Learning Information Theory

Dan Hammer

May 30, 2012

Introduction

The idea behind this interactive, org-mode document is to track the learning of information theory. I will sound increasingly knowledgeable about information theory. For now, though, I probably sound like an idiot. But I intend to learn.

My understanding of information theory is limited to the use of techniques to recover information from noisy or indirect data. There are degrees of parameterization required for estimation; some techniques require more levels of assumptions about the distributions and parameters than others. For example, maximum likelihood estimation requires that the distribution be specified, which is then parameterized to reflect the data. Generalized method of moments estimation, however, does not require a complete distribution to be specified; only certain moments are needed for GMM estimation. First, as a background, I will review GMM theory, since underidentified models serve as a basis for much of Judge's recent book *An Information Theoretic Approach to Econometrics*.

Method of Moments

Generalized Method of Moments

We first construct an example to apply GMM, based on Max Auffhammer's 2012 final exam. Assume the following data generating process:

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \eta_i \tag{1}$$

Let the true value for the parameter vector β be [1, 1, 2, -4, 1]. We generate the sample of x_{1i}, x_{1i}, x_{2i}, x_{3i}, x_{4i}, z_{1i}, and z_{2i} by drawing outcomes from a joint normal distribution with mean zero for all variables and covariance matrix

$$\Sigma = \begin{bmatrix} 1 & a & 0 & 0 & 0 \\ a & 1 & b & c & d \\ 0 & b & 1 & 0 & 0 \\ 0 & c & 0 & 1 & e \\ 0 & d & 0 & e & 1 \end{bmatrix} \tag{2}$$

The following function builds a 5×5 covariance matrix.

A random $n \times k$ multivariate normal $\mathbf{X}$ can be generated quickly in two steps. First, generate a matrix $\mathbf{Z} = [z_1, z_2, \ldots, z_k]$, where each z_j is distributed standard normal. (I take a shortcut, here, and rely on the `R` function `rnorm()`.) In general, $\mathbf{X} = \mathbf{ZQ} + \mathbf{J}\mu'$, where $\mathbf{Q}'\mathbf{Q} = \Sigma$, a covariance matrix which is assumed to be symmetric and positive definite; $\mathbf{J}$ is a column vector of ones; and μ is a vector of k means. We can take advantage of the constraints within the problem – most notably that the variables all have zero mean – and employ the following function to generate $\mathbf{X}$, which is used at the top of the for-loop in the function `est.bias`:

```
vcov.fn <- function(rho.24, rho.34, rho.3z) {
  mat <- diag(7)
  mat[3,1] <- rho.24; mat[1,3] <- rho.2t4
  mat[2,3] <- rho.34; mat[3,2] <- rho.34
  mat[2,4] <- rho.3z; mat[4,2] <- rho.3z
```

```
  return(mat)
}

rmvn.chol <- function(n, vcov.mat) {
  k <- ncol(vcov.mat)
  Q <- chol(vcov.mat)
  Z <- matrix(rnorm(k*n), nrow=n, ncol=k)
  return(Z %*% Q)
}
```

There are many different ways to decompose Σ. I used the built in function for Cholesky decomposition, but you could also use singular value or spectral decomposition. I took a shortcut here, too, by using the built-in function for decomposition, but it seems like this wasn't exactly the point of the problem. We can check the function by making sure that the variance of each variable is approximately equal to one.

```
vcov <- vcov.fn(0, 0.5, 0.5)
X <- rmvn.chol(500, vcov)
print(apply(X, 2, function(i){var(i)}))
```

```
 Error in vcov.fn(0, 0.5, 0.5) : object 'rho.2t4' not found
 Error in chol.default(vcov.mat) :
   the leading minor of order 2 is not positive definite
 (Intercept)        d$z1        d$z2        d$z3        d$z4        d$z5
   0.0000000   1.3434784   1.0503100   0.5734663   0.8291680   1.4201850
```

```
library(gmm)

n = 10
k = 1
l = 2
d =  matrix(rnorm( (1+k+l)*n ), ncol = 1 + k + l)
colnames(d) = c("y", "x1", "z1", "z2", "z3", "z4", "z5")

d = d + 3

d[, 2:(1+k)]         = 2 + rowSums( d[, (1+k+1):(1+k+l)] )  + rnorm(n, 0, 0.2)
d[, 1] = 2 +  d[, (1 + 1):(1+k  )]   + rnorm(n, 0, 0.2)

fs = d$y ~ d$x1
fi =     ~ d$z1 + d$z2 + d$z3 + d$z4 + d$z5

###### result from package gmm #################

### good gmm #################
source("gmm_linear.R")
gmm.linear(list(structural = fs, instruments = fi), data = d)


### big code ###
n <- 10000
x <- rnorm(n)
z1 <- 3 + 0.50*x + rnorm(n, 0, 0.1)
z2 <- 1 + 0.25*x + rnorm(n, 0, 0.1)
y <- 2 + 3*x + 4*z1 + 5*z2 + rnorm(n)

X <- cbind(1, x)
Z <- cbind(1, z1, z2)
W = function(G) solve( (1/n) * (t(G)%*%G) )
```

```
A = function(WW) ((1/n) * t(X) %*% Z) %*% WW %*% ((1/n) * t(Z) %*% X )
B = function(WW) ((1/n) * t(X) %*% Z) %*% WW %*% ((1/n) * t(Z) %*% y )
betaHat = function(WW) {
  AA = A(WW)
  BB = B(WW)
  b = solve(AA, BB)
  return(b)
}

eHat = function(betaHat) y - X %*% betaHat # the residual

### 1st stage: W = (X'X)^-1
W1 = W(Z)
betaHat1 = betaHat(W1)

### 2nd stage: W = (G G' - g g')^-1
e = eHat(betaHat1)
G2 = Z * matrix(rep(e, 3), ncol = 3)
G2Bar = colMeans(G2)
W2 = solve( (t(G2)%*%G2)/n   -  G2Bar %*% t(G2Bar)  )
betaHat2 = betaHat(W2)

### variance-covariance matrix
Q = crossprod(Z, X)/n
V = solve( t(Q) %*%  W2 %*% Q )
s = sqrt(diag(V)/n)

summ = cbind(betaHat2, s, betaHat2/s, 2*(1-pnorm(abs(betaHat2/s))) )
colnames(summ) = c("coefficient", "s.d.", "Z", "p-value")
print(summ)

data = d
formula = list(structural = fs, instruments = fi)
data1 = model.frame(formula$structural, data)
data2 = model.frame(formula$instruments, data)

##### organize the data ############

# y: endogenous variable, (n * 1) vector
# Z: explanatory variable, (n * k) matrix
# X: instruments, (n*l) matrix

y = model.response(data1, type = "numeric")

Z = model.matrix(formula$structural, data = data1)
X = model.matrix(formula$instrument, data = data2)

l = ncol(X)
k = ncol(Z)

if (k >l ) stop( "# of regressors must <= # of instruments" )

############### start gmm computation #############

## prepare the functions
```

```
## J stat and test ###
J = n * t(G2Bar) %*% W2 %*% G2Bar
cat("\n J-stat = ", J,
    ". DF = ", l-k,
    ". p-value = ", 1-pchisq(J, df = l-k), ".\n")

 Error in `colnames<-`(`*tmp*`, value = c("y", "x1", "z1", "z2", "z3",  :
  length of 'dimnames' [2] not equal to array extent
 Error in model.frame.default(formula$structural, data) :
  'data' must be a data.frame, not a matrix or an array
  coefficient       s.d.         Z p-value
    19.007249 0.01187528 1600.5729       0
x    6.346382 0.01199629  529.0286       0
 Error in model.frame.default(formula$structural, data) :
  'data' must be a data.frame, not a matrix or an array
Error in model.frame.default(formula$instruments, data) :
  'data' must be a data.frame, not a matrix or an array

 J-stat =  513.7977 . DF =  4 . p-value =  0 .
```