Self-tuning Experience-weighted Attraction (EWA) model applied to median action game

GS Verhoeven, Dutch Healthcare Authority

Summary

This notebook implements the self-tuning EWA model from Ho, Camerer and Chong (Journal of Economic Theory, 2007). We use it to reproduce the results presented in Chapter 7 of the review by Ho, Lim and Camerer (2006) of behavioral economics applied to consumer and firm behavior.

Load packages

```r
library(data.table)
```

```
## Warning: package 'data.table' was built under R version 3.4.4
```

```r
library(ggplot2)
library(readODS)
```

```
## Warning: package 'readODS' was built under R version 3.4.4
```

Read data

The data was created by van Huyck et al in their 1991 paper "strategic uncertainty, equilibrium selection, and coordination failure in average opinion games". The table in Appendix B contains the data. The data was entered in LibreOffice Calc.

In the experiments, each of the nine subjects chooses an action simultaneously and the median is then announced. Hence subjects have only histories of their own actions and the medians.

```r
df <- read_ods("vanHuyck_median_action.ods")
df <- data.table(df)
setnames(df, "round", "session")

mdf <- data.table(melt(df, id.vars = c("session", "subject")))
setnames(mdf, "variable", "round")
mdf <- mdf[, round := as.numeric(substr(round, 2,3))]
```

Calculate table 14

```r
res <- mdf[, median(value), .(session, round)]
dcast(res, session ~ round)
```

```
## Using 'V1' as value column. Use 'value.var' to override
##    session 1 2 3 4 5 6 7 8 9 10
## 1:       1 4 4 4 4 4 4 4 4 4  4
## 2:       2 5 5 5 5 5 5 5 5 5  5
## 3:       3 5 5 5 5 5 5 5 5 5  5
## 4:       4 4 4 4 4 4 4 4 4 4  4
## 5:       5 4 4 4 4 4 4 4 4 4  4
## 6:       6 5 5 5 5 5 5 5 5 5  5
```

```
res <- mdf[, .(sd = sd(value)), .(session, round)]
res[, asterisk := "."]
res[sd == 0, asterisk := "*"]
dcast(res, session ~ round, value.var = "asterisk")
```

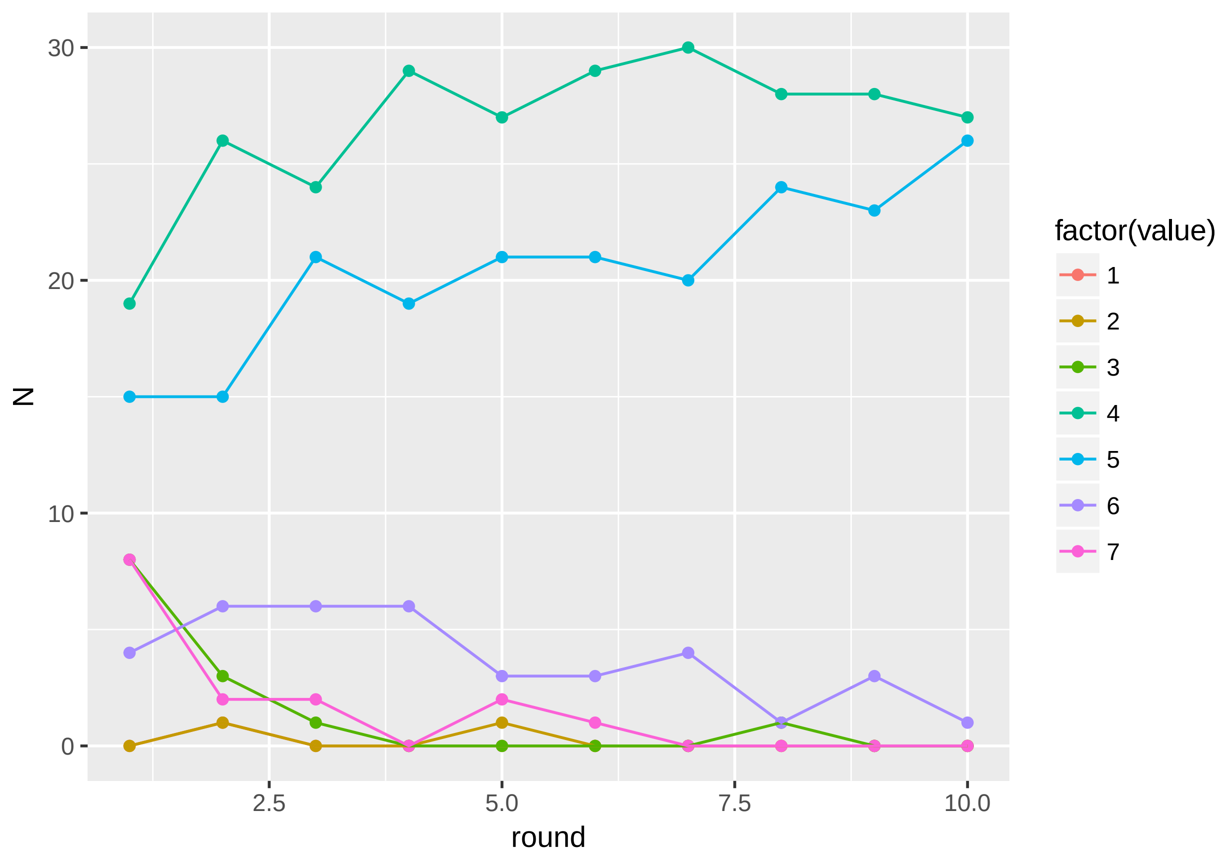
```
##    session 1 2 3 4 5 6 7 8 9 10
## 1:       1 . . . . . . * . *  *
## 2:       2 . . . . . . . . .  .
## 3:       3 . . . . . . . . .  *
## 4:       4 . . . . . * * * *  *
## 5:       5 . . . * * * * * *  *
## 6:       6 . . . . . . . * *  *
```

This reproduces table 14 on page 45.

Recreate figure 3: Empirical Frequency

The data from all six sessions are pooled.

```
res <- mdf[, .(.N), .(value, round)]
res <- merge(data.table(value = rep(1:7, 10), round = rep(1:10, each = 7)),
             res, by = c("value", "round"), all.x = T)
res[is.na(res)] <- 0
ggplot(res, aes(x = round, y = N, group = factor(value), col = factor(value))) +
  geom_point() + geom_line()
```



We now apply the self tuning EWA model to see if it can capture the overall behavior of the players.

Setup data structures

```
n_players <- 9 # index i
n_strategies <- 7 # index j
n_rounds <- 10 # index t
n_sessions <- 6

#lambda <- 5.64 # ML best fit for Median action game (Ho et al JET 2007)
lambda <- 11
```

Action matrix

Create and fill action matrix for a particular session.

```
action <- matrix(nrow = n_players, ncol = n_rounds)

for(t in 1:n_rounds){
  action[, t] <- unname(unlist(df[session == 1, c(t+2), with = F]))
}
```

Payoffs

calculate foregone payoffs of player i for strategy j in round t, GIVEN the strategies played by all other players (-i). This is the conditional payoff function.

We also make a function that calculates for each player the actual payoffs for a set of strategies played by all players.

The payoff functions rewards conformity, but also coordination on a high value.

```
# payoffs for player i playing j, given the actions of other players
cond_payoff <- function(i, j, actions){
    actions[i] <- j
    M <- median(actions)
  payoff <- 0.1 * M - 0.05 * (M - actions)^2 + 0.6
  return(payoff[i])
}

# calculate actual payoff for all players
actual_payoff <- function(actions){
  M <- median(actions)
  payoff <- 0.1 * M - 0.05 * (M - actions)^2 + 0.6
  return(payoff)
}
```

Test payoffs:

```
actual_payoff(action[,1])
```

```
## [1] 0.95 1.00 0.95 1.00 0.95 0.95 0.95 0.95 0.95
```

This is correct.

Calculate payoff when player 1 plays strategy 1, given the actions of all other players in round 1.

```
median(c(1,action[-1,1]))
```

```
## [1] 4
```

```
cond_payoff(1, 1, action[,1])
```

```
## [1] 0.55
```

This is correct (comparing with table 13).

Attention function delta

The attention function serves to reinforce chosen and unchosen strategies that give better payoffs after playing round t.

```
create_delta <- function(actions){
  delta <- array(data = NA,
                 dim = c(n_players, n_strategies, n_rounds))
  for(t in 1:n_rounds){
    for(i in 1:n_players){
      for(j in 1:n_strategies){
        act_payoff_i <- actual_payoff(actions[,t][i])
        cond_payoff <- cond_payoff(i, j, actions[,t])
        delta[i, j, t] <- 0
        if(cond_payoff >= act_payoff_i) delta[i, j, t] <- 1
      }
    }
  }
  return(delta)
}
```

Test delta.

```
delta <- create_delta(action)
action[,8]
```

```
## [1] 4 4 4 4 3 4 4 4 4
```

```
delta[, , 8]
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7]
##  [1,]    0    0    0    1    0    0    0
##  [2,]    0    0    0    1    0    0    0
##  [3,]    0    0    0    1    0    0    0
##  [4,]    0    0    0    1    0    0    0
##  [5,]    0    0    1    1    1    0    0
##  [6,]    0    0    0    1    0    0    0
##  [7,]    0    0    0    1    0    0    0
##  [8,]    0    0    0    1    0    0    0
##  [9,]    0    0    0    1    0    0    0
```

In round 8, playing strategy 4 or 5 had given equal or greater payoff compared to its actual strategy (playing 3). For all other players, there exist no other strategy than strategy four that gives equal or higher payoffs.

Works as expected.

```
mdelta <- c()

for(s in 1:n_sessions){
  action_tst <- matrix(nrow = n_players, ncol = n_rounds)

  for(t in 1:n_rounds){
    action_tst[, t] <- unname(unlist(df[session == s, c(t+2), with = F]))
  }
  delta <- create_delta(action_tst)
  mdelta[s] <- mean(delta[, , 8:10])
}
mean(mdelta)
```

```
## [1] 0.1455026
```

For delta, Ho et al report a value of 0.15. We can reproduce this if we take the average over all delta values for the last three rounds, the out-of-sample rounds.

Change detector function phi

Also called the decay rate ("forgetting"). It serves to detect if other players are changing their strategy.

```r
# create historical frequencies normalized to 1
create_cumulative_history <- function(action){
  history <- array(data = NA,
                   dim = c(n_players, n_strategies, n_rounds))
  for(t in 1:n_rounds){
    for(i in 1:n_players){
      for(j in 1:n_strategies){
        freq <- table(action[-i, 1:t])
        # create counts for all strategies (also non-chosen ones)
        freq <- merge(data.frame(Var1 = 1:n_strategies),
                      data.frame(freq), by = "Var1", all.x= T)
        freq[is.na(freq)]<-0
        history[i, , t]  <- freq[,2]/sum(freq[,2])
      }
    }
  }
  return(history)
}

create_recent_history <- function(action){
  history <- array(data = NA,
                   dim = c(n_players, n_strategies, n_rounds))
  for(t in 1:n_rounds){
    for(i in 1:n_players){
      for(j in 1:n_strategies){
        # note here we only look at strategies in round t
        freq <- table(action[-i, t])
        # create counts for all strategies (also non-chosen ones)
        freq <- merge(data.frame(Var1 = 1:n_strategies),
                      data.frame(freq), by = "Var1", all.x= T)
        freq[is.na(freq)]<-0
        history[i, , t]  <- freq[,2]/sum(freq[,2])
      }
    }
  }
  return(history)
}

calculate_surprise_index <- function(cum_history, rec_history){
  surprise <- array(data = NA,
                   dim = c(n_players, n_rounds))
  for(t in 1:n_rounds){
    for(i in 1:n_players){
      for(j in 1:n_strategies){
        if(j == 1) {
          surprise[i, t] = (cum_history[i, j, t] - rec_history[i, j, t])^2
        } else {surprise[i, t] = surprise[i, t] +
          (cum_history[i, j, t] - rec_history[i, j, t])^2}
      }
    }
  }
  return(surprise)
}
```

```r
calculate_phi <- function(action){
  cum_history <- create_cumulative_history(action)
  rec_history <- create_recent_history(action)
  surprise <- calculate_surprise_index(cum_history, rec_history)
  # map surprise to phi
  phi <- 1 - 0.5 * surprise
  return(phi)
}
```

Test history functions.

```r
cum_history <- create_cumulative_history(action)

# normalized counts of actions of players 1:8 in first round
table(action[-9,1:3])/sum(table(action[-9,1:3]))
```

```
##
##          1          3          4          5          6          7
## 0.04166667 0.08333333 0.41666667 0.37500000 0.04166667 0.04166667
```

```r
cum_history[9, ,3]
```

```
## [1] 0.04166667 0.00000000 0.08333333 0.41666667 0.37500000 0.04166667
## [7] 0.04166667
```

Is correct.

```r
rec_history <- create_recent_history(action)

table(action[-9,2])/sum(table(action[-9,2]))
```

```
##
##     1     4     5
## 0.125 0.375 0.500
```

```r
rec_history[9, , 2]
```

```
## [1] 0.125 0.000 0.000 0.375 0.500 0.000 0.000
```

Is correct.

Test surprise function.

```r
surprise <- calculate_surprise_index(cum_history, rec_history)

phi <- calculate_phi(action)
```

For phi, Ho et al report an (average? final?) value of 0.85.

```r
mphi <- c()

for(s in 1:n_sessions){
  action_tst <- matrix(nrow = n_players, ncol = n_rounds)

  for(t in 1:n_rounds){
    action_tst[, t] <- unname(unlist(df[session == s, c(t+2), with = F]))
  }
  phi <- calculate_phi(action_tst)
  mphi[s] <- mean(phi[ , 8:10])
}
mean(mphi)
```

```
## [1] 0.9586559
```

Our phi is too large compared to the reported value.

Check examples in paper:

```
n_row <- dim(action)[1]
n_col <- dim(action)[2]
action_tst <- action

for(i in 1:n_row){
  for(j in 1:n_col)
    action_tst[i, j] <- 4
}
action_tst[2:n_row, n_col] <- 1

phi <- calculate_phi(action_tst)
action_tst
```

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

```
phi
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]     [,10]
##  [1,]    1    1    1    1    1    1    1    1    1 0.1900000
##  [2,]    1    1    1    1    1    1    1    1    1 0.3798438
##  [3,]    1    1    1    1    1    1    1    1    1 0.3798438
##  [4,]    1    1    1    1    1    1    1    1    1 0.3798438
##  [5,]    1    1    1    1    1    1    1    1    1 0.3798438
##  [6,]    1    1    1    1    1    1    1    1    1 0.3798438
##  [7,]    1    1    1    1    1    1    1    1    1 0.3798438
##  [8,]    1    1    1    1    1    1    1    1    1 0.3798438
##  [9,]    1    1    1    1    1    1    1    1    1 0.3798438
```

Experience weights

Although not clear, we assume that the experience weight is player specific, since it needs to normalize over a set of attractions for a particular player. Since the experience weight is updated by a player specific phi value, that also works on the player's attractions, this leads us to a player specific experience weight.

```
calculate_experience_weights <- function(phi) {
  exp_weights <- array(data = NA,
                       dim = c(n_players, n_rounds + 1))
  for(t in 1:(n_rounds + 1)){ # here t' = t+1
    for(i in 1:n_players){
      if(t == 1) exp_weights[i, t] <- 1
      else{
        exp_weights[i, t] <- (phi[i, t-1] * exp_weights[i, t-1]) + 1
      }
    }
  }
  return(exp_weights)
}
```

```
phi <- calculate_phi(action)
exp_weights <- calculate_experience_weights(phi)
```

Attraction array

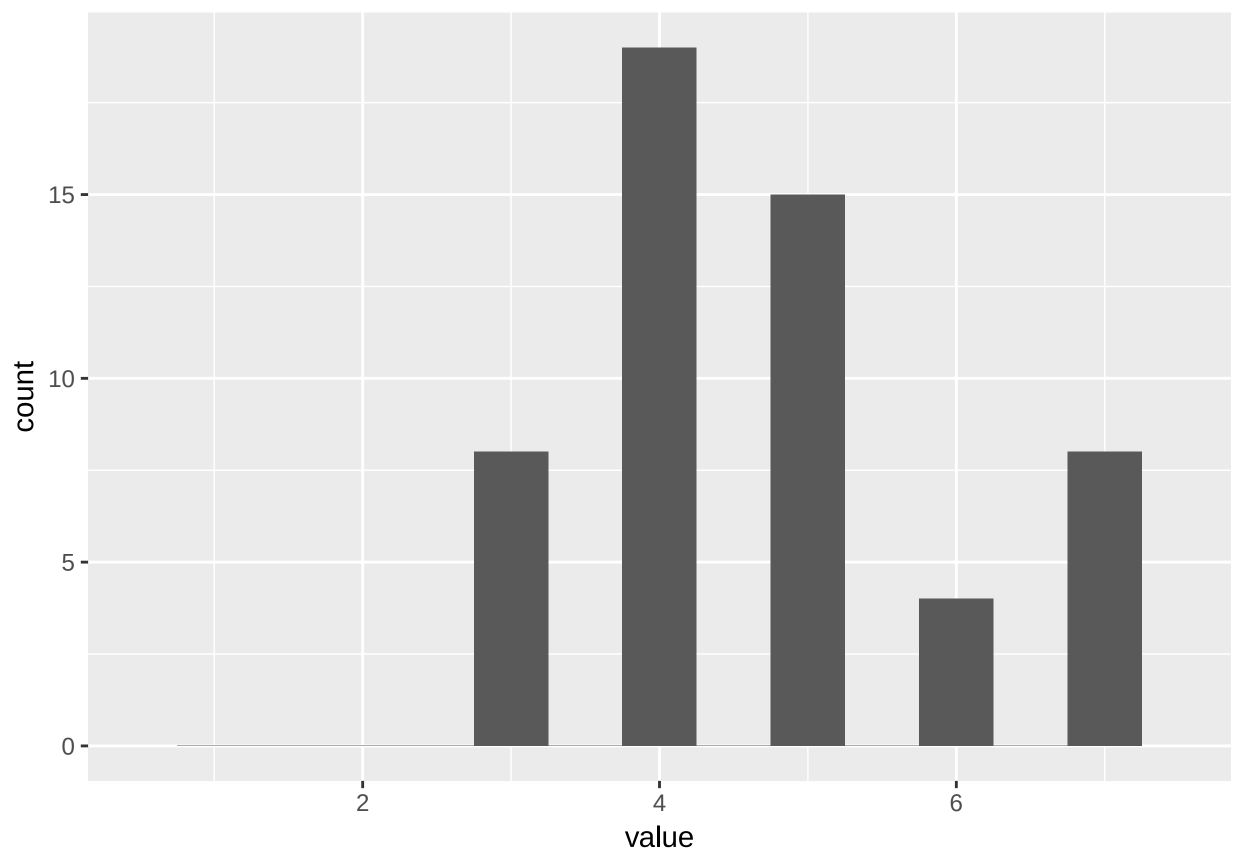
Create and fill attraction array. This holds for each player, each strategy and each round its attraction at the end of the round.

We need to provide it with initial attractions. Ho et al use the CH model with t = 1.5. We use the result of their calculations by estimating them from Figure 4 for period 1. They hold for all players.

```
# for each strategy
init_attractions <- c(1, 2, 4, 5, 4, 2, 1)
# normalize
init_attractions <- init_attractions/sum(init_attractions)
```

An alternative is to use the distribution of strategies chosen in the first period of each session.

```
ggplot(mdf[round == 1], aes(x = value)) + geom_histogram(binwidth = 0.5) +
  xlim(0.5, 7.5)
```



```
calc_attractions <- function(action, phi, delta, exp_weights, init_attractions){
  # create empty attraction array
  attraction <- array(data = NA,
                      dim = c(n_players, n_strategies, n_rounds))

  for(i in 1:n_players){
    attraction[i, , 1] <- init_attractions
  }
  for(t in 2:n_rounds){
    for(i in 1:n_players){
      for(j in 1:n_strategies){
        term1 <- phi[i, t] * exp_weights[i, t] * attraction[i, j, t-1]
        term2 <- (delta[i, j, t] + (1 - delta[i, j, t] * (j == action[i,t]))) *
          cond_payoff(i, j, action[,t])
        numerator <- term1 + term2
        denominator <- phi[i, t] * exp_weights[i, t] + 1
        attraction[i, j, t] <- numerator/denominator
      }
    }
  }
  return(attraction)
}
```

```
attraction <- calc_attractions(action, phi, delta, exp_weights,
                               init_attractions)
```

Logistic stochastic response function

```r
calc_probabilities <- function(lambda, attraction){
  # create empty attraction array
  probability <- array(data = NA,
                       dim = c(n_players, n_strategies, n_rounds))
  numerator <- array(data = NA,
                       dim = c(n_players, n_strategies, n_rounds))
  denominator <- array(data = NA,
                       dim = c(n_players, n_rounds))
  for(t in 1:n_rounds){
    for(i in 1:n_players){
      for(j in 1:n_strategies){
          numerator[i, j, t] <- exp(lambda * attraction[i, j, t])
      }
    }
  }
  # calculate normalization by summing all numerators
  for(t in 1:n_rounds){
    for(i in 1:n_players){
      denominator[i, t] <- 0
      for(j in 1:n_strategies){
          denominator[i, t] <- denominator[i, t] + numerator[i, j, t]
      }
    }
  }
  # combine numerators and denominators
    for(t in 1:n_rounds){
    for(i in 1:n_players){
      for(j in 1:n_strategies){
          probability[i, j, t] <- numerator[i, j, t]/denominator[i, t]
      }
    }
  }
  return(probability)
}
```

Test function. Expect that probs sum to 1 for a each player in a round.

```r
probability <- calc_probabilities(lambda, attraction)
sum(probability[1, , 1])
```

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

It works.

Complete program

```r
list_of_probs <- list()

for(s in 1:n_sessions){
  action <- matrix(nrow = n_players, ncol = n_rounds)
  for(t in 1:n_rounds){
    action[, t] <- unname(unlist(df[session == s, c(t+2), with = F]))
  }
  phi <- calculate_phi(action)
  delta <- create_delta(action)

  exp_weights <- calculate_experience_weights(phi)

  # for each strategy
  init_attractions <- c(1, 2, 4, 5, 4, 2, 1)
  # normalize
  init_attractions <- init_attractions/sum(init_attractions)

  attraction <- calc_attractions(action, phi, delta, exp_weights,
                               init_attractions)

  probability <- calc_probabilities(lambda, attraction)
  list_of_probs[[s]] <- probability
}
```

Plot self-tuning EWA predicted frequencies

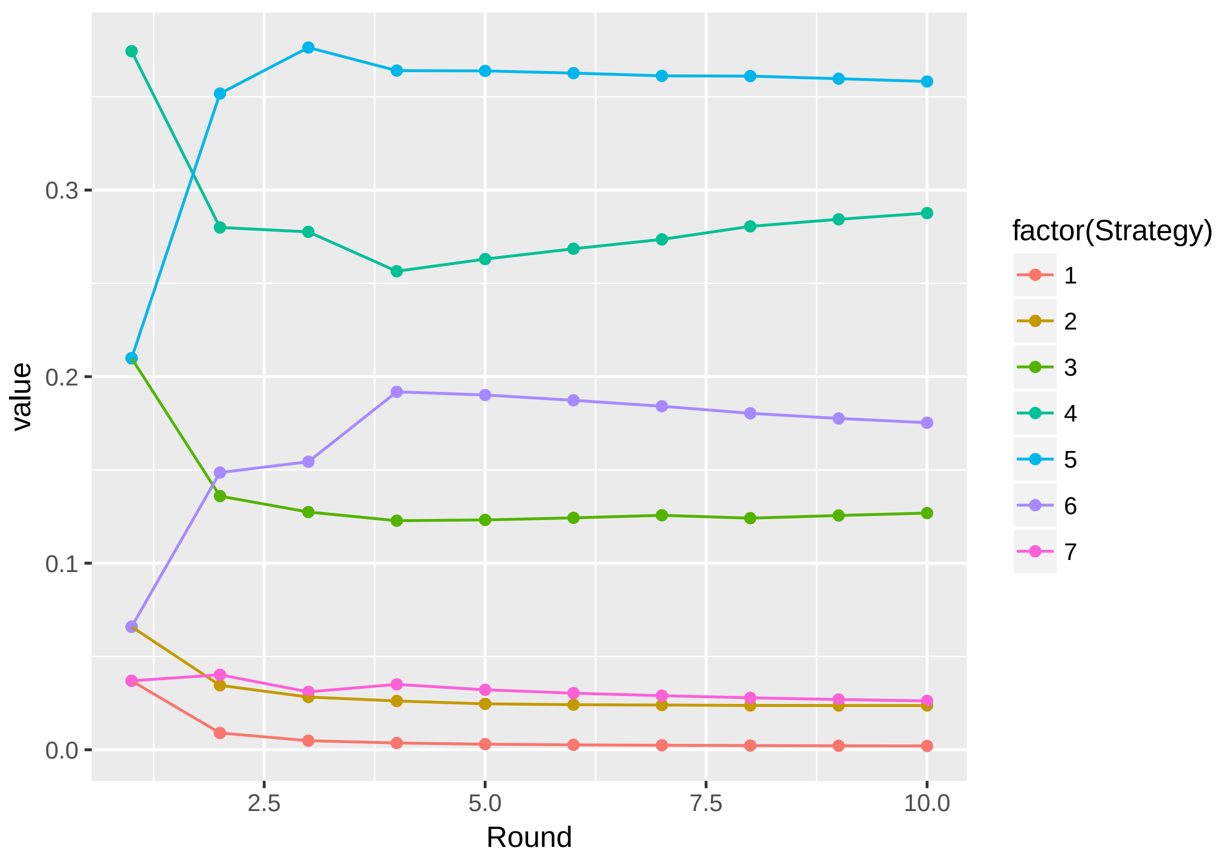
For each round, and each strategy, sum all probability density over the players.

```r
calc_frequencies <- function(probability){
  frequency <- array(data = NA,
                     dim = c(n_strategies, n_rounds))
  for(t in 1:n_rounds){
    for(j in 1:n_strategies){
      frequency[j, t] <- 0
      for(i in 1:n_players){
        frequency[j, t] <- frequency[j, t] + probability[i, j, t]
      }
    }
  }
  return(frequency/n_players)
}
```

```r
frequency <- calc_frequencies(list_of_probs[[1]])

for(s in 2:n_sessions){
  frequency <- frequency + calc_frequencies(list_of_probs[[s]])
}
frequency <- frequency/n_sessions
```

```r
mfreq <- data.table(melt(frequency))
setnames(mfreq, "Var1", "Strategy")
setnames(mfreq, "Var2", "Round")
mfreq <- mfreq[, type := "model"]
ggplot(mfreq, aes(x = Round, y = value, group = factor(Strategy),
                  col = factor(Strategy))) + geom_point() + geom_line()
```



Combine model with data:

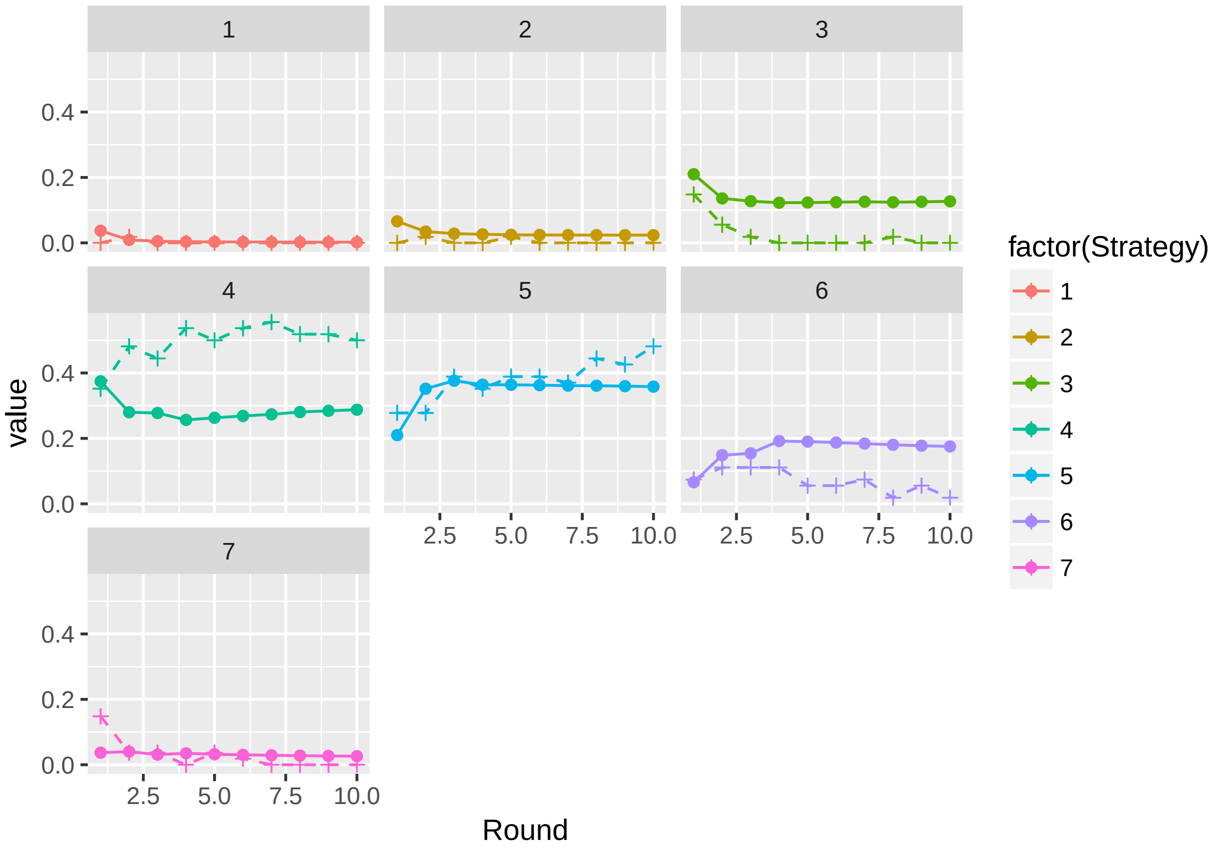
```
res <- mdf[, .(N = .N/(n_players*n_sessions)), .(value, round)]
res <- merge(data.table(value = rep(1:7, 10), round = rep(1:10, each = 7)),
             res, by = c("value", "round"), all.x = T)
res[is.na(res)] <- 0

setnames(res, "value", "Strategy")
setnames(res, "round", "Round")
setnames(res, "N", "value")
setkey(res, Round)

res <- res[, type := "experiment"]

res <- rbind(res, mfreq)

ggplot(res[type == "model"], aes(x = Round, y = value, group = factor(Strategy),
                                 col = factor(Strategy))) + geom_point() +
  geom_line() + geom_point(data = res[type == "experiment"], shape = 3) +
  geom_line(data = res[type == "experiment"], linetype = 2) + facet_wrap(~ Strategy)
```


1
2
3
4
5
6
7
value
0.0
0.2
0.4
Round
2.5
5.0
7.5
10.0
factor(Strategy)
1
2
3
4
5
6
7