2019 10 19 Order Probit

- Name: Jikhan Jeong
- Reference: https://www.cambridge.org/features/econmodelling/ (https://www.cambridge.org/features/econmodelling/)
- Reference: https://www.nber.org/papers/w7847 (https://www.nber.org/papers/w7847)
- Published: Hamilton, James D. and Oscar Jorda. **"A Model Of The Federal Funds Rate Target,"** Journal of Political Economy, 2002, v110(5,Oct), 1135-1167

Shurtcut in R markdown

1)**chunk n: Ctrl + Alt + I**
2)**knit: Ctrl + Shift + k**
3)**run: Ctrl + Enter**

- The ordered probit of dependent variable is a dummy variable in ordered rank as follows:
- n = decision maker
- The main idea is that there is a latent countinuous metric underling the ordinal dependent variables observed by the researcher.
- Thresholds partition the real line into a series of regions corresponding to the variouse ordinal categories.

$$y_n = d_n = x_n\beta + e_n, e_n \sim N(0,1), \forall n = 1....N,\ d_n \subset 0,1,2,3,4 - (1)$$

Dependent variable are ordinal category variable in here denote as d_i

$$d_n = \begin{cases} 0 : y_n = -0.5 \\ 1 : y_n = -0.25 \\ 2 : y_n = 0.00 \\ 3 : y_n = 0.25 \\ 4 : y_n = 0.50 \end{cases}$$

Probability of dependent variable is 0

$$\begin{aligned} p(d_n = 0) &= P[-oo < d_n < \mu_0] \\ &= P[d_n < \mu_0]\ - plug\,(1) \\ &= p(x_n b + e_n < \mu_0) \\ &= p(e_n < \mu_0 - x_n b) \\ &= \Phi(\mu_0 - x_n b) \end{aligned}$$

C is the intercepts of each regime

$$c_0 < c_1 < c_2 < c_3$$

Normal standard distrubiton of cdf based on different dependent variables

$$d_t = \begin{cases} \Phi_{0,n} = \Pr(d_n = 0) = \Phi(c_0 - x_n\beta) \\ \Phi_{1,n} = \Pr(d_n = 1) = \Phi(c_1 - x_n\beta) - \Phi(c_0 - x_n\beta) \\ \Phi_{2,n} = \Pr(d_n = 2) = \Phi(c_2 - x_n\beta) - \Phi(c_1 - x_n\beta) \\ \Phi_{3,n} = \Pr(d_n = 3) = \Phi(c_3 - x_n\beta) - \Phi(c_2 - x_n\beta) \\ \Phi_{4,n} = \Pr(d_n = 4) = \Phi(c_4 - x_n\beta) - \Phi(c_3 - x_n\beta) \end{cases}$$

The log-likelihood function for a sample size N is

$$

$$
\begin{aligned}
\ln L_N(\theta) =& \frac{1}{N}\sum_{n=1}^{N}[d_{0,n}\ln\Phi_{0,n} + d_{1,n}\ln\Phi_{1,n} + d_{2,n}\ln\Phi_{2,n} + d_{3,n}\ln\Phi_{3,n} + d_{4,n}\ln\Phi_{4,n}] \\
\theta =& \beta, c_0, c_1, c_2, c_3
\end{aligned}
$$

$$

Code for setting

```
rm (list = ls(all=TRUE)) #rm(list=ls()) removes all objects from the current workspace (R memory)
graphics.off() # shuts down all open graphics devices
```

Inverse Matrix

```
#---------------------------------------------------------------------
# Matrix inverse function
# Wrapper function for computing matrix inverse
#---------------------------------------------------------------------
inv <- function (A) {
  return(solve(A)) # solve(A) = Inverse Matrix
}
# Solve: This generic function solves the equation a %*% x = b for x, where b can be either a vector or a matrix. in here so
lve(a) = I^{-1}, ax=I, x=solve(a)
# Ref: https://www.rdocumentation.org/packages/base/versions/3.6.1/topics/solve
```

Example. Solve Example for calculating inverse matrix

```
a_ex =matrix(c(1,2,3,4), nrow=2, ncol=2, byrow=TRUE)
c=solve(a_ex)
a_ex%*%c
```

```
##      [,1]         [,2]
## [1,]    1 1.110223e-16
## [2,]    0 1.000000e+00
```

c is the inverse matrix of a by obtaining c=solve(a)

```
c
```

```
##      [,1] [,2]
## [1,] -2.0  1.0
## [2,]  1.5 -0.5
```

The invrse matrix of a by using inv(a) which defined in the above inv function the results indicates that the same reults with inv(a) = c as follows

```
inv(a_ex)
```

```
##      [,1] [,2]
## [1,] -2.0  1.0
## [2,]  1.5 -0.5
```

Example: rowSums

- Sum values of Raster objects by row or column.
- Ref: https://www.rdocumentation.org/packages/raster/versions/3.0-7/topics/rowSums (https://www.rdocumentation.org/packages/raster/versions/3.0-7/topics/rowSums)

```
(a_ex <-matrix(c(1:10), nrow=2, ncol=5, byrow=TRUE))
```

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

```
class(a_ex)
```

```
## [1] "matrix"
```

```
as.matrix(a_ex)
```

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

```
rowSums(a_ex)
```

```
## [1] 15 40
```

- (example) Basic for pnrom and LL form

```
(a1<-pnorm(0,0,1)) # 0
```

```
## [1] 0.5
```

```
(a2<-pnorm(1,0,1) -a1) # 1
```

```
## [1] 0.3413447
```

```
(a3<-pnorm(2,0,1) -a1 -a2) # 2
```

```
## [1] 0.1359051
```

```
(a4<-1-a1-a2-a3)
```

```
## [1] 0.02275013
```

```
sum(a1,a2,a3,a4) # cdf of sum is 1
```

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

```
(a<-cbind(a1,a2,a3,a4)) # qnorm of each label
```

```
##       a1        a2        a3         a4
## [1,] 0.5 0.3413447 0.1359051 0.02275013
```

```
(log(a))
```

```
##              a1        a2        a3        a4
## [1,] -0.6931472 -1.074862 -1.995798 -3.783184
```

```
(dependent_variable_mark <- c(1,1,1,1)) # actually this should be dummy in real situation but just for practice
```

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

```
(t_dependent_variable_mark <-dependent_variable_mark*log(a))
```

```
##             a1        a2        a3        a4
## [1,] -0.6931472 -1.074862 -1.995798 -3.783184
```

```
(log_liklihood <--mean(t_dependent_variable_mark))
```

```
## [1] 1.886748
```

The log-likelihood function for a sample size N is

$$

$$
\ln L_N(\theta) = \frac{1}{N}\sum_{n=1}^{N}[d_{0,n}\ln\Phi_{0,n} + d_{1,n}\ln\Phi_{1,n} + d_{2,n}\ln\Phi_{2,n} + d_{3,n}\ln\Phi_{3,n} + d_{4,n}\ln\Phi_{4,n}]
$$

$$
\theta = \beta, c_0, c_1, c_2, c_3
$$

$$

Unrestricted Probit *negative* log-likelihood function

- reference: (about pnrom) : http://seankross.com/notes/dpqr/ (http://seankross.com/notes/dpqr/)

```
lprobit <- function(b,x,d){ # b =coefficient, x = feature,
  # Cut off points = regime change
  c<- b[1:4] # cut points parameters (unknown)

  # Regression part excluding the intercepts
  xb <- x[,2:4] %*% b[5:7] # x is feature data and b is coefficient

  # Cut off points
  f1 <- pnorm(c[1] - xb,0,1) # pnorm(0,0,1) CDF of standard normal = 0.5
  f2 <- pnorm(c[2] - xb,0,1) -f1
  f3 <- pnorm(c[3] - xb,0,1) -f1 -f2
  f4 <- pnorm( c[4] - xb,0,1) - f1 - f2 - f3
  f5 <- 1 - f1 - f2 - f3 - f4   # the sum of cdf = 1
  f <-cbind(f1, f2, f3, f4, f5) # matrix

  # Negative Log-Likelihood fuction -> to minimize negative LL = maximization of LL
  tp <- d*log(f) # d = dependent variables with ordered dummay type {d=0, d=1,....d=4}
  lf <- -mean(rowSums(tp)) # negative log likelihood for minimization rather than maximization with positive log likelihood
  return(lf)
}
```

- apply function Ref: https://www.guru99.com/r-apply-sapply-tapply.html (https://www.guru99.com/r-apply-sapply-tapply.html)

apply(X, MARGIN, FUN) x: an array or matrix MARGIN: take a value or range between 1 and 2 to define where to apply the function, 1=rows, 2=columns, c(1,2)= all, FUN: tells which function to apply

```
(m1 <- matrix(C<-(1:10),nrow=5, ncol=6))
```

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

```
(a_m1 <- apply(m1, 2, sum)) # apply(matrix, margin =2 = columns, fucntion = sum) = calcuating column sum
```

```
## [1] 15 40 15 40 15 40
```

Restricted Probit *negative* log-likelihood function : without covariate: only critical points without covariates

```
lOprobit <- function(b,d) { # b = coefficient = unknown parameter = cutting points = scalars = vector =~ matrix, more simpl
er code than the unrestriced case
  # d = multilabel
  # b = b[1,4] =cutting points
  # t = sample size, t(matrix) = transpose, I don't know how it works in this case
  # Cut off points
  c<- b[1:4]

  f1 <- pnorm(c[1]-0,0,1) # -0 for point out that there is no xb in here
  f2 <- pnorm(c[2]-0,0,1) - f1
  f3 <- pnorm(c[3]-0,0,1) - f1 - f2
  f4 <- pnorm(c[4]-0,0,1) - f1 - f2 - f3
  f5 <- 1 - f1 - f2 - f3 - f4
  f  <- cbind(f1, f2,  f3,  f4,  f5)

  # Log-likelihood function

  #tp <- t(apply(d, 1, '*', log(f))) # dependent variable * log (f) transpose (apply function), 1= rows function * multiplic
ation with log(f) ust transpos(d*log(f))
  #tp <- d%*%log(f)
  tp <- d%*%t(log(f)) # d is n = sample size x 5 | log(f) = 1 x 5, t(log(f)) = 5 x 1 | d*log(f) = 1 x 1 in each point, then
 column sum
  # lf <- -mean( rowSums(tp) )
  # print(tp)
  lf <- -mean(rowSums(tp))
  return(lf)
}
```

Ordered Probit with data load

Ref: The data file is from the Hamilton and Jorda (2002) data set.

Dataload

```
setwd("C:/Users/jikhan.jeong/Documents/R/Econ_Modelling_R/New folder/")
getwd()
```

```
## [1] "C:/Users/jikhan.jeong/Documents/R/Econ_Modelling_R/New folder"
```

```
usmoney <- as.matrix(read.table("C:/Users/jikhan.jeong/Documents/R/Econ_Modelling_R/New folder/usmoney.dat"))
```

```
target  <- usmoney[,2]
bin     <- usmoney[,4]  # to data
fomc    <- usmoney[,8]
spread6 <- usmoney[,20]
spread <- -spread6      # to data
inf     <- usmoney[,23] # to data
gdp     <- usmoney[,28] # to data

ind       <- fomc == 1  # Boolen operation Ture and False based on FOMC ==1

# Choose data based on fomc days
data      <- cbind(bin, spread, inf, gdp)

#       bin     spread        inf        gdp
# 0.0000000 -0.2900000  0.0489853  0.4000000

data_fomc <- data[ind,] # sliceing fomc ==1 rows
# data_fomc
# > length(data)
# [1] 2772
# > length(data_fomc) # data_fomc is a subset of data that meet the condition "fomc ==1" condition
# [1] 424

# Dependent and independent variables
y <- data_fomc[,1] # first column is dependent variagle
n<- length(y) # sample size
x <-cbind(rep(1,n), data_fomc[,2:4]) # rep : replicates the values in x. # this is for constant
# rep(1,n) # for constant

  # Create dummy variables for each interest rate change
d1 <- as.numeric(y == -0.50) # true 1 if not 0, example  1 = as.numeric(1 == 1),  0 = as.numeric(1 == 0) making dummy
d2 <- as.numeric(y == -0.25)
d3 <- as.numeric(y ==  0.00)
d4 <- as.numeric(y ==  0.25)
d5 <- as.numeric(y ==  0.50)
d  <- cbind(d1, d2, d3, d4, d5)
```

```
# raw_data <-cbind(y,x)
# dim(raw_data)
# write.csv(raw_data,'nov_21_2019_order_probit_raw_data.csv')
```

Estimation

```
# Estimate model by OLS (ie ignore that the data are ordered
reg <- lm(y ~ x - 1)   # no constant
b <- reg$coef          # coefficient
u <- reg$residuals     # residual
s <- sqrt( mean(u^2) ) # mean square error
summary(reg)
```

```
##
## Call:
## lm(formula = y ~ x - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.49858 -0.06383 -0.00077  0.07748  0.31150
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## x       -0.0006804  0.0834120  -0.008   0.9935
## xspread  0.1709798  0.0308328   5.545 2.32e-07 ***
## xinf     0.7902920  2.1013518   0.376   0.7076
## xgdp     0.0142407  0.0077021   1.849   0.0674 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1375 on 102 degrees of freedom
## Multiple R-squared:  0.3139, Adjusted R-squared:  0.287
## F-statistic: 11.67 on 4 and 102 DF,  p-value: 7.692e-08
```

```
print(s)
```

```
## [1] 0.1349221
```

Unresctricted Ordered Probit

```
# Compute the unconditional probabilities of each regime
# d = multilabel
p <- cumsum(colMeans(d)) # for initial value of cutting points

# > colMeans(d)
#         d1         d2         d3         d4         d5
# 0.02830189 0.10377358 0.77358491 0.05660377 0.03773585
# > cumsum(colMeans(d))
#         d1         d2         d3         d4         d5
# 0.02830189 0.13207547 0.90566038 0.96226415 1.00000000

# Estimate the unrestricted ordered probit regression model by MLE
theta0 <- c(qnorm(p[1:4],0,1), b[2:4]/s ) #qnorm(p[1:4],0,1) is for cutting point value
# theta0 # intial values

# minimization
estResults <- optim(theta0, lprobit, x=x, d=d, method="BFGS", hessian=T) # BFGS or Nelder-Mead
# theta0 = initial value
# lprobit = nagative probit ll
# x = independent
# d = multilabel
# method = optimization methods
# Hessian = True

theta1 <- estResults$par # coefficient
l1 <- estResults$val # nagative log likelihood
h <- estResults$hessian

cat('₩nUnrestricted parameter estimates')
```

```
##
## Unrestricted parameter estimates
```

```
cat('₩n', theta1 )
```

```
##
##  -2.537094 -1.61858 1.59053 2.369967 1.650514 5.862124 0.1266434
```

```
cov <- (1/n)*inv(h) # covariance matrix
sd_matrix <-as.matrix(sqrt(cov))
sd_matrix
```

```
##                d1        d2        d3        d4    xspread       xinf
## d1      0.8763188 0.8384845 0.8000517 0.7983883 0.23636125  4.0334181
## d2      0.8384845 0.8396043 0.7969932 0.7959278 0.24507397  4.0137064
## d3      0.8000517 0.7969932 0.8248059 0.8232864 0.34848638  3.9939107
## d4      0.7983883 0.7959278 0.8232864 0.8747022 0.37683918  4.0199309
## xspread 0.2363612 0.2450740 0.3484864 0.3768392 0.33819846  1.7521257
## xinf    4.0334181 4.0137064 3.9939107 4.0199309 1.75212566 20.3724852
## xgdp    0.1736433 0.1739791 0.1864490 0.1932791 0.07846975  0.7797129
##               xgdp
## d1      0.17364326
## d2      0.17397914
## d3      0.18644900
## d4      0.19327914
## xspread 0.07846975
## xinf    0.77971286
## xgdp    0.07789864
```

```
variance  = diag(cov) # variance of each independent variable

standard.error = sqrt(variance) # standard.error of each variables
cat('₩n₩n standard error of coeff=    ',standard.error)
```

```
##
##
##  standard error of coeff=     0.8763188 0.8396043 0.8248059 0.8747022 0.3381985 20.37249 0.07789864
```

```
cat('₩n₩nCovariance Matrix of Unrestricted =     ',cov)
```

```
##
##
## Covariance Matrix of Unrestricted =      0.7679346 0.7030562 0.6400828 0.6374238 0.05586664 16.26846 0.03015198 0.7030562
0.7049353 0.6351982 0.6335011 0.06006125 16.10984 0.03026874 0.6400828 0.6351982 0.6803048 0.6778005 0.1214428 15.95132 0.03
476323 0.6374238 0.6335011 0.6778005 0.7651039 0.1420078 16.15984 0.03735683 0.05586664 0.06006125 0.1214428 0.1420078 0.114
3782 3.069944 0.006157502 16.26846 16.10984 15.95132 16.15984 3.069944 415.0382 0.6079521 0.03015198 0.03026874 0.03476323
0.03735683 0.006157502 0.6079521 0.006068197
```

```
cat('₩n₩nStandard error of each variable of Unrestricted =     ',standard.error)
```

```
##
##
## Standard error of each variable of Unrestricted =      0.8763188 0.8396043 0.8248059 0.8747022 0.3381985 20.37249 0.07789
864
```

```
l1 <- -l1 # loglikelihood = (-1) x negative loglikelihood

cat('₩n₩nUnrestricted log-likelihood function =     ',l1)
```

```
##
##
## Unrestricted log-likelihood function =      -0.642981
```

```
cat('₩n T x unrestricted log-likelihood function = ',n*l1)
```

```
##
##  T x unrestricted log-likelihood function =  -68.15599
```

z-state

- Ref: http://logisticregressionanalysis.com/1577-what-are-z-values-in-logistic-regression/ (http://logisticregressionanalysis.com/1577-what-are-z-values-in-logistic-regression/)
- Diagonal element of covraiance matrix is variance of coefficient
- cov <- (1/n)*inv(h) # covariance matrix
- variance = diag(cov) # variance of each independent variable
- standard.error = sqrt(variance) # standard.error of each variables

$$X = \frac{\hat{\beta} - 0}{\hat{\sigma}_{\hat{\beta}}} = \frac{\hat{\beta}}{\hat{\sigma}_{\hat{\beta}}}$$

Z-score for var1 (=xspread)

sd = 0.3381985 coefficient = 0.33819846

```
sd = 0.3381985
coef = 1.650514
(z.test.var1 =  coef/sd)
```

```
## [1] 4.880311
```

p-value calcuation

- Ref: https://www.cyclismo.org/tutorial/R/pValues.html (https://www.cyclismo.org/tutorial/R/pValues.html)

```
# options("scipen"=-100, "digits"=4) for -e format
options("scipen"=100, "digits"=4)
2*pnorm(-abs(z.test.var1 ))
```

```
## [1] 0.000001059
```

unrestrictued stata output : (stata) oprobit y x1 x2 x3, nolog

- stata using BFGS optimization

```
. oprobit y x1 x2 x3, nolog

Ordered probit regression                   Number of obs     =        106
                                            LR chi2(3)        =      37.70
                                            Prob > chi2       =     0.0000
Log likelihood = -68.154038                 Pseudo R2         =     0.2167

------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |   1.641085   .3379364     4.86   0.000     .9787419    2.303428
          x2 |   4.590844   20.37261     0.23   0.822    -35.33873    44.52042
          x3 |   .1248224   .0779046     1.60   0.109    -.0278678    .2775126
-------------+----------------------------------------------------------------
       /cut1 |  -2.586887   .8773858                     -4.306531   -.8672421
       /cut2 |   -1.66787   .8406923                     -3.315597   -.0201437
       /cut3 |   1.541877   .8241242                      -.073377    3.157131
       /cut4 |   2.320755   .8740189                      .6077091      4.0338
------------------------------------------------------------------------------
```

stata

Estimate the restricted probit regression model by MLE

```
theta0 <- qnorm(p[1:4],0,1) # inital value for cutting point
estResults <- optim(theta0, lOprobit, d=d, method="BFGS", hessian=T)

theta <- estResults$par # cutting points
l0 <- estResults$val     # negative likelihood
h <- estResults$hessian # hesiian

cat('₩n₩nRestricted parameter estimates')
```

```
##
##
## Restricted parameter estimates
```

```
cat('₩n', theta )
```

```
##
##  -1.906 -1.117 1.314 1.778
```

```
l0 <- -l0 # likelihood = (-1) x negative likelihood
cat('₩n')
```

```
cat('₩n Restricted log-likelihood function =     ',l0) # already n is multplied
```

```
##
##  Restricted log-likelihood function =      -0.8208
```

```
cat('₩n T x restricted log-likelihood function = ',n*l0)
```

```
##
##  T x restricted log-likelihood function =  -87.01
```

Stata output for unrestriced ordered probit : (stata) oprobit y,nolog

- stata using BFGS optimization

```
. oprobit y

Iteration 0:   log likelihood = -87.005147
Iteration 1:   log likelihood = -87.005147

Ordered probit regression                       Number of obs     =        106
                                                LR chi2(0)        =      -0.00
                                                Prob > chi2       =          .
Log likelihood = -87.005147                     Pseudo R2         =    -0.0000

------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       /cut1 |  -1.906358   .2484661                     -2.393343   -1.419373
       /cut2 |  -1.116634   .1537601                     -1.417998   -.8152695
       /cut3 |   1.314496   .1688401                      .9835758    1.645417
       /cut4 |   1.777587   .2252198                      1.336164     2.21901
------------------------------------------------------------------------------
```

stata2

Likelihood Ratio Test

```
# Likelihood ratio test
lr <- -2*n*(l0 - l1)

cat('\n\nLR Statistic          = ',lr)
```

```
##
##
## LR Statistic          =  37.7
```

```
cat('\np-value               = ',1-pchisq(lr,ncol(x)-1))
```

```
##
## p-value               =  0.00000003274
```

```
cat('\n')
```

Wald Test

```r
# Wald test
r <- matrix(c(0,   0,   0,   0,   1,   0,   0,
              0,   0,   0,   0,   0,   1,   0,
              0,   0,   0,   0,   0,   0,   1), byrow=T, nrow=3)

q <- rbind(0, 0, 0)

wd <- t( r %*% theta1 - q) %*% inv(r %*% cov %*% t(r)) %*% (r %*% theta1 - q)

cat('₩nWald Statistic       = ',wd)
```

```
##
## Wald Statistic       =  29.8
```

```r
cat('₩np-value              = ',1-pchisq(wd,ncol(x)-1))
```

```
##
## p-value              =  0.000001523
```