Assignment 6: Entry and Exist

Kohei Kawaguchi

2019/4/25

Simulate data

In this assignment, we consider a Berry-type entry model. Suppose that there are M markets indexed by $m = 1, \cdots, M$. In each market, there are N_m potential entrants such that $N_m \leq \overline{N}$. Let x_m be the K dimensional market attributes and z_{im} be the L dimensional potential entrant attributes. The size of Monte Carlo simulations in the estimation is R.

1. Set the constants as follows:

```
# set the seed
set.seed(1)
# number of markets
M <- 100
# the upper bound of the number of potential entrants
N <- 10
# the dimension of market attributes
K <- 2
# the dimension of potential entrant attributes
L <- 2
# the number of Monte Carlo simulations
R <- 100
```

The payoff of entrant i in market m is:

$$\pi_{im}(y_m) = x_m'\beta - \delta \ln\left(\sum_{i=1}^{N_m} y_{im}\right) + z_{im}'\alpha + \sqrt{1-\rho^2}\nu_{im} + \rho\epsilon_m,$$

where $y_{im} \in \{0, 1\}$ is the indicator for entrant i in market m to enter the market, and ν_{im} and ϵ_m are entrant- and market-specific idiosyncratic shocks that are drawn from an i.i.d. standard normal distribution. In each market, all the attributes and idiosyncratic shocks are observed by the potential entrants. N_m, x_m, z_{im}, and y_m are observed to econometrician but ν_{im} and ϵ_m are not.

2. Set the parameters as follows:

```
# parameters of interest
beta <- abs(rnorm(K)); beta
```

```
## [1] 0.6264538 0.1836433
```

```
alpha <- abs(rnorm(L)); alpha
```

```
## [1] 0.8356286 1.5952808
```

```
delta <- 1; delta
```

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

```
rho <- abs(rnorm(1)); rho
```

```
## [1] 0.3295078
```

```r
# auxiliary parameters
x_mu <- 1
x_sd <- 3
z_mu <- 0
z_sd <- 4
```

3. Draw exogenous variables as follows:

```r
# number of potential entrants
E <- purrr::rdunif(M, 1, N); E
```

```
##   [1]  3  2  7  4  8  5  8 10  4  8 10  3  7  2  3  4  1  4  9  4  5  6  5
##  [24]  2  9  7  8  2  8  5  9  7  8  6  6  8  1  5  8  7  5  9  5  3  1  1
##  [47]  4  6  7  5 10  3  5  4  7  3  5  8  1  9  4  9  4  4  5  9  9  4  8
##  [70] 10  5  8  4  4  8  3  8  2  3  2  3  1  7  9  8  8  5  5  9  7  7  4
##  [93]  3 10  7  3  2  5 10  6
```

```r
# market attributes
X <- matrix(
  rnorm(M * K, x_mu, x_sd),
  nrow = M
)
colnames(X) <- paste("x", 1:K, sep = "_")
X
```

```
##               x_1          x_2
##   [1,]  6.94119970 -2.225576890
##   [2,] -0.10166443  4.000086411
##   [3,] -2.13240388 -0.863800084
##   [4,]  2.70915888 -3.153280542
##   [5,]  0.59483619  6.607871867
##   [6,]  8.20485328  2.275301132
##   [7,]  0.88227999  0.284058697
##   [8,]  3.06921809  4.175449146
##   [9,]  1.08400648  3.659267954
##  [10,] -1.22981963 -0.857729145
##  [11,]  1.56637690  7.618307394
##  [12,] -4.41487589  0.234918910
##  [13,]  5.39666458 -3.273483951
##  [14,]  1.45976001  0.566801194
##  [15,]  7.51783501  1.622615018
##  [16,]  2.42652859  7.923935197
##  [17,] -1.12983929  1.317407104
##  [18,]  2.83217906  2.370996416
##  [19,] -1.80229289  0.768541194
##  [20,] -2.76090020 -0.002002527
##  [21,]  1.87433871  0.895821915
##  [22,] -0.32987562  3.362918817
##  [23,]  1.00331605  7.225735026
##  [24,]  1.22302397  4.082177316
##  [25,] -0.76856284  4.623725195
##  [26,] -0.70600620 -2.693970265
##  [27,]  0.59446415  3.951686710
##  [28,]  4.53426099  1.659774411
##  [29,] -3.57070040 -3.401750087
##  [30,]  2.78183856  2.563068228
```

```
## [31,]  1.99885111  0.523736186
## [32,]  4.18929951  5.393761936
## [33,]  0.08744823 -1.298245999
## [34,]  2.11005643 -0.290635262
## [35,]  1.80129637 -1.778328492
## [36,] -0.62756009  0.468688116
## [37,]  4.62360342  2.206035338
## [38,]  4.48120785 -1.195244519
## [39,]  3.10064095  3.491119504
## [40,]  5.76050036 -2.624248359
## [41,]  2.67545928 -2.143953238
## [42,] -2.82977663  5.323473121
## [43,] -0.71979624 -2.047542396
## [44,] -2.67383784  2.235924137
## [45,] -0.42020191 -0.143228153
## [46,] -0.86110003  2.228205519
## [47,]  1.12634762  6.066619859
## [48,] -1.73276495  5.759765300
## [49,]  1.47408632  0.007276598
## [50,] -0.96375393 -5.855706606
## [51,]  6.30186181  8.492984770
## [52,]  3.15012243  3.001198500
## [53,]  3.73052269  2.623982008
## [54,]  2.15255607  0.959801431
## [55,]  6.04652824  2.530325269
## [56,] -0.90720936  0.506872505
## [57,] -0.38493419  2.262083930
## [58,]  5.29684672 -0.200740232
## [59,] -0.95208906 -3.110623633
## [60,]  0.37785777  3.963514802
## [61,] -0.17842379  5.559235076
## [62,]  0.04002139  0.073778292
## [63,]  0.16266009 -2.759869267
## [64,]  2.48256499  2.926723917
## [65,]  0.46800855  0.865872589
## [66,] -0.51787239 -4.199655220
## [67,]  5.02911648  1.006395579
## [68,]  0.35626177 -0.890901002
## [69,]  0.46133041 -0.022905740
## [70,]  0.69942778 -2.469717088
## [71,]  3.13799892  6.409425724
## [72,]  0.77930679  0.006603891
## [73,]  0.88709749 -3.816540237
## [74,] -1.04498144  1.591580316
## [75,]  0.02718918  1.789526939
## [76,]  1.18048132 -1.957480101
## [77,] -0.76668346 -7.666762015
## [78,]  2.59448858 -0.921445108
## [79,] -3.55518225  2.711522908
## [80,]  1.91967358  0.820830172
## [81,] -3.60934947  0.705463768
## [82,]  0.09707162  2.682462186
## [83,] -0.58483971 -2.559375916
## [84,] -0.95628434  4.290331133
```

```
##  [85,]  0.82930967  0.983967915
##  [86,] -4.74307828  3.121932002
##  [87,]  4.52974994  4.102323204
##  [88,] -3.99491731  1.670441245
##  [89,] -0.39059120 -1.636122839
##  [90,] -2.34776032  4.488893668
##  [91,] -1.25245700 -5.000494834
##  [92,]  7.26149964 -0.634372220
##  [93,]  1.05218686  0.232987873
##  [94,] -2.85890159  0.501636890
##  [95,] -3.92181660  4.061391726
##  [96,]  2.35056130  1.408665679
##  [97,]  0.94432050  2.221502810
##  [98,]  0.04579488  0.791035561
##  [99,] -1.78808644  0.257006975
## [100,] -3.46238093  3.086652420
```

```r
# entrant attributes
Z <-
  foreach (m = 1:M) %dopar% {
    Z_m <- matrix(
      rnorm(E[m] * L, z_mu, z_sd),
      nrow = E[m]
    )
    colnames(Z_m) <- paste("z", 1:L, sep = "_")
    return(Z_m)
  }
Z[[1]]
```

```
##            z_1       z_2
## [1,] -4.106418 -2.950715
## [2,]  2.239022  5.221025
## [3,]  7.472220  1.511501
```

```r
# unobserved market attributes
EP <- matrix(
  rnorm(M),
  nrow = M
)
EP
```

```
##                [,1]
##   [1,]  1.146228357
##   [2,] -2.403096215
##   [3,]  0.572739555
##   [4,]  0.374724407
##   [5,] -0.425267722
##   [6,]  0.951012808
##   [7,] -0.389237182
##   [8,] -0.284330662
##   [9,]  0.857409778
##  [10,]  1.719627299
##  [11,]  0.270054901
##  [12,] -0.422184010
##  [13,] -1.189113295
##  [14,] -0.331032979
```

```
## [15,] -0.939829327
## [16,] -0.258932583
## [17,]  0.394379168
## [18,] -0.851857092
## [19,]  2.649166881
## [20,]  0.156011676
## [21,]  1.130207267
## [22,] -2.289123980
## [23,]  0.741001157
## [24,] -1.316245160
## [25,]  0.919803678
## [26,]  0.398130155
## [27,] -0.407528579
## [28,]  1.324258630
## [29,] -0.701231669
## [30,] -0.580614304
## [31,] -1.001072181
## [32,] -0.668178607
## [33,]  0.945184953
## [34,]  0.433702150
## [35,]  1.005159218
## [36,] -0.390118664
## [37,]  0.376370292
## [38,]  0.244164924
## [39,] -1.426257342
## [40,]  1.778429287
## [41,]  0.134447661
## [42,]  0.765598999
## [43,]  0.955136677
## [44,] -0.050565701
## [45,] -0.305815420
## [46,]  0.893673702
## [47,] -1.047298149
## [48,]  1.971337386
## [49,] -0.383632106
## [50,]  1.654145302
## [51,]  1.512212694
## [52,]  0.082965734
## [53,]  0.567220915
## [54,] -1.024548480
## [55,]  0.323006503
## [56,]  1.043612458
## [57,]  0.099078487
## [58,] -0.454136909
## [59,] -0.655781852
## [60,] -0.035922423
## [61,]  1.069161461
## [62,] -0.483974930
## [63,] -0.121010111
## [64,] -1.294140004
## [65,]  0.494312836
## [66,]  1.307901520
## [67,]  1.497041009
## [68,]  0.814702731
```

```
##  [69,] -1.869788790
##  [70,]  0.482029504
##  [71,]  0.456135603
##  [72,] -0.353400286
##  [73,]  0.170489471
##  [74,] -0.864035954
##  [75,]  0.679230774
##  [76,] -0.327101015
##  [77,] -1.569082185
##  [78,] -0.367450756
##  [79,]  1.364434929
##  [80,] -0.334281365
##  [81,]  0.732750042
##  [82,]  0.946585640
##  [83,]  0.004398704
##  [84,] -0.352322306
##  [85,] -0.529695509
##  [86,]  0.739589226
##  [87,] -1.063457415
##  [88,]  0.246210844
##  [89,] -0.289499367
##  [90,] -2.264889356
##  [91,] -1.408850456
##  [92,]  0.916019329
##  [93,] -0.191278951
##  [94,]  0.803283216
##  [95,]  1.887474463
##  [96,]  1.473881181
##  [97,]  0.677268492
##  [98,]  0.379962687
##  [99,] -0.192798426
## [100,]  1.577891795
```

```r
# unobserved entrant attributes
NU <-
  foreach (m = 1:M) %dopar% {
    NU_m <- matrix(
      rnorm(E[m]),
      nrow = E[m]
    )
    return(NU_m)
  }
NU[[1]]
```

```
##            [,1]
## [1,] -0.8566941
## [2,]  1.0451666
## [3,]  1.2279516
```

4. Write a function `compute_payoff(y_m, X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho)` that returns the vector of payoffs of the potential entrants when the vector of entry decisions is `y_m`.

```r
# compute payoff
compute_payoff <-
  function(y_m, X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho) {
    N_m <- length(y_m)
```

```r
    if (sum(y_m) == 0) {
      payoff_m <- 0 * y_m
    } else {
      payoff_m <- matrix(rep(1, N_m)) %*% (X_m %*% beta - delta * log(sum(y_m)) + rho * EP_m) + Z_m %*%
      payoff_m <- payoff_m * y_m
    }
    return(payoff_m)
  }
m <- 1
N_m <- dim(Z[[m]])[1]
y_m <- as.matrix(rep(1, N_m))
y_m[length(y_m)] <- 0
X_m <- X[m, , drop = FALSE]
Z_m <- Z[[m]]
EP_m <- EP[m, , drop = FALSE]
NU_m <- NU[[m]]
compute_payoff(y_m, X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho)
```

```
##           [,1]
## [1,] -5.323337
## [2,] 14.810961
## [3,]  0.000000
```

5. Assume that the order of entry is predetermined. Assume that the potential entrants sequentially decide entry according to the order of the payoff excluding the competitive effects, i.e.:

$$x_m'\beta + z_{im}'\alpha + \sqrt{1-\rho^2}\nu_{im} + \rho\epsilon_m.$$

   Write a function `compute_sequential_entry(X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho)` that returns the equilibrium vector of entry at a market.

```r
# compute sequential entry
compute_sequential_entry <-
  function(X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho) {
    N_m <- dim(Z_m)[1]
    y_m <- rep(0, N_m)
    N_m <- dim(Z_m)[1]
    # compute the baseline payoff
    payoff_baseline <- matrix(rep(1, N_m)) %*% (X_m %*% beta + rho * EP_m) + Z_m %*% alpha + sqrt(1 - rh
    # baseline payoff ranking
    ranking <- rank(-payoff_baseline)
    # initial y_m
    y_m <- rep(0, N_m)
    for (index in 1:N_m) {
      i <- which(ranking == index)
      y_m0 <- y_m
      y_m0[i] <- 1
      payoff <- compute_payoff(y_m0, X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho)
      payoff_i <- payoff[i]
      y_m[i] <- as.integer(payoff_i >= 0)
    }
    y_m <- as.matrix(y_m)
    return(y_m)
  }
compute_sequential_entry(X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho)
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    1
```

6. Next, assume $\rho = 0$. Assume that potential entrants simultaneously decide entry. Write a function `compute_simultaneous_entry(X_m, Z_m, EP_m, NU_m, beta, alpha, delta)` that returns the equilibrium vector of entry at a market.

```
# compute simultaneous entry
compute_simultaneous_entry <-
  function(X_m, Z_m, EP_m, NU_m, beta, alpha, delta) {
    N_m <- dim(Z_m)[1]
    y_m <- rep(1, N_m)
    y_m_old <- rep(0, N_m)
    while (!identical(y_m, y_m_old)) {
      y_m_old <- y_m
      for (i in 1:N_m) {
        # counterfactual choice
        y_m0 <- y_m
        y_m0[i] <- 1 - y_m0[i]
        # payoffs
        payoff <- compute_payoff(y_m, X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho = 0)
        payoff0 <- compute_payoff(y_m0, X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho = 0)
        payoff_i <- payoff[i]
        payoff_i0 <- payoff0[i]
        # check improvement
        if (payoff_i0 > payoff_i) {
          y_m <- y_m0
        }
      }
    }
    y_m <- as.matrix(y_m)
    return(y_m)
  }
compute_simultaneous_entry(X_m, Z_m, EP_m, NU_m, beta, alpha, delta)
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    1
```

7. Write a function `compute_sequential_entry_across_markets(X, Z, EP, NU, beta, alpha, delta, rho)` compute the equilibrium entry vectors under the assumption of sequential entry. The output should be a list of entry vectors across markets. Write a function to compute the equilibrium payoffs across markets, `compute_payoff_across_markets(Y, X, Z, EP, NU, beta, alpha, delta, rho)` and check that the payoffs under the equilibrium entry vectors are non-negative. Otherwise, there are some bugs in the code.

```
# compute payoff across markets
compute_payoff_across_markets <-
  function(Y, X, Z, EP, NU, beta, alpha, delta, rho) {
    payoff <-
      foreach (m = 1:length(Y)) %dopar% {
        y_m <- Y[[m]]
        # extract
```

```r
        X_m <- X[m, , drop = FALSE]
        Z_m <- Z[[m]]
        EP_m <- EP[m, , drop = FALSE]
        NU_m <- NU[[m]]
        payoff <- compute_payoff(y_m, X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho)
        return(payoff)
      }
    return(payoff)
  }
# compute sequential entry across markets
compute_sequential_entry_across_markets <-
  function(X, Z, EP, NU, beta, alpha, delta, rho) {
    Y <-
      foreach (m = 1:length(Z)) %do% {
        # extract
        X_m <- X[m, , drop = FALSE]
        Z_m <- Z[[m]]
        EP_m <- EP[m, , drop = FALSE]
        NU_m <- NU[[m]]
        # compute entry
        y_m <- compute_sequential_entry(X_m, Z_m, EP_m, NU_m, beta, alpha, delta, rho)
        # return
        return(y_m)
      }
    return(Y)
  }

Y_sequential <-
  compute_sequential_entry_across_markets(X, Z, EP, NU, beta, alpha, delta, rho)
Y_sequential[[1]]
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    1
```

```r
Y_sequential[[M]]
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    0
## [4,]    0
## [5,]    1
## [6,]    0
```

```r
payoff_sequential <-
  compute_payoff_across_markets(Y_sequential, X, Z, EP, NU, beta, alpha, delta, rho)
min(unlist(payoff_sequential))
```

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

8. Write a function `compute_simultaneous_entry_across_markets(X, Z, EP, NU, beta, alpha, delta, rho = 0)` compute the equilibrium entry vectors under the assumption of simultaneous entry. The output should be a list of entry vectors across markets. Check that the payoffs under the equilibrium entry vectors are non-negative. Otherwise, there are some bugs in the code. I also

recommend to write this function with

```
# compute simultaneous entry across markets
compute_simultaneous_entry_across_markets <-
  function(X, Z, EP, NU, beta, alpha, delta) {
    Y <-
      foreach (m = 1:length(Z)) %do% {
        # extract
        X_m <- X[m, , drop = FALSE]
        Z_m <- Z[[m]]
        EP_m <- EP[m, , drop = FALSE]
        NU_m <- NU[[m]]
        # compute entry
        y_m <- compute_simultaneous_entry(X_m, Z_m, EP_m, NU_m, beta, alpha, delta)
        # return
        return(y_m)
      }
    return(Y)
  }

# compute simultaneous entry across markets
Y_simultaneous <-
  compute_simultaneous_entry_across_markets(X, Z, EP, NU, beta, alpha, delta)
Y_simultaneous[[1]]
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    1
```

```
Y_simultaneous[[M]]
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    0
## [4,]    0
## [5,]    1
## [6,]    0
```

```
payoff_simultaneous <-
  compute_payoff_across_markets(Y_simultaneous, X, Z, EP, NU, beta, alpha, delta, rho = 0)
min(unlist(payoff_simultaneous))
```

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

Estimate the parameters

We estimate the parameters by matching the actual and predicted number of entrants in each market. To do so, we simulate the model for R times. Under the assumption of the sequential entry, we can uniquely predict the equilibrium identify of the entrants. So, we consider the following objective function:

$$\frac{1}{RM}\sum_{r=1}^{R}\sum_{m=1}^{M}\left[\sum_{i=1}^{N_m}|y_{im}-y_{im}^{(r)}|\right]^2,$$

where $y_{im}^{(r)}$ is the entry decision in r-th simulation. On the other hand, under the assumption of the simultaneous entry, we can only uniquely predict the equilibrium number of the entrants. So, we consider the following objective function:

$$\frac{1}{RM}\sum_{r=1}^{R}\sum_{m=1}^{M}\left[\sum_{i=1}^{N_m}(y_{im}-y_{im}^{(r)})\right]^2,$$

1. Draw R unobserved shocks:

```
set.seed(1)
# unobserved market attributes
EP_mc <-
  foreach (r = 1:R) %dopar% {
    EP <- matrix(
      rnorm(M),
      nrow = M
    )
    return(EP)
}
# unobserved entrant attributes
NU_mc <-
  foreach (r = 1:R) %dopar% {
  NU <-
    foreach (m = 1:M) %do% {
      NU_m <- matrix(
        rnorm(E[m]),
        nrow = E[m]
      )
      return(NU_m)
    }
  return(NU)
}
```

2. Write a function `compute_monte_carlo_sequential_entry(X, Z, EP_mc, NU_mc, beta, alpha, delta, rho)` that returns the Monte Carlo simulation. Then, write function `compute_objective_sequential_entry(Y, X, Z, EP_mc, NU_mc, theta)` that calls`compute_monte_carlo_sequential_entry` and returns the value of the objective function given data and parameters under the assumption of sequential entry.

```
# compute monte carlo simulations of sequential entry model
compute_monte_carlo_sequential_entry <-
  function(X, Z, EP_mc, NU_mc,
           beta, alpha, delta, rho) {
    # Monte Carlo Simulation
    Y_mc <-
      foreach (r = 1:length(EP_mc)) %dopar% {
        # extract
        EP_r <- EP_mc[[r]]
        NU_r <- NU_mc[[r]]
        Y_r <-
          compute_sequential_entry_across_markets(X, Z, EP_r, NU_r, beta, alpha, delta, rho)
        return(Y_r)
      }
    # return
    return(Y_mc)
  }
# compute the objective function of sequential entry model
```

```r
compute_objective_sequential_entry <-
  function(Y, X, Z, EP_mc, NU_mc, theta) {
    # extract parameters
    K <- dim(X)[2]
    L <- dim(Z[[1]])[2]
    beta <- theta[1:K]
    alpha <- theta[(K + 1):(K + L)]
    delta <- theta[K + L + 1]
    rho <- theta[K + L + 2]
    # compute monte carlo simulations of sequential entry model
    Y_mc <- compute_monte_carlo_sequential_entry(X, Z, EP_mc, NU_mc,
                                                 beta, alpha, delta, rho)
    # compute the square difference
    objective <-
      foreach (r = 1:length(EP_mc), .combine = "rbind") %dopar% {
        Y_mc_r <- Y_mc[[r]]
        diff_r <- purrr::map2(Y_mc_r, Y, `-`) %>%
          purrr::map(., ~ sum(abs(.))) %>%
          purrr::map(., ~ .^2) %>%
          purrr::reduce(`+`)
        diff_r <- diff_r / length(Y_mc_r)
        return(diff_r)
      }
    objective <- mean(objective)
    # return
    return(objective)
  }

# sequential entry
theta <- theta_sequential <-
  c(beta, alpha, delta, rho)
Y <- Y_sequential
# compute monte carlo simulations
Y_mc <-
  compute_monte_carlo_sequential_entry(
    X, Z, EP_mc, NU_mc, beta, alpha, delta, rho)
Y_mc[[1]][[1]]
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    1
```

```r
# compute objective function
compute_objective_sequential_entry(Y, X, Z, EP_mc, NU_mc, theta)
```

```
## [1] 0.4458
```

3. Write a function `compute_objective_simultaneous_entry(Y, X, Z, EP_mc, NU_mc, theta)` that returns the value of the objective function given data and parameters under the assumption of simultaneous entry.

```r
# compute monte carlo simulations of simultaneous entry model
compute_monte_carlo_simultaneous_entry <-
  function(X, Z, EP_mc, NU_mc,
           beta, alpha, delta) {
```

```r
    # Monte Carlo Simulation
    Y_mc <-
      foreach (r = 1:length(EP_mc)) %dopar% {
        # extract
        EP_r <- EP_mc[[r]]
        NU_r <- NU_mc[[r]]
        Y_r <-
          compute_simultaneous_entry_across_markets(X, Z, EP_r, NU_r, beta, alpha, delta)
        return(Y_r)
      }
    return(Y_mc)
  }

# compute the objective function of simultaneous entry model
compute_objective_simultaneous_entry <-
  function(Y, X, Z, EP_mc, NU_mc, theta) {
    # extract parameters
    K <- dim(X)[2]
    L <- dim(Z[[1]])[2]
    beta <- theta[1:K]
    alpha <- theta[(K + 1):(K + L)]
    delta <- theta[K + L + 1]
    # Monte Carlo Simulation
    Y_mc <-
      compute_monte_carlo_simultaneous_entry(
        X, Z, EP_mc, NU_mc, beta, alpha, delta)
    # compute the square difference
    objective <-
      foreach (r = 1:length(EP_mc), .combine = "rbind") %dopar% {
        Y_mc_r <- Y_mc[[r]]
        diff_r <- purrr::map2(Y_mc_r, Y, `-`) %>%
          purrr::map(., sum) %>%
          purrr::map(., ~ .^2) %>%
          purrr::reduce(`+`)
        diff_r <- diff_r / length(Y_mc_r)
        return(diff_r)
      }
    objective <- mean(objective)
    # return
    return(objective)
  }

# simultaneous entry
theta <- theta_simultaneous <-
  c(beta, alpha, delta)
Y <- Y_simultaneous
# compute monte carlo simulations
Y_mc <- compute_monte_carlo_simultaneous_entry(X, Z, EP_mc, NU_mc, beta, alpha, delta)
Y_mc[[1]][[1]]
```

```
##      [,1]
## [1,]    0
## [2,]    1
## [3,]    1
```

```r
# compute objective function
compute_objective_simultaneous_entry(Y, X, Z, EP_mc, NU_mc, theta)
```

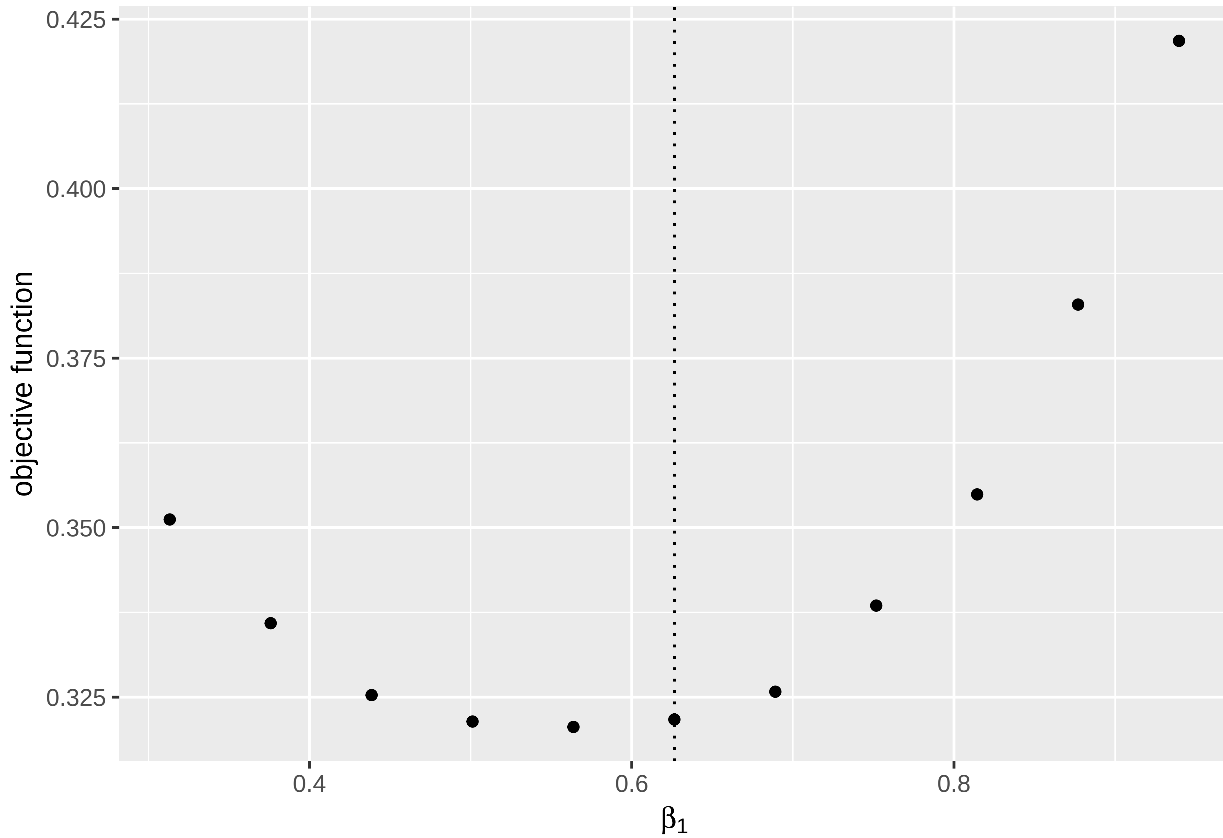
```
## [1] 0.2904
```

4. Check the value of the objective function around the true parameter under the assumption of the sequential entry.

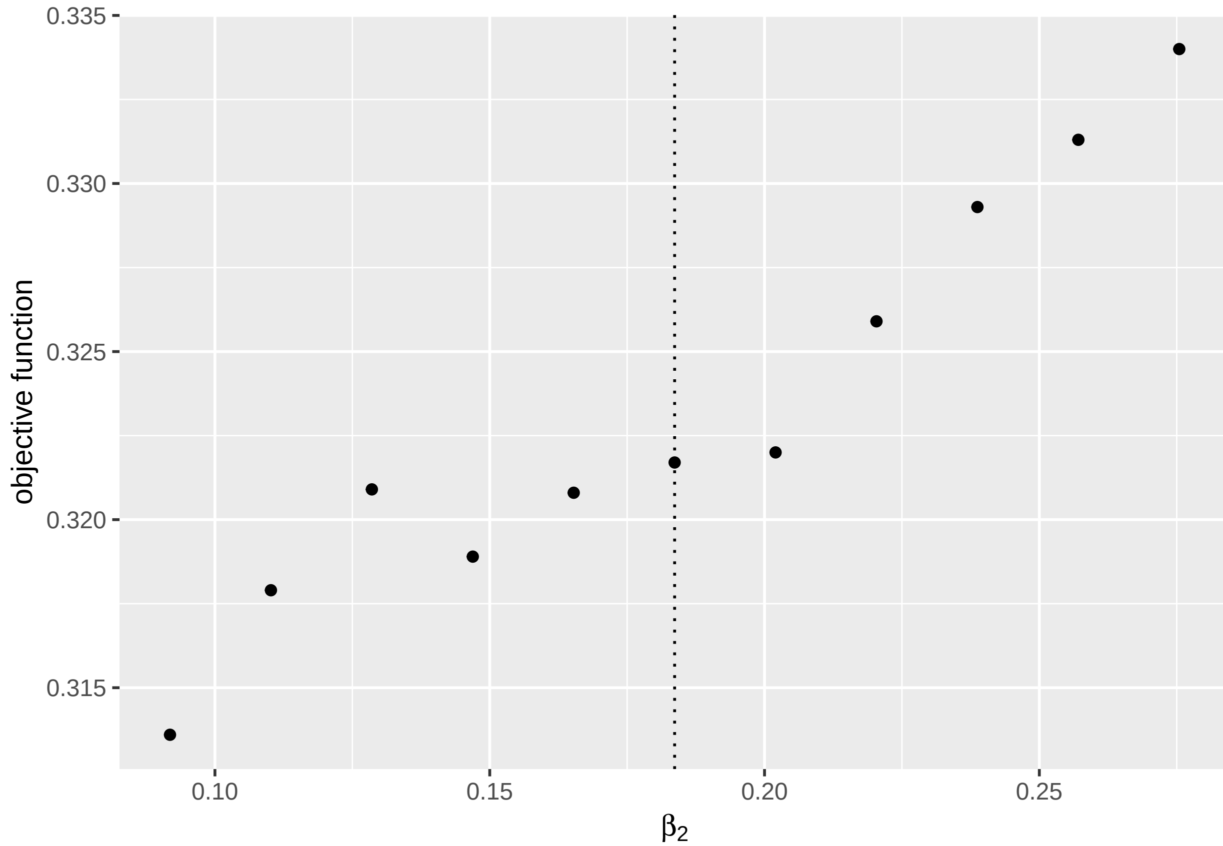
```r
# sequential entry
theta <- theta_sequential <-
  c(beta, alpha, delta, rho)
Y <- Y_sequential
model <- compute_sequential_entry_across_markets
label <- c(paste("\\beta_", 1:K, sep = ""),
           paste("\\alpha_", 1:L, sep = ""),
           "\\delta",
           "\\rho")
label <- paste("$", label, "$", sep = "")
# compute the graph
graph <- foreach (i = 1:length(theta)) %do% {
  theta_i <- theta[i]
  theta_i_list <- theta_i * seq(0.5, 1.5, by = 0.1)
  objective_i <-
    foreach (j = 1:length(theta_i_list),
             .combine = "rbind") %do% {
               theta_ij <- theta_i_list[j]
               theta_j <- theta
               theta_j[i] <- theta_ij
               objective_ij <-
                 compute_objective_sequential_entry(Y, X, Z, EP_mc, NU_mc, theta_j)
               return(objective_ij)
             }
  df_graph <- data.frame(x = theta_i_list, y = objective_i)
  g <- ggplot(data = df_graph, aes(x = x, y = y)) +
    geom_point() +
    geom_vline(xintercept = theta_i, linetype = "dotted") +
    ylab("objective function") + xlab(TeX(label[i]))
  return(g)
}
save(graph, file = "data/A6_graph_sequential.RData")
```

```r
load(file = "data/A6_graph_sequential.RData")
graph
```

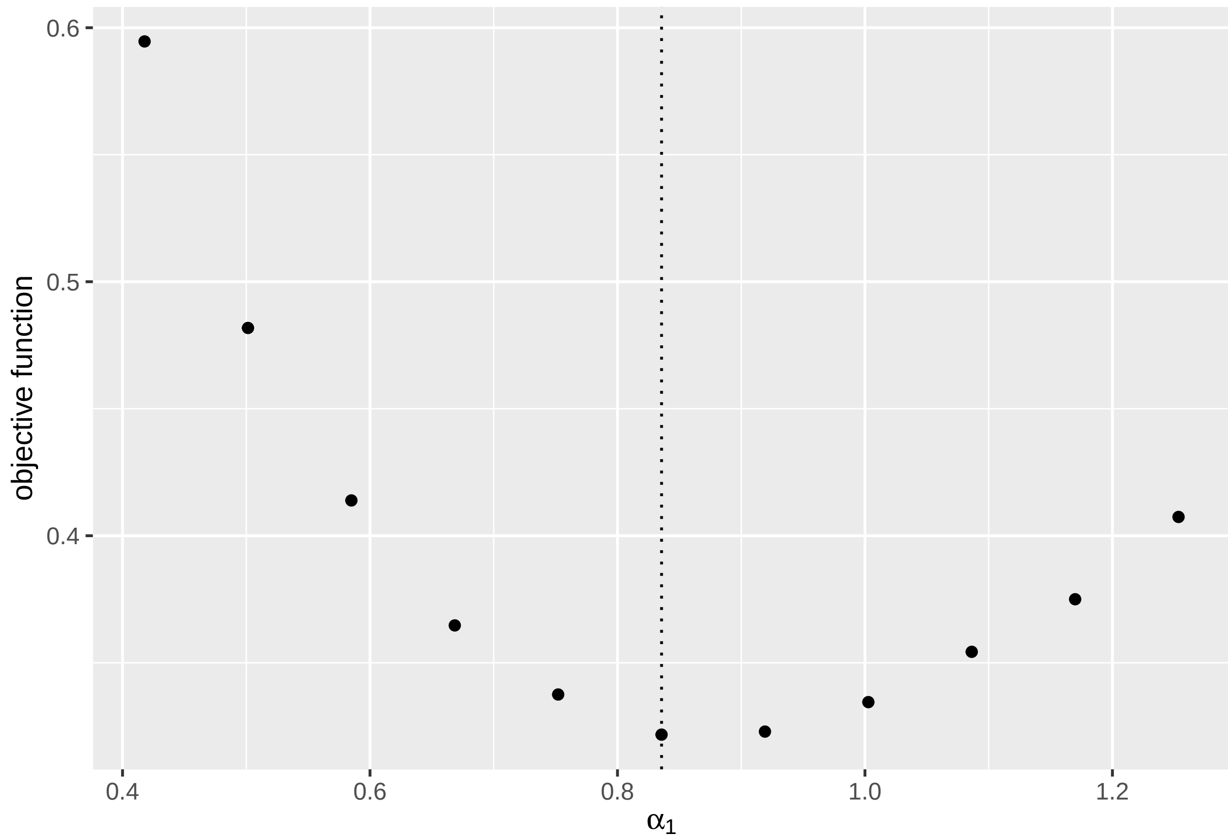
```
## [[1]]
```



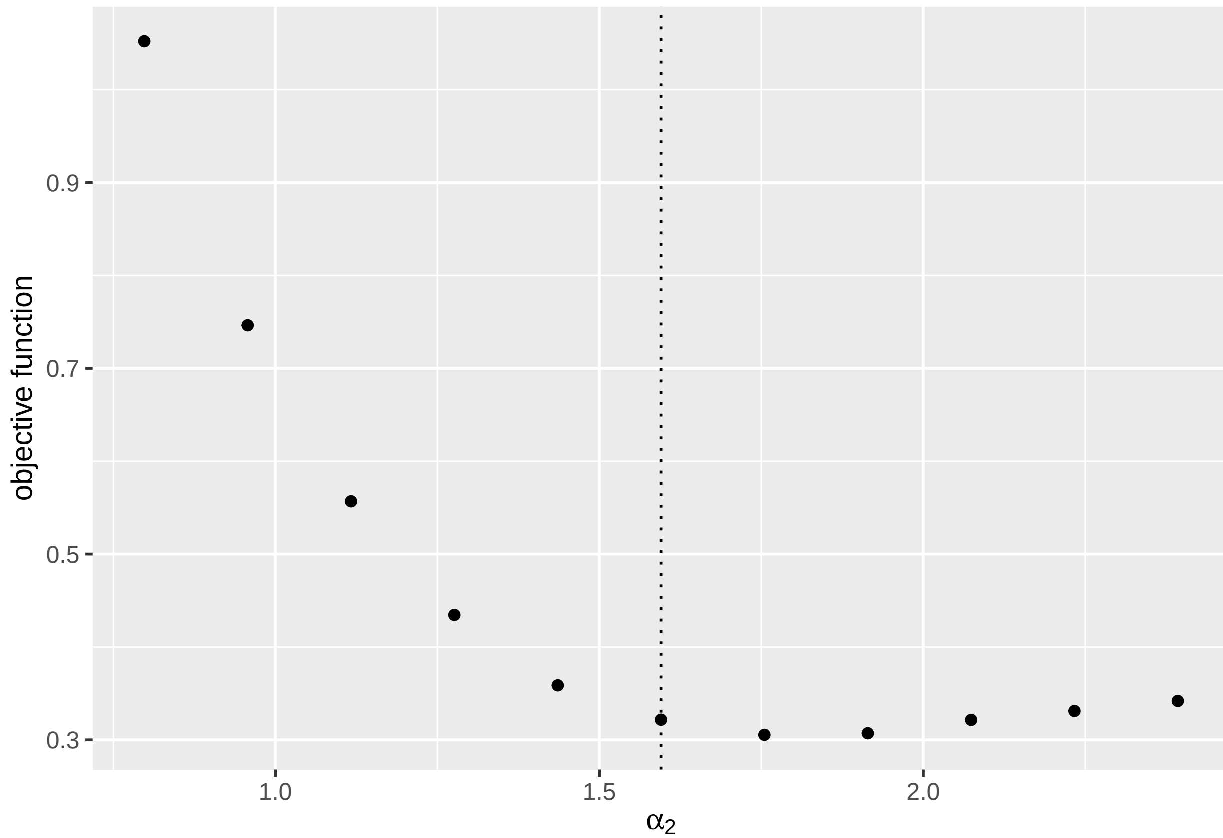

```
##
## [[2]]
```


0.335
0.330
0.325
0.320
0.315
objective function
0.10
0.15
0.20
0.25
β2

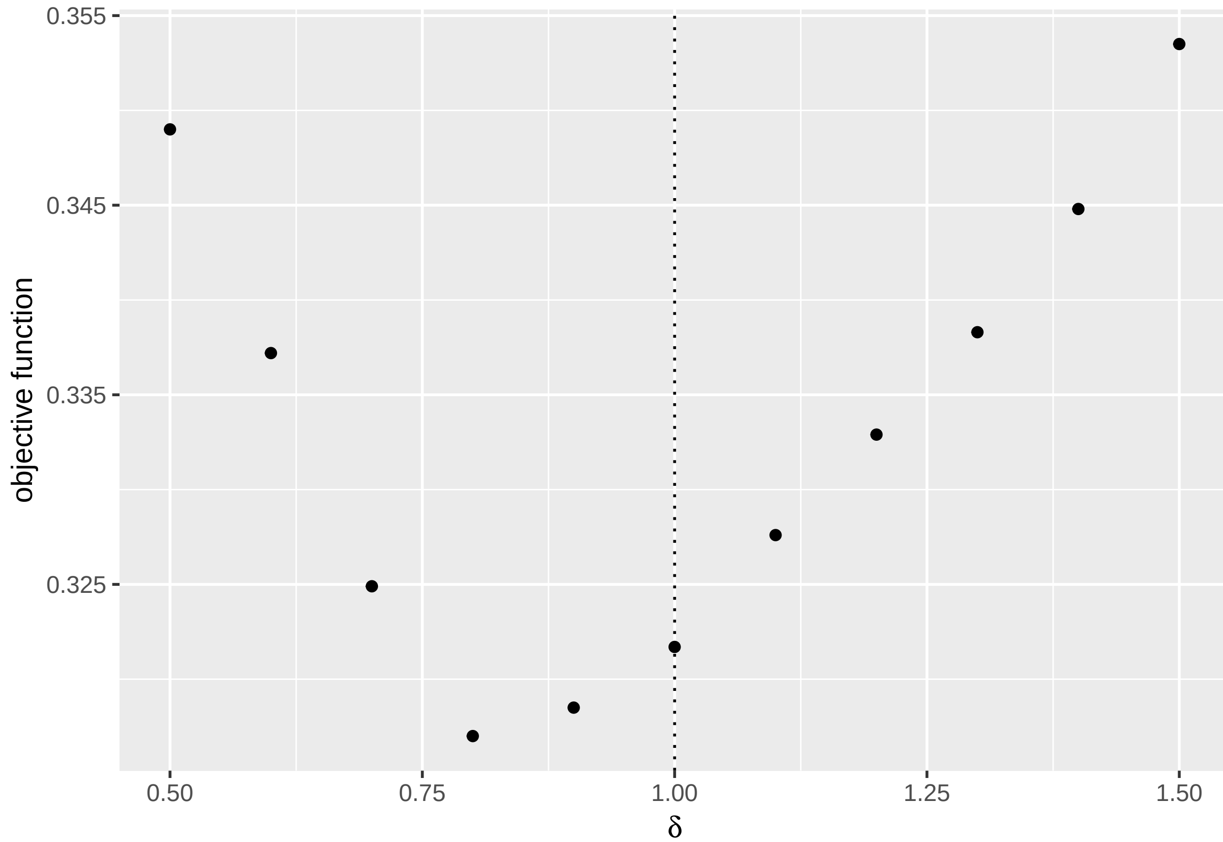
```
##
## [[3]]
```



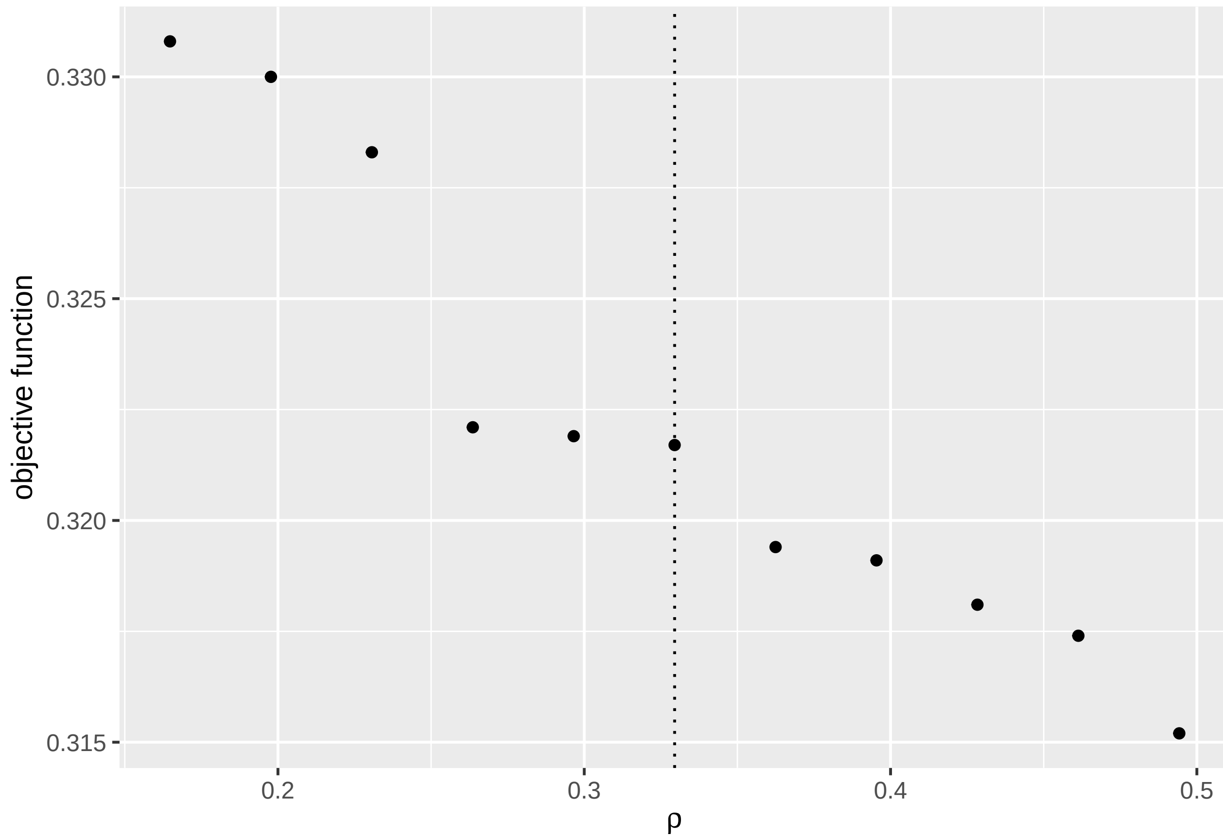

```
##
## [[4]]
```




```
##
## [[5]]
```




```
##
## [[6]]
```



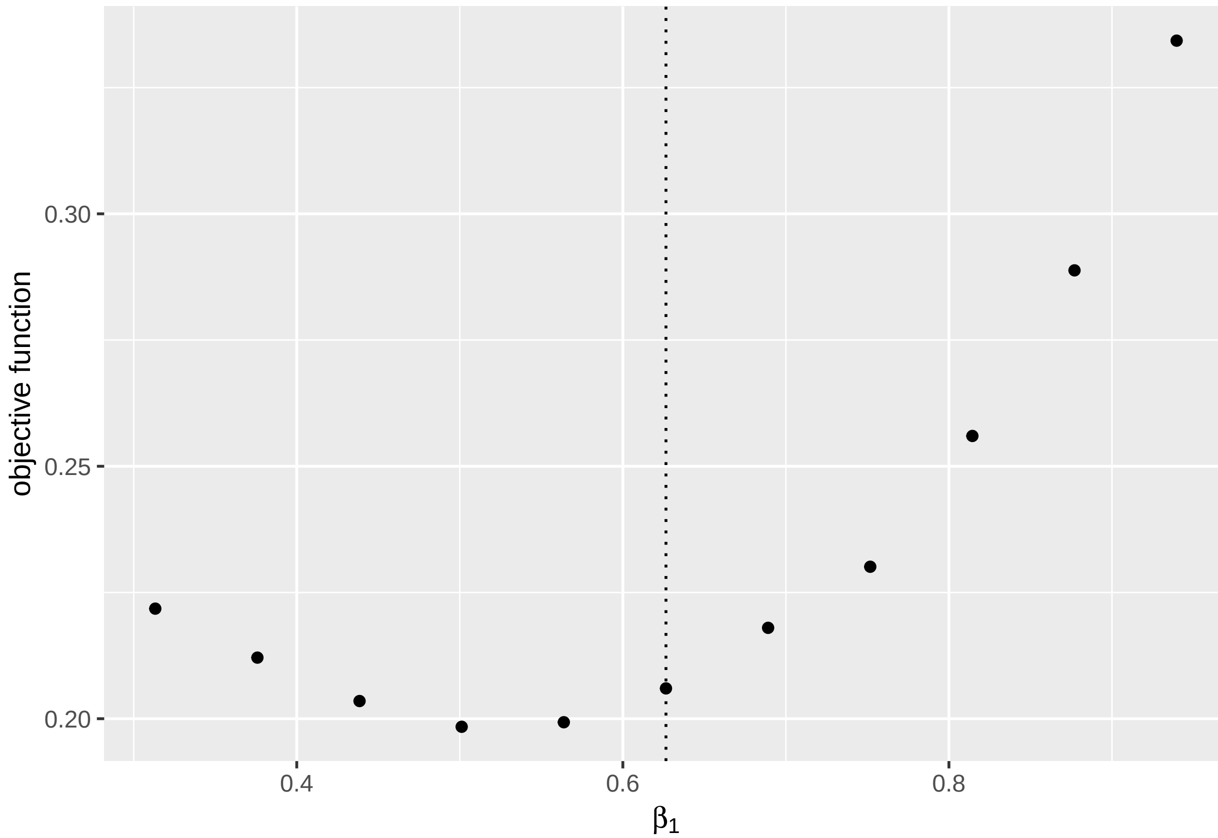
5. Check the value of the objective function around the true parameter under the assumption of the simultaneous entry.

```r
# simultaneous entry
theta <- theta_simultaneous <-
  c(beta, alpha, delta)
Y <- Y_simultaneous
model <- compute_simultaneous_entry_across_markets
label <- c(paste("\\beta_", 1:K, sep = ""),
           paste("\\alpha_", 1:L, sep = ""),
           "\\delta")
label <- paste("$", label, "$", sep = "")
# compute the graph
graph <- foreach (i = 1:length(theta)) %do% {
  theta_i <- theta[i]
  theta_i_list <- theta_i * seq(0.5, 1.5, by = 0.1)
  objective_i <-
    foreach (j = 1:length(theta_i_list),
             .combine = "rbind") %do% {
               theta_ij <- theta_i_list[j]
               theta_j <- theta
               theta_j[i] <- theta_ij
               objective_ij <-
                 compute_objective_simultaneous_entry(Y, X, Z, EP_mc, NU_mc, theta_j)
               return(objective_ij)
             }
  df_graph <- data.frame(x = theta_i_list, y = objective_i)
  g <- ggplot(data = df_graph, aes(x = x, y = y)) +
```

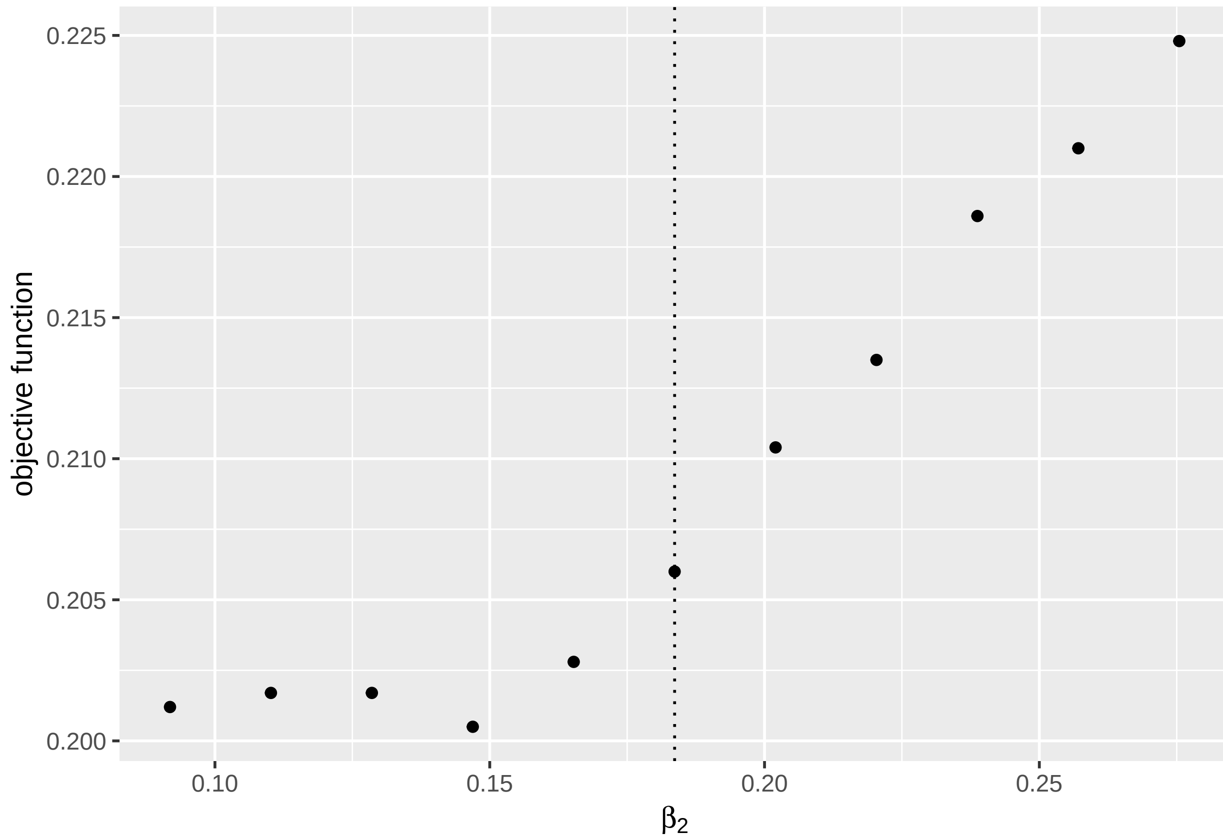
```r
    geom_point() +
    geom_vline(xintercept = theta_i, linetype = "dotted") +
    ylab("objective function") + xlab(TeX(label[i]))
  return(g)
}
save(graph, file = "data/A6_graph_simultaneous.RData")
```

```r
load(file = "data/A6_graph_simultaneous.RData")
graph
```

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

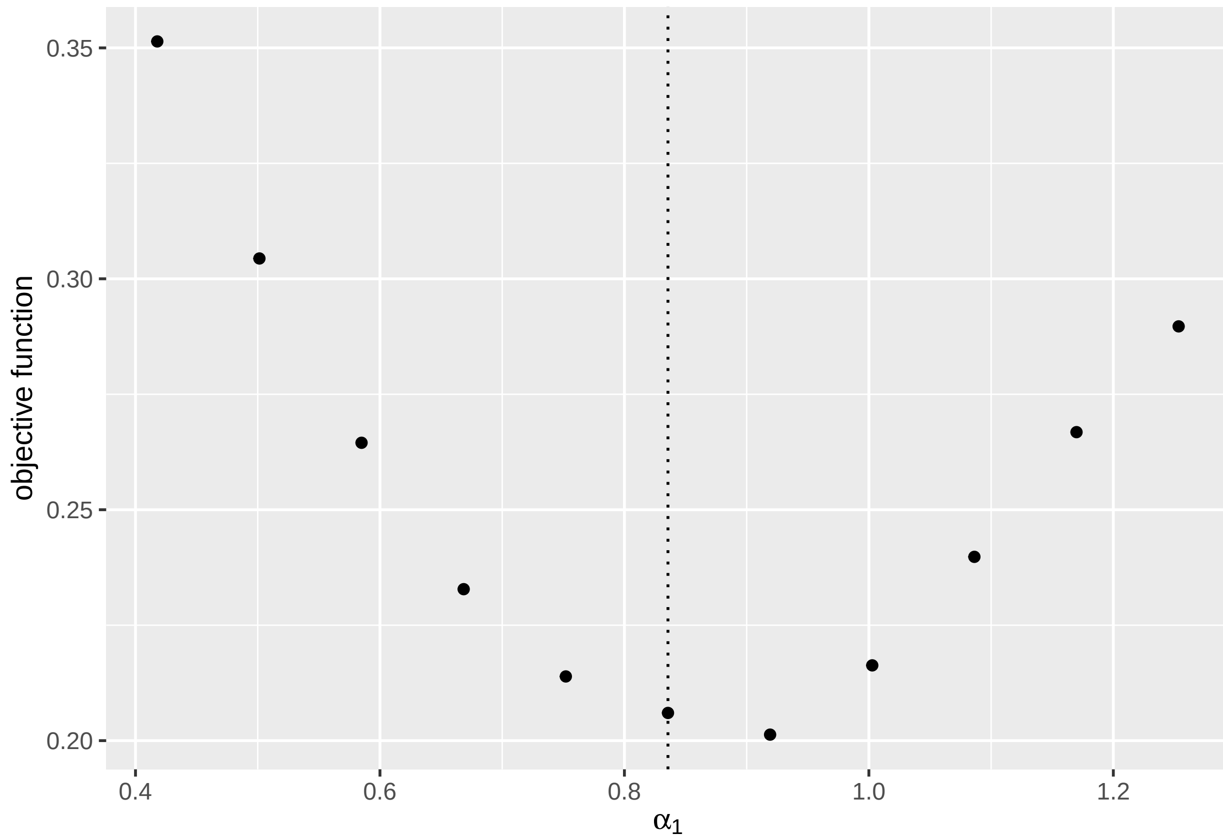



```
##
## [[2]]
```

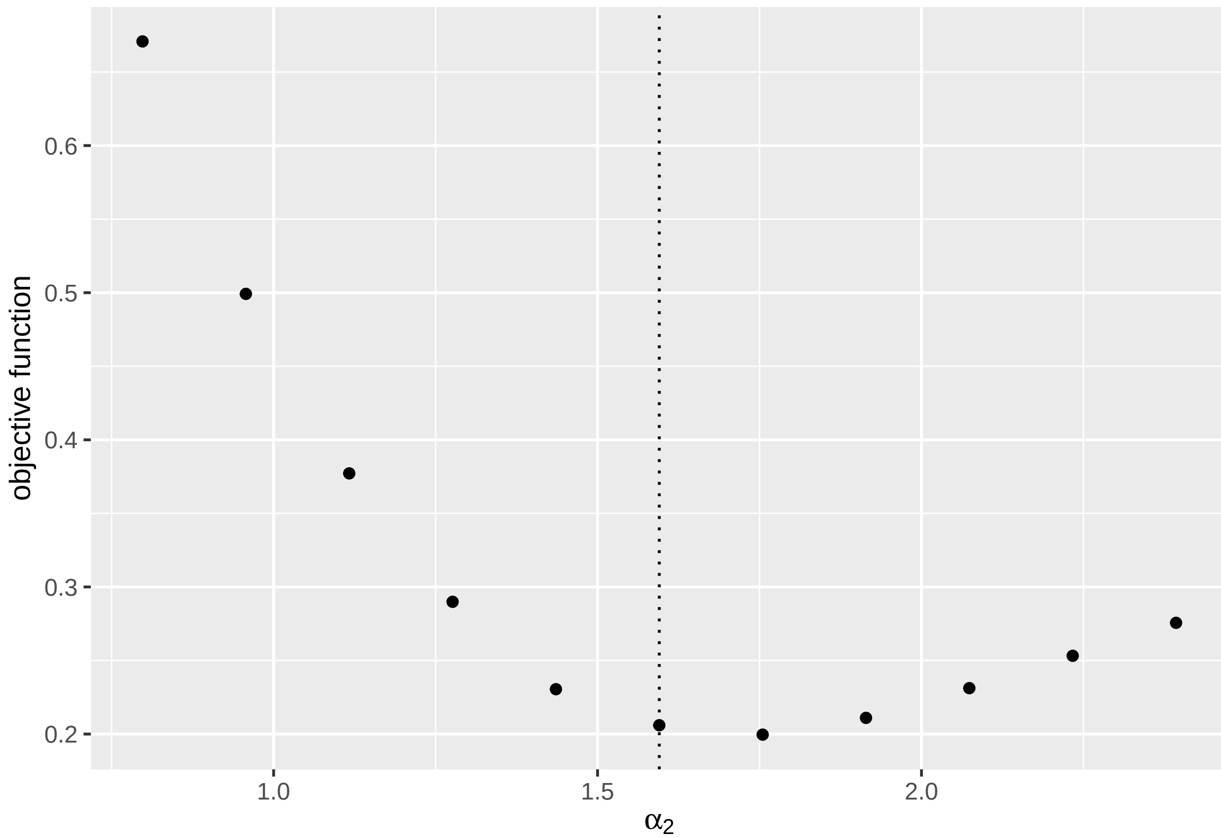


0.225
0.220
0.215
0.210
0.205
0.200
objective function
0.10
0.15
0.20
0.25
β_2

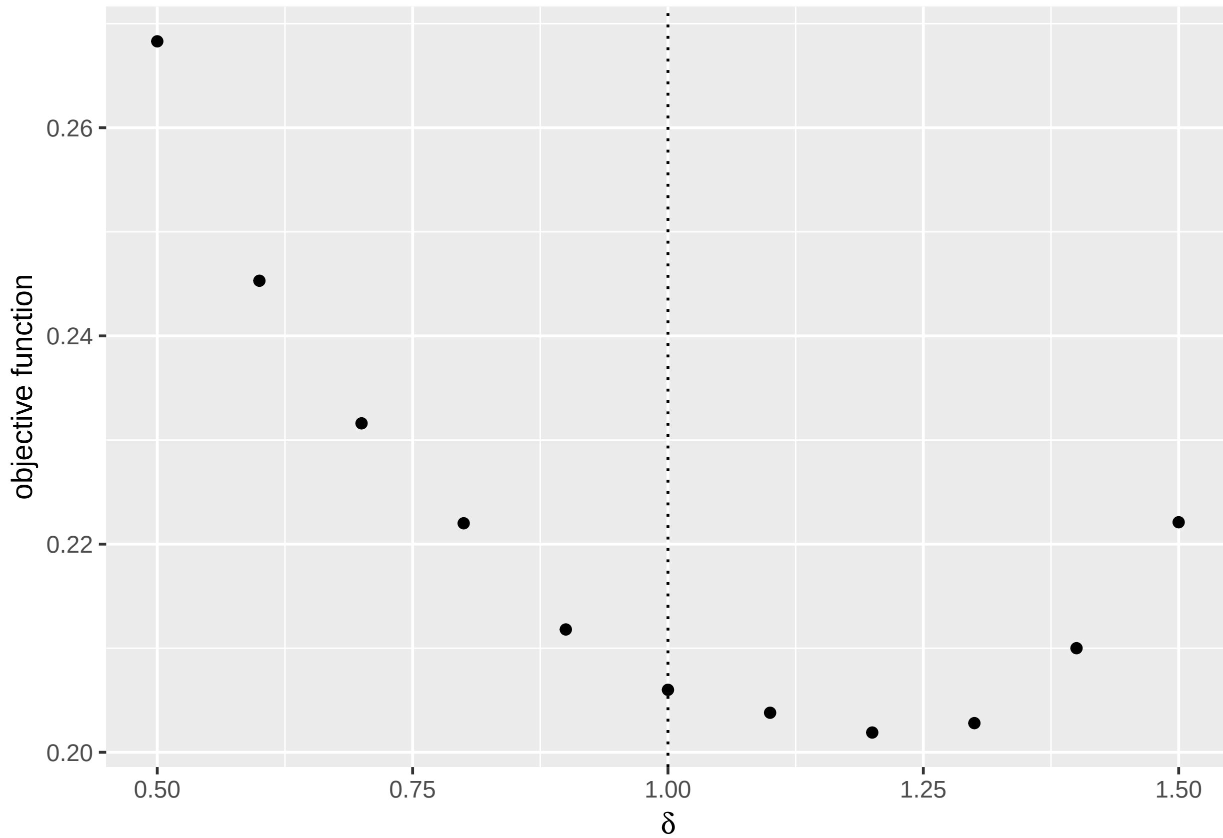

```
##
## [[3]]
```




```
##
## [[4]]
```




```
##
## [[5]]
```



6. Estimate the parameters under the assumption of the sequential entry.

```
# sequential entry
theta <- theta_sequential <-
  c(beta, alpha, delta, rho)
```

```
Y <- Y_sequential
result_sequential <-
  optim(par = theta,
        fn = compute_objective_sequential_entry,
        method = "Nelder-Mead",
        Y = Y,
        X = X,
        Z = Z,
        EP_mc = EP_mc,
        NU_mc = NU_mc)
save(result_sequential, file = "data/A6_estimate_sequential.RData")
```

```
load(file = "data/A6_estimate_sequential.RData")
result_sequential
```

```
## $par
## [1] 0.59704343 0.04938142 0.93904853 1.80598524 1.04483638 0.39060511
##
## $value
## [1] 0.2762
##
## $counts
## function gradient
```

```
##      155       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
comparison <-
  data.frame(
    actual = theta,
    estimate = result_sequential$par
  )
comparison
```

```
##      actual   estimate
## 1 0.6264538 0.59704343
## 2 0.1836433 0.04938142
## 3 0.8356286 0.93904853
## 4 1.5952808 1.80598524
## 5 1.0000000 1.04483638
## 6 0.3295078 0.39060511
```

7. Estimate the parameters under the assumption of the simultaneous entry. Set the lower bound for δ at 0.

```r
# simultaneous entry
theta <- theta_simultaneous <-
  c(beta, alpha, delta, rho)
```

```r
Y <- Y_sequential
result_simultaneous <-
  optim(par = theta,
        fn = compute_objective_simultaneous_entry,
        method = "Nelder-Mead",
        Y = Y,
        X = X,
        Z = Z,
        EP_mc = EP_mc,
        NU_mc = NU_mc)
save(result_simultaneous, file = "data/A6_estimate_simultaneous.RData")
```

```r
load(file = "data/A6_estimate_simultaneous.RData")
result_simultaneous
```

```
## $par
## [1] 0.5787563 0.1216856 0.9521359 1.8443535 1.2177670 0.3075741
##
## $value
## [1] 0.1623
##
## $counts
## function gradient
##      141       NA
##
## $convergence
## [1] 0
```

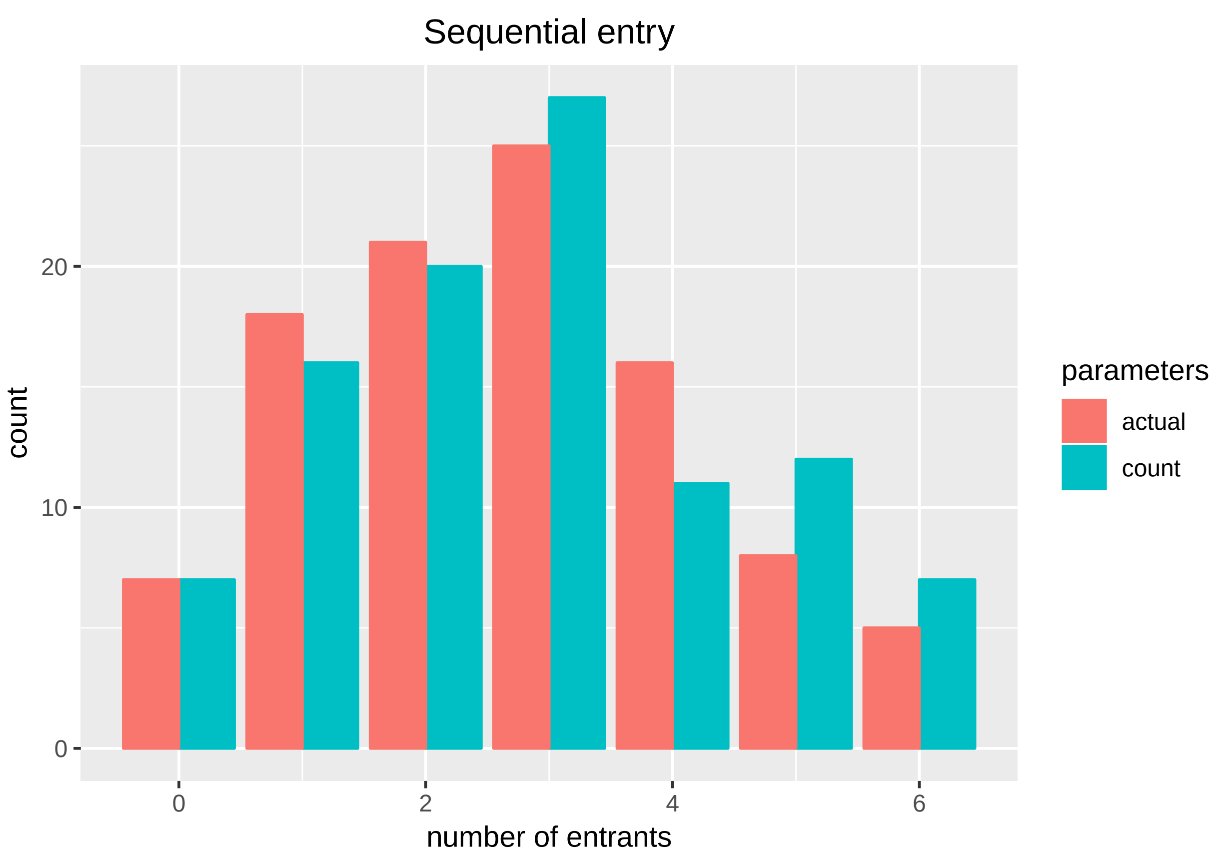
```
##
## $message
## NULL
```

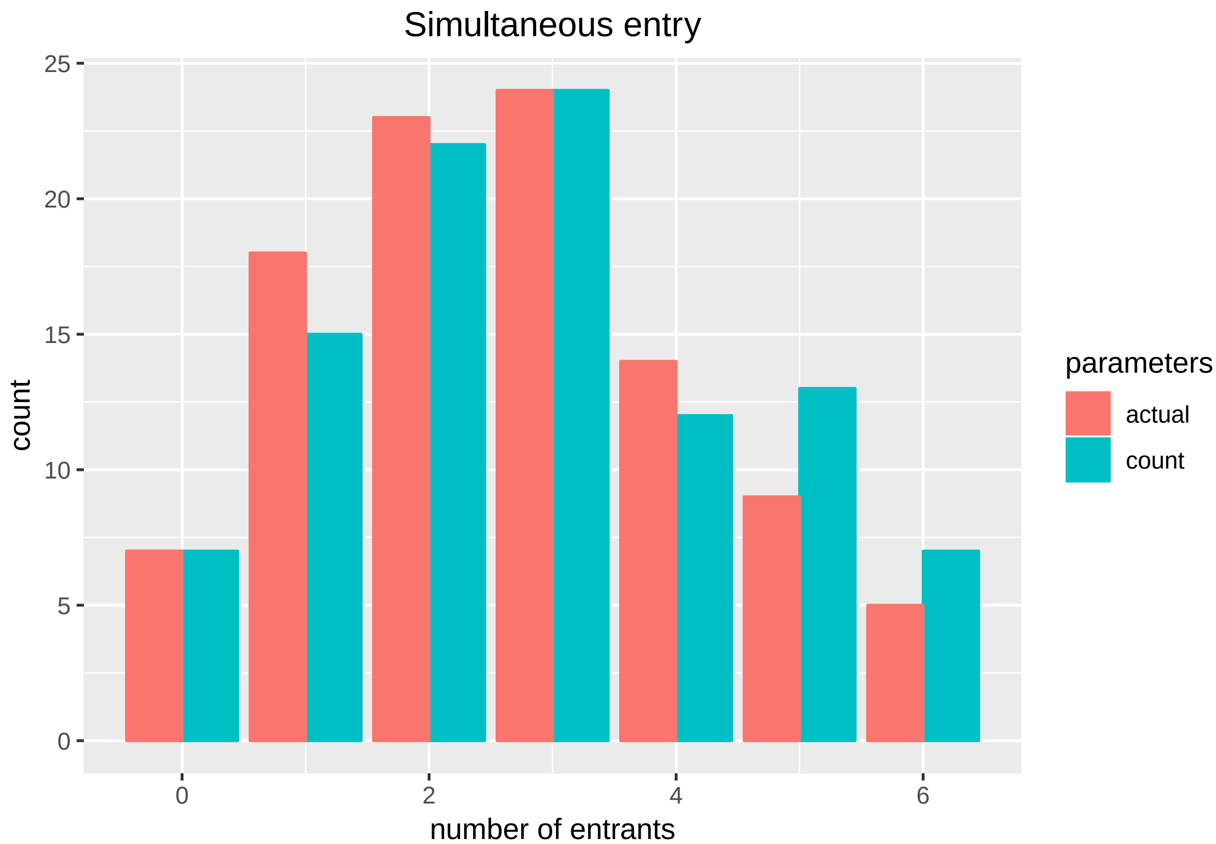
```
comparison <-
  data.frame(
    actual = theta,
    estimate = result_simultaneous$par
  )
comparison
```

```
##      actual  estimate
## 1 0.6264538 0.5787563
## 2 0.1836433 0.1216856
## 3 0.8356286 0.9521359
## 4 1.5952808 1.8443535
## 5 1.0000000 1.2177670
## 6 0.3295078 0.3075741
```

Conduct counterfactual simulations

1. Fix the first draw of the Monte Carlo shocks. Suppose that the competitive effect becomes mild, i.e. δ is changed to 0.5. Under these shocks, compute the equilibrium number of entrants across markets and plot the histogram with the estimated and counterfactual parameters. Conduct this analysis under the assumptions of sequential and simultaneous entry.




Simultaneous entry
count
25
20
15
10
5
0
0
2
4
6
number of entrants
parameters
actual
count