Estimating Utility Functions

```r
# install.packages("tidyverse")
library(tidyverse)
library(purrr)
library(tidyr)
```

Data

The data for estimating the utility function of player B is from Charness and Rabin (2002: 829), Table 1, Two-person dictator games.

```r
# Spalten der Tabelle als Vektoren erstellen
Game <- c("Berk29", "Barc2", "Berk17", "Berk23", "Barc8", "Berk15", "Berk26")
LeftPayoffA <- c(400, 400, 400, 800, 300, 200, 0)
LeftPayoffB <- c(400, 400, 400, 200, 600, 700, 800)
RightPayoffA <- c(750, 750, 750, 0, 700, 600, 400)
RightPayoffB <- c(400, 375, 375, 0, 500, 600, 400)
ObsProbRight <- c(.69, .48, .50, 0, .33, .73, .22)

# Vektoren in eine Tabelle packen
dataRaw <- data.frame(Game,
                   LeftPayoffA,
                   LeftPayoffB,
                   RightPayoffA,
                   RightPayoffB,
                   ObsProbRight)

# Die Rohdaten ausgeben
print(dataRaw)
```

```
##     Game LeftPayoffA LeftPayoffB RightPayoffA RightPayoffB ObsProbRight
## 1 Berk29         400         400          750          400         0.69
## 2  Barc2         400         400          750          375         0.48
## 3 Berk17         400         400          750          375         0.50
## 4 Berk23         800         200            0            0         0.00
## 5  Barc8         300         600          700          500         0.33
## 6 Berk15         200         700          600          600         0.73
## 7 Berk26           0         800          400          400         0.22
```

Utility function of player B

The model is a simplified version from Charness and Rabin (2002: 822).

```r
# Nutzenfunktion definieren
Utility <- function(ownPayoff, # Input1
```

```r
                    weightOtherPayoff,# Input2
                    otherPayoff) { # Input3
  weightOwnPayoff <- 1 - weightOtherPayoff # Calculation
  return(weightOtherPayoff * otherPayoff + weightOwnPayoff * ownPayoff) # Output
}

# Anwendung der Nutzenfunktion
Utility(ownPayoff = 5,
        weightOtherPayoff = 0.5,
        otherPayoff = 40)
```

```
## [1] 22.5
```

We assume a fixed value for the parameter `weightPayoffA` and calculate the utility of B for our data. Then we predict which option will be chosen and calculate the sum of squared differences.

```r
# Das Gewicht, das Spieler B auf die Auszahlung von Spieler A legt
parameter = .2

# Berechnung der Nutzen von Spieler B für die Alternativen Left und Right
# Berechnung der vorhergesagten Wahlwahrscheinlichkeit für die Alternative Right
# Berechung der quadrierten Abweichungen zwischen vorhergesagten Wahlwahrscheinlichkeiten und der beoba
dataFix <- dataRaw %>%
  mutate(LeftUtilityB = Utility(ownPayoff = LeftPayoffB,
                                weightOtherPayoff = parameter,
                                otherPayoff = LeftPayoffA),
         RightUtilityB = Utility(ownPayoff = RightPayoffB,
                                 weightOtherPayoff = parameter,
                                 otherPayoff = RightPayoffA)) %>%
  mutate(PredProbRight = ifelse(test = RightUtilityB > LeftUtilityB, yes = 1,
                                ifelse(test = RightUtilityB == LeftUtilityB, yes = .5, no = 0)),
         SquaredDiff = (ObsProbRight - PredProbRight)^2)

# Summe der quadrierten Abweichungen
sum(dataFix$SquaredDiff)
```

```
## [1] 0.7467
```

Model with one free parameter

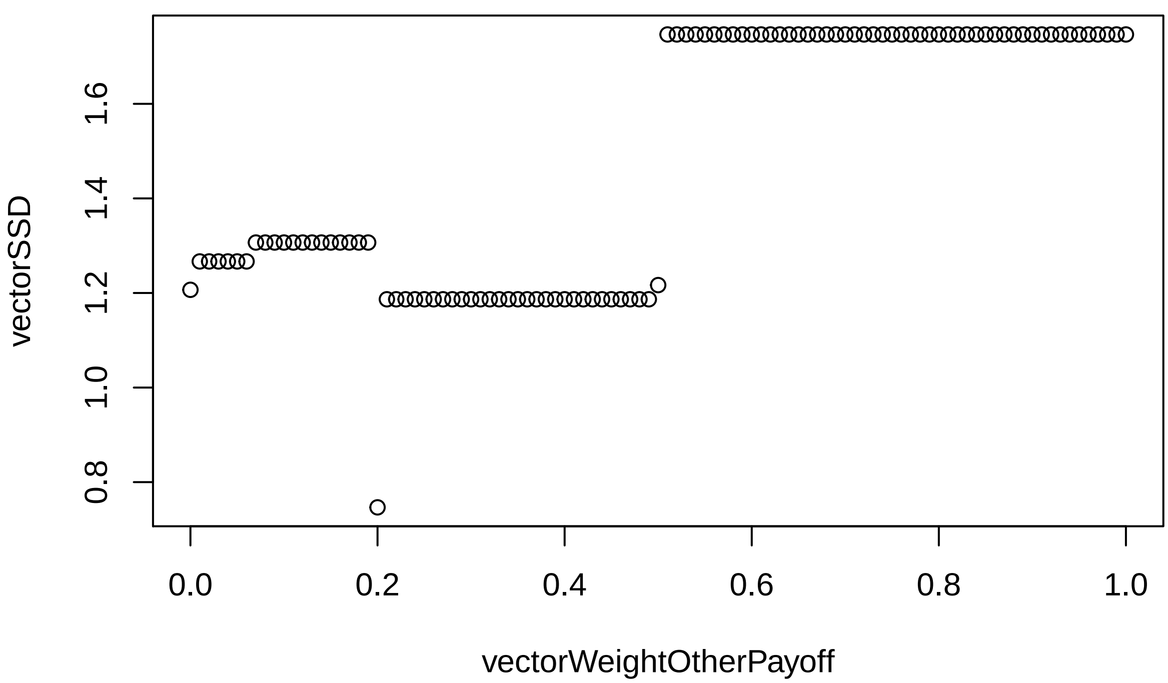
We search for the parameter `weightPayoffA` that maximizes the fit to the data.

```r
vectorWeightOtherPayoff <- 0:100 / 100
#vectorWeightOtherPayoff

vectorSSD = rep(NA, length(vectorWeightOtherPayoff))
count <- 1
for (parameter in vectorWeightOtherPayoff) {
dataEst <- dataRaw %>%
  mutate(LeftUtilityB = Utility(ownPayoff = LeftPayoffB,
                                weightOtherPayoff = parameter,
                                otherPayoff = LeftPayoffA),
         RightUtilityB = Utility(ownPayoff = RightPayoffB,
                                 weightOtherPayoff = parameter,
                                 otherPayoff = RightPayoffA)) %>%
```

```
  mutate(PredProbRight = ifelse(test = RightUtilityB > LeftUtilityB, yes = 1,
                                ifelse(test = RightUtilityB == LeftUtilityB, yes = .5, no = 0)),
         SquaredDiff = (ObsProbRight - PredProbRight)^2)

SSD = sum(dataEst$SquaredDiff) # Sum of squared differences

vectorSSD[count] = SSD

count <- count + 1
}

dataEstimation <- data.frame(vectorWeightOtherPayoff, vectorSSD)
```

We plot the optimization search results

```
plot(dataEstimation)
```



We print the table of the optimization results

```
dataEstimation
```

```
##     vectorWeightOtherPayoff vectorSSD
## 1                      0.00    1.2067
## 2                      0.01    1.2667
## 3                      0.02    1.2667
## 4                      0.03    1.2667
## 5                      0.04    1.2667
## 6                      0.05    1.2667
```

```
## 7                      0.06    1.2667
## 8                      0.07    1.3067
## 9                      0.08    1.3067
## 10                     0.09    1.3067
## 11                     0.10    1.3067
## 12                     0.11    1.3067
## 13                     0.12    1.3067
## 14                     0.13    1.3067
## 15                     0.14    1.3067
## 16                     0.15    1.3067
## 17                     0.16    1.3067
## 18                     0.17    1.3067
## 19                     0.18    1.3067
## 20                     0.19    1.3067
## 21                     0.20    0.7467
## 22                     0.21    1.1867
## 23                     0.22    1.1867
## 24                     0.23    1.1867
## 25                     0.24    1.1867
## 26                     0.25    1.1867
## 27                     0.26    1.1867
## 28                     0.27    1.1867
## 29                     0.28    1.1867
## 30                     0.29    1.1867
## 31                     0.30    1.1867
## 32                     0.31    1.1867
## 33                     0.32    1.1867
## 34                     0.33    1.1867
## 35                     0.34    1.1867
## 36                     0.35    1.1867
## 37                     0.36    1.1867
## 38                     0.37    1.1867
## 39                     0.38    1.1867
## 40                     0.39    1.1867
## 41                     0.40    1.1867
## 42                     0.41    1.1867
## 43                     0.42    1.1867
## 44                     0.43    1.1867
## 45                     0.44    1.1867
## 46                     0.45    1.1867
## 47                     0.46    1.1867
## 48                     0.47    1.1867
## 49                     0.48    1.1867
## 50                     0.49    1.1867
## 51                     0.50    1.2167
## 52                     0.51    1.7467
## 53                     0.52    1.7467
## 54                     0.53    1.7467
## 55                     0.54    1.7467
## 56                     0.55    1.7467
## 57                     0.56    1.7467
## 58                     0.57    1.7467
## 59                     0.58    1.7467
## 60                     0.59    1.7467
```

```
## 61                         0.60    1.7467
## 62                         0.61    1.7467
## 63                         0.62    1.7467
## 64                         0.63    1.7467
## 65                         0.64    1.7467
## 66                         0.65    1.7467
## 67                         0.66    1.7467
## 68                         0.67    1.7467
## 69                         0.68    1.7467
## 70                         0.69    1.7467
## 71                         0.70    1.7467
## 72                         0.71    1.7467
## 73                         0.72    1.7467
## 74                         0.73    1.7467
## 75                         0.74    1.7467
## 76                         0.75    1.7467
## 77                         0.76    1.7467
## 78                         0.77    1.7467
## 79                         0.78    1.7467
## 80                         0.79    1.7467
## 81                         0.80    1.7467
## 82                         0.81    1.7467
## 83                         0.82    1.7467
## 84                         0.83    1.7467
## 85                         0.84    1.7467
## 86                         0.85    1.7467
## 87                         0.86    1.7467
## 88                         0.87    1.7467
## 89                         0.88    1.7467
## 90                         0.89    1.7467
## 91                         0.90    1.7467
## 92                         0.91    1.7467
## 93                         0.92    1.7467
## 94                         0.93    1.7467
## 95                         0.94    1.7467
## 96                         0.95    1.7467
## 97                         0.96    1.7467
## 98                         0.97    1.7467
## 99                         0.98    1.7467
## 100                        0.99    1.7467
## 101                        1.00    1.7467
```

Why is a weight of .2 optimal?

```
parameter = .2 # change to .2 and change to .21 - What do you observe?

dataEst <- dataRaw %>%
  mutate(LeftUtilityB = Utility(ownPayoff = LeftPayoffB,
                               weightOtherPayoff = parameter,
                               otherPayoff = LeftPayoffA),
         RightUtilityB = Utility(ownPayoff = RightPayoffB,
                                weightOtherPayoff = parameter,
```

```
                                  otherPayoff = RightPayoffA)) %>%
  mutate(PredProbRight = ifelse(test = RightUtilityB > LeftUtilityB, yes = 1,
                                 ifelse(test = RightUtilityB == LeftUtilityB, yes = .5, no = 0)),
         SquaredDiff = (ObsProbRight - PredProbRight)^2)

dataEst
```

```
##     Game LeftPayoffA LeftPayoffB RightPayoffA RightPayoffB ObsProbRight
## 1 Berk29         400         400          750          400         0.69
## 2  Barc2         400         400          750          375         0.48
## 3 Berk17         400         400          750          375         0.50
## 4 Berk23         800         200            0            0         0.00
## 5  Barc8         300         600          700          500         0.33
## 6 Berk15         200         700          600          600         0.73
## 7 Berk26           0         800          400          400         0.22
##   LeftUtilityB RightUtilityB PredProbRight SquaredDiff
## 1          400           470           1.0      0.0961
## 2          400           450           1.0      0.2704
## 3          400           450           1.0      0.2500
## 4          320             0           0.0      0.0000
## 5          540           540           0.5      0.0289
## 6          600           600           0.5      0.0529
## 7          640           400           0.0      0.0484
```

The model predicts that the player will choose Left if weight < 0.2 or Right if weight > 0.2. Both predictions are suboptimal to weight == 0, which predicts that player B is indifferent between Left and Right in both games Barc8 and Berk15.

Model with two free parameters

We introduce a second parameter gamma (see Charness and Rabin 2002: 839). The precision parameter gamma measures the sensitivity of player B to differences in utility.

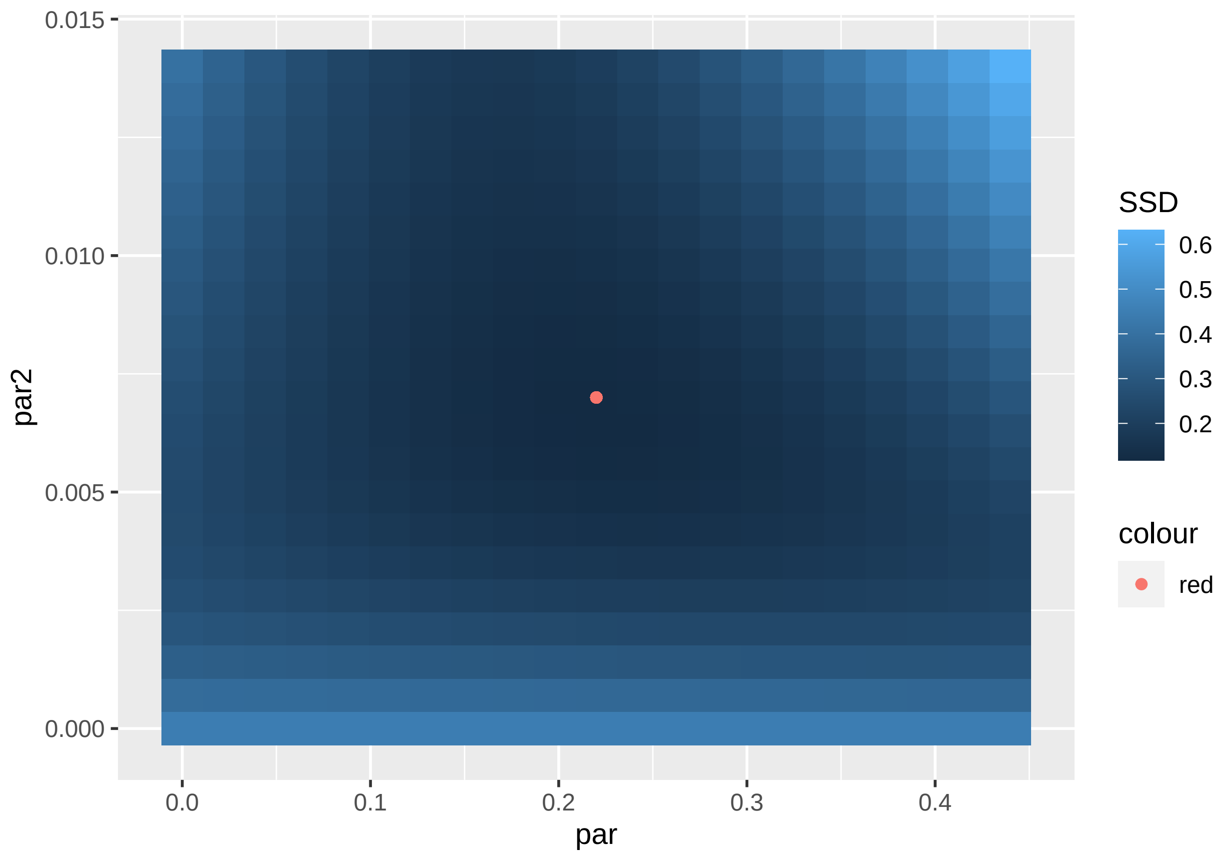
```
sumSqDiff <- function(parameter, parameter2){

dataEst <- dataRaw %>%
  mutate(LeftUtilityB = Utility(ownPayoff = LeftPayoffB,
                                 weightOtherPayoff = parameter,
                                 otherPayoff = LeftPayoffA),
         RightUtilityB = Utility(ownPayoff = RightPayoffB,
                                  weightOtherPayoff = parameter,
                                  otherPayoff = RightPayoffA)) %>%
  mutate(PredProbRight = exp(RightUtilityB * parameter2) /
                        (exp(LeftUtilityB * parameter2) + exp(RightUtilityB * parameter2)),
         SquaredDiff = (ObsProbRight - PredProbRight)^2)

SSD = sum(dataEst$SquaredDiff) # Sum of squared differences is returned by function
}

print(sumSqDiff(parameter = .1, parameter2 = 0))
```

```
## [1] 0.4467
```

```
### Heatmap
myheatmap <- function(parameter, parameter2, dev){
```

```r
# Grid
comparison_grid <- expand.grid(par = seq(parameter * (1 - dev),
                                         parameter * (1 + dev),
                                         length.out = 21),
                               par2 = seq(parameter2 * (1 - dev),
                                          parameter2 * (1 + dev),
                                          length.out = 21)) %>%
  group_by(par, par2) %>%
  nest()

# Results
grid_results <-
  comparison_grid %>%
  mutate(SSD = map2(par, par2, ~sumSqDiff(parameter = .x, parameter2 = .y))) %>%
  unnest(cols = SSD) %>%
  arrange(SSD)

# Best fitting paramters
opt_par <- grid_results$par[1]
opt_par2 <- grid_results$par2[1]

# Heatmap
ggplot(data = grid_results) +
  geom_tile(aes(x = par, y = par2, fill = SSD)) +
  geom_point(aes(x = opt_par, y = opt_par2, color = "red"))
}
```

```r
myheatmap(parameter = 0.22, # Gewicht auf Auszahlung von A
          parameter2 = .007, # Gamma*
          dev = 1)# Intervall der Paramtervariation
```




```
# * Wie stark richte ich mich nach meinen Präferenzen?
# 0 = Random Choice
# hohes Gamma = Nutzenmaximierung
```

The red dot shows the best fitting parameter combination

```
#help(optim)

sumSqDiff2 <- function(par){
  parameter <- par[1]
  parameter2 <- par[2]

dataEst <- dataRaw %>%
  mutate(LeftUtilityB = Utility(ownPayoff = LeftPayoffB,
                                weightOtherPayoff = parameter,
                                otherPayoff = LeftPayoffA),
         RightUtilityB = Utility(ownPayoff = RightPayoffB,
                                 weightOtherPayoff = parameter,
                                 otherPayoff = RightPayoffA)) %>%
  mutate(PredProbRight = exp(RightUtilityB * parameter2) /
                      (exp(LeftUtilityB * parameter2) + exp(RightUtilityB * parameter2)),
         SquaredDiff = (ObsProbRight - PredProbRight)^2)

SSD = sum(dataEst$SquaredDiff) # Sum of squared differences is returned by function
}

optim(par = c(0, .1), # Startwerte für Parameter weightOtherPayoff, gamma
```

```
      fn = sumSqDiff2) # zu minimierende funktion
```

```
## $par
## [1] 0.218409321 0.006681957
##
## $value
## [1] 0.1176641
##
## $counts
## function gradient
##       83       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Model with three free parameters

We estimate a noisy utility function with different weights on other payoff depending on whether player B is better off or worse off.

```
# Nutzenfunktion definieren
Utility2 <- function(ownPayoff, # Input1
                    weightOtherPayoffBetterOff,# Input2
                    weightOtherPayoffWorseOff,# Input3
                    otherPayoff) { # Input4

  weightOwnPayoff <- 1 - weightOtherPayoffBetterOff # Calculation

  if (otherPayoff > ownPayoff) {
    weightOwnPayoff <- 1 - weightOtherPayoffWorseOff # Calculation
  }

  return( (1 - weightOwnPayoff)  * otherPayoff + weightOwnPayoff * ownPayoff) # Output
}

sumSqDiff3 <- function(par){
  parameter <- par[1] # gewicht anderer payoff, wenn besser gestellt
  parameter2 <- par[2] # gewicht anderer payoff, wenn schlechter gestellt
  parameter3 <- par[3]# gamma

dataEst <- dataRaw %>%
  mutate(LeftUtilityB = Utility2(ownPayoff = LeftPayoffB,
                                weightOtherPayoffBetterOff = parameter,
                                weightOtherPayoffWorseOff = parameter2,
                                otherPayoff = LeftPayoffA),
         RightUtilityB = Utility2(ownPayoff = RightPayoffB,
                                 weightOtherPayoffBetterOff = parameter,
                                 weightOtherPayoffWorseOff = parameter2,
                                 otherPayoff = RightPayoffA)) %>%
  mutate(PredProbRight = exp(RightUtilityB * parameter3) /
```

```r
                          (exp(LeftUtilityB * parameter3) + exp(RightUtilityB * parameter3)),
         SquaredDiff = (ObsProbRight - PredProbRight)^2)

SSD = sum(dataEst$SquaredDiff) # Sum of squared differences is returned by function
}

optim(par = c(0, 0, 0), # Startwerte für Parameter = *
      fn = sumSqDiff3)
```

```
## $par
## [1] 0.24385967 0.06086561 0.04251490
##
## $value
## [1] 0.04987446
##
## $counts
## function gradient
##      126       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
# * weightOtherPayoffBetterOff, weightOtherPayoffWorseOff, gamma
```

- Player B puts a higher weight on the other payoff when he is better off
- The model fits the data better than a model that does not take the direction of inequality into account
- In comparison to Charness and Rabin we used less games to estimate the utility function and a different criterion: minimize sum squared differences instead of maximize log likelihood