Estimating Models by The Generalized Method of Moments

Elad Guttman

Outline

Working with panel data

`dplyr` basic verbs

- `mutate()` - adds new variables that are functions of existing variables
- `select()` - picks variables based on their names.
- `filter()` - picks cases based on their values.
- `summarise()` - reduces multiple values down to a single summary.
- `arrange()` - changes the ordering of the rows.

Leads and Lags

Another two useful `dplyr` functions for working with panel data are the `lead` and `lag` functions:

- `lag` - Find the "previous" values in a vector.
- `lead` - Find the "next" values in a vector.

```
library(tidyverse)
x = 1:10
lag(x, n = 1)
```

```
##  [1] NA  1  2  3  4  5  6  7  8  9
```

```
lag(x, n = 2)
```

```
##  [1] NA NA  1  2  3  4  5  6  7  8
```

```
lead(x, n = 1)
```

```
##  [1]  2  3  4  5  6  7  8  9 10 NA
```

Combining all together

Let's combine it all together to calculate the daily log return ($r_t = log(\frac{p_t}{p_{t-1}})$) on several stock prices:

```
library(tidyquant)
stocks = c("AAPL", "NFLX", "AMZN")
prices = tq_get(stocks,
                from = "2020-01-01",
                to = "2020-03-01",
                get = "stock.prices")
head(prices, 4)
```

```
## # A tibble: 4 x 8
##   symbol date        open  high   low close    volume adjusted
##   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>     <dbl>    <dbl>
## 1 AAPL   2020-01-02  74.1  75.2  73.8  75.1 135480400     74.4
## 2 AAPL   2020-01-03  74.3  75.1  74.1  74.4 146322800     73.7
## 3 AAPL   2020-01-06  73.4  75.0  73.2  74.9 118387200     74.3
## 4 AAPL   2020-01-07  75.0  75.2  74.4  74.6 108872000     74.0
```

Combining all together

```
prices = prices %>%
  arrange(symbol, date) %>%
  group_by(symbol) %>%
  mutate(log_return = log(open/lag(open))) %>%
  ungroup()

head(prices %>% filter(symbol == "AAPL"), 3)
```

```
## # A tibble: 3 x 9
##   symbol date        open  high   low close    volume adjusted log_return
##   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>     <dbl>    <dbl>      <dbl>
## 1 AAPL   2020-01-02  74.1  75.2  73.8  75.1 135480400     74.4   NA
## 2 AAPL   2020-01-03  74.3  75.1  74.1  74.4 146322800     73.7    0.00307
## 3 AAPL   2020-01-06  73.4  75.0  73.2  74.9 118387200     74.3   -0.0114
```

```
head(prices %>% filter(symbol == "NFLX"), 3)
```

```
## # A tibble: 3 x 9
##   symbol date        open  high   low close  volume adjusted log_return
##   <chr>  <date>     <dbl> <dbl> <dbl> <dbl>   <dbl>    <dbl>      <dbl>
## 1 NFLX   2020-01-02  326.  330.  325.  330. 4485800     330.   NA
## 2 NFLX   2020-01-03  327.  330.  326.  326. 3806900     326.    0.00208
## 3 NFLX   2020-01-06  323.  336.  321.  336. 5663100     336.   -0.0113
```

GMM

A Reminder

The Estimator

- We defined the objective function:

$$J_N(\theta) = [\tfrac{1}{N} \sum g(w_i, \theta)]' \, W \, [\tfrac{1}{N} \sum g(w_i, \theta)]$$

- Then we defined the GMM estimator as:

$$\hat{\theta}_{GMM} = argmin \, J_N(\theta)$$

- For every weighting matrix W we get a consistent estimator for θ, but the following is the optimal one:

$$\hat{W}_{opt} = \hat{\Lambda}^{-1} \text{ where } \hat{\Lambda} = \tfrac{1}{N} \sum [g(w_i, \hat{\theta}) \, g(w_i, \hat{\theta})']$$

The Estimator

- Initially, we can't calculate $\hat{W}_{opt}$ since we don't know $\hat{\theta}$
- The solution is to estimate GMM in two steps:

1. Estimate $\hat{\theta}_{step1}$ using the identity matrix (or any other matrix) as a weighting matrix.
2. Estimate $\hat{\theta}_{GMM}$ using $\hat{\Lambda}^{-1}_{step1}$ as a weighting matrix, to get the efficient **Two-Step GMM Estimator**

- Another option is to keep iterating until convergence is obtained (the **Iterated GMM Estimator**)

Examples

The `momentfit` package

- `momentfit` is the R package for estimating models by GMM
- The package supports 3 different ways (or classes) to represent the moment conditions:

   1. The “linearModel” class - for the special case of linear models
   2. The “formulaModel” class - for the special case of Minimum Distance Estimation
   3. The “functionModel” class - for the general case

- We'll use the `bwght` dataset, where we consider the following model:

  $$log(bwght) = \beta_0 + \beta_1 male + \beta_2 parity + \beta_3 log(faminc) + \beta_4 packs + u$$

  to estimate the effect of cigarette smoking on the weight of newborns, using cigarette price as an instrument

- In PS9 you saw that in fact cigarette price is not a good instrument, so we'll not try to interpret the results

The “linearModel” class

```r
library(wooldridge)
library(momentfit)
library(lfe)

data("bwght")

linModel = momentModel(g = lbwght ~ male + parity + lfaminc + packs,
                       x =  ~ male + parity + lfaminc + cigprice,
                       data = bwght,
                       vcov = "iid") #vcov under homoskedasticity
linModel
```

```
## Model based on moment conditions
## *********************************
## Moment type: linear
## Covariance matrix: iid
## Number of regressors: 5
## Number of moment conditions: 5
## Number of Endogenous Variables: 1
## Sample size:  1388
```

The "linearModel" class

```r
gmm = gmmFit(linModel, type = "twostep")

#for comparison:
iv = felm(lbwght ~ male + parity + lfaminc | 0 | (packs ~  cigprice), data = bwght)
```

What would happen if we set `type = "onestep"` instead?

The Just-Identified Case

- Remember that in case when g is linear, i.e., $g(w_i, \theta) = z_i'(y_i - x_i\theta)$, we have a closed formula for $\hat{\theta}_{GMM}$:

$$\hat{\theta}_{GMM} = ((X'Z) \cdot W \cdot (Z'X))^{-1} \cdot (X'Z) \cdot W \cdot Z'y$$

- When $K = L$ (the just-identified case), $X'Z$ and W are square matrices, and therefore the GMM estimator reduces to:

$$\hat{\theta}_{GMM} = (Z'X)^{-1} Z'y$$

- So in this case W plays no rule

The Just-Identified Case

```
summary(iv)$coefficients
```

```
##                   Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)    4.467861478 0.25882893 17.26183221 1.286386e-60
## male           0.029820508 0.01777901  1.67728754 9.371224e-02
## parity        -0.001239075 0.02193217 -0.05649578 9.549550e-01
## lfaminc        0.063645997 0.05701281  1.11634549 2.644682e-01
## `packs(fit)`   0.797106270 1.08627520  0.73379772 4.631964e-01
```

```
summary(gmm)@coef
```

```
##                  Estimate Std. Error     t value     Pr(>|t|)
## (Intercept)   4.467861478 0.25845543 17.28677707 5.916934e-67
## male          0.029820508 0.01775335  1.67971137 9.301349e-02
## parity       -0.001239075 0.02190052 -0.05657742 9.548818e-01
## lfaminc       0.063645997 0.05693054  1.11795871 2.635846e-01
## packs         0.797106270 1.08470771  0.73485812 4.624259e-01
```

The “functionModel” class

We can estimate the same model by defining g directly:

```
g = function(theta, dat){
  #extract the relevant variables:
  X = dat %>%
    select(intercept, male, parity, lfaminc, packs) %>%
    as.matrix()
  Z = dat %>%
    select(intercept, male, parity, lfaminc, cigprice) %>%
    as.matrix()
  y = matrix(dat$lbwght, ncol = 1)
  beta = as.matrix(theta, ncol = 1)
  u = as.vector(y - X%*%beta)
  return(Z*u)
}

#add an intercept
bwght = bwght %>% mutate(intercept = 1)
#initial values for the numerical algorithm:
theta0 = rnorm(5)
names(theta0) = c("intercept", "male", "parity", "lfaminc", "packs")
funModel = momentModel(g = g,
                       x = bwght,
                       theta0 = theta0,
                       vcov = "iid") #vcov under heteroskedasticity
```

The “functionModel” class

```
funModel
```

```
## Model based on moment conditions
## *********************************
## Moment type: function
## Covariance matrix: iid
## Number of regressors: 5
## Number of moment conditions: 5
## Number of Endogenous Variables: 0
## Sample size:  1388
```

The “functionModel” class

```
gmm2 = gmmFit(funModel, type = "twostep")
summary(iv, robust = T)$coefficients
```

```
##                   Estimate Robust s.e     t value     Pr(>|t|)
## (Intercept)    4.467861478 0.25631403 17.43120118 1.142829e-61
## male           0.029820508 0.01722088  1.73164842 8.355917e-02
## parity        -0.001239075 0.02537546 -0.04882966 9.610621e-01
## lfaminc        0.063645997 0.05707269  1.11517428 2.649695e-01
## `packs(fit)`   0.797106270 1.11322077  0.71603611 4.740899e-01
```

```
summary(gmm2)@coef
```

```
##                Estimate Std. Error     t value     Pr(>|t|)
## intercept   4.467861773 0.25585173 17.46269922 2.755897e-68
## male        0.029820502 0.01718982  1.73477673 8.278036e-02
## parity     -0.001239055 0.02532974 -0.04891702 9.609854e-01
## lfaminc     0.063645932 0.05696979  1.11718742 2.639142e-01
## packs       0.797105077 1.11121370  0.71732834 4.731715e-01
```

The Over-Identified Case

- Now things are getting more interesting…
- We'll estimate the model using 2 different weighting matrices: the 2SLS matrix and the optimal matrix (Do we expect to get different results?)
- But first need to modify g:

```
g = function(theta, dat){
  #extract the relevant variables:
  X = dat %>%
    select(intercept, male, parity, lfaminc, packs) %>%
    as.matrix()
  Z = dat %>%
    select(intercept, male, parity, lfaminc, cigprice, cigprice2) %>%
    as.matrix()
  y = matrix(dat$lbwght, ncol = 1)
  beta = as.matrix(theta, ncol = 1)
  u = as.vector(y - X%*%beta)
  return(Z*u)
}
```

The Over-Identified Case

```
#generate more instrument
bwght = bwght %>% mutate(cigprice2 = cigprice^2)
model = momentModel(g = g,
                    x = bwght,
                    theta0 = theta0,
                    vcov = "iid")

#estimate with the optimal matrix
res_with_optimal_mat = gmmFit(model, type = "twostep")

#estimate with the 2sls matrix

#define (Z'Z)^-1
Z = bwght %>%
  select(intercept, male, parity, lfaminc, cigprice, cigprice2) %>%
  as.matrix()
W = solve(t(Z)%*%Z)
res_with_2sls_mat = gmmFit(model, type = "onestep", weights = W)
```

The Over-Identified Case

```
summary(res_with_optimal_mat)@coef
```

```
##                Estimate Std. Error    t value     Pr(>|t|)
## intercept   4.457063590 0.21032171 21.1916476 1.140364e-99
## male        0.029931177 0.01775034  1.6862310 9.175132e-02
## parity     -0.002423397 0.01942087 -0.1247831 9.006952e-01
## lfaminc     0.066078306 0.04652062  1.4204089 1.554887e-01
## packs       0.846892950 0.87865412  0.9638525 3.351199e-01
```

```
summary(res_with_optimal_mat)@specTest
```

```
##
##  J-Test
##                  Statistics  df   pvalue
## Test E(g)=0:      0.0044147   1  0.94702
```

The Over-Identified Case

```
summary(res_with_2sls_mat)@coef
```

```
##                 Estimate Std. Error     t value      Pr(>|t|)
## intercept   4.458513135 0.20885780 21.34712300 4.146951e-101
## male        0.029986374 0.01765379  1.69857987  8.939837e-02
## parity     -0.001966193 0.02006417 -0.09799525  9.219361e-01
## lfaminc     0.065698507 0.04621798  1.42149246  1.551736e-01
## packs       0.836839472 0.87549936  0.95584248  3.391518e-01
```

```
summary(res_with_2sls_mat)@specTest
```

```
##
##  J-Test
##                   Statistics  df   pvalue
## Test E(g)=0:       0.0053864   1  0.94149
```

Minimum Distance Estimation

- This is a special case when moments have data separately from parameters
- For example, consider the problem of estimating the parameters of the normal distribution μ and σ^2
- The moments conditions are:

  1. $x_i - \mu$
  2. ${x_i}^2 - \mu^2 - \sigma^2$

- Let's go back to the prices data and estimate these parameters for the distribution of log-returns (which in this specific case seems like the normal distribution)

The “formulaModel” class

```
moment_conditions = list(log_return ~ mu,
                         log_return^2 ~ mu^2 + sigma2)

theta0 = c(0.1, 0.1)
names(theta0) = c("mu", "sigma2")
formulaModel = momentModel(g = moment_conditions,
                           theta0 = theta0,
                           data = prices,
                           vcov = "CL", #clustered SE
                           #also need to provide the clustering variable:
                           vcovOptions = list(cluster = ~ symbol))

formulaModel
```

```
## Model based on moment conditions
## *********************************
## Moment type: formula
## Covariance matrix: CL
## Clustered based on: symbol
## Number of regressors: 2
## Number of moment conditions: 2
## Number of Endogenous Variables: 0
## Sample size:  117
```

The “formulaModel” class

```
res = gmmFit(formulaModel, type = "twostep")
summary(res)@coef
```

```
##              Estimate   Std. Error    t value     Pr(>|t|)
## mu     -0.0005408820 1.868354e-03 -0.2894965 7.722014e-01
## sigma2  0.0006429251 5.811574e-05 11.0628390 1.899870e-28
```

```
#convergence status for the numerical optimization algorithm:
summary(res)@convergence
```

```
## [1] 0
```