, where X_i is $k \times 1$ and Z_i is $q \times 1$ with $q \geq k$. The only difference is that $\varepsilon_i(\theta)$ is a nonlinear function of the coefficient vector θ. For this case, the same three possibilities exist for the asymptotic variance.

3. The functional case: Is we cannot represent the model in a regression format with instruments, we simply write the moment conditions as $\mathrm{E}[g_i(\theta)]$ with $g_i(\theta)$ being a continuous and differentiable function from $\mathbb{R}^k$ to $\mathbb{R}^q$, with $q \geq k$. Here, we do not distinguish "iid" from "MDS". We therefore have two possible cases:

   a) "iid" or "MDS": The asymptotic variance is $V = E[g_i(\theta)g_i(\theta)']$ and can be estimated by its sample counterpart.

   b) "HAC": Same as for the linear case with $\Gamma_i = E[g_t(\theta)g_{t-i}(\theta)']$.

Since the moment conditions are defined differently, we have three difference classes to represent the three models. Their common slots are all the arguments that specify V, which include the specifications of th HAC estimator if needed, the names of the coefficients, the names of the moment conditions, k, q, n, and the argument "isEndo", a k logical vector that indicates which regressors in X_i is considered endogenous. It is considered endogenous is it is not part of Z_i. Of course, it makes no sense when $g_i(\theta)$ is a general function.

The main difference is the slots that define $g_i(\theta)$. For "linearGmm" class, the slots "modelF" and "instF" are model.frame's that define the regression model and the instruments. For "nonlinearGmm", we have the following slots: "modelF" is a data.frame for the nonlinear regression, "instF" is as for the linear case, and "fRHS" and "fLHS" are expressions to compute the right and left hand sides of the nonlinear regression. The function D() can be used to obtain analytical derivatives. Finally, the "functionGmm" class contains the slot "fct", which is a function of two arguments, the first being θ, and returns a $n \times q$ matrix with the ith row being $g_i(\theta)'$. The slot "dfct" is an optional function with the same two arguments which returns the $q \times k$ matrix of first derivatives of $\bar{g}(\theta)$. The slot "X" is whatever is needed as second argument of "fct" and "dfct". The last two classes also contain the slot "theta0", which is mainly used to validate the object. It is also used latter as starting values for "optim" if no other starting values are provided. For the nonlinear regression, it must be a named vector.

Consider the following model:
$$y = \theta_0 + \theta_1 x_{1i} + \theta_2 x_{2i} + \varepsilon_i$$
with the instruments $Z_i = \{1, x_{2i}, z_{1i}, z_{2i}\}'$ and iid errors. We could create an object of class "linarGmm" as follows:

```
library(gmm4)
data(simData)
modelF <- model.frame(y~x1+x2, simData)
instF <- model.frame(~x2+z1+z2, simData)
mod1 <- new("linearGmm", modelF=modelF, instF=instF, k=3L, q=4L, vcov="iid",
            parNames=c("(Intercept)", "x1","x2"), n=50L,
            momNames=c("(Intercept)", "x2", "z1", "z2"),
            isEndo=c(FALSE, TRUE, FALSE, FALSE))
```

The print method describes the model.

```
mod1

## GMM Model
## *********
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 3
```

```
## Number of moment conditions: 4
## Number of Endogenous Variables: 1
## Sample size:  50
```

Although there is a validity procedure when the object is created, it is not recommended to create if this way. Small error not detected by the validity method could result in estimation problems. The constructor is the method "gmmModel", with the signature "gmmModels", which is the union class of all above types. The above model can be created as follows:

```
mod1 <- gmmModel(y~x1+x2, ~x2+z1+z2, data=simData, vcov="iid")
mod1

## GMM Model
## *********
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 3
## Number of moment conditions: 4
## Number of Endogenous Variables: 1
## Sample size:  50
```

The two other classes of object can be created the same way. Consider the following model:

$$y_i = e^{\theta_0 + \theta_1 x_{1i} + \theta_2 x_{2i}} + \varepsilon_i$$

using the same instruments. The nonlinear model can be created as follows:

```
theta0 <- c(theta0=1, theta1=1, theta2=2)
mod2 <- gmmModel(y~exp(theta0+theta1*x1+theta2*x2), ~x2+z1+z2, theta0,
                 data=simData, vcov="iid")
mod2

## GMM Model
## *********
## Moment type: nonlinear
## Covariance matrix: iid
## Number of regressors: 3
## Number of moment conditions: 4
## Number of Endogenous Variables: 2
## Sample size:  50
```

(Wrong number of endogenous variables. Needs to be fixed. The problem is that the variable names "(Intercept)" is in Z_i but not in the right hand side expression)

For the functional case, suppose we want to estimate the mean and variance of a normal distribution using the following moment condition:

$$E \begin{pmatrix} x_i - \mu \\ (x_i - \mu)^2 - \sigma^2 \\ (x_i - \mu)^3 \\ (x_i - \mu)^4 - 3\sigma^4 \end{pmatrix} = 0$$

The functions "fct" and "dfct" would be

```
fct <- function(theta, x)
    cbind(x-theta[1], (x-theta[1])^2-theta[2],
          (x-theta[1])^3, (x-theta[1])^4-3*theta[2]^2)
dfct <- function(theta, x)
    {
        m1 <- mean(x-theta[1])
```

```
        m2 <- mean((x-theta[1])^2)
        m3 <- mean((x-theta[1])^3)
        matrix(c(-1, -2*m1, -3*m2, -4*m3,
                 0, -1, 0, -6*theta[2]), 4, 2)
    }
```

The object can than be created:

```
theta0=c(mu=1,sig2=1)
x <- simData$x3
mod3 <- gmmModel(fct, x, theta0, grad=dfct, vcov="iid")
mod3

## GMM Model
## *********
## Moment type: function
## Covariance matrix: iid
## Number of regressors: 2
## Number of moment conditions: 4
## Sample size:  50
```

1.2   Methods for gmmModels Classes

- *residuals*: Only for linearGMM and nonlinearGMM, it returns $\varepsilon(\theta)$:

  ```
  theta0 <- c(theta0=1, theta1=1, theta2=2)
  e1 <- residuals(mod1, c(1,2,3))
  e2 <- residuals(mod2, theta0)
  ```

- *Dresiduals*: Only for linearGMM and nonlinearGMM, it returns the $n \times k$ matrix $d\varepsilon(\theta)/d\theta$:

  ```
  theta0 <- c(theta0=1, theta1=1, theta2=2)
  e1 <- Dresiduals(mod1)
  e2 <- Dresiduals(mod2, theta0)
  ```

  Notice that the coefficient θ is not required for linear models, for no error is returned if it is. It is just not used. For nonlinear regressions, the derivatives are obtained analytically using D() from the *utils* package.

- *model.matrix*: For linearGMM and nonlinearGmm only. For both classes, it ca be used to get the matrix of instruments:

  ```
  Z <- model.matrix(mod1, type="instruments")
  ```

  For linearGMM only, it can be used to get the matrix of regressors X

  ```
  X <- model.matrix(mod1)
  ```

- *modelResponse*: For linear model only, it returns the vector of response. It is not defined for nonlinearGMM classes because the left hand side is not always defined.

  ```
  Y <- modelResponse(mod1)
  ```

- "]": It creates a new object of the same class with a subset of moment conditions:

  ```
  mod1[1:3]
  ```

```
## GMM Model
## *********
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 3
## Number of moment conditions: 3
## Number of Endogenous Variables: 1
## Sample size:  50

mod2[c(1,2,4)]

## GMM Model
## *********
## Moment type: nonlinear
## Covariance matrix: iid
## Number of regressors: 3
## Number of moment conditions: 3
## Number of Endogenous Variables: 2
## Sample size:  50

mod3[-1]

## GMM Model
## *********
## Moment type: function
## Covariance matrix: iid
## Number of regressors: 2
## Number of moment conditions: 3
## Sample size:  50
```

- *as*: linearGmm can be converted into a nonlinearGmm or functionGmm. The former is userful when we impose nonlinear restrictions on the coefficients.

```
mod4 <- as(mod1, "nonlinearGmm")
```

Notice, however, that coefficient names and the variable names in modelF change in this case. It is done to avoid invalid variable and parameter names in the expressions. It will happens with the intercept or if there are interactions or transformations using the identity function I().

```
mod4@parNames

## [1] "theta1" "theta2" "theta3"

mod4@fLHS

## expression(Y)

mod4@fRHS

## expression(theta1*X1+theta2*X2+theta3*X3)
```

- *subset*: As for the S3 method, it creates the same class of object with a subset of the sample:

```
subset(mod1, simData$x1>4)

## GMM Model
## *********
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 3
```

```
## Number of moment conditions: 4
## Number of Endogenous Variables: 1
## Sample size:  31
```

- *evalMoment*: It computes the $n \times q$ matrix of moments, with the ith row being $g_i(\theta)'$:

```
gt <- evalMoment(mod1, 1:3)
```

- *evalDMoment*: It computes the $p \times k$ matrix of derivatives of the sample mean of $g_i(\theta)$ (the matrix G above):

```
theta0 <- c(theta0=.1, theta1=1, theta2=-2)
evalDMoment(mod2, theta0)

##               theta0     theta1     theta2
## (Intercept) -471.3350 -5128.568 -245.9807
## x2          -245.9807 -2651.763 -161.5198
## z1          -554.1436 -6026.748 -293.1144
## z2          -180.9458 -1964.310 -103.2112
```

- *momentVcov*: It computes $\hat{V}$ using the specification of the model as described in the previous section. For example, if the model is linear with MDS error, it computes $\hat{V} = \frac{1}{n}\sum_{i=1}^{n} \hat{\varepsilon}_i^2 Z_i Z_i'$.

```
momentVcov(mod1, theta=1:3)

##             (Intercept)        x2        z1        z2
## (Intercept)    85.76003  520.3819  88.88919 119.33071
## x2            520.38190 3981.2274 509.27177 919.50037
## z1             88.88919  509.2718 167.01913  91.37298
## z2            119.33071  919.5004  91.37298 273.68146
```

- *momentStrength*: For linearGmm only (for now), it computes the first stage F-test to measure the strength of the instruments:

```
momentStrength(mod1)

## $strength
##        Stats df1 df2         pv
## x1 4.113798   2  46 0.02271759
##
## $mess
## [1] "Instrument strength based on the F-Statistics of the first stage OLS"
```

Other methods will be presented below as they require to define other classes.

1.3 Restricted models

We can create objects of class "rlinearGmm", "rnonlinearGMM" or "rfunctionGMM" using the method *restGmmModel* and print the restrictions using the *printRestrict* method.

Lets first create a new model with more regressors:

```
UR.mod1 <- gmmModel(y~x1+x2+x3+z1, ~x1+x2+z1+z2+z3+z4, data=simData)
```

We can impose restrictions in two ways. Using $R\theta = q$ format:

```
R1 <- matrix(c(1,1,0,0,0,0,0,2,0,0,0,0,0,1,-1),3,5, byrow=TRUE)
q1 <- c(0,1,3)
R1.mod1 <- restGmmModel(UR.mod1, R1, q1)
R1.mod1
```

```
## GMM Model
## *********
## Moment type: rlinear
## Covariance matrix: HAC with  Quadratic Spectral  kernel and Andrews bandwidth
## Number of regressors: 2
## Number of moment conditions: 7
## Number of Endogenous Variables: 1
## Sample size:  50
## Constraints:
##   (Intercept) + x1 = 0
##    2 x2 = 1
##    x3 - z1 = 3
## Restricted regression:
##   (y-0.5x2-3x3) = (-(Intercept)+x1)+(x3+z1)
```

Or using character vectors. As long as it uses the parameter names, it will work fine.

```
R2 <- c("x1","2*x2+z1=2", "4+x3*5=3")
R2.mod1 <- restGmmModel(UR.mod1, R2)
printRestrict(R2.mod1)

## Constraints:
##    x1 = 0
##    2 x2 + z1 = 2
##    5 x3 = -1
## Restricted regression:
##   (y-x2+0.2x3) = (Intercept)+(-0.5x2+z1)
```

If parameters have special names because of the way the regression is defined, it will also w ork fine:

```
UR.mod2 <- gmmModel(y~x1*x2+exp(x3)+I(z1^2), ~x1+x2+z1+z2+z3+z4, data=simData)
R3 <- c("x1","exp(x3)+2*x1:x2", "I(z1^2)=3")
R3.mod2 <- restGmmModel(UR.mod2, R3)
printRestrict(R3.mod2)

## Constraints:
##    x1 = 0
##    exp(x3) + 2x1:x2 = 0
##    I(z1^2) = 3
## Restricted regression:
##   (y-3I(z1^2)) = (Intercept)+x2+(-2exp(x3)+x1:x2)
```

For nonlinearGmm, only character vector or list of formulas are allowed. The restriction must also be written as one coefficient as a function of the others.

```
R1 <- c("theta1=theta2^2")
restGmmModel(mod2, R1)

## GMM Model
## *********
## Moment type: rnonlinear
## Covariance matrix: iid
## Number of regressors: 2
## Number of moment conditions: 4
## Number of Endogenous Variables: 2
## Sample size:  50
## Constraints:
##   theta1 ~ theta2^2

printRestrict(restGmmModel(mod2, theta1~theta2))
```

```
## Constraints:
##  theta1 ~ theta2
```

Restrictions can also be imposed on functionGmm:

```
restGmmModel(mod3, "mu=0.5")

## GMM Model
## *********
## Moment type: rfunction
## Covariance matrix: iid
## Number of regressors: 1
## Number of moment conditions: 4
## Sample size:  50
## Constraints:
##  mu ~ 0.5
```

All methods described in the previous subsections also apply to restricted models. However, when θ is need, it must be of the right length, which is k minus the number of restrictions. Many of these methods use the *coef* method to obtain the unrestricted version of the coefficients and call the method for unrestricted models.

For example, in the following model

```
printRestrict(R2.mod1)

## Constraints:
##    x1 = 0
##    2 x2 + z1 = 2
##    5 x3 = -1
## Restricted regression:
##  (y-x2+0.2x3) = (Intercept)+(-0.5x2+z1)
```

There are only 2 restricted coefficients, the intercept and the coefficient of $(-0.5x_2 + z_1)$. Suppose there are respectively equal to 1.5 and 0.5, then the unrestricted version is

```
coef(R2.mod1, c(1.5,.5))

## (Intercept)          x1          x2          x3          z1
##        1.50        0.00        0.75       -0.20        0.50
```

Notice that any restricted class object contains its unrestricted version. For example, rlinearGmm is a class that contains a linearGMM class object plus a few additional slots. We can therefore use the *as* method directly to convert a restricted model to its unrestricted counterpart. We can therefore compute the residuals from the restricted model as follows:

```
e1 <- residuals(as(R2.mod1, "linearGmm"),
                coef(R2.mod1, c(1.5,.5)))
```

It is identical to use the "rlinearGmm" method directly:

```
e2 <- residuals(R2.mod1, c(1.5,.5))
all.equal(e1,e2)

## [1] TRUE
```

Other methods that behave in the same way include *evalMoment* and *momentVcov*. The methods that will produce different results include *Dresiduals*, *evalDMoment*, *model.matrix*, and *modelResponse*. Restrictions affect derivatives and the left and right hand sides of regression models. Fo example:

```
R1 <- c("theta1=theta2^2")
R1.mod2 <- restGmmModel(mod2, R1)
evalDMoment(mod2, c(theta0=1, theta1=1, theta2=1))

##                   theta0        theta1        theta2
## (Intercept)    -81045584    -879800997    -763376712
## x2            -763376712   -8146652309   -7316573837
## z1             -67269892    -726185991    -616728543
## z2            -215480202   -2340842161   -2071011945

evalDMoment(R1.mod2, c(theta0=1, theta2=1))

##                   theta0        theta2
## (Intercept)    -81045584    -2522978706
## x2            -763376712   -23609878455
## z1             -67269892    -2069100525
## z2            -215480202    -6752696267
```

Every method uses the method *modelDims* to extract the information for a model. For example, the slot "parNames" of mod2 and R1.mod2 are the same even if *theta*1 is not present in the restricted model.

```
mod2@parNames

## [1] "theta0" "theta1" "theta2"

R1.mod2@parNames

## [1] "theta0" "theta1" "theta2"
```

When we need the right specifications of the model, we need to extract that information using *modelDims*.

```
modelDims(mod2)$parNames

## [1] "theta0" "theta1" "theta2"

modelDims(mod2)$k

## [1] 3

modelDims(R1.mod2)$parNames

## [1] "theta0" "theta2"

modelDims(R1.mod2)$k

## [1] 2
```

1.4 A class object for GMM Weights

Now that we have our model classes well defined, we need a way to construct a weighting matrix. We could simply define W as a matrix and move on to the estimation section, but in an attempt to make the estimation more computationally efficient and more numerically stable, we construct the weights in a particular way depending on its structure. There is in fact an optimal choice for W that minimizes the asymptotic variance of the GMM estimator. If $W = V^{-1}$, the above property becomes:

$$\sqrt{n}\left(\hat{\theta}(V^{-1}) - \theta\right) \xrightarrow{d} N\left(0, [G'V^{-1}G]^{-1}\right),$$

The new covariance matrix $[G'V^{-1}G]^{-1}$ is smaller than the one based on other W in the sense that the difference (the second minust the first) is negative definite. The inverse V^{-1} may have to be computed

several times for inference or simply for estimation if we use iterative GMM of CUE. It is therefore worth finding a way to reduce the number of potentially unstable operations. For example, in the linear or nonlinear model with iid errors, $V^{-1} = [\sigma^2 \mathrm{E}(Z_i Z_i')]^{-1}$, and can be estimated by

$$\hat{V} = \frac{1}{\hat{\sigma}^2} \left(\frac{1}{n} \sum_{i=1}^{n} Z_i Z_i' \right)^{-1}$$

Therefore two $\hat{V}$'s differ only by their estimates of σ^2. It is therefore not necessary to recompute the second term each time. In fact, it is even not necessary to compute the sum. A more stable way would be to store the QR decomposition of the $n \times q$ matrix Z. The "gmmWeights" class store only what is needed. It can be created by the *evalWeights* method. It is a method for the union class "gmmModels", which includes all restricted models.The method has three arguments, the "gmmModels", the vector of coefficients, and the type of weights. The third argument can be a matrix, if we want to provide our own fixed one, the character "ident", to create an identity matrix or, which is the default, the character "optimal". In the latter case, the efficient weighting matrix is computed based on the characteristics of the "gmmModels" specified when the object was created.

There are two ways of creating an identity. The first way is to use the character "ident". In this case, it is not necessary to provide a vector of coefficients.

```
model <- gmmModel(y~x1, ~z1+z2, data=simData, vcov="iid") ## lets create a simple model
wObj <- evalWeights(model, w="ident")
```

The *show* method for the "gmmWeights" object prints the matrix as it should look like. If it is the efficient matrix, the inverse is computed and printed. It is not too efficient but when do we really need to see it? For the one we just created, we get

```
wObj
```

```
## GMM weights matrix object
## [1] "Identity"
```

Only a character string is printed because the identity is not actually created. After all, why should we? If we need to compute $G'IG$, we do not want to create I and do the operation, but rather compute $G'G$. That's how things are done in the package. For this reason, the second way of creating an identity weighting matrix is not recommended:

```
evalWeights(model, w=diag(3))
```

```
## GMM weights matrix object
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

The optimal matrix at θ can be obtained without specifying w.

```
wObj <- evalWeights(model, theta=c(1,2))
```

The type slot indicates how the weighting matrix is stored.

```
wObj@type
```

```
## [1] "qr"
```

Here the QR decomposition is store because vcov="iid". For any "gmmModels" including "functionGmm" classes, with vcov="MDS", the QR decomposition of the $n \times q$ matrix of moment conditions is stored. It avoids having to compute $g(\theta)'g(\theta)$. For HAC, there is no gain in storing the QR decomposition. The type is then "chol", which indicates that the Cholesky upper triangular matrix is stored:

```
model2 <- gmmModel(y~x1, ~z1+z2, data=simData, vcov="HAC")
evalWeights(model2, c(1,2))@type
```

```
## [1] "chol"
```

When the matrix is provided, the type is "weights", which indicates that no inversion is needed

```
evalWeights(model, w=diag(3))@type
```

```
## [1] "weights"
```

The weights matrix is used to compute the vector of estimates, its covariance matrix and to do inference. Most operations ar in the form $A'WB$ for matrices A and B. How do we compute those knowing that it depends on how W is stored in the object. The method *quadra* does it for us. Consider the following optimal weighting matrix, which is stored as a QR decomposition:

```
wObj <- evalWeights(model, theta=1:2)
```

Let compute G and $\bar{g}(\theta)$

```
G <- evalDMoment(model, theta=1:2)
gbar <- colMeans(evalMoment(model, theta=1:2))
```

If we need to compute $\bar{g}'W\bar{g}$, which is the objective function that we want to minimize, we do the following:

```
quadra(wObj, gbar)
```

```
## [1] 0.8478471
```

To compute $G'W\bar{g}$, which is the first order condition of the minimization problem, we proceed as follows:

```
quadra(wObj, G, gbar)
```

```
##            [,1]
## [1,] 0.1316962
## [2,] 0.7043764
```

If we only want W, we only use the weights as argument.

```
quadra(wObj)
```

```
##              [,1]         [,2]         [,3]
## [1,]   0.11425728 -0.041052353 -0.036112517
## [2,] -0.04105235  0.028230539  0.008474443
## [3,] -0.03611252  0.008474443  0.019640578
```

It is what the *print* method calls before printing the object. Finally, the "[" method can be used to create another "gmmWeights" object with a subset of the moment conditions. Only one argument is needed, and the slot "type" of the object is converted into "weights".

```
wObj[1:2]
```

```
## GMM weights matrix object
##             [,1]        [,2]
## [1,]  0.11425728 -0.04105235
## [2,] -0.04105235  0.02823054
```

We just saw a way of computing the objective function using *quadra*, but is can also be done using the *evalObjective* method. In this case, the weights is not necessarily based on the same coefficient as $\bar{g}$, which is often the case in GMM estimations:

```
theta0 <- 1:2
wObj <- evalWeights(model, theta0)
theta1 <- 3:4
evalObjective(model, theta1, wObj)

## [1] 374.6209
```

Notive that the method returns $n\bar{g}'W\bar{g}$.

1.5 The *solveGmm* Method

We now have all we need to estimate our models. The main method to estimate a model for a given W is *solveGmm*. The available signatures are:

```
showMethods("solveGmm")

## Function: solveGmm (package gmm4)
## object="allNLGmm", wObj="gmmWeights"
## object="functionGmm", wObj="gmmWeights"
##     (inherited from: object="allNLGmm", wObj="gmmWeights")
## object="linearGmm", wObj="gmmWeights"
## object="nonlinearGmm", wObj="gmmWeights"
##     (inherited from: object="allNLGmm", wObj="gmmWeights")
## object="rlinearGmm", wObj="gmmWeights"
##     (inherited from: object="linearGmm", wObj="gmmWeights")
## object="rnonlinearGmm", wObj="gmmWeights"
##     (inherited from: object="allNLGmm", wObj="gmmWeights")
## object="rslinearGmm", wObj="sysGmmWeights"
## object="slinearGmm", wObj="sysGmmWeights"
## object="snonlinearGmm", wObj="sysGmmWeights"
```

The last three are for systems of equations that we will cover later. The first is for "nonlinearGmm" and "functionGmm", and the second for "linearGmm". The methods require a gmmWeights object as second argument. For "nonlinearGmm" and "functionGmm" classes, there is a third optional argument, "theta0", which is the starting value to pass to *optim*. If not provided, the one stored in the GMM model object is used.

The method simply minimizes $\bar{g}(\theta)'W\bar{g}(\theta)$ for a given W. For "linearGmm" classes, the analytical solution is used. It is therefore the prefered class to use when it is possible. For all other classes, the solution is obtained by *optim*, and the argument "..." is used to pass options to it. For "nonlinearGmm", the gradian of the objective function, $2nG'W\bar{g}$ is passed to *optim* using the analytical derivative of the moment conditions (the *evalDMoment* method). For "functionGMM" classes, G is computed numerically using *numericDeriv* unless *dfct* was provided when the object was created. The *solveGmm* method returns a vector of coefficients and a convergence code. The latter is null for linear models and is the code from *optim* otherwise.

Consider the following linear model:

```
mod <- gmmModel(y~x1, ~z1+z2, data=simData, vcov="MDS")
```

We can estimate the model using the identity matrix as weights as follows:

```
wObj0 <- evalWeights(mod, w="ident")
res0 <- solveGmm(mod, wObj0)
res0$theta

## (Intercept)          x1
##   0.1049242   0.9553511
```

For two-step GMM, we just need to recompute the weighting matrix and call the method again.

```
wObj1 <- evalWeights(mod, res0$theta)
res1 <- solveGmm(mod, wObj1)
res1$theta

## (Intercept)          x1
##   0.1505614   0.9503860
```

We could iterate and get the iterative GMM estimator. The result may be different if we express the linear model in a nonlinear way or using a function, which is not recommended.

```
solveGmm(as(mod, "nonlinearGmm"), wObj1)$theta

##    theta1    theta2
## 0.1505604 0.9503862

solveGmm(as(mod, "functionGmm"), wObj1)$theta

##    theta1    theta2
## 0.1505614 0.9503860
```

Consider now the above nonlinear model that we repeat here.

```
theta0 <- c(theta0=0, theta1=0, theta2=0)
mod2 <- gmmModel(y~exp(theta0+theta1*x1+theta2*x2), ~x2+z1+z2, theta0,
                 data=simData, vcov="MDS")
wObj0 <- evalWeights(mod2, w="ident")
res1 <- solveGmm(mod2, wObj0, control=list(maxit=2000))
res1

## $theta
##       theta0       theta1       theta2
##   0.43293969   0.20638573  -0.01283577
##
## $convergence
## [1] 0
```

Notice that there is no signature for restricted models. However, it is not needed since they inherit from their unrestricted counterpart and the same procedure is needed to estimate them. Suppose, for example, that we want to impose the restriction $\theta_1 = \theta_2^2$.

```
R1 <- c("theta1=theta2^2")
rmod2 <- restGmmModel(mod2, R1)
res2 <- solveGmm(rmod2, wObj0, control=list(maxit=2000))
res2

## $theta
##     theta0     theta2
##   2.269389 -0.118783
##
## $convergence
## [1] 0
```

The unrestricted version can be extracted using *coef*.

```
coef(rmod2, res2$theta)

##       theta0       theta1       theta2
##   2.26938888   0.01410941  -0.11878305
```

15

1.6 GMM Estimation: the *gmmFit* method

For most users, what we presented above will rarely be used. What they want is a way to estimate their models without worrying about how it is done. The *gmmFit* method is the main method to estimate models. The only requirement is to first create a "gmmModels". Before going into all the details, the most important arguments to set is the object, which is a "gmmModels" class, and a type of GMM. The different types are: (1) "twostep" for two-step GMM, which is the default, (2) "iter" for iterative GMM, (3) "cue" for continuously updated GMM , or (4) "onestep" for th one-step GMM.

In this package, the one-step GMM means the estimation using the identity matrix as W. It is therefore not an efficient GMM. The two-step GMM, without any other argument is computed as follows:

1. Define W_0 as being the identity matrix.
2. Get $\hat{\theta}_1 \equiv \hat{\theta}(W_0)$
3. Compute $W_1 = [\hat{V}(\hat{\theta}_1)]^{-1}$.
4. Get $\hat{\theta}_2 \equiv \hat{\theta}(W_1)$.

For the iterative GMM we proceed as follows:

1. Define W_0 as being the identity matrix.
2. Get $\hat{\theta}_1 \equiv \hat{\theta}(W_0)$
3. Compute $W_1 = [\hat{V}(\hat{\theta}_1)]^{-1}$.
4. Get $\hat{\theta}_2 \equiv \hat{\theta}(W_1)$.
5. If $\|\hat{\theta}_1 - \hat{\theta}_2\|/(1 + \|\hat{\theta}_1\|) < itertol$, where *itertol* is a user defined tolerance level, stop. Otherwise, set $\hat{\theta}_1 = \hat{\theta}_2$ and go back to 3. By default, $itertol = 10^{-7}$.

CUE is a one step efficient GMM method in which $W = \hat{V}(\theta)$. The solution is obtained by minimizing $n\bar{g}(\theta)[\hat{V}(\theta)]^{-1}\bar{g}(\theta)$.

There are two special cases that are worth mentioning. The first case applies to all "gmmModels". If $q = k$, the model is just-identified. In that case, the choice of W has no effect on the solution. Therefore, *gmmFit* will automatically set W to the identity and return the one-step GMM solution. Setting the argument type to another value will therefore have no effect on the result.

Second, when "vcov" is set to "iid" in either a linearGMM or a "nonlinearGmm" model, the matrices W_1 and W_2 are proportional to each other. They therefore lead to the same solution. As a result, the two-step GMM, iterative GMM and CUE produce identical solution. In particular, if the model is linear, the solution corresponds to the two-stage least squares solution. In fact, *gmmFit* calls the method *tsls* in that case. We will look at the method below.

The *gmmFit* method returns the S4 class object "gmmfit". The object contains the vector of coefficient estimates, the "gmmWeights" used to obtain it, the model object and other information about the method and convergence. We will cover its methods in the next section. The only one we introduce now is the *show* method which prints the model info, the estimation method and the coefficient estimates. To avoid printing the model, we can set the argument "model" of print to FALSE.

```
mod <- gmmModel(y~x1, ~z1+z2, data=simData, vcov="MDS")
gmmFit(mod, type="onestep")

## GMM Model
## *********
## Moment type: linear
## Covariance matrix: MDS
## Number of regressors: 2
## Number of moment conditions: 3
## Number of Endogenous Variables: 1
## Sample size:  50
##
```

```
## Estimation:  One-Step GMM with fixed weights
## coefficients:
## (Intercept)            x1
##    0.1049242     0.9553511

print(gmmFit(mod, type="twostep"), model=FALSE)

##
## Estimation:  Two-Step GMM
## coefficients:
## (Intercept)            x1
##    0.1505614     0.9503860

print(gmmFit(mod, type="iter"), model=FALSE)

##
## Estimation:  Iterated GMM
## Convergence Iteration:  0
## coefficients:
## (Intercept)            x1
##    0.1604345     0.9487049
```

For nonlinear models, it is possible to pass arguments to *optim* and to set a different starting value with the argument "start".

```
theta0 <- c(theta0=0, theta1=0, theta2=0)
mod2 <- gmmModel(y~exp(theta0+theta1*x1+theta2*x2), ~x2+z1+z2, theta0,
                 data=simData, vcov="MDS")
res1 <- gmmFit(mod2)
print(res1, model=FALSE)

##
## Estimation:  Two-Step GMM
## Convergence Optim:  0
## coefficients:
##       theta0         theta1         theta2
##   0.61713001     0.18549461    -0.01975427

theta0 <- c(theta0=0.5, theta1=0.5, theta2=-0.5)
res2 <- gmmFit(mod2, start=theta0, control=list(reltol=1e-8))
print(res2, model=FALSE)

##
## Estimation:  Two-Step GMM
## Convergence Optim:  0
## coefficients:
##       theta0         theta1         theta2
##   0.61712992     0.18549462    -0.01975426
```

For the iterative GMM, we can control the tolerance level and the maximum number of iterations with the arguments "itertol" and "itermaxit". The argument "weights" is equal to the character string "optimal", which implies that by default W is set to the estimate of V^{-1}. If "weights" is set to "ident", *gmmFit* returns the one-step GMM. Alternatively, we can provide *gmmFit* with a fixed weighting matrix. It could be a matrix or a "gmmWeights" object. When the weighting matrix is provided, it returns a one-step GMM based on that matrix. The "gmmfit" object contains a slot "efficientGmm" of type logical. It is TRUE if the model has been estimated by efficient GMM. By default it is TRUE, since "weights" is set to "optimal". If "weights" takes any other value or if "type" is set to "onestep", it is set to FALSE. There is one exception. It is set to TRUE if we provide the method with a weighting matrix and we set the argument "efficientWeights" to TRUE. For example, the optimal weighting matrix of the minimum distance method does not depend on any coefficient. It

is probably a good idea in this case to compute it before and pass it to the *gmmFit* method. The value of the "efficientGmm" slot will be used by the *vcov* method to determine whether it should return the robust (or sandwich) covariance matrix.

1.7   Methods for "gmmfit" classes

- *meatGmm*: It returns the meat of the sandwich covariance matrix. The only other argument is "robust". A non robust meat assumes that $W = V^{-1}$, which is true if the model has been estimated by efficient GMM. Since W is usually a first step weighting matrix, it is not numerically identical to the estimate of V^{-1} based on the final estimate. However, it is a common practice to ignore it. The meat will in this case be equal to $(G'\hat{V}^{-1}G)$. If "robust" is TRUE, we do not assume that $W = V^{-1}$ and the meat becomes $(G'W\hat{V}WG)$.

- *bread*: It returns the bread of the sandwich covariance matrix, $(G'WG)^{-1}$, where W is the weighting matrix used to get the final estimate..

- *vcov*: It returns the covariance matrix of the coefficient. By default, it returns a sandwich matrix if the argument "efficienGmm" of the object is FALSE or if the model is just identified, and a non sandwich estimator otherwise. Here are all the possibilities:
  - Efficient and over-identified GMM: $(G'\hat{V}^{-1}G)^{-1}/n$
  - Just-identified GMM: $G^{-1}\hat{V}G^{-1'}/n$
  - Any other sandwich estimator: $(G'WG)^{-1}G'W\hat{V}WG(G'WG)^{-1}/n$.
  - The argument "breadonly" is set to TRUE: $(G'WG)^{-1}/n$. For efficient GMM, it is asymptotically equivalent to $(G'\hat{V}^{-1}G)^{-1}/n$. It is particularly useful for efficient and fixed weighting matrices.

  The method is flexible enough that you may hand up with a non-valid covariance matrix if not careful. For example, setting "sandwich" to FALSE would lead to non valid covariance matrix if the model was not estimated by efficient GMM. It is important to understand that we assume here that the specifications of the model are valid. If you set "vcov" to iid and that the errors are heteroscedastic, GMM is no longer efficient and a sandwich matrix is required. But event if you set "sandwich" to TRUE, it will not solve the problem because the meat will be computed assuming the errors are iid. You can, however, set the argument "modelVcov" to "MDS" which will set "sandwich" to TRUE and compute the meat properly.

  The argument "df.adj" can be set to TRUE if degrees of freedom adjustment is needed. In that case, the covariance matrix is multiplied by $n/(n-k)$. It is only included in the package to reproduce textbook examples. This adjustment is not really justified in the GMM context.

- *specTest*: It tests the null hypothesis $\mathrm{E}[g_i(\theta)] = 0$ using the J-test. The statistics is $n\bar{g}'\hat{V}^{-1}\bar{g}$ and it is asymptotically distributed as a χ^2_{q-k} under the null. The model must have been estimated by efficient GMM for this test to be valid. The method returns an S4 class object.

```
mod <- gmmModel(y~x1, ~z1+z2+z3, data=simData, vcov="MDS")
res <- gmmFit(mod)
specTest(res)

##
##   J-Test
##               Statistics  df  pvalue
## Test E(g)=0:      1.0333   2  0.5965
```

It is also possible to test subsets of instruments. Suppose we suspect z_2 to be invalid. We would estimate the model without z_2 and compute the difference between the J-tests $(J_1 - J_2)$, where J_1 is the J-test with z_2 and J_2 is the test without. The distribution is the number of instruments that we want to test, which is one in this example. To test it using the *specTest* method, we specify which instrument we want to test (z_2 is the third instrument if we include the intercept):

```
specTest(res, 3)

##
##  Testing the following subset of moments:
##  {z2}
##                Statistics  df  pvalue
## Test E(g)=0:        1.023   1  0.3118
```

- *summary*: It computes important information about the estimated model. It is an S4 class object with a *print* method that shows the results in the usual way.

```
summary(res)

## GMM Model
## *********
## Moment type: linear
## Covariance matrix: MDS
## Number of regressors: 2
## Number of moment conditions: 4
## Number of Endogenous Variables: 1
## Sample size:  50
##
## Estimation:  Two-Step GMM
## Sandwich vcov: FALSE
## coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.17154    0.51304  0.3344   0.7381
## x1           0.94690    0.10072  9.4011   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##   J-Test
##                Statistics  df  pvalue
## Test E(g)=0:       1.0333   2  0.5965
##
##
## Instrument strength based on the F-Statistics of the first stage OLS
## x1 : F( 3 ,  46 ) =  7.603642  (P-Vavue =  0.0003126709 )
```

The argument "..." can be used to pass options to the *vcov* method. For example, we can used the bread only to compute the standard errors:

```
summary(res, breadOnly=TRUE)@coef

##              Estimate Std. Error    t value      Pr(>|t|)
## (Intercept) 0.1715368  0.5188364 0.3306183 7.409328e-01
## x1          0.9469048  0.1018860 9.2937705 1.489173e-20
```

- *hypothesisTest*: Method to perform hypothesis tests on the coefficients. Consider the following unrestricted model:

```
mod <- gmmModel(y~x1+x2+x3+z1, ~x1+x2+z1+z2+z3+z4, data=simData, vcov="iid")
res <- gmmFit(mod)
```

We want to test the hypothesis

```
R <- c("x1=1", "x2=x3", "z1=-0.7")
rmod <- restGmmModel(mod, R)
printRestrict(rmod)
```

```
## Constraints:
##    x1 = 1
##    x2 - x3 = 0
##    z1 = -0.7
## Restricted regression:
##    (y-x1+0.7z1) = (Intercept)+(x2+x3)
```

There are three ways to do it. The Wald test only requires us to estimate the unrestricted model. It is performed as follows:

```
hypothesisTest(object.u=res, R=R)

## Wald Test
## ***********
## The Null Hypothesis:
##    x1 = 1
##    x2 - x3 = 0
##    z1 = -0.7
## Distribution: Chi-square with 3 degrees of freedom
##    Statistics       Pvalue
## 1   15.97411 0.001147931
```

The statistics is $(R\hat{\theta}-q)'[R\hat{\Omega}R']^{-1}(R\hat{\theta}-q)$, where $\hat{\Omega}$ is the covariance matrix of $\hat{\theta}$, and is distributed as a chi-square with degrees of freedom equal to the number of restrictions. Here R and q are given in the restricted model:

```
rmod@cstLHS

##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    1    0    0    0
## [2,]    0    0    1   -1    0
## [3,]    0    0    0    0    1

rmod@cstRHS

## [1]  1.0  0.0 -0.7
```

We can also test it using the LM test, which test if the score of the GMM objective is close enough to zero when evaluated at the restricted coefficient estimates. The statistics is

$$n\bar{g}(\tilde{\theta})'\hat{V}^{-1}\tilde{G}\hat{\Omega}\tilde{G}'\hat{V}^{-1}\bar{g}(\tilde{\theta}),$$

where the tilde implies that it is evaluated at the restricted coefficient estimates. The asymptotic distribution is the same as the Wald test. To perform the test, we need to estimate the restricted model.

```
res.r <- gmmFit(rmod)
```

Then, we perform the test

```
hypothesisTest(object.r=res.r)

## LM Test
## ***********
## The Null Hypothesis:
##    x1 = 1
##    x2 - x3 = 0
##    z1 = -0.7
## Distribution: Chi-square with 3 degrees of freedom
##    Statistics       Pvalue
## 1   11.58521 0.008947939
```

The LR test, compares the values of the GMM objective function at the restricted and unrestricted coefficient estimates. It is in fact the restricted minus the unrestricted one. The distribution is also the same in large samples. We therefore need both the restricted and unrestricted model:

```
hypothesisTest(object.r=res.r, object.u=res)

## Wald Test
## ***********
## The Null Hypothesis:
##    x1 = 1
##    x2 - x3 = 0
##    z1 = -0.7
## Distribution: Chi-square with 3 degrees of freedom
##    Statistics       Pvalue
## 1    15.97411 0.001147931
```

Alternatively, we can give both model and specify the test.

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

- *coef*: Returns the coefficient estimate.

```
coef(res.r)

## (Intercept)      (x2+x3)
##   1.24288790 -0.09512986
```

- *residuals*: Returns the residuals. Only for "linearGmm" and "nonlinearGmm".

```
e <- residuals(res)
e.r <- residuals(res.r)
```

- *DWH*: It performs the Durbin-Wu-Hausman test. In general, the purpose of the test is to compare an efficient estimator, $\hat{\theta}$, with an inefficient one, $\tilde{\theta}$. Under the null hypothesis, both are consistent estimators of θ and under the alternative only $\tilde{\theta}$ is consistent. It is well known in the linear GMM setup as a way of comparing OLS with GMM. We want to test if it is worth instrumenting the suspected endogenous vaiables among th regressors. The method with signature $\{gmmfit, lm\}$ performs such test.

```
mod <- gmmModel(y~x1, ~z1+z2, data=simData, vcov="iid")
res1 <- gmmFit(mod)
res2 <- lm(y~x1, simData)
DWH(res1,res2)

##
##   Hausman Test
##
##                  Statistics  df    pvalue
## Hausman Test:        1.1039   2   0.57581
```

Used this way, the test is defined as $(\theta_{ols} - \theta_{gmm})'\Sigma(\theta_{ols} - \theta_{gmm})$, where Σ is the generalized inverse of $[\widehat{\text{Var}(\theta_{gmm})} - \widehat{\text{Var}(\theta_{ols})}]$. The degrees of freedom is the rank of difference between the two covariance matrices. The argument "tol" is the tolerance level for the Moore-Penrose generalized inverse (for singular values less than "tol", their inverse is set to zero). The degrees of freedom should be 1 here because there is only one endogenous variable. That approach is therefore not too stable. Below, we consider a regression approach. The method with signature $\{gmmfit, gmmfit\}$ is used to compare two GMM estimators applied on the same regression model, using the same approach.

For the signature $\{gmmfit, missing\}$, the test is done using an auxiliary regression. The fitted endogenous regressors are added to the regression model and a joint significance test on their coefficients is performed. For the example we have here, we would regress x_1 on z_1 and z_2 with an intercept, regress y on x_1 and the fitted value $\hat{x}_1$ and test the coefficient of $\hat{x}_1$. Using DWH we obtain:

```
DWH(res1)

##
##  Durbin-Wu-Hausman Test
##              Statistics  df   pvalue
## DWH Test:       1.1983   1  0.27367
```

Notice that the Wald test is robust in the sense that the covariance matrix is based on the specification of the "gmmModels". For example, if "vcov" was set to "MDS", an HCCM covariance matrix would be used.

1.8   The *tsls* method

This method is to estimate linear models with two-stage least squares. It returns a "tsls" class object which inherits from "gmmfit". Most "gmmfit" methods are the same with the eception of *bread*, *meatGmm* and *vcov*. They just use the structure of 2LSL to make them more computationally efficient. They may be removed in future version and included in the main "gmmfit" methods.

If the model has iid error, *gmmFit* and *tsls* are numerically identical. In fact, the function is called by *gmmFit* in that case. The main reason for using it is if we have a more complex variance structure but want to avoid using a fully efficient GMM, which may have worse small sample properties. Therefore, "sandwich" is set to TRUE in the *vcov* method for "tsls" objects. In the following example, errors are assumed heteroscedastic, and the model is estimated by 2SLS. The *summary* method returns, however, robust standard errors because "sandwich=TRUE" is the default in the *vcov* method of "tsls".

```
mod <- gmmModel(y~x1, ~z1+z2+z3, data=simData, vcov="MDS")
res <- tsls(mod)
summary(res)@coef

##               Estimate Std. Error   t value      Pr(>|t|)
## (Intercept) 0.3310664  0.5240580 0.6317361 5.275593e-01
## x1          0.9218663  0.1024185 9.0009738 2.237243e-19
```

1.9   *gmm4*: A function to fit them all

If you still think that the *gmmFit* method is not simple enough because you have to crate a model first, the *gmm4* function will do everything for you. It is the function that looks the most like its ancestor function *gmm* from the gmm package. It is still required to specify the structure of variance for the moment conditions. In fact, it combines all arguments of the *gmmModel* constructor and *gmmFit* method. Here are a few examples.

You want to estimate

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \varepsilon$$

using the instruments $\{x_2, z_1, z_2, z_3\}$. We do not want to assume homoscedasticity, so we want to set "vcov" to "MDS". We want to estimate the model by two-step GMM.

```
res1 <- gmm4(y~x1+x2, ~x2+z1+z2+z3, type="twostep", vcov="MDS", data=simData)
res1

## GMM Model
## *********
## Moment type: linear
```

```
## Covariance matrix: MDS
## Number of regressors: 3
## Number of moment conditions: 5
## Number of Endogenous Variables: 1
## Sample size:  50
##
## Estimation:  Two-Step GMM
## coefficients:
## (Intercept)            x1            x2
##  1.12231930    0.87408142   -0.09146589
```

We want to compare it with iterative GMM:

```
res2 <- gmm4(y~x1+x2, ~x2+z1+z2+z3, type="iter", vcov="MDS", data=simData)
```

Now, we want to estimate the model with the restrictions $\theta_1 = \theta_2$

```
res1.r <- gmm4(y~x1+x2, ~x2+z1+z2+z3, type="twostep", vcov="MDS",
               data=simData, cstLHS="x1=x2")
res1.r

## GMM Model
## *********
## Moment type: rlinear
## Covariance matrix: MDS
## Number of regressors: 2
## Number of moment conditions: 5
## Number of Endogenous Variables: 1
## Sample size:  50
## Constraints:
##    x1 - x2 = 0
## Restricted regression:
##  y = (Intercept)+(x1+x2)
##
## Estimation:  Two-Step GMM
## coefficients:
## (Intercept)       (x1+x2)
##  4.70393880    0.02575094
```

Since the function returns a "gmmfit" object, all methods work with the output. We for example test the restriction:

```
hypothesisTest(res1, res1.r, type="LR")

## LR Test
## ***********
## The Null Hypothesis:
##    x1 - x2 = 0
## Distribution: Chi-square with 1 degrees of freedom
##    Statistics Pvalue
## 1    126.9516      0
```

There is also a *tsls* method for "formula", which works the same way:

```
res3 <- tsls(y~x1+x2, ~x2+z1+z2+z3, vcov="MDS", data=simData)
res3

## GMM Model
## *********
## Moment type: linear
```

```
## Covariance matrix: MDS
## Number of regressors: 3
## Number of moment conditions: 5
## Number of Endogenous Variables: 1
## Sample size:  50
##
## Estimation:  Two-Stage Least Squares
## coefficients:
## (Intercept)             x1             x2
##  1.14484351     0.87623904   -0.09597286
```

It is still important to specify the variance structure in order to obtain the appropriate coefficient standard errors. To estimate a nonlinear model, *gmm4* will recognize it by the way the formula is set along with the names vector "tet0".

```
res3 <- gmm4(y~theta0+exp(theta1*x1+theta2*x2), ~x2+z1+z2+z3+z4, vcov="iid",
             tet0=c(theta0=1, theta1=0, theta2=0), data=simData)
res3

## GMM Model
## *********
## Moment type: nonlinear
## Covariance matrix: iid
## Number of regressors: 3
## Number of moment conditions: 6
## Number of Endogenous Variables: 2
## Sample size:  50
##
## Estimation:  Two-Step GMM
## Convergence Optim:   0
## coefficients:
##      theta0         theta1         theta2
##  1.25158662     0.23850834   -0.02834336
```

1.10   Textbooks Applications

In this section, we cover a few examples from major textbooks. Since it is meant to help users who care less about the structure of the package, we use, when possible, the quicker functions that we just intruduced in the last section.

1.10.1   Stock-Watson

In this section, we cover examples from Stock and Watson (2015). In Chapter 12, the demand for cigarettes is estimated for 1985 using a panel. The following data change is required

```
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")
```

In equation 12.15, the demand is estimated using sales tax as an instrument for price. In order to get the same standard errors, we need to assume "MDS", and use a sandwich matrix with degrees of freedom adjustment.

```
res1 <- gmm4(log(packs)~log(rprice)+log(rincome),
             ~log(rincome)+tdiff, data = c1995, vcov="MDS")
summary(res1, sandwich=TRUE, df.adj=TRUE)@coef

##               Estimate Std. Error    t value      Pr(>|t|)
## (Intercept)  9.4306583  1.2593926  7.4882595  6.979292e-14
## log(rprice) -1.1433751  0.3723027 -3.0710902  2.132787e-03
## log(rincome) 0.2145153  0.3117469  0.6881071  4.913853e-01
```

Equation 12.16, for which both cigarettes and sales taxes are used as instruments, can be reproduced using the same specifications. We also have to set "centeredVcov" to FALSE. We have not seen that argument yet. When set to TRUE, the moments are centered before computing the weights. For more details on when it should be centered, see Hall (2005).

```
res2<- tsls(log(packs)~log(rprice)+log(rincome),
            ~log(rincome)+tdiff+I(tax/cpi), data = c1995,
            centeredVcov=FALSE, vcov="MDS")
summary(res2, sandwich=TRUE, df.adj=TRUE)@coef

##               Estimate Std. Error    t value      Pr(>|t|)
## (Intercept)  9.8949555  0.9592169 10.315660  5.986874e-25
## log(rprice) -1.2774241  0.2496100 -5.117680  3.093166e-07
## log(rincome) 0.2804048  0.2538897  1.104436  2.694041e-01
```

In Table 12.1, the long-run demand elasticity is estimated over a 10 year period. They compare a model with only sales tax as instrument, a model with cigarettes tax only and one with both.

```
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))
res1 <- tsls(dQ~dP+dInc, ~dInc+dTs, vcov="MDS", data=data)
res2 <- tsls(dQ~dP+dInc, ~dInc+dT, vcov="MDS", data=data)
res3 <- tsls(dQ~dP+dInc, ~dInc+dTs+dT, vcov="MDS", data=data)
```

You can print the summary to see the result, but I use the texreg package of Leifeld (2013),with an home made *extact* method (see Appendix), to make it more compact and more like Table 12.1 of the textbook. Table 1 presents the results. There is a small difference in the first stage F-test, which could be explained by the way they compute the covariance matrix. We cannot figure out our to get the same first two digits. Using *lm* manually and "HC1" type of HCCM, the test is identical to what we get here and it is the closest we can get from their results. It is the same for the other two F-tests.

For the J-test, the difference is a little larger. But, we have to notice that if we assume "MDS", 2SLS is not efficient and the J-test is not valid. If we estimate the model by efficient GMM, the J-test gets closer to what the authors get.

```
res4 <- gmm4(dQ~dP+dInc, ~dInc+dTs+dT, vcov="MDS", data=data)
specTest(res4)

##
##  J-Test
##                Statistics  df    pvalue
## Test E(g)=0:       4.8726   1  0.027286
```

```
res4 <- gmm4(dQ~dP+dInc, ~dInc+dTs+dT, vcov="MDS", data=data)
```

	Model 1	Model 2	Model 3
(Intercept)	-0.12	-0.02	-0.05
	(0.07)	(0.07)	(0.06)
dP	-0.94^{***}	-1.34^{***}	-1.20^{***}
	(0.21)	(0.23)	(0.20)
dInc	0.53	0.43	0.46
	(0.34)	(0.30)	(0.31)
J-test Statistics			4.29
J-test p-value			0.04
First Stage F-stats(dP)	33.67	107.18	88.62

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

Table 1: Table 12.1 of Stock and Watson textbook

1.10.2 Greene

In this section, we want to reproduce results from Greene (2012).

In Example 13.7, the author estimates a nonlinear model with (1) the method of moments, one-step GMM and efficient GMM. To reproduce the results, we first need to create a dataset for 1988 remove zero income observations, and scale income.

```
data(HealthRWM)
dat88 <- subset(HealthRWM, year==1988 & hhninc>0)
dat88$hhninc <- dat88$hhninc/10000
```

The model is

$$hhninc = \exp[b_0 + b_1 age + b_2 educ + b_3 female] + \varepsilon$$

We want to reproduce Table 13.2. The NLS estimates shows that we have the same data used by the author.

```
thet0 <- c(b0=log(mean(dat88$hhninc)),b1=0,b2=0,b3=0)
g <- hhninc~exp(b0+b1*age+b2*educ+b3*female)
res0 <- nls(g, dat88, start=thet0, control=list(maxiter=100))
summary(res0)$coef

##         Estimate   Std. Error      t value      Pr(>|t|)
## b0 -1.693313418 0.0441023947 -38.3950447 4.622165e-279
## b1  0.002066691 0.0006060923   3.4098617  6.557037e-04
## b2  0.047916516 0.0024693628  19.4044048  1.332381e-80
## b3 -0.006581704 0.0137357112  -0.4791674  6.318430e-01
```

The second column is the method of moment (or just identified GMM), using the regressors as instruments.

```
h1 <- ~age+educ+female
model1 <- gmmModel(g, h1, thet0, vcov="MDS", data=dat88)
res1 <- gmmFit(model1, control=list(reltol=1e-10, abstol=1e-10))
```

The third column is first step GMM using the instruments $\{age, educ, female, hstat, married\}$.

```
h2 <- ~age+educ+female+hsat+married
model2 <- gmmModel(g, h2, thet0, vcov="MDS", data=dat88)
res2 <- gmmFit(model2, type="onestep")
```

The third is efficient GMM using the same instruments.

26

	Model 1	Model 2	Model 3
b0	-1.69258^{***}	-1.45552^{***}	-1.61908^{***}
	(0.04214)	(0.10102)	(0.04156)
b1	0.00178^{**}	-0.00028	0.00097
	(0.00057)	(0.00100)	(0.00056)
b2	0.04861^{***}	0.03731^{***}	0.04688^{***}
	(0.00262)	(0.00518)	(0.00261)
b3	0.00069	-0.02205	-0.01487
	(0.01384)	(0.01445)	(0.01357)

$^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

Table 2: Attempt to reproduce Table 13.2 from Greene (2012)

```
res3 <- gmmFit(model2)
```

The results (column 2 to 4 of Table 13.2) are presented in Table 2. The results are not identical, which is expected since results from nonlinear models depends on how the optimizer used is tuned. Only the last column cannot be explained by rounding errors or optimizer tuning. We have tried different tuning parameters in *optim* and it never gets closer. Even if we start with the author's solution, *optim* finds a solution with smaller value of the objective function.

In the Example 8.7, the author computes the Hausman test for a consumption function. The efficient estimator is the OLS estimator and the inefficient but consistent is 2SLS with lag income and consumption as instruments. We first estimate the models:

```
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)
model <- gmmModel(C~Y, ~Y1+C1, data=dat, vcov="iid")
```

We then estimate them with OLS and 2SLS.

```
res1 <- tsls(model)
res2 <- lm(C~Y)
```

Result of the test from Example 8.7-2:

```
DWH(res1)

##
##  Durbin-Wu-Hausman Test
##            Statistics  df     pvalue
## DWH Test:      8.811   1  0.0029942
```

The difference is explained by rounding errors. We get the same as the author if we square the t ratio using only three digits. For example 8.7-1, we first try to adjust the covariance for the degrees of freedom.

```
DWH(res1, res2, df.adj=TRUE)

##
##  Hausman Test
##              Statistics  df     pvalue
## Hausman Test:    8.0659   1  0.0045106
```

The result is a little different (the author reports 8.481). To reproduce the same results we need to specify the variance.

```
X <- model.matrix(model)
Xhat <- qr.fitted(res1@wObj@w, X)
s2 <- sum(residuals(res2)^2)/(res2$df.residual)
v1 <-  solve(crossprod(Xhat))*s2
v2 <- solve(crossprod(X))*s2
DWH(res1, res2, v1=v1, v2=v2)

##
##  Hausman Test
##                 Statistics  df    pvalue
## Hausman Test:       8.4814   1  0.003588
```

What we do above is to assume that the variance of the 2SLS and OLS coefficients are respectively $\hat{\sigma}^2(\hat{X}'\hat{X})^{-1}$ and $\hat{\sigma}^2(X'X)^{-1}$, where $\hat{X}$ is the fitted values of the regression of X on the instruments and $\hat{\sigma}^2$ is the estimated variance of the error terms using the unbiased OLS estimator. We therefore need the same estimate to obtain the same results.

1.10.3   Wooldridge

In this section, we want to reproduce results from Wooldridge (2016).

2   Systems of Equations

We consider two type of system of equations. The linear system:

$$Y_{ji} = X'_{ji}\theta_j + \varepsilon_{ji}$$

or

$$Y_{ji}(\theta_j) = X_{ji}(\theta_j) + \varepsilon_{ji}$$

for $j = 1, ..., m$, the number of equations, and $i = 1, ..., n$, the number of observations, with θ_j being a $k_j \times 1$ vector. We assume that for each equation j, there is a $q_j \times 1$ vector of instruments Z_{ji} that satisfies $\mathrm{E}[\varepsilon_{ji}Z_{ji}] = 0$. The moment conditions can therefore be written as:

$$E[g_i(\theta)] \equiv E \begin{bmatrix} \varepsilon_{1i}Z_{1i} \\ \varepsilon_{2i}Z_{2i} \\ \varepsilon_{3i}Z_{3i} \\ \vdots \\ \varepsilon_{mi}Z_{mi} \end{bmatrix} = 0$$

The model is just-identified if $k_j = q_j$ for all j, and it is over-identified if $k_j < q_j$ for ar least one j. For now, we offer two possible variance structures. We refer to "iid" models in which the errors are conditionally homoscedastic. In that case, the asymptotic variance of the moment condition is: 1

$$Var[\sqrt{(n)}\bar{g}(\theta)] \xrightarrow{p} S \equiv \begin{pmatrix} \sigma_1^2 E[Z_{1i}Z'_{1i}] & \sigma_{12}E[Z_{1i}Z'_{2i}] & \cdots & \sigma_{1m}E[Z_{1i}Z'_{mi}] \\ \sigma_{21}E[Z_{2i}Z'_{1i}] & \sigma_2^2 E[Z_{2i}Z'_{2i}] & \cdots & \sigma_{2m}E[Z_{2i}Z'_{mi}] \\ \vdots & \vdots & \vdots & \vdots \\ \sigma_{m1}E[Z_{mi}Z'_{1i}] & \sigma_{m2}E[Z_{mi}Z'_{2i}] & \cdots & \sigma_m^2 E[Z_{mi}Z'_{mi}] \end{pmatrix}$$

We can estimate $E[Z_{li}Z'_{ji}]$ by $\frac{1}{n}\sum_{i=1}^n Z_{li}Z'_{ji}$ and σ_{lj} by $\frac{1}{n}\sum_{i=1}^n \hat{\varepsilon}_{li}\hat{\varepsilon}_{ji}$. We label this estimate $\hat{S}$. If $Z_{li} = Z_{ji}$ for all l and j, which implies that all equations have the same instruments, we can simplify the expression. Let $\Sigma = \mathrm{E}[\varepsilon_i\varepsilon'_i]$, where $\varepsilon_i = \{\varepsilon_{1i}, ..., \varepsilon_{mi}\}'$ and $Z_i = Z_{ji}$ for all $j = 1, ..., m$. The asymptotic variance can be written as:

$$Var[\sqrt{(n)}\bar{g}(\theta)] \xrightarrow{p} S \equiv \Sigma \otimes E[Z_iZ'_i],$$

28

where $\otimes$ is the kronecker product. S can be estimated by $\hat{S} = \hat{\Sigma} \otimes \left[\frac{1}{n} \sum_{i=1}^{n} Z_i Z_i' \right]$, where $\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} \hat{\varepsilon}_i \hat{\varepsilon}_i'$. If we relax the homoscedasticity, the variance structure is labeled "MDS". In that case, the asymptotic variance of the moments are:

$$Var[\sqrt{(n)}\bar{g}(\theta)] \xrightarrow{p} S \equiv \begin{pmatrix} E[\varepsilon_{1i}^2 Z_{1i} Z_{1i}'] & E[\varepsilon_{1i}\varepsilon_{2i} Z_{1i} Z_{2i}'] & \cdots & E[\varepsilon_{1i}\varepsilon_{mi} Z_{1i} Z_{mi}'] \\ E[\varepsilon_{2i}\varepsilon_{1i} Z_{2i} Z_{1i}'] & E[\varepsilon_{2i}^2 Z_{2i} Z_{2i}'] & \cdots & E[\varepsilon_{2i}\varepsilon_{mi} Z_{2i} Z_{mi}'] \\ \vdots & \vdots & \vdots & \vdots \\ E[\varepsilon_{mi}\varepsilon_{1i} Z_{mi} Z_{1i}'] & E[\varepsilon_{mi}\varepsilon_{2i} Z_{mi} Z_{2i}'] & \cdots & E[\varepsilon_{mi}^2 Z_{mi} Z_{mi}'] \end{pmatrix}$$

It can be estimated by

$$\hat{S} = \frac{1}{n} \sum_{i=1}^{n} \begin{pmatrix} \hat{\varepsilon}_{1i}^2 Z_{1i} Z_{1i}' & \hat{\varepsilon}_{1i}\hat{\varepsilon}_{2i} Z_{1i} Z_{2i}' & \cdots & \hat{\varepsilon}_{1i}\hat{\varepsilon}_{mi} Z_{1i} Z_{mi}' \\ \hat{\varepsilon}_{2i}\hat{\varepsilon}_{1i} Z_{2i} Z_{1i}' & \hat{\varepsilon}_{2i}^2 Z_{2i} Z_{2i}' & \cdots & \hat{\varepsilon}_{2i}\hat{\varepsilon}_{mi} Z_{2i} Z_{mi}' \\ \vdots & \vdots & \vdots & \vdots \\ \hat{\varepsilon}_{mi}\hat{\varepsilon}_{1i} Z_{mi} Z_{1i}' & \hat{\varepsilon}_{mi}\hat{\varepsilon}_{2i} Z_{mi} Z_{2i}' & \cdots & \hat{\varepsilon}_{mi}^2 Z_{mi} Z_{mi}' \end{pmatrix}$$

Another type of systems considered in the package are the ones in which each equation has the same instruments and that these instruments are the union of all regressors from all equations. This is called the SUR assumption (Seemingly Unrelated Regressions). We will compare the estimation of the different models below. Notice that there is no function type of system yet because we don't see any specific applications. Suggestions are welcome if you have examples in mind.

2.1   A class object for System of Equations

The two classes are "slinearGmm" and "snonlinearGmm" and the union class is "sgmmModels". For most, the slots are the same with the exception that they are lists. The other difference is that the whole data.frame for all equations is store in the slot "data". For "slinearGmm", the equations and instruments are defined in the slots "modelT" and "instT", the latter being also the format for "snonlinearGmm" classes. They are lists of terms for each formula. There are two extra slots in system classes, "eqnNames", which labels each equation, and "SUR", which is TRUE if the SUR assumption is satisfied.The constructor is *sysGmmModel* and works as the *gmmModel* constructor. A *show* method prints the most important specification of the system of equations. Here is an example.

```
data(simData)
g <- list(Supply=y1~x1+z2, Demand1=y2~x1+x2+x3, Demand2=y3~x3+x4+z1)
h <- list(~z1+z2+z3, ~x3+z1+z2+z3+z4, ~x3+x4+z1+z2+z3)
smod1 <- sysGmmModel(g, h, vcov="iid", data=simData)
smod1

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
```

If we do not name the equations as we did, the default names $Eqnj$ for $j = 1, ..., m$ will be given. As for single equations, the "vcov" argument defines the assumption we make on the structure of the moment conditions variance. "snonlinearGmm" are constructed the same way with the exception that "theta0", a list of named starting coefficient vectors, must be provided. If we only provide one formula for the instruments, the same instruments will be used in all equations.

```
smod2 <- sysGmmModel(g, ~x2+x4+z1+z2+z3+z4, vcov="iid", data=simData)
smod2
```

```
## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=7, number of Endogenous: 1
## Demand1: coefs=4, moments=7, number of Endogenous: 2
## Demand2: coefs=4, moments=7, number of Endogenous: 1
## Sample size:  50
```

To impose the SUR assumption, we just ignore the instrument argument. In that case, instruments will be constructed using the union of all regressors.

```
smod3 <- sysGmmModel(g, vcov="iid", data=simData)
smod3

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=7, number of Endogenous: 0
## Demand1: coefs=4, moments=7, number of Endogenous: 0
## Demand2: coefs=4, moments=7, number of Endogenous: 0
## Sample size:  50
```

There is one other way to create a system classes. If one tries to create a "linearGmm" class using a matrix as the left hand side of the regression, the model will automatically converted to a system of equation with the same regressors and same instruments. Here is an example using simulated data.

```
dat <- list(y=matrix(rnorm(150),50,3),
            x=rnorm(50), z1=rnorm(50),
            z2=rnorm(50))
mod <- gmmModel(y~x, ~z1+z2, data=dat, vcov="iid")
mod

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Eqn1: coefs=2, moments=3, number of Endogenous: 1
## Eqn2: coefs=2, moments=3, number of Endogenous: 1
## Eqn3: coefs=2, moments=3, number of Endogenous: 1
## Sample size:  50
```

We could therefore create a multivariate regression in the following way:

```
mod <- gmmModel(y~x, ~x, vcov="iid", data=dat)
```

2.2 Methods for "sgmmModels" classes

The methods are very similar to the ones described above for "gmmModels" classes. Here, we briefly describe the difference.

- [: The method has two arguments. The first is an vector of integers to select the equations, and the second is a list of integers to select the instruments in each of the selected equation. For example, the following creates a system of equations from the "smod1" object with the first two equations, and using the first 3 instruments in the first equation and the first 4 for the second.

```
smod1[1:2, list(1:3,1:4)]

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=3, number of Endogenous: 1
## Demand1: coefs=4, moments=4, number of Endogenous: 2
## Sample size:  50
```

If the second argument is missing, all instruments are selected. If only one equation is selected, the object if converted to a single equation class. We can therefore estimate each equation separately.

```
gmmFit(smod1[1])

## GMM Model
## *********
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 3
## Number of moment conditions: 4
## Number of Endogenous Variables: 1
## Sample size:  50
##
## Estimation:  Two-Stage Least Squares
## coefficients:
## (Intercept)          x1           z2
##   1.0080239    0.8349503   -0.1696884
```

- *model.matrix* and *modelResponse*. The methods return the model.matrix and modelResponse of each equation in a list. Basically, the following are equivalent:

```
mm <- model.matrix(smod1)
mm <- lapply(1:3, function(i) model.matrix(smod1[i]))
```

- *evalMoment*, *evalDMoment*, *Dresiduals*: The methods are applied to each equation and returned in a list. Notice that *theta* must be stored in a list.

```
theta <- list(1:3, 1:4, 1:4)
gt <- evalMoment(smod1, theta)
```

- *residuals*: It returns a $n \times m$ matrix of residuals. We can therefore estimate Σ directly:

```
Sigma <- crossprod(residuals(smod1, theta))/smod1@n
```

- *momentVcov*: It returns the $Q \times Q$ matrix $\hat{S}$, where $Q = \sum_{j=1}^{m} q_j$. The way it is computed depends on the structure of the variance as described above.

- *merge*: The method is used to merge single equations into a system class, or to add equations to an already created system class. The "smod1" object could have been created this way.

```
eq1 <- gmmModel(g[[1]], h[[1]], data=simData, vcov="iid")
eq2 <- gmmModel(g[[2]], h[[2]], data=simData, vcov="iid")
eq3 <- gmmModel(g[[3]], h[[3]], data=simData, vcov="iid")
smod <- merge(eq1,eq2,eq3)
smod

## System GMM Model
## ****************
## Moment type: linear
```

```
## Covariance matrix: iid
## Eqn1: coefs=3, moments=4, number of Endogenous: 1
## Eqn2: coefs=4, moments=6, number of Endogenous: 2
## Eqn3: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
```

We can also add an equation to "smod1".

```
eq1 <- gmmModel(y~x1, ~x1+z4, data=simData, vcov="iid")
merge(smod1, eq1)

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Eqn4: coefs=2, moments=3, number of Endogenous: 0
## Sample size:  50
```

Notice that the equations are merged to the first argument. It the "vcov" differes, the one from the first argument is kept.

2.3   Restricted models

As for the single equation case, we can create an object with restrictions imposed on the coefficients. For now, only linear restrictions on linear models are implemented in the package. The class is "rslinearGmm" and it contains its unrestricted counterpart. Restrictions are imposed in the same way they are imposed in the single equation case. We can impose cross-equation restrictions, or simply impose restrictions equation by equation. The method *restGmmModel* is used to create the restricted models. In the following example, restrictions are imposed equation by equation.

```
R1 <- list(c("x1=-12*z2"), character(), c("x3=0.8", "z1=0.3"))
rsmod1 <- restGmmModel(smod1, R1)
rsmod1

## System GMM Model
## ****************
## Moment type: rlinear
## Covariance matrix: iid
## Supply: coefs=2, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=2, moments=6, number of Endogenous: 0
## Sample size:  50
## **Equation by Equation restrictions**
## **Supply**
## Constraints:
##   x1 + 12z2 = 0
## Restricted regression:
##   y1 = (Intercept)+(-12x1+z2)
##
## **Demand2**
## Constraints:
##   x3 = 0.8
##   z1 = 0.3
## Restricted regression:
##   (y3-0.8x3-0.3z1) = (Intercept)+x4
```

For now, R is a list of the same length as the number of equations. For equations with no restrictions, an empty character vector must be provided. (Eventually, we will allow R to be a named list with the names being the equation names.) For cross-equation restrictions, we need to add to the coefficient names the equation names.

```
R2<- c("Supply.x1=1", "Demand1.x3=Demand2.x3")
rsmod1.ce <- restGmmModel(smod1, R2)
rsmod1.ce

## System GMM Model
## ****************
## Moment type: rlinear
## Covariance matrix: iid
## combinedEqns: coefs=9, moments=16, number of Endogenous: 2
## Sample size:  150
## Constraints:
##   Supply.x1 = 1
##   Demand1.x3 - Demand2.x3 = 0
```

Notice that the model contains only one equation in the print output. That's because we can no longer consider equations to be distinct. All methods that exist for "sGmmModels" can also be applied to "rslinearGmm" objects. When a vector of coefficient is required, the dimension of *theta* must reflect the new number of coefficients implied by the restrictions. For example, in "rsmod1" there are only two coefficients in the restricted supply and demand2 equations.

```
e <- residuals(rsmod1, theta=list(1:2, 1:4, 1:2))
dim(e)

## [1] 50  3
```

Notice that in order to compute the residuals in restricted models, the method converts the restricted coefficients in their unrestricted format and calls the *residuals* method for the unrestricted model. The method *coef* is used to do the conversion. We could therefore reproduce what the method for "rslinearGMM" computes as follows:

```
(b <- coef(rsmod1, theta=list(1:2, 1:4, 1:2)))

## $Supply
## (Intercept)          x1          z2
##           1         -24           2
##
## $Demand1
## (Intercept)          x1          x2          x3
##           1           2           3           4
##
## $Demand2
## (Intercept)          x3          x4          z1
##         1.0         0.8         2.0         0.3

e <- residuals(as(rsmod1, "slinearGmm"), b)
```

The same is done for all methods that can be computed using the converted coefficient vector. These methods include *evalMoment* and *momentVcov*. All derivatives methods, however, reflect the change in the models. For example, *evalDMoment* will produce lists of matrices with different dimensions:

```
evalDMoment(rsmod1, theta=list(1:2,1:4,1:2))[[1]]

##             (Intercept) (-12x1+z2)
## (Intercept)  -1.0000000   59.47345
## z1           -1.0364874   75.33542
```

```
## z2            -1.3914491    70.58468
## z3             0.1586131   -16.66339
```

The method *Dresiduals* will also be affected the same way. Of course, the methods *model.matrix* and *modelResponse* are also affected by the restrictions because the latter modify the left and/or the right hand sides of the equations.

When cross-equation restrictions are imposed, we treat the object as being a system with one equation by providing a list with one single coefficient vector. However, the output of the methods will be the one implied by the system of equations by converting the retricted coefficient vector into its unrestricted counterpart. It is the case of *residuals* and *momentVcov*. For example, the residuals:

```
e <- residuals(rsmod1.ce, theta=list(1:9))
e[1:3,]

##      Supply   Demand1   Demand2
## 1 -1.633415 -22.81319  1.645366
## 2 -9.492418 -79.40969 -3.076222
## 3 -6.127287 -71.84968 10.935640
```

is an $n \times m$ matrix, one column for each equation. As to the case with no cross-equation restriction, the residuals can be computed this way:

```
(b <- coef(rsmod1.ce, theta = list(1:9)))

## $Supply
## (Intercept)          x1          z2
##           1           1           2
##
## $Demand1
## (Intercept)          x1          x2          x3
##           3           4           5           7
##
## $Demand2
## (Intercept)          x3          x4          z1
##           6           7           8           9

e <- residuals(as(rsmod1.ce, "slinearGmm"), b)
```

The methods *evalDMoment*, *Dresiduals*, *model.matrix* and *modelResponse* outputs are, however, lists with ony one element, the combined equations.

```
G <- evalDMoment(rsmod1.ce, list(1:9))
names(G)

## [1] "combinedEqns"

dim(G[[1]])

## [1] 16  9
```

The *[* method works the same way. We can therefore get the first equation as a "rlinearGmm" object as follows:

```
rsmod1[1]

## GMM Model
## *********
## Moment type: rlinear
## Covariance matrix: iid
## Number of regressors: 2
```

```
## Number of moment conditions: 4
## Number of Endogenous Variables: 1
## Sample size:  50
## Constraints:
##   x1 + 12z2 = 0
## Restricted regression:
##  y1 = (Intercept)+(-12x1+z2)
```

2.4   A class for GMM weights

As for the single equation case, the weighting matrices must have a particular class in order to work with all model fitting methods. The constructor is the method *evalWeights*. The class for system of equations is "sysGmmWeights". The simplest weighting matrix is the identity matrix and can be created as follows:

```
wObj1 <- evalWeights(smod1, w="ident")
wObj1

## GMM weights matrix object
## [1] "Identity"
```

The object contains slots with information about the type of moments. When the slot "sameMom" is TRUE, it indicates that all instruments are the same in each equation.

```
wObj1@sameMom

## [1] FALSE
```

This information allows the different methods to treat the weighting matrix in a more efficient way. The other slots are:

```
wObj1@type

## [1] "weights"
```

which also help to choose an efficient way to do operations, and

```
wObj1@eqnNames

## [1] "Supply"  "Demand1" "Demand2"

wObj1@momNames

## [[1]]
## [1] "(Intercept)" "z1"          "z2"          "z3"
##
## [[2]]
## [1] "(Intercept)" "x3"          "z1"          "z2"          "z3"
## [6] "z4"
##
## [[3]]
## [1] "(Intercept)" "x3"          "x4"          "z1"          "z2"
## [6] "z3"
```

There are two slots to store the weighting matrix, "w" and "Sigma". The way it is stored depends on the "vcov" type of the "sysGmmModels" object and on the value of the argument "w" of *evalWeights*. If we provide a fixed matrix, it must be $Q \times Q$:

```r
wObj2 <- evalWeights(smod1, w=diag(16))
```

In that case, "Sigma" is NULL and the slot "w" is equal to the provided weighting matrix. Also, the "type" slot is equal to "weights", which indicates that operations like $G'WG$ will be computed without having to do additional oparations on W. If the argument "w" is set to "optimal", which is the default, the optimal weights matrix is computed based on the slot "vcov" of the model.

If "vcov" is equal to "MDS", we obtain the following.

```r
smod1 <- sysGmmModel(g,h,vcov="MDS", data=simData)
wObj <- evalWeights(smod1, theta=list(1:3,1:4,1:4))
is(wObj@w)

## [1] "qr"

wObj@Sigma

## NULL
```

In that case, there is no benefit of computing $\hat{\Sigma}$. The slot "w" is the QR decomposition of the $n \times Q$ matrix $g(\theta)/\sqrt{n}$ so that $R'R = \hat{S} \equiv \frac{1}{n}\sum_{i=1}^{n} g_i(\theta)g_i'(\theta)$, where R is the upper triangular matrix from the decomposition. Stored this way, it is easy to compute, for example, $G'\hat{S}^{-1}G$.

When "vcov" is set to "iid", the format of the slot "w" depends on whether the instruments are the same across equations or not. In any case, the slot "Sigma" is equal to $\hat{\Sigma}$. When the instruments are not the same, there is no benefit of storing a QR decomposition because it cannot be used to invert the weighting matrix. In that case, the slot "w" is $Z'Z/n$, where Z is a $n \times Q$ matrix that contains all instruments for all equations. If all instruments are the same, "w" is equal to the QR decomposition of the $n \times q_1$ matrix $Z_1/\sqrt{n}$, which facilitates the computation of, for example, $G'WG = G'[\hat{\Sigma}^{-1} \otimes (Z_1'Z_1/n)^{-1}]G$. Also, it is possible to set the "wObj" argument of *evalWeights* to a previously estimated object to avoid recomputing the slot "w". It is particularly usefull in iterative GMM or CUE.

As for the single equation case, any operation $A'WB$ are done using the *quadra* method. We can therefore compute the value of the objective function using the following operation:

```r
gt <- evalMoment(smod1, theta=list(1:3, 1:4, 1:4)) ## this is a list
gbar <- colMeans(do.call(cbind, gt))
obj <- smod1@n*quadra(wObj, gbar)
obj

## [1] 47.87927
```

An easier way to compute the objective function is to use the *evalObjective* method.

```r
evalObjective(smod1, theta=list(1:3,1:4,1:4), wObj=wObj)

## [1] 47.87927
```

2.5 The *solveGmm* method for systems of equations

The method computes the GMM estimates for a given weighting matrix. A two-step GMM can be obtained manually this way:

```r
smod1 <- sysGmmModel(g,h,vcov="MDS", data=simData)
wObj1 <- evalWeights(smod1, w="ident")
theta0 <- solveGmm(smod1, wObj1)$theta
wObj2 <- evalWeights(smod1, theta=theta0)
solveGmm(smod1, wObj2)

## $theta
## $theta$Supply
```

```
## (Intercept)           x1           z2
##  0.56967887  0.90211804 -0.09465356
##
## $theta$Demand1
## (Intercept)           x1           x2           x3
##   1.3965328    1.9181508   -0.1077914   -0.1265638
##
## $theta$Demand2
## (Intercept)           x3           x4           z1
##   3.9449762    0.1213295   -0.2315603   -0.6483119
##
##
## $convergence
## NULL
```

The method also applies to restricted models.

```
R1 <- list(c("x1=-12*z2"), character(), c("x3=0.8", "z1=0.3"))
rsmod1 <- restGmmModel(smod1, R1)
wObj1 <- evalWeights(rsmod1, w="ident")
theta0 <- solveGmm(rsmod1, wObj1)$theta
wObj2 <- evalWeights(rsmod1, theta=theta0)
theta1 <- solveGmm(rsmod1, wObj2)$theta
theta1

## $Supply
## (Intercept)   (-12x1+z2)
##   0.77168040 -0.06976301
##
## $Demand1
## (Intercept)           x1           x2           x3
##   0.98101095  1.96328515 -0.07861969 -0.09507590
##
## $Demand2
## (Intercept)           x4
##   2.8454840   -0.3158484
```

We can recover the values of the coefficients of the original equations using the *coef* method.

```
coef(rsmod1, theta1)

## $Supply
## (Intercept)           x1           z2
##   0.77168040  0.83715613 -0.06976301
##
## $Demand1
## (Intercept)           x1           x2           x3
##   0.98101095  1.96328515 -0.07861969 -0.09507590
##
## $Demand2
## (Intercept)           x3           x4           z1
##   2.8454840    0.8000000   -0.3158484    0.3000000
```

The way we estimate models with cross-equation restrictions, is identical, but the result is a list with one element, all coefficients in a single vector.

```
R2<- c("Supply.x1=1", "Demand1.x3=Demand2.x3")
rsmod1<- restGmmModel(smod1, R2)
wObj1 <- evalWeights(rsmod1, w="ident")
theta0 <- solveGmm(rsmod1, wObj1)$theta
```

```
wObj2 <- evalWeights(rsmod1, theta=theta0)
theta1 <- solveGmm(rsmod1, wObj2)$theta
theta1

## $combinedEqns
##        Supply.Intercept              Supply.z2       Demand1.Intercept
##             -0.15063318             0.04773087              0.65820272
##             Demand1.x1              Demand1.x2       Demand2.Intercept
##              1.97884902            -0.04182288              4.02036885
## (Demand1.x3+Demand2.x3)              Demand2.x4              Demand2.z1
##             -0.05775232            -0.22495580             -0.66896061
```

Again, we can recover the equation by equation coefficients:

```
coef(rsmod1, theta1)

## $Supply
## (Intercept)          x1          z2
## -0.15063318  1.00000000  0.04773087
##
## $Demand1
## (Intercept)          x1          x2          x3
##   0.65820272  1.97884902 -0.04182288 -0.05775232
##
## $Demand2
## (Intercept)          x3          x4          z1
##   4.02036885 -0.05775232 -0.22495580 -0.66896061
```

2.6 The *gmmFit* method for system of equations

This is the main algorithm to obtain GMM estimates of systems of equations. The method returns an
object of class "sgmmfit". The latter has a *show* method that print the essential of the model fit. We
can estimate a system by two step GMM as follows:

```
smod1 <- sysGmmModel(g,h,vcov="MDS", data=simData)
gmmFit(smod1, type="twostep")

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: MDS
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
##
## Estimation: Two-Step GMM
## coefficients:
## Supply:
## (Intercept)          x1          z2
##   0.56967887   0.90211804  -0.09465356
##
## Demand1:
## (Intercept)          x1          x2          x3
##   1.3965328   1.9181508  -0.1077914  -0.1265638
##
## Demand2:
## (Intercept)          x3          x4          z1
##   3.9449762   0.1213295  -0.2315603  -0.6483119
```

38

If "vcov" is "iid" and the instruments differ across equations, we obtain the FIVE estimator (Full-Information Instrumental Variable Efficient).

```
smod1 <- sysGmmModel(g,h,vcov="iid", data=simData)
gmmFit(smod1, type="twostep")

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
##
## Estimation: Full-Information Instrumental Variables Efficient
## coefficients:
## Supply:
##   (Intercept)            x1               z2
##   0.628525314    0.865621252   -0.008753291
##
## Demand1:
## (Intercept)           x1              x2               x3
##   0.55815808    2.00337671   -0.04895639   -0.10295591
##
## Demand2:
## (Intercept)           x3              x4               z1
##    3.6553507     0.0336418    -0.4139044    -0.4141462
```

If "vcov" is "iid", the instruments are the same and first step weights are obtained using an equation by equation 2SLS, it returns the 3SLS estimates.

```
smod1 <- sysGmmModel(g,~z1+z2+z3+z4+z5,vcov="iid", data=simData)
gmmFit(smod1, type="twostep", initW="tsls")

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=6, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 3
## Demand2: coefs=4, moments=6, number of Endogenous: 2
## Sample size:  50
##
## Estimation: Three-Stage Least Squares
## coefficients:
## Supply:
## (Intercept)           x1              z2
##   0.54015729    0.90271025   -0.08044127
##
## Demand1:
##   (Intercept)            x1              x2               x3
## -0.110276952    2.029106112    0.038818975   -0.008191462
##
## Demand2:
## (Intercept)           x3              x4               z1
##    3.7535684     0.7354099    -1.5225474    -0.4584757
```

If, on top of that, the instruments are the union of all regressors, we get the SUR estimates.

```
smod1 <- sysGmmModel(g, vcov="iid", data=simData)
gmmFit(smod1, type="twostep", initW="tsls")

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=7, number of Endogenous: 0
## Demand1: coefs=4, moments=7, number of Endogenous: 0
## Demand2: coefs=4, moments=7, number of Endogenous: 0
## Sample size:  50
##
## Estimation: Seemingly Unrelated Regression
## coefficients:
## Supply:
## (Intercept)            x1             z2
## -0.19348394    1.02109118    0.01528935
##
## Demand1:
## (Intercept)            x1             x2             x3
##   0.54220198    2.02617101   -0.06557172   -0.08227634
##
## Demand2:
## (Intercept)            x3             x4             z1
##   3.66017017    0.03142988   -0.39036984   -0.42055399
```

It is also possible to obtain the first step weighting matrix using the equation by equation efficient GMM estimates

```
smod1 <- sysGmmModel(g,h,vcov="MDS", data=simData)
res <- gmmFit(smod1, type="twostep", initW="EbyE")
```

As for the single equation case, a type "onestep" is a one step with the identity matrix, which is the same as setting the argument "weights" to "ident". If the argument "weights" is set to a matrix or a "sysGmmWeights" object, the method will return a one step GMM with a fixed weighting matrix. Finally, we can obtain the equation by equation estimtes that uses a specific type, initW and weights. In the latter case, it is possible to inform the method that the weighting matrix is optimal by setting the argument "efficientWeights" to TRUE.

Finally, it is possible to obtain an equation by equation GMM estimates. The estimates are obtained using the same argument provided. For example, the following is a two-step efficient equation by equation GMM estimates:

```
gmmFit(smod1, EbyE=TRUE) ## type is 'twostep' by default

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: MDS
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
##
## Estimation: Equation by Equation Two-Step GMM
## coefficients:
## Supply:
## (Intercept)            x1             z2
##   1.0087826     0.8327111    -0.1668389
##
```

```
## Demand1:
## (Intercept)             x1              x2              x3
##   0.13604759    2.00220300     0.01956147   -0.07244969
##
## Demand2:
## (Intercept)             x3              x4              z1
##   3.7030449     0.1483739     -0.1614735    -0.5243800
```

As another example, the following is an equation by equation one-step GMM.

```
res <- gmmFit(smod1,  EbyE=TRUE, weights="ident")
```

Restricted models are estimated in exactly the same way.

```
R1 <- list(c("x1=-12*z2"), character(), c("x3=0.8", "z1=0.3"))
rsmod1 <- restGmmModel(smod1, R1)
gmmFit(rsmod1)@theta

## $Supply
## (Intercept)  (-12x1+z2)
##   0.77168040 -0.06976301
##
## $Demand1
## (Intercept)           x1             x2             x3
##   0.98101095   1.96328515 -0.07861969 -0.09507590
##
## $Demand2
## (Intercept)           x4
##    2.8454840   -0.3158484

R2<- c("Supply.x1=1", "Demand1.x3=Demand2.x3")
rsmod1<- restGmmModel(smod1, R2)
gmmFit(rsmod1)@theta

## $combinedEqns
##        Supply.Intercept                  Supply.z2         Demand1.Intercept
##            -0.15063318                 0.04773087                0.65820272
##              Demand1.x1                 Demand1.x2          Demand2.Intercept
##             1.97884902                -0.04182288                4.02036885
## (Demand1.x3+Demand2.x3)                 Demand2.x4                Demand2.z1
##            -0.05775232                -0.22495580               -0.66896061
```

2.7 The *tsls* and *ThreeSLS* methods

A system of equation can be estimated by 2SLS equation by equation using the *tsls* method.

```
smod1 <- sysGmmModel(g,h,vcov="MDS", data=simData)
res <- tsls(smod1)
res

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: MDS
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
##
```

```
## Estimation: Equation by Equation Two-Stage Least Squares
## coefficients:
## Supply:
## (Intercept)              x1               z2
##    1.0080239      0.8349503      -0.1696884
##
## Demand1:
##   (Intercept)              x1               x2               x3
##   0.172446231    2.013394313     0.005950135    -0.072060387
##
## Demand2:
## (Intercept)              x3               x4               z1
##   3.61876967     0.04268109     -0.44315266    -0.37700899
```

It is also possible to estimate a system of equations using the *ThreeSLS* method. This is only possible if all instruments are the same.

```
smod2 <- sysGmmModel(g,~z1+z2+z3+z4+z5,vcov="MDS", data=simData)
res <- ThreeSLS(smod2)
```

If the instruments are the union of the regressors, the function returns the SUR estimates.

```
smod2 <- sysGmmModel(g,,vcov="MDS", data=simData)
res <- ThreeSLS(smod2)
```

The difference between the 3SLS and SUR using *ThreeSLS* instead of *gmmFit* is that the latter is an efficient GMM, while the former will only be efficient if the "vcov" of the model is "iid". Since the "vcov" of the above model is set to "MDS", the 3SLS and SUR are not efficient GMM estimates. As a result, the covariance matrix of the coefficient estimates will be computed using a sandwich matrix by deafult. If vcov is set to "iid", the following produce identical results.

```
smod2 <- sysGmmModel(g,~z1+z2+z3+z4+z5,vcov="iid", data=simData)
gmmFit(smod2, initW="tsls")@theta

## $Supply
## (Intercept)              x1               z2
##   0.54015729    0.90271025    -0.08044127
##
## $Demand1
##   (Intercept)              x1               x2               x3
## -0.110276952    2.029106112    0.038818975   -0.008191462
##
## $Demand2
## (Intercept)              x3               x4               z1
##    3.7535684     0.7354099     -1.5225474     -0.4584757


ThreeSLS(smod2)@theta

## $Supply
## (Intercept)              x1               z2
##   0.54015729    0.90271025    -0.08044127
##
## $Demand1
##   (Intercept)              x1               x2               x3
## -0.110276952    2.029106112    0.038818975   -0.008191462
##
## $Demand2
## (Intercept)              x3               x4               z1
##    3.7535684     0.7354099     -1.5225474     -0.4584757
```

The *tsls* method returns an object of class "stsls" which inherits from "sgmmfit", and *ThreeSLS* returns an object of class "sgmmfit".

2.8 Methods for "sgmmfit" class objects

- *meatGmm*: It returns the $K \times K$ matrix $G'W\hat{V}WG$, where G is the block diagonal matrix with the jth block being the $q_j \times k_j$ matrix $G_j = \frac{1}{n}\sum_{i=1}^{n} dg_{ji}(\hat{\theta}_j)/d\theta_j$ for $j = 1, ..., m$. As for single equation models, if the argument "robust" is FALSE, it is assumed that $W = V^{-1}$ and it returns $G'\hat{V}^{-1}G/n$, where V is the covariance matrix computed using the final coefficient estimates. If TRUE, is returns $G'W\hat{V}WG$, with $\hat{V}$ computed with the coefficient estimates and W being the weigthing matrix used to get it.

- *bread*: It returns $(G'WG)^{-1}$, where W is the last weights used to compute the final coefficient estimates. If the model is estimated by efficient GMM, the bread is a consistent estimator of the covariance matrix of the coefficients.

- *vcov*: It returns the covariance matrix of the vectorized coefficients. It is therefore $K \times K$. As for single equation, it returns the sandwich matrix $(G'WG)^{-1}G'W\hat{V}WG(G'WG)^{-1}/n$ (or the robust one) if the model was not estimated by efficient GMM, and $(G'\hat{V}^{-1}G)^{-1}/n$ otherwise. Alternatively, it is possible to force *vcov* to return a sandwich matrix by setting the argument "sandwich" to TRUE, or to force it to not be a sandwich by setting the argument to FALSE. It is also possible to change the specification of the model by setting the argument "modelVcov" to another "vcov" type. If different from the fitted model, a sandwich is automatically computed. It is also possible to adjust the covariance matrix for the degrees of freedom by setting the argument "adj.df" to TRUE, which multiplies the covariance matrix by $n/(n-K)$, or to compute only the bread by setting the argument "breadOnly" to TRUE.

- *specTest*: As for single equation, it tests the null hypothesis that $E[g_i(\theta)] = 0$. The degrees of freedom is $Q - K$, where $Q = \sum_{i=1}^{m} q_i$ and $K = \sum_{i=1}^{m} k_i$. It returns an object of class "specTest" which has its own *show* and *print* methods. For the test to be valid, the model must be estimated by efficient GMM. The only signature available for now is ("sgmmfit", "missing"), so we cannot test subsets of the instruments.

```
smod1 <- sysGmmModel(g, h, vcov="iid", data=simData)
res <- gmmFit(smod1)
specTest(res)

##
##   J-Test
##                 Statistics  df      pvalue
## Test E(g)=0:        34.937   5  1.5491e-06
```

- *summary*: Summarizes the estimation results with an equation by equation coefficient matrix, the *specTest* result and an equation by equation first stage F-test. It returns an object of class "summarySysGmm" with its own *show* and *print* methods.

```
summary(res)

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: iid
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
##
## Estimation:  Full-Information Instrumental Variables Efficient
## Sandwich vcov: FALSE
```

43

```
## coefficients:
##
## Supply:
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  0.6285253  0.8913480  0.7051     0.4807
## x1           0.8656213  0.1388769  6.2330 4.576e-10 ***
## z2          -0.0087533  0.1774090 -0.0493     0.9606
##
## Instrument strength based on the F-Statistics of the first stage OLS
## x1 : F( 2 ,  46 ) =  3.761398  (P-Vavue =  0.03069289 )
##
## Demand1:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.558158   0.794222  0.7028   0.4822
## x1           2.003377   0.106229 18.8590   <2e-16 ***
## x2          -0.048956   0.062138 -0.7879   0.4308
## x3          -0.102956   0.148744 -0.6922   0.4888
##
## Instrument strength based on the F-Statistics of the first stage OLS
## x1 : F( 4 ,  44 ) =  3.209718  (P-Vavue =  0.02136437 )
## x2 : F( 4 ,  44 ) =  10.04988  (P-Vavue =  7.243974e-06 )
##
## Demand2:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.655351   0.266694 13.7061  < 2e-16 ***
## x3           0.033642   0.172851  0.1946  0.84568
## x4          -0.413904   0.163711 -2.5283  0.01146 *
## z1          -0.414146   0.189369 -2.1870  0.02874 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##   J-Test
##               Statistics  df      pvalue
## Test E(g)=0:      34.937   5  1.5491e-06
```

The method works also for restricted models.

```
smod1 <- sysGmmModel(g,h,vcov="iid", data=simData)
R1 <- list(c("x1=-12*z2"), character(), c("x3=0.8", "z1=0.3"))
rsmod1 <- restGmmModel(smod1, R1)
summary(gmmFit(rsmod1))@coef

## $Supply
##               Estimate   Std. Error    t value      Pr(>|t|)
## (Intercept)  0.96992584 0.570211480   1.700993 8.894427e-02
## (-12x1+z2)  -0.06787761 0.009262412 -7.328287 2.331139e-13
##
## $Demand1
##                 Estimate Std. Error     t value      Pr(>|t|)
## (Intercept)  0.0890126761 0.74734568   0.11910509 9.051921e-01
## x1           2.0356106953 0.10258712  19.84275096 1.272987e-87
## x2           0.0006860502 0.05863956   0.01169944 9.906654e-01
## x3          -0.0241290806 0.14362398  -0.16800176 8.665819e-01
##
## $Demand2
##               Estimate Std. Error    t value      Pr(>|t|)
## (Intercept)  2.8746969  0.1974970 14.555647 5.377437e-48
## x4          -0.4452847  0.1804412 -2.467755 1.359632e-02

R2<- c("Supply.x1=1", "Demand1.x3=Demand2.x3")
```

```
rsmod1<- restGmmModel(smod1, R2)
summary(gmmFit(rsmod1))@coef

## $combinedEqns
##                           Estimate Std. Error    t value      Pr(>|t|)
## Supply.(Intercept)     -0.315174361 0.23410119 -1.34631676  1.782004e-01
## Supply.z2               0.179626282 0.13646274  1.31630278  1.880724e-01
## Demand1.(Intercept)    -0.549969735 0.68644402 -0.80118658  4.230236e-01
## Demand1.x1              2.107668969 0.09106087 23.14571600 1.605600e-118
## Demand1.x2              0.045496733 0.06357770  0.71560836  4.742332e-01
## Demand2.(Intercept)     3.555754012 0.28066550 12.66900978  8.781592e-37
## (Demand1.x3+Demand2.x3) 0.004203493 0.14013057  0.02999697  9.760695e-01
## Demand2.x4             -0.422872679 0.16449421 -2.57074512  1.014800e-02
## Demand2.z1             -0.315744677 0.20653403 -1.52877799  1.263195e-01
```

- *hypothesisTest*: For hypothesis testing, the method can test any linear restriction using either
  LM, LR or Wald tests. Consider the following unrestricted and restricted models.

```
smod1 <- sysGmmModel(g, h, vcov="MDS", data=simData)
res.u <- gmmFit(smod1)
R1 <- list(c("x1=-12*z2"), character(), c("x3=0.8", "z1=0.3"))
rsmod1 <- restGmmModel(smod1, R1)
res.r <- gmmFit(rsmod1)
```

The methods works as for single equations. We can just provide the unrestricted model and the
R and q to get the Wald test, provide only the restricted fit for the LR test, or provide both and
choose among the three tests by setting the argument "type" to the appropriate value. We only
show the latter case.

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

## Wald Test
## ***********
## The Null Hypothesis:
##    Supply.x1 + 12Supply.z2 = 0
##    Demand2.x3 = 0.8
##    Demand2.z1 = 0.3
## Distribution: Chi-square with 3 degrees of freedom
##    Statistics       Pvalue
## 1    30.84396 9.168458e-07
```

It is as easy to test cross-equation restrictions.

```
R2<- c("Supply.x1=1", "Demand1.x3=Demand2.x3")
rsmod1<- restGmmModel(smod1, R2)
res2.r <- gmmFit(rsmod1)
hypothesisTest(res.u, res2.r, type="LR")

## LR Test
## ***********
## The Null Hypothesis:
##    Supply.x1 = 1
##    Demand1.x3 - Demand2.x3 = 0
## Distribution: Chi-square with 2 degrees of freedom
##    Statistics    Pvalue
## 1    4.226464 0.1208468
```

2.9 Direct estimation with *gmm4*

Again, we can do everything at once using the *gmm4* function. For example, if we want to estimate a model by two-step GMM with "MDS" errors, we proceed as follows:

```
res <- gmm4(g, h, type="twostep", vcov="MDS", data=simData)
res

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: MDS
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
##
## Estimation: Two-Step GMM
## coefficients:
## Supply:
## (Intercept)           x1             z2
##   0.56967887    0.90211804  -0.09465356
##
## Demand1:
## (Intercept)           x1             x2             x3
##   1.3965328     1.9181508    -0.1077914    -0.1265638
##
## Demand2:
## (Intercept)           x3             x4             z1
##   3.9449762     0.1213295    -0.2315603    -0.6483119
```

It produces an object of class "sgmmfit" so all of its methods can be apply to the output. The function *gmm4* recognizes that it is a system because the first argument is a list of formulas. You can estimate it equation by equation by setting the argument "EbyE" to TRUE:

```
res <- gmm4(g, h, type="twostep", vcov="MDS", EbyE=TRUE, data=simData)
res

## System GMM Model
## ****************
## Moment type: linear
## Covariance matrix: MDS
## Supply: coefs=3, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=4, moments=6, number of Endogenous: 0
## Sample size:  50
##
## Estimation: Equation by Equation Two-Step GMM
## coefficients:
## Supply:
## (Intercept)           x1             z2
##   1.0087826     0.8327111    -0.1668389
##
## Demand1:
## (Intercept)           x1             x2             x3
##   0.13604759    2.00220300    0.01956147  -0.07244969
##
## Demand2:
## (Intercept)           x3             x4             z1
##   3.7030449     0.1483739    -0.1614735    -0.5243800
```

To estimate a model by 3SLS, or SUR, we just need the right model:

```r
res <- gmm4(g, ~z1+z2+z3+z4+z5, type="twostep", vcov="iid", initW="tsls", data=simData) #3SLS
res <- gmm4(g, NULL, type="twostep", vcov="iid", initW="tsls", data=simData) #SUR
```

To estimate a restricted model, simply add the restrictions

```r
R1 <- list(c("x1=-12*z2"), character(), c("x3=0.8", "z1=0.3"))
res <- gmm4(g, h, data=simData, cstLHS=R1) #two-step by default
res

## System GMM Model
## ****************
## Moment type: rlinear
## Covariance matrix: MDS
## Supply: coefs=2, moments=4, number of Endogenous: 1
## Demand1: coefs=4, moments=6, number of Endogenous: 2
## Demand2: coefs=2, moments=6, number of Endogenous: 0
## Sample size:  50
## **Equation by Equation restrictions**
## **Supply**
## Constraints:
##   x1 + 12z2 = 0
## Restricted regression:
##  y1 = (Intercept)+(-12x1+z2)
##
## **Demand2**
## Constraints:
##   x3 = 0.8
##   z1 = 0.3
## Restricted regression:
##  (y3-0.8x3-0.3z1) = (Intercept)+x4
##
##
## Estimation: Two-Step GMM
## coefficients:
## Supply:
## (Intercept)   (-12x1+z2)
##  0.77168040  -0.06976301
##
## Demand1:
## (Intercept)          x1             x2             x3
##  0.98101095   1.96328515   -0.07861969   -0.09507590
##
## Demand2:
## (Intercept)          x4
##  2.8454840   -0.3158484
```

2.10 Textbooks Applications

2.10.1 Greene

In this section, we kind of reproduce results from Greene (2012). Textbook with GMM estimation of systems of equations are not common.

In Table 10.3, the author estimates the following system of equations:

$$s_k = \beta_k + \delta_{kk} \log\left(\frac{p_k}{p_m}\right) + \delta_{kl} \log\left(\frac{p_l}{p_m}\right) + \delta_{ke} \log\left(\frac{p_e}{p_m}\right) + u_k$$

$$s_l = \beta_l + \delta_{lk} \log\left(\frac{p_k}{p_m}\right) + \delta_{ll} \log\left(\frac{p_l}{p_m}\right) + \delta_{le} \log\left(\frac{p_e}{p_m}\right) + u_l$$

$$s_e = \beta_e + \delta_{ek} \log\left(\frac{p_k}{p_m}\right) + \delta_{el} \log\left(\frac{p_l}{p_m}\right) + \delta_{ee} \log\left(\frac{p_e}{p_m}\right) + u_e$$

where k, l, e and m stand for capital, labor, energy and materials. The dependent variables are shares and the regressors are prices. The equation for materials is omitted because it is equal to one minus the sum of the other three. The system with all four would be singular. Without any restriction, this is just a multivariate regression in which all equation are just identified. Any GMM estimation would therefore be identical to OLS. If we impose restrictions on the coefficients, however, some equations become over-identified and GMM deviates from OLS. In particular, if we assume iid errors, efficient GMM becomes SUR since all regressors are considered exogenous. It turns out that the theory behind the model implies that $\delta_{kl} = \delta_{lk}$, $\delta_{le} = \delta_{el}$ and $\delta_{ke} = \delta_{ek}$. SUR estimation of the restricted model should lead to results that are close to Table 10.3 which were generated by restricted feasible GLS.

First, we normalize the prices and take the log.

```
data(ManufactCost)
price <- c("Pk","Pl","Pe")
ManufactCost[,price] <- log(ManufactCost[,price]/ManufactCost$Pm)
```

In the dataset, the shares are labeled K, L and E. The unrestricted model can be defined as follows.

```
g <- list(Sk=K~Pk+Pl+Pe,
          Sl=L~Pk+Pl+Pe,
          Se=E~Pk+Pl+Pe)
mod <- sysGmmModel(g, NULL, data=ManufactCost, vcov="iid")
```

Notice that the second argument is NULL because we want the instruments to be the regressors. We can now create the restricted model by adding the equation names to each coefficient names.

```
R <- c("Sk.Pl=Sl.Pk", "Sk.Pe=Se.Pk", "Sl.Pe=Se.Pl")
rmod <- restGmmModel(mod, R=R)
```

We can then estimate the model and print the coefficient matrix:

```
res <- gmmFit(rmod)
summary(res)@coef

## $combinedEqns
##                    Estimate    Std. Error       t value      Pr(>|t|)
## Sk.(Intercept)   5.681333e-02 0.0013276579  42.79214499 0.000000e+00
## Sk.Pk            2.975632e-02 0.0057775474   5.15033696 2.600189e-07
## Sl.(Intercept)   2.535064e-01 0.0020668406 122.65406201 0.000000e+00
## (Sk.Pl+Sl.Pk)   -7.430699e-06 0.0037726370  -0.00196963 9.984285e-01
## Sl.Pl            7.496081e-02 0.0066683728  11.24124493 2.557500e-29
## Se.(Intercept)   4.378201e-02 0.0008438713  51.88233158 0.000000e+00
## (Sk.Pe+Se.Pk)   -8.089997e-03 0.0033890092  -2.38712739 1.698061e-02
## (Sl.Pe+Se.Pl)   -3.152070e-03 0.0022537698  -1.39857691 1.619399e-01
## Se.Pe            3.057416e-02 0.0053117169   5.75598354 8.613878e-09
```

We can also test the restriction:

```
res.u <- gmmFit(mod)
hypothesisTest(res.u, res)

## Wald Test
## ***********
## The Null Hypothesis:
##    Sk.Pl - Sl.Pk = 0
##    Sk.Pe - Se.Pk = 0
##    Sl.Pe - Se.Pl = 0
## Distribution: Chi-square with 3 degrees of freedom
##    Statistics        Pvalue
## 1   16.56175 0.0008696287
```

In Table 10.5, the author estimates the macro model of Klein (1950):

$$
\begin{aligned}
C_t &= \theta_{c0} + \theta_{c1}P_t + \theta_{c2}P_{t-1} + \theta_{c3}(W_t^p + W_t^g) + \varepsilon_{ct} \\
I_t &= \theta_{i0} + \theta_{i1}P_t + \theta_{i2}P_{t-1} + \theta_{i3}K_{t-1} + \varepsilon_{it} \\
W_t^D &= \theta_{w0} + \theta_{w1}X_t + \theta_{w2}X_{t-1} + \theta_{w3}A_t + \varepsilon_{wt}
\end{aligned}
$$

The exogenous and predetermined variables that are used as instruments in each equation are $Z_t = \{G_t, T_t, W_t^g, A_t, K_{t-1}, P_{t-1}, X_{t-1}\}$. The data are annual observations from 1920 to 1941. We therefore have only 22 observations (21 because of the lags).

The table reports the results for many estimation method. We can reproduce 2SLS and 3SLS, but we only consider the latter because we have covered 2SLS cases in Section 1.10. First we arrange the data to get lags and A_t.

```
data(Klein)
Klein1 <- Klein[-22,]
Klein <- Klein[-1,]
dimnames(Klein1) <- list(rownames(Klein), paste(colnames(Klein),"1",sep=""))
Klein <- cbind(Klein, Klein1)
Klein$A <- (Klein$YEAR-1931)
```

We can then estimate it by 3SLS. To reproduce the same standard errors, we need to use the bread only. In other words, using the final estimate to compute the weights leads to slightly different standard errors. In other words, the covariance matrix must be estimated using

$$
\left[G'\left(\tilde{\Sigma}^{-1} \otimes (Z'Z/n)^{-1}\right)G\right]^{-1}/n,
$$

where $\tilde{\Sigma}$ is computed using the 2SLS estimates. By default, the package updates $\tilde{\Sigma}$ using the final estimates. The following is identical to Table 10.5, section 3SLS of Greene (2012).

```
g <- list(C=C~P+P1+I(WP+WG),
          I=I~P+P1+K1,
          Wp=WP~X+X1+A)
h <- ~G+T+WG+A+K1+P1+X1
res <- ThreeSLS(g, h, vcov="iid", data=Klein)
summary(res, breadOnly=TRUE)@coef

## $C
##                Estimate Std. Error    t value      Pr(>|t|)
## (Intercept) 16.4407901 1.30454876  12.602664 2.041306e-36
## P            0.1248905 0.10812905   1.155013 2.480850e-01
## P1           0.1631441 0.10043819   1.624323 1.043068e-01
## I(WP + WG)   0.7900809 0.03793791  20.825634 2.535479e-96
##
## $I
```

```
##                 Estimate Std. Error     t value       Pr(>|t|)
## (Intercept) 28.17784687 6.79377017   4.14760084 3.359775e-05
## P           -0.01307918 0.16189624  -0.08078744 9.356110e-01
## P1           0.75572396 0.15293313   4.94153209 7.751106e-07
## K1          -0.19484825 0.03253069  -5.98967376 2.102624e-09
##
## $Wp
##                 Estimate Std. Error    t value      Pr(>|t|)
## (Intercept) 1.7972177 1.11585498   1.610619 1.072627e-01
## X           0.4004919 0.03181341 12.588774 2.434243e-36
## X1          0.1812910 0.03415878   5.307304 1.112584e-07
## A           0.1496741 0.02793524   5.357897 8.419628e-08
```

References

W. H. Greene. *Econometric Analysis, 7th edition.* Prentice Hall, 2012.

A. R. Hall. *Generalized Method of Moments (Advanced Texts in Econometrics).* Oxford University Press, 2005.

L. P. Hansen. Large sample properties of generalized method of moments estimators. *Econometrica*, 50:1029–1054, 1982.

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

Philip Leifeld. texreg: Conversion of statistical model output in R to LaTeX and HTML tables. *Journal of Statistical Software*, 55(8):1–24, 2013. URL http://www.jstatsoft.org/v55/i08/.

J.H. Stock and M.W. Watson. *Introduction to Econometrics.* Pearson, 2015.

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

Appendix

A Some extra codes

A.1 The *Extract* method

```r
library(texreg)
setMethod("extract", "gmmfit",
        function(model, includeJTest=TRUE, includeFTest=TRUE, ...)
            {
                s <- summary(model, ...)
                spec <- modelDims(model@model)
                coefs <- s@coef
                names <- rownames(coefs)
                coef <- coefs[, 1]
                se <- coefs[, 2]
                pval <- coefs[, 4]
                n <- model@model@n
                gof <- numeric()
                gof.names <- character()
                gof.decimal <- logical()
                if (includeJTest) {
                    if (spec$k == spec$q)
                        {
                            obj.fcn <- NA
                            obj.pv <- NA
                        } else {
                            obj.fcn <- s@specTest@test[1]
                            obj.pv <- s@specTest@test[3]
                        }
                    gof <- c(gof, obj.fcn, obj.pv)
                    gof.names <- c(gof.names, "J-test Statistics", "J-test p-value")
                    gof.decimal <- c(gof.decimal, TRUE, TRUE)
                }
                if (includeFTest) {
                    str <- s@strength$strength
                    if (is.null(str))
                        {
                            gof <- c(gof, NA)
                            gof.names <- c(gof.names, "First Stage F-stats")
                            gof.decimal <- c(gof.decimal, TRUE)
                        } else {
                            for (i in 1:nrow(str))
                                {
                                    gof <- c(gof, str[i,1])
                                    gofn <- paste("First Stage F-stats(",
                                                    rownames(str)[i], ")", sep="")
                                    gof.names <- c(gof.names, gofn)
                                    gof.decimal <- c(gof.decimal, TRUE)
                                }
                        }
                }
                tr <- createTexreg(coef.names = names, coef = coef, se = se,
                                pvalues = pval, gof.names = gof.names, gof = gof,
                                gof.decimal = gof.decimal)
                return(tr)
```

```
})
```