```
> # Exercise 1
> # Set a seed
> set.seed(008)
> # create X1 which is a vector of 10,000 draws from a unif
orm distribution with range 1:3.
> X1 <- as.vector(runif(10000, min = 1, max = 3))
> # create X2 which is a vector of 10,000 draws from a gamm
a distribution with shape 3 and scale 2
> X2 <- as.vector(rgamma(10000, shape = 3, scale = 2))
> # create X3 which is a vector of 10,000 draws from a bino
mial distribution with probability 0.3
> X3 <- as.vector(rbinom(10000, size = 1, prob = 0.3))
> # create eps which is a vector of 10,000 draws from a nor
mal distribution with mean 2 and sd 1
> eps <- as.vector(rnorm(10000, mean = 2, sd = 1))
> # create Y and ydum
> Y <- as.vector(0.5 + 1.2*X1 + -0.9*X2 + 0.1*X3 + eps)
> ydum <- as.matrix(as.numeric(Y>mean(Y)))
>
>
> # Exercise 2
> cor(Y, X1)
[1] 0.2021716
> # The output is 0.1452196, which is quite different from
1.2.
> # create a 10000*1 vector of number 0.5
> X0 <- data.frame(num=1)
> X0 <- as.vector(X0[rep(1:nrow(X0), each=10000),])
> # combine the four vectors
> X <- cbind(X0, X1, X2, X3)
> # calculate the coefficient
> beta1 <- solve(t(X)%*%X)%*%t(X)%*%Y
>
> # Calculate the estimator
> # Calculate the eps at first
> # s2 = (e'e)/n-k(the residual matrix/the degree of freed
om)
> eps2 <- as.matrix(Y - X%*%beta1)
> s2 <- as.numeric((t(eps2)%*%eps2)/9996)
> # calculate covariance matrix and standard error
> X_X <- t(X)%*%X
> X_X_2 <- solve(X_X)
> se_b <- s2*X_X_2
> se_b <- sqrt(se_b)
```

```
Warning message:
In sqrt(se_b) : NaNs produced
> # We only need the data on the diagonal
> se__b <- as.matrix(diag(se_b))
>
> # Bootstrap rep = 49
> # combine Y and X
> XY <- cbind(Y,1,X1,X2,X3)
> X_boot <- matrix(c(0),nrow = 4, ncol = 49)
> # use for loop to resample 49 times, put the data into x_
boot
> loop = 49
> for(y in 1:loop){
+   # generate samples, replace the data each time
+   i <- sample(1:10000, size = 10000, replace = TRUE)
+   Y <- as.matrix(XY[i,1])
+   X0 <- as.vector(XY[i,2])
+   X1 <- as.vector(XY[i,3])
+   X2 <- as.vector(XY[i,4])
+   X3 <- as.vector(XY[i,5])
+   X4 <- as.matrix(cbind(X0,X1,X2,X3))
+   # calculate beta = (X'X)-1*X'Y
+   beta_boot <- solve(t(X4)%*%X4)%*%t(X4)%*%Y
+   X_boot[,y] <- beta_boot
+ }
> # calculate standard error
> sd_X_boot <- cbind(sd(X_boot[1,]),sd(X_boot[2,]), sd(X_b
oot[3,]), sd(X_boot[4,]))
> # result:
> # 0.03601531
> # 0.01533849
> # 0.003224517
> # 0.01940847
>
>
> # Bootstrap rep = 499
> # repeat the process above
> X_boot2 <- matrix(c(0),nrow = 4, ncol = 499)
> # use for loop to resample 499 times, put the data into x
_boot2
> for(y in 1:499){
+   i <- sample(1:10000, size = 10000, replace = TRUE)
+   Y <- as.matrix(XY[i,1])
+   X0 <- as.vector(XY[i,2])
```

```
+   X1 <- as.vector(XY[i,3])
+   X2 <- as.vector(XY[i,4])
+   X3 <- as.vector(XY[i,5])
+   X4 <- as.matrix(cbind(X0,X1,X2,X3))
+   beta_boot <- solve(t(X4)%*%X4)%*%t(X4)%*%Y
+   X_boot2[,y]<- beta_boot
+ }
> sd_X_boot2 <- cbind(sd(X_boot2[1,]),sd(X_boot2[2,]), sd
(X_boot2[3,]), sd(X_boot2[4,]))
> # result:
> # 0.03838917
> # 0.01649101
> # 0.002774781
> # 0.02084976
>
>
> # exercise 3
> # get the log likelihood function
> est_4 <- function(beta,x,y){
+   y <- sum(ydum*log(pnorm(X%*%beta)) + sum((1-ydum)*log
(1-pnorm(X%*%beta))
+   return(-y)
+ }
>
>
> # set an eps
> d <- 0.00000001
> # set an initial b
> b <- c(3, 1, -1, 0.05)
> # generate the matrix of b+eps
> bn <- as.matrix(cbind(b,b,b,b))
> bn2 <- diag(d, 4)
> bn3 <- bn + bn2
> # calculate the derivative of log likelihood function an
d determine the direction
> p1 <- (est_4(bn3[,1])-est_4(bn[,1]))/d
> p2 <- (est_4(bn3[,2])-est_4(bn[,2]))/d
> p3 <- (est_4(bn3[,3])-est_4(bn[,3]))/d
> p4 <- (est_4(bn3[,4])-est_4(bn[,4]))/d
> p <- as.vector(c(p1,p2,p3,p4))
> # give an initial change rate
> diff <- 1
> while(diff > 0.0000001){
```

```
+   # generate a new b based on ak and dk, set initial ak = 
0.000001
+   b0 <- matrix(b, ncol = 1)
+   b <- matrix(b - 0.000001*p, ncol = 1)
+   # calculate the change rate of b
+   diff <- (est_4(b)-est_4(b0))/est_4(b0)
+ }
> 
> # print b
> b
           [,1]
[1,]  3.00216087
[2,]  1.00449938
[3,] -0.98763998
[4,]  0.05062369
> # [,1]
> # [1,]  3.00216087
> # [2,]  1.00449938
> # [3,] -0.98763998
> # [4,]  0.05062369
> # The estimated coefficient of X2 is more different from 
that in true parameters, compared with the others.
> 
> 
> 
> # exercise 4
> # write the log likelihood function of logit model
> logit_log <- function(beta, x, y){
+   y <- sum(ydum*log(exp(X%*%beta)/(1+exp(X%*%beta)))) + 
sum((1-ydum)*log(1-exp(X%*%beta)/(1+exp(X%*%beta))))
+   return(-y)
+ }
> # use optimize function to get the MLE
> xmin_log <- optim(c(0,0,0,0), logit_log, x = X, y = ydum)
$par
> # use glm function to check the result, the two results a
re same
> ydumX <- as.data.frame(cbind(ydum, X))
> logit <- glm(ydum ~ 0 + X, family=binomial(link="logit
"), data=ydumX)
Warning message:
glm.fit:拟合機率算出来是数值零或一
> summary(logit)
```

```
Call:
glm(formula = ydum ~ 0 + X, family = binomial(link = "logit
"),
    data = ydumX)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.4351  -0.1430   0.0394   0.2609   3.2958

Coefficients:
    Estimate Std. Error z value Pr(>|z|)
XX0  5.30949    0.18634  28.494   <2e-16 ***
XX1  2.22403    0.08216  27.071   <2e-16 ***
XX2 -1.61703    0.03672 -44.040   <2e-16 ***
XX3  0.05262    0.08566   0.614    0.539
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1
‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 13862.9  on 10000  degrees of freedom
Residual deviance:  4315.5  on  9996  degrees of freedom
AIC: 4323.5

Number of Fisher Scoring iterations: 7

> # glm(formula = ydum ~ 0 + X, family = binomial(link = "l
ogit"),
> #     data = ydumX)
> #
> # Deviance Residuals:
> #   Min       1Q   Median       3Q      Max
> # -3.4351  -0.1430   0.0394   0.2609   3.2958
> #
> # Coefficients:
> #   Estimate Std. Error z value Pr(>|z|)
> # XX0  5.30949    0.18634  28.494   <2e-16 ***
> #   XX1  2.22403    0.08216  27.071   <2e-16 ***
> #   XX2 -1.61703    0.03672 -44.040   <2e-16 ***
> #   XX3  0.05262    0.08566   0.614    0.539
> # ---
> #   Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’
0.1 ‘ ’ 1
```

```
> #
> # (Dispersion parameter for binomial family taken to be
1)
> #
> # Null deviance: 13862.9  on 10000  degrees of freedom
> # Residual deviance:  4315.5  on  9996  degrees of freedo
m
> # AIC: 4323.5
> #
> # Number of Fisher Scoring iterations: 7
>
> # interpretation
> # 2.22403 is positive, which means that y will be more li
kely to happen when x1 = 1
> # -1.61703 is negative, which means that y will be less l
ikely to happen when x2 = 1
> # 0.05262 is positive, which means that y will be more li
kely to happen when x3 = 1
> # "***" in the summarty means that the estimation of X1 a
nd X2 are signigicant, while that of X3 is not quite signif
icant.
>
> # repeat the process above
> xmin_pro <- optim(c(0,0,0,0), est_4, x = X, y = ydum)$par
> probit <- glm(ydum ~ 0 + X, family=binomial(link="probit
"), data=ydumX)
Warning message:
glm.fit:拟合機率算出来是数值零或一
> summary(probit)

Call:
glm(formula = ydum ~ 0 + X, family = binomial(link = "probi
t"),
    data = ydumX)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.8746  -0.1081   0.0087   0.2525   3.6264

Coefficients:
    Estimate Std. Error z value Pr(>|z|)
XX0  2.92587    0.09857  29.683   <2e-16 ***
XX1  1.23282    0.04383  28.129   <2e-16 ***
XX2 -0.89277    0.01818 -49.118   <2e-16 ***
```

```
XX3  0.02977    0.04750   0.627    0.531
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1
‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 13862.9  on 10000  degrees of freedom
Residual deviance:  4314.1  on  9996  degrees of freedom
AIC: 4322.1

Number of Fisher Scoring iterations: 7

> # glm(formula = ydum ~ 0 + X, family = binomial(link = "p
robit"),
> #     data = ydumX)
> #
> # Deviance Residuals:
> #   Min       1Q   Median       3Q      Max
> # -3.8746  -0.1081   0.0087   0.2525   3.6264
> #
> # Coefficients:
> #   Estimate Std. Error z value Pr(>|z|)
> # XX0  2.92587    0.09857  29.683   <2e-16 ***
> #   XX1  1.23282    0.04383  28.129   <2e-16 ***
> #   XX2 -0.89277    0.01818 -49.118   <2e-16 ***
> #   XX3  0.02977    0.04750   0.627    0.531
> # ---
> #   Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’
0.1 ‘ ’ 1
> #
> # (Dispersion parameter for binomial family taken to be
1)
> #
> # Null deviance: 13862.9  on 10000  degrees of freedom
> # Residual deviance:  4314.1  on  9996  degrees of freedo
m
> # AIC: 4322.1
> #
> # Number of Fisher Scoring iterations: 7
>
> # interpretation
> # 1.23282 is positive, which means that y will be more li
kely to happen when x1 = 1
```

```
> # -0.89277 is negative, which means that y will be less l
ikely to happen when x2 = 1
> # 0.02977 is positive, which means that y will be more li
kely to happen when x3 = 1
> # "***" in the summary means that the estimation of X1 an
d X2 are significant, while that of X3 is not significant.
> # Logit and probit models yield almost the same result.
>
> # repeat the process above
> linear_function <- function(beta, x, y){
+   y <- t(ydum-X%*%beta)%*%(ydum-X%*%beta)
+   return(y)
+ }
> xmin_linear <- as.matrix(optim(c(0,0,0,0), linear_functi
on, y = ydum, x = X)$par)
> linear_probability <- lm(ydum~0 + X, data=ydumX)
> summary(linear_probability)

Call:
lm(formula = ydum ~ 0 + X, data = ydumX)

Residuals:
     Min       1Q   Median       3Q      Max
-0.94585 -0.26742  0.05788  0.24721  2.02094

Coefficients:
      Estimate Std. Error  t value Pr(>|t|)
XX0  0.8868259  0.0133142   66.608   <2e-16 ***
XX1  0.1501367  0.0056936   26.370   <2e-16 ***
XX2 -0.1038651  0.0009539 -108.879   <2e-16 ***
XX3  0.0059472  0.0072690    0.818    0.413
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1
‘ ’ 1

Residual standard error: 0.3306 on 9996 degrees of freedom
Multiple R-squared:  0.8064,     Adjusted R-squared:  0.806
3
F-statistic: 1.041e+04 on 4 and 9996 DF,  p-value: < 2.2e-1
6

> # lm(formula = ydum ~ 0 + X, data = ydumX)
> #
> # Residuals:
```

```
> #   Min      1Q   Median      3Q     Max
> # -0.94585 -0.26742  0.05788  0.24721  2.02094
> #
> # Coefficients:
> #   Estimate Std. Error  t value Pr(>|t|)
> # XX0  0.8868259  0.0133142   66.608   <2e-16 ***
> #   XX1  0.1501367  0.0056936   26.370   <2e-16 ***
> #   XX2 -0.1038651  0.0009539 -108.879   <2e-16 ***
> #   XX3  0.0059472  0.0072690    0.818    0.413
> # ---
> #   Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’
0.1 ‘ ’ 1
> #
> # Residual standard error: 0.3306 on 9996 degrees of free
dom
> # Multiple R-squared:  0.8064, Adjusted R-squared:  0.806
3
> # F-statistic: 1.041e+04 on 4 and 9996 DF,  p-value: < 2.
2e-16
>
> # interpretation
> # Since X1 is continuous, 0.1501 is the change in the pro
bability of success given one unit increase in x1.
> # Since X2 is continuous, 0.1501 is the change in the pro
bability of success given one unit increase in x2.
> # Since X3 is discrete, 0.0059 is the difference in the p
robability of success when x3 = 1 and x3 = 0, holding other
 xj fixed. It seems that X3 almost doesn't impact the proba
bility.
> # "***" in the summarty means that the estimation of X1 a
nd X2 are signigicant, while that of X3 is not quite signif
icant.
>
> # exercise 5
>
> # write the derivative of function of logit
> est_5 <- function(x){
+   y <- exp(x)/(1+exp(x))^2
+   return(y)
+ }
>
> # calculate the mean for each column of X
> est_6 <- function(beta){
+   # calculate f'(XB)
```

```
+     y1 <- mean(X%*%beta)
+   # calculate the marginal effect
+     y2 <- est_5(y1)*beta
+     y <- as.matrix(y2)
+   return(y)
+ }
>
> # run the result of Q4
> beta3 <- matrix(c(5.30949, 2.22403, -1.61703, 0.05262),
ncol = 1)
> me1 <- est_6(beta3)
>
>
> # repeat the process above
> est_7 <- function(beta){
+   y1 <- mean(X%*%beta)
+   p1 <- dnorm(y1)
+   y <- p1*beta
+   y <- as.matrix(y)
+   return(y)
+ }
>
> beta4 <- matrix(c(2.86589, 1.22988, -0.88943, 0.09990),
ncol = 1)
> me2 <- est_7(beta4)
>
> # calculate the standard deviations using the delta meth
od
> # generate the Jacobian matrix, calculate each partial d
erivative
> est_8 <- function(beta){
+   # calculate the mean of the X*beta and f'(mean(XB))
+   y1 <- X%*%beta
+   y2 <- mean(y1)
+   # calculate the marginal effect
+   y3 <- est_5(y2)*beta[1,]
+   return(y3)
+ }
>
> # write the similar functions to calculate the marginal e
ffects respective to beta1, 2, 3 and 4
> # Let's call them ME1_l, ME2_l, ME3_l, ME4_l
> est_9 <- function(beta){
+   y1 <- X%*%beta
```

```
+   y2 <- mean(y1)
+   y3 <- est_5(y2)*beta[2,]
+   return(y3)
+ }
>
> est_10 <- function(beta){
+   y1 <- X%*%beta
+   y2 <- mean(y1)
+   y3 <- est_5(y2)*beta[3,]
+   return(y3)
+ }
>
> est_11 <- function(beta){
+   y1 <- X%*%beta
+   y2 <- mean(y1)
+   y3 <- est_5(y2)*beta[4,]
+   return(y3)
+ }
>
> # calculate the partial derivative of ME1_l repective to
 beta1
> beta5 <- matrix(c(5.30950, 2.22403, -1.61703, 0.05262),
ncol = 1)
> logit_11 <- (est_8(beta3)-est_8(beta5))/0.00001
> # calculate the partial derivative of ME1_l repective to
 beta2
> beta6 <- matrix(c(5.30949, 2.22404, -1.61703, 0.05262),
ncol = 1)
> logit_12 <- (est_8(beta3)-est_8(beta6))/0.00001
> # calculate the partial derivative of ME1_l repective to
 beta3
> beta7 <- matrix(c(5.30949, 2.22403, -1.61702, 0.05262),
ncol = 1)
> logit_13 <- (est_8(beta3)-est_8(beta7))/0.00001
> # calculate the partial derivative of ME1_l repective to
 beta4
> beta8 <- matrix(c(5.30949, 2.22403, -1.61703, 0.05263),
ncol = 1)
> logit_14 <- (est_8(beta3)-est_8(beta8))/0.00001
> # generate the first row of the Jacobian matrix of logit
model
> logit_1 <- t(as.matrix(c(logit_11, logit_12, logit_13, l
ogit_14)))
>
```

```
> # repeat the process above to generate the second line of
 this matrix
> logit_21 <- (est_9(beta3)-est_9(beta5))/0.00001
> logit_22 <- (est_9(beta3)-est_9(beta6))/0.00001
> logit_23 <- (est_9(beta3)-est_9(beta7))/0.00001
> logit_24 <- (est_9(beta3)-est_9(beta8))/0.00001
> logit_2 <- t(as.matrix(c(logit_21, logit_22, logit_23, l
ogit_24)))
>
> # repeat the process above to generate the third line of
this matrix
> logit_31 <- (est_10(beta3)-est_10(beta5))/0.00001
> logit_32 <- (est_10(beta3)-est_10(beta6))/0.00001
> logit_33 <- (est_10(beta3)-est_10(beta7))/0.00001
> logit_34 <- (est_10(beta3)-est_10(beta8))/0.00001
> logit_3 <- t(as.matrix(c(logit_31, logit_32, logit_33, l
ogit_34)))
>
> # repeat the process above to generate the fourth line of
 this matrix
> logit_41 <- (est_11(beta3)-est_11(beta5))/0.00001
> logit_42 <- (est_11(beta3)-est_11(beta6))/0.00001
> logit_43 <- (est_11(beta3)-est_11(beta7))/0.00001
> logit_44 <- (est_11(beta3)-est_11(beta8))/0.00001
> logit_4 <- t(as.matrix(c(logit_41, logit_42, logit_43, l
ogit_44)))
>
> # generate the whole Jacobian matrix
> logit_j <- as.matrix(rbind(logit_1, logit_2, logit_3, lo
git_4))
>
> # calculate the sd of logit model: the sd of the diagonal
 of J(T)*vcovmatrix*J
> sd_delta_logit <- t(logit_j)%*%vcov(logit)%*%logit_j
> sd_delta_logit <- sqrt(diag(sd_delta_logit))
> sd_delta_logit
[1] 0.04044099 0.02404779 0.04557324 0.02167735
> # output: [1] 0.04044099 0.02404779 0.04557324 0.0216773
5
>
> # write the similar functions to calculate the marginal e
ffects respective to beta1, 2, 3 and 4
> # Let's call them ME1_p, ME2_p, ME3_p, ME4_p
> est_12 <- function(beta){
```

```
+   y1 <- X%*%beta
+   y2 <- mean(y1)
+   p1 <- dnorm(y2)
+   y <- p1*beta[1,]
+   return(y)
+ }
>
> est_13 <- function(beta){
+   y1 <- X%*%beta
+   y2 <- mean(y1)
+   p1 <- dnorm(y2)
+   y <- p1*beta[2,]
+   return(y)
+ }
>
> est_14 <- function(beta){
+   y1 <- X%*%beta
+   y2 <- mean(y1)
+   p1 <- dnorm(y2)
+   y <- p1*beta[3,]
+   return(y)
+ }
>
> est_15 <- function(beta){
+   y1 <- X%*%beta
+   y2 <- mean(y1)
+   p1 <- dnorm(y2)
+   y <- p1*beta[4,]
+   return(y)
+ }
>
> # calculate the partial derivative of ME1_p repective to
 beta1
> beta9 <- matrix(c(2.86590, 1.22988, -0.88943, 0.09990),
ncol = 1)
> probit_11 <- (est_12(beta4)-est_12(beta9))/0.00001
> # calculate the partial derivative of ME1_p repective to
 beta2
> beta10 <- matrix(c(2.86589, 1.22989, -0.88943, 0.09990),
 ncol = 1)
> probit_12 <- (est_12(beta4)-est_12(beta10))/0.00001
> # calculate the partial derivative of ME1_p repective to
 beta3
```

```
> beta11 <- matrix(c(2.86589, 1.22988, -0.88942, 0.09990),
 ncol = 1)
> probit_13 <- (est_12(beta4)-est_12(beta11))/0.00001
> # calculate the partial derivative of ME1_p repective to
 beta4
> beta12 <- matrix(c(2.86589, 1.22988, -0.88943, 0.09991),
 ncol = 1)
> probit_14 <- (est_12(beta4)-est_12(beta12))/0.00001
> # generate the first row of the Jacobian matrix of probit
 model
> probit_1 <- t(as.matrix(c(probit_11, probit_12, probit_1
3, probit_14)))
>
> # repeat the process above to generate the second line of
 this matrix
> probit_21 <- (est_13(beta4)-est_13(beta9))/0.00001
> probit_22 <- (est_13(beta4)-est_13(beta10))/0.00001
> probit_23 <- (est_13(beta4)-est_13(beta11))/0.00001
> probit_24 <- (est_13(beta4)-est_13(beta12))/0.00001
> probit_2 <- t(as.matrix(c(probit_21, probit_22, probit_2
3, probit_24)))
>
> # repeat the process above to generate the third line of
this matrix
> probit_31 <- (est_14(beta4)-est_14(beta9))/0.00001
> probit_32 <- (est_14(beta4)-est_14(beta10))/0.00001
> probit_33 <- (est_14(beta4)-est_14(beta11))/0.00001
> probit_34 <- (est_14(beta4)-est_14(beta12))/0.00001
> probit_3 <- t(as.matrix(c(probit_31, probit_32, probit_3
3, probit_34)))
>
> # repeat the process above to generate the fourth line of
 this matrix
> probit_41 <- (est_15(beta4)-est_15(beta9))/0.00001
> probit_42 <- (est_15(beta4)-est_15(beta10))/0.00001
> probit_43 <- (est_15(beta4)-est_15(beta11))/0.00001
> probit_44 <- (est_15(beta4)-est_15(beta12))/0.00001
> probit_4 <- t(as.matrix(c(probit_41, probit_42, probit_4
3, probit_44)))
>
> # generate the whole Jacobian matrix
> probit_j <- as.matrix(rbind(probit_1, probit_2, probit_
3, probit_4))
>
```

```
> # calculate the sd of probit model: the sd of the diagona
l of J(T)*vcovmatrix*J
> sd_delta_probit <- t(probit_j)%*%vcov(probit)%*%probit_
j
> sd_delta_probit <- sqrt(diag(sd_delta_probit))
> sd_delta_probit
[1] 0.03858910 0.01769946 0.01116097 0.01897350
> # output: [1] 0.03858910 0.01769946 0.01116097 0.0189735
0
>
> # calculate the standard deviations using bootstrap
> # set an empty matrix to combine data
> me_bootlogit <- matrix(c(0,0,0,0),nrow = 1, ncol = 4)
> loop3 <- 450
> for(r in 1:loop3){
+   # use bootstrap to generate data for 10000 times
+     i <- sample(1:10000, size = 10000, replace = TRUE)
+     Y <- as.matrix(ydum[i,])
+     X0 <- as.vector(XY[i,2])
+     X1 <- as.vector(XY[i,3])
+     X2 <- as.vector(XY[i,4])
+     X3 <- as.vector(XY[i,5])
+     X4 <- as.matrix(cbind(X0,X1,X2,X3))
+     # calculate the coefficient of logit model, then calc
ulate the marginal effect, put the data into the empty matr
ix
+   betaend <- as.matrix(optim(c(0,0,0,0), logit_log, x =
X4, y = Y, method = "BFGS")$par)
+   r <- as.matrix(est_5(mean(X4%*%betaend))*t(betaend))
+   me_bootlogit <- rbind(me_bootlogit, r)
+ }
> # calculate sd
> sd_boot_logit <- cbind(sd(me_bootlogit[,1]),sd(me_bootl
ogit[,2]), sd(me_bootlogit[,3]), sd(me_bootlogit[,4]))
> # output:
> # 0.06246317
> # 0.02616447
> # 0.01902341
> # 0.0006190331
>
> # repeat the process above for probit model
> me_bootprobit <- matrix(c(0,0,0,0), nrow = 1, ncol = 4)
> loop4 = 450
> for(r in 1:loop4){
```

```
+   i <- sample(1:10000, size = 10000, replace = TRUE)
+   Y <- as.matrix(ydum[i,])
+   X0 <- as.vector(XY[i,2])
+   X1 <- as.vector(XY[i,3])
+   X2 <- as.vector(XY[i,4])
+   X3 <- as.vector(XY[i,5])
+   X4 <- as.matrix(cbind(X0,X1,X2,X3))
+   betaend2 <- as.matrix(optim(c(0,0,0,0), est_4, x = X4,
 y = Y, method = "BFGS")$par)
+   r2 <- as.matrix(dnorm(mean(X4%*%betaend2))*t(betaend
2))
+   me_bootprobit <- rbind(me_bootprobit, r2)
+ }
> sd_boot_probit <- cbind(sd(me_bootprobit[,1]),sd(me_boo
tprobit[,2]), sd(me_bootprobit[,3]), sd(me_bootprobit[,
4]))
> # output:
> # 0.05492565
> # 0.02314306
> # 0.01675942
> # 0.0005589166
```