Instrumental Variables Tools for the Case of Weak or Many Instruments

Pierre Chausse*

Abstract

This vignette explains the different tools included in the package to deal with the weak or the many instruments problem. For example, it presents estimation methods like the LIML or the Fuller method and some improved inference methods for TSLS and GMM. It is in early stage of development, so comments and recommendations are welcomed.

This document is incomplete. It will improve as we add more functionalities to the package.

1 Weak Instruments (For later use)

The following function is used to generate dataset with k instruments and different level of strength. The DGP is

$$y_1 = \beta y_2 + u$$
$$y_2 = \pi' Z + e\,,$$

where $Z \in \mathbb{R}^k$, $\text{Var}(u) = \text{Var}(e) = 1$, $\text{Cor}(e,u) = \rho$, $\pi_i = \eta$ for all $i = 1, ..., k$ and $Z \sim N(0, I)$. The R^2 of the first stage regression is therefore equal to

$$R^2 = \frac{k\eta^2}{k\eta^2 + 1}\,,$$

which implies

$$\eta = \sqrt{\frac{R^2}{k(1-R^2)}}$$

We can therefore set R^2 and k and let the function get η.

```
getIVDat <- function(n, R2, k, rho, b0=0)
{
    eta <- sqrt(R2/(k*(1-R2)))
    Z <- sapply(1:k, function(i) rnorm(n))
    sigma <- chol(matrix(c(1,rho,rho,1),2,2))
    err <- cbind(rnorm(n), rnorm(n))%*%sigma
    y2 <- rowSums(Z)*eta+err[,2]
    y1 <- b0*y2 + err[,1]
    dat <- data.frame(y1=y1, y2=y2, u=err[,1], e=err[,2])
    for (i in 1:k) dat[[paste("Z",i,sep="")]] <- Z[,i]
```

*University of Waterloo, pchausse@uwaterloo.ca

```
    dat
}
```

```
library(momentfit)
```

```
## Loading required package: sandwich
```

```
set.seed(112233)
k <- 10
rho <- .3
R2 <- .001
g <- y1~y2
n <- 500
h <- reformulate(paste("Z", 1:k, sep=""))
dat <- getIVDat(n, R2, k, rho)
m <- momentModel(g, h, data=dat, vcov="MDS")
```

2 K-class Estimator and LIML

The package `ivmodel` implements many useful methods including the K-class estimators. Let's see if we can borrow some of their codes.

2.1 Computing $\hat{\kappa}$

```
library(ivmodel)
data(card.data)
## from the ivmodel examples
Z <- card.data[,c("nearc4","nearc2")]
Y <- card.data[,"lwage"]
D <- card.data[,"educ"]
Xname <- c("exper", "expersq", "black", "south", "smsa", "reg661",
           "reg662", "reg663", "reg664", "reg665", "reg666", "reg667",
           "reg668", "smsa66")
X <- card.data[,Xname]
mod <- ivmodel(Y=Y,D=D,Z=Z,X=X)
```

To get $\hat{\kappa}$ for LIML and the modified LIML of Fuller (1977), the package only provides an option for the case of one endogenous regressor. We can easily extend it to more general models using `linearModel` objects from the `momentfit` package. The above model replicated as follows:

```
g <- reformulate(c("educ", Xname), "lwage")
h <- reformulate(c(c("nearc4","nearc2"), Xname))
mod2 <- momentModel(g, h, data=card.data)
```

I use the default `vcov="iid"` for now. We'll discuss inference later. The `getK` function generates $\hat{\kappa}$ for the original LIML and the modified one. No effort is done to make it efficient for now. The modified LIML is $\hat{\kappa} - \alpha/(n-k)$, where k is the number of exogenous variables (included and excluded).

We can compare the values with the ones computed by `ivmodel`. They are identical:

```
getK(mod2)
```

```
##     LIML   Fuller
## 1.000409 1.000075
```

```
mod$Fuller$k
```

```
## [1] 1.000075
```

```
mod$LIML$k
```

```
## [1] 1.000409
```

We can also have more than one endogenous regressor. For this model, we can interact `educ` with, say, `exper`, which is like having a second endogenous variable. The package can recognize that `educ:exper` is endogenous because it is not part of the set of instruments.

```
g2 <- reformulate(c("educ", "educ:exper", Xname), "lwage")
h2 <- reformulate(c(c("nearc4","nearc2", "nearc2:exper", "nearc4:exper"), Xname))
mod3 <- momentModel(g2, h2, data=card.data)
getK(mod3)
```

```
##     LIML   Fuller
## 1.000702 1.000368
```

For just-identified models, $\hat{\kappa} = 1$. The `getK` function does check the number of overidentifying restrictions before computing $\hat{\kappa}$. What happens in `ivmodel` (it works)?

```
Z <- card.data[,c("nearc2")]
mod4 <- ivmodel(Y=Y,D=D,Z=Z,X=X)
mod4$LIML$k
```

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

2.2 Computing the K-Class estimators

A K-Class estimator is the solution to

$$X'(I - kM_w)(y - X\beta) = 0\,,$$

where $M_w = I - P_w$, P_w is the projection matrix for W, the matrix of exogenous variables (included and excluded). It is therefore a just-identified IV with the instrument $Z = (I - kM_w)X$ (because M_w is symmetric). Note that if $X = \{X_1, X_2\}$ with X_1 being the matrix of included exogenous variables, $M_w X_1 = 0$, so that

$$Z = \begin{pmatrix}(I - kM_w)X_1 & (I - kM_w)X_2\end{pmatrix} = \begin{pmatrix}X_1 & (I - kM_w)X_2\end{pmatrix}$$

We can easily compute the instruments since $M_w X_2$ is the matrix of residuals from the first stage regression. Let $U_2 = M_w X_2$, then the instruments are:

$$Z = \begin{pmatrix}X_1 & (X_2 - kU_2)\end{pmatrix}$$

We can compute the standard errors using the asymptotic properties of just identified IV. In the case of iid errors (no heteroskedasticity), the variance can be estimated as:

$$\hat{\Sigma}_{iid} = \hat{\sigma}^2 (Z'X)^{-1} Z'Z (X'Z)^{-1}\,,$$

where $\hat{\sigma}^2$ is the estimated variance of the residuals. Since $\hat{\kappa}$ converges to 1 as n goes to infinity, $Z'Z \approx Z'X \equiv X'Z$, the latter being true only for this specific matrix of instruments, for large enough `n`, so we could estimate it as

$$\hat{\Sigma}_{iid} = \hat{\sigma}^2 (Z'X)^{-1} .$$

However, I choose to keep the first version and treat the method as a general just-identified estimation. This allows me to use the tools included in the package for inference. In the case of `MDS`, the standard errors are based on the following expression:

$$\hat{\Sigma}_{HC} = (Z'X)^{-1} \hat{\Omega}_{HC} (X'Z)^{-1} ,$$

where $\hat{\Omega}_{HC}$ is an HCCM estimator of the variance of $Z'u$. The K-Class estimators have two special cases. It is OLS when $\kappa = 0$ and two-stage least squares (TSLS) when $\kappa = 1$. The main `kclassfit` function uses the `tsls` method when `k=1` and `lm` if `k=0` (should we use `all.equal` instead? what if it is almost 1 or 0?). For the latter, we need a method that returns LS estimates and that can be directly applied to `linearModel` objects. The `lse` method returns an `lsefit` object that contains the `lm` object from the estimation:

```
lse(mod2)
```

```
## Model based on moment conditions
## *********************************
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 16
## Number of moment conditions: 17
## Number of Endogenous Variables: 1
## Sample size:  3010
##
## Estimation: Least Squares
##
## Coefficients:
## (Intercept)         educ        exper      expersq        black        south
##   4.7393766    0.0746933    0.0848320   -0.0022870   -0.1990123   -0.1479550
##        smsa       reg661       reg662       reg663       reg664       reg665
##   0.1363845   -0.1185698   -0.0222026    0.0259703   -0.0634942    0.0094551
##      reg666       reg667       reg668       smsa66
##   0.0219476   -0.0005887   -0.1750058    0.0262417
```

The function 'kclassfit' computes the K-Class estimator. For now, it is a function that can only be applied to linear models. The function returns an object of class `kclassfit`, which contains a `gmmfit` class object. The additional slots are used to store the method, κ and the original model. The function generates the matrix of instruments $Z = (I - kM_w)X$, use it to create a just-identified linear model and estimate the new model using `gmmFit`. If `k` is missing, it is computed for either the LIML or Fuller method.

```
(liml <- kclassfit(mod2))
```

```
## Model based on moment conditions
## *********************************
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 16
## Number of moment conditions: 17
## Number of Endogenous Variables: 1
## Sample size:  3010
##
## Estimation: LIML (k = 1.000409)
## coefficients:
##  (Intercept)          educ         exper       expersq         black
```

```
##  3.221269443   0.164027756   0.121689917  -0.002362359  -0.116870463
##        south          smsa        reg661        reg662        reg663
## -0.142791708   0.097738480  -0.101656724   0.001630403   0.048731041
##       reg664        reg665        reg666        reg667        reg668
## -0.054724308   0.055061606   0.074061888   0.042413909  -0.199985585
##       smsa66
##  0.014116798
```

```
(fuller <- kclassfit(mod2, type="Fuller"))
```

```
## Model based on moment conditions
## *********************************
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 16
## Number of moment conditions: 17
## Number of Endogenous Variables: 1
## Sample size:  3010
##
## Estimation: Fuller (k = 1.000075)
## coefficients:
##   (Intercept)          educ          exper        expersq          black
##  3.319304e+00  1.582588e-01   1.193098e-01  -2.357495e-03  -1.221749e-01
##         south          smsa         reg661         reg662         reg663
## -1.431251e-01  1.002341e-01  -1.027489e-01   9.134797e-05   4.726123e-02
##        reg664         reg665         reg666         reg667         reg668
## -5.529064e-02  5.211649e-02   7.069652e-02   3.963694e-02  -1.983725e-01
##        smsa66
##  1.489978e-02
```

We see that the LIML and Fuller estimates I get are identical to the ones from the `ivmodel` package.

```
print(mod$LIML$point.est,digits=10)
```

```
##          Estimate
## [1,] 0.1640277561
```

```
print(coef(liml)[2], digits=10)
```

```
##         educ
## 0.1640277561
```

```
print(mod$Fuller$point.est,digits=10)
```

```
##          Estimate
## [1,] 0.1582588323
```

```
print(coef(fuller)[2], digits=10)
```

```
##         educ
## 0.1582588323
```

2.3 Inference

Since the `kclassfit` object contains a just-identified `gmmfit` object, we can do inference as if it was an IV. The `summary` method for `kclassfit` objects is in fact the same as for `gmmfit` objects, but it contains additional information about the original model and the method. It returns an object of class `summaryKclass`.

```
summary(liml)
```

```
## Model based on moment conditions
## *********************************
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 16
## Number of moment conditions: 16
## Number of Endogenous Variables: 1
## Sample size:  3010
##
## Estimation: LIML (k = 1.00040942731651)
## Sandwich vcov: TRUE
## coefficients:
##                Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)  3.22126944  0.98048104  3.2854 0.0010184 **
## educ         0.16402776  0.05763981  2.8457 0.0044309 **
## exper        0.12168992  0.02482322  4.9023 9.474e-07 ***
## expersq     -0.00236236  0.00035189 -6.7133 1.903e-11 ***
## black       -0.11687046  0.05656732 -2.0660 0.0388245 *
## south       -0.14279171  0.02879080 -4.9596 7.063e-07 ***
## smsa         0.09773848  0.03329490  2.9355 0.0033297 **
## reg661      -0.10165672  0.04410858 -2.3047 0.0211838 *
## reg662       0.00163040  0.03468374  0.0470 0.9625071
## reg663       0.04873104  0.03349713  1.4548 0.1457294
## reg664      -0.05472431  0.03968009 -1.3791 0.1678523
## reg665       0.05506161  0.04942349  1.1141 0.2652459
## reg666       0.07406189  0.05544273  1.3358 0.1816059
## reg667       0.04241391  0.05143408  0.8246 0.4095836
## reg668      -0.19998559  0.05348458 -3.7391 0.0001847 ***
## smsa66       0.01411680  0.02278641  0.6195 0.5355691
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##  Anderson and Rubin
##                 Statistics  df   pvalue
## Test E(g)=0:        1.2321   1  0.26699
##
##
## Instrument strength based on the F-Statistics of the first stage OLS
## educ : F( 1 ,  2994 ) =  13.42398  (P-Vavue =  0.0002527353 )
```

Note that the specification test is based on Anderson and Rubin. It is a likelihood ration test equal to $n \log(\hat{\kappa})$ and is distributed as a chi-square with the degrees of freedom equal to the number of over-identifying restrictions. It calls the `specTest` method for `kclassfit` objects:

```
specTest(liml)
```

```
##
##  Anderson and Rubin
##                 Statistics  df   pvalue
## Test E(g)=0:        1.2321   1  0.26699
```