R Notebook

For fitting the Block-Marschack and the other inequalities, we need a function that solves quadratic equations with inequality constraints. R offers several such packages, see section ‘Quadratic Optimization’ on: <https://cran.r-project.org/view=Optimization> Initially, we used the most popular package, quadprog (<https://cran.r-project.org/package=quadprog>) for this purpose. However, in some cases the function got stuck. Consequently, we tried other packages. Frr example, we observed good, albeit slow, results with Dykstra (<https://cran.r-project.org/package=Dykstra>). So all results reported here should replicate with Dykstra as well. We finally settled on quadprogpp which is a new and fast implementation of quadprog, which currently is only available from github: <https://github.com/fnoorian/quadprogpp> It requires a C++ compiler (e.g., Rtools on Windows or Xtools on Mac) and can then be installed via devtools

## Demographics

## Prepare Data

din$correct <- if\_else(din$Test\_T1\_originalIdentification == "F68", 1, 0)  
din$n <- if\_else(!is.na(din$Test\_T1\_originalIdentification) == "F68",1,0)  
  
  
dat <- din %>%   
 group\_by(Test\_T1\_lineupSize) %>%  
 summarise(correct = sum(correct),  
 n = n())  
  
dvector <- dat %>%   
 group\_by(Test\_T1\_lineupSize) %>%  
 summarise(corr = sum(correct),  
 incorrect = sum(n) - sum(correct)) %>%   
 select(-Test\_T1\_lineupSize) %>%   
 as.matrix() %>%   
 t() %>%   
 as.vector  
  
dvector

## [1] 296 38 262 82 223 108 209 113 185 145 182 149 365 361

prop\_vector <- dat %>%   
 group\_by(Test\_T1\_lineupSize) %>%   
 summarise(out = sum(correct)/sum(n)) %>%   
 {.$out}  
  
prop\_vector

## [1] 0.8862275 0.7616279 0.6737160 0.6490683 0.5606061 0.5498489 0.5027548

## Fit Block-Marschak

## inequality matrix

getGsq <- function(Q,dd){  
 ee <- c(rbind(Q, 1-Q))\*sum(dd[1:2])  
 # compute Gsq  
 Gsq <- 2\*sum(dd[dd!=0]\*(log(dd[dd!=0])-log(ee[dd!=0])))  
 return(Gsq)  
}  
  
#the following fits to something that I don't understand  
qpfit\_BM <- -1\*QP.Solve(  
 G = diag(7), #n by n matrix appearing in the quadratic function to be minimized.  
 g0 = prop\_vector, #vector on length n appearing in the quadratic function to be minimized.  
 CI = -1\*t(MM[-c(42:48),-c(1,9)]), #n by m constraints matrix.   
 ci0 = -1\*MM[-c(42:48),9] #constraints constants, with size m.   
 )  
qpfit\_BM

## [1] 0.8711353 0.7687420 0.6885933 0.6264624 0.5781227 0.5393473 0.5059095

#the following fits to the full Block-Marschak inequalities  
qpfit\_BMML <- -1\*QP.Solve(  
 G = diag(7),   
 g0 = prop\_vector,   
 CI = -1\*t(MM[,-c(1,9)]),   
 ci0 = -1\*MM[,9]  
 )  
qpfit\_BMML

## [1] 0.8719132 0.7694413 0.6887421 0.6259732 0.5772925 0.5388578 0.5068268

prop\_vector

## [1] 0.8862275 0.7616279 0.6737160 0.6490683 0.5606061 0.5498489 0.5027548

### yields rankings

rnko <- c(choose(7,0:7) \*   
 (c(MM[MM[,"max"] == 8,][-c(2),-c(1,9)]%\*%qpfit\_BMML) +   
 c(rep(0,7),1)))  
rnko

## [1] 5.068268e-01 2.242170e-01 1.344781e-01 1.344781e-01 -5.051515e-14  
## [6] -5.362377e-14 -1.632028e-14 -1.199041e-14

rnkn <- (1-rnko)/7  
rnkn

## [1] 0.07045331 0.11082615 0.12364599 0.12364599 0.14285714 0.14285714  
## [7] 0.14285714 0.14285714

cumsum(rnkn)

## [1] 0.07045331 0.18127946 0.30492544 0.42857143 0.57142857 0.71428571  
## [7] 0.85714286 1.00000000

cumsum(rnko)

## [1] 0.5068268 0.7310438 0.8655219 1.0000000 1.0000000 1.0000000 1.0000000  
## [8] 1.0000000

## G-square:

# Block- Marschak

fitdata\_BM <- getGsq(qpfit\_BM, dvector)  
fitdata\_BM

## [1] 1106.618

# Block-Marschak with monotonic likelihood

fitdata\_BMML <- getGsq(qpfit\_BMML, dvector)  
fitdata\_BMML

## [1] 1106.602

### multiplicative inequalities (note \*\* = ^)

epc <- c(1,qpfit\_BMML)  
  
all(c(  
epc[3] > epc[2]\*\*2 , # 2 > 1+1  
epc[4] > epc[2]\*\*3 , # 3 > 1+1+1  
epc[4] > epc[2]\*epc[3] , # 3 > 1+2  
epc[5] > epc[2]\*\*4 , # 4 > 1+1+1+1  
epc[5] > epc[3]\*epc[2]\*\*2 , # 4 > 2+1+1  
epc[5] > epc[3]\*\*2 , # 4 > 2+2  
epc[5] > epc[4]\*epc[2] , # 4 > 3+1  
epc[6] > epc[2]\*\*5 , # 5 > 1+1+1+1+1  
epc[6] > epc[2]\*\*3\*epc[3] , # 5 > 2+1+1+1  
epc[6] > epc[3]\*\*2\*epc[2], # 5 > 2+2+1  
epc[6] > epc[4]\*epc[2]\*\*2 , # 5 > 3+1+1  
epc[6] > epc[4]\*epc[3] , # 5 > 3+2  
epc[6] > epc[5]\*epc[2] , # 5 > 4+1  
epc[7] > epc[2]\*\*6 , # 6 > 1+1+1+1+1+1  
epc[7] > epc[3]\*epc[2]\*\*4 , # 6 > 2+1+1+1+1  
epc[7] > epc[3]\*\*2\*epc[2]\*\*2, # 6 > 2+2+1+1  
epc[7] > epc[3]\*\*3 , # 6 > 2+2+2  
epc[7] > epc[4]\*epc[2]\*\*3 , # 6 > 3+1+1+1  
epc[7] > epc[4]\*epc[3]\*epc[2], # 6 > 3+2+1  
epc[7] > epc[4]\*\*2 , # 6 > 3+3  
epc[7] > epc[5]\*epc[2]\*\*2 , # 6 > 4+1+1  
epc[7] > epc[5]\*epc[3] , # 6 > 4+2  
epc[7] > epc[6]\*epc[2] , # 6 > 5+1  
epc[8] > epc[2]\*\*7 , # 7 > 1+1+1+1+1+1+1  
epc[8] > epc[3]\*epc[2]\*\*5 , # 7 > 2+1+1+1+1+1  
epc[8] > epc[3]\*\*2\*epc[2]\*\*3 , # 7 > 2+2+1+1+1  
epc[8] > epc[3]\*\*3\*epc[2] , # 7 > 2+2+2+1  
epc[8] > epc[4]\*epc[2]\*\*4 , # 7 > 3+1+1+1+1  
epc[8] > epc[4]\*epc[3]\*epc[2]\*\*2 , # 7 > 3+2+1+1  
epc[8] > epc[4]\*epc[3]\*\*2, # 7 > 3+2+2  
epc[8] > epc[4]\*\*2\*epc[2] , # 7 > 3+3+1  
epc[8] > epc[5]\*epc[2]\*\*3, # 7 > 4+1+1+1  
epc[8] > epc[5]\*epc[3]\*epc[2] , # 7 > 4+2+1  
epc[8] > epc[5]\*epc[4] , # 7 > 4+3  
epc[8] > epc[6]\*epc[2]\*\*2 , # 7 > 5+1+1  
epc[8] > epc[6]\*epc[3] , # 7 > 5+2  
epc[8] > epc[7]\*epc[2] # 7 > 6+1  
))

## [1] TRUE

## Therefore, G-Square equal to 0, p-value equal to 1.

## —————————————————————

## get bootstrap p-value -

## —————————————————————

nsim <- 100000  
gfitdata\_BM <- vector("numeric", nsim)  
gfitdata\_BMML <- vector("numeric", nsim)  
  
get\_np\_sample <- function(dvec) {  
 out <- vector("numeric", length(dvec))  
 for (i in seq\_len(length(dvec)/2)) {  
 sel <- (i-1)\*2 + (1:2)  
 out[sel] <- rmultinom(1, sum(dvec[sel]), dvec[sel]/sum(dvec[sel]) )[,1]  
 }  
 out  
}  
  
get\_p\_sample <- function(dvec, prob) {  
 out <- vector("numeric", length(dvec))  
 for (i in seq\_len(length(dvec)/2)) {  
 sel <- (i-1)\*2 + (1:2)  
 out[sel] <- rmultinom(1, sum(dvec[sel]), c(prob[i], 1-prob[i]) )[,1]  
 }  
 out  
}  
  
set.seed(666)  
for(iii in 1:nsim){  
  
 gdd <- get\_np\_sample(dvector)  
   
 g\_qpfit <- -1\*QP.Solve(diag(7), gdd[seq(1,13,2)]/sum(dvector[1:2]),   
 -1\*t(MM[-c(42:48),-c(1,9)]), -1\*MM[-c(42:48),9])  
   
   
 gdd <- get\_p\_sample(dvector, g\_qpfit)  
   
 g\_qpfit <- -1\*QP.Solve(diag(7), gdd[seq(1,13,2)]/sum(dvector[1:2]),   
 -1\*t(MM[-c(42:48),-c(1,9)]), -1\*MM[-c(42:48),9])   
   
   
 gfitdata\_BM[iii] <- getGsq(g\_qpfit,gdd)  
   
}  
## p-value ##  
mean(gfitdata\_BM > fitdata\_BM)

## [1] 1

## [1] 0.98182  
describe(gfitdata\_BM)

## vars n mean sd median trimmed mad min max range  
## X1 1 1e+05 1394.82 27.43 1392.99 1393.77 26.88 1305.28 1553.18 247.89  
## skew kurtosis se  
## X1 0.42 0.38 0.09

set.seed(999)  
for(iii in 1:nsim){  
  
 gdd <- get\_np\_sample(dvector)  
   
 g\_qpfit <- -1\*QP.Solve(diag(7), gdd[seq(1,13,2)]/sum(dvector[1:2]),   
 -1\*t(MM[,-c(1,9)]), -1\*MM[,9])  
   
   
 gdd <- get\_p\_sample(dvector, g\_qpfit)  
   
 g\_qpfit <- -1\*QP.Solve(diag(7), gdd[seq(1,13,2)]/sum(dvector[1:2]),   
 -1\*t(MM[,-c(1,9)]), -1\*MM[,9])   
   
   
 gfitdata\_BMML[iii] <- getGsq(g\_qpfit,gdd)  
   
}  
## p-value ##  
mean(gfitdata\_BMML > fitdata\_BMML)

## [1] 1

# [1] 0.92689

## Figure

library("ggplot2", lib.loc="~/R/win-library/3.5")  
library("grDevices", lib.loc="C:/Program Files/R/R-3.5.3/library")  
library("graphics", lib.loc="C:/Program Files/R/R-3.5.3/library")  
dd <- dvector  
print(dd)

## [1] 296 38 262 82 223 108 209 113 185 145 182 149 365 361

par(mfrow=c(1,2))  
h <- dd[seq(1,13,2)]  
h

## [1] 296 262 223 209 185 182 365

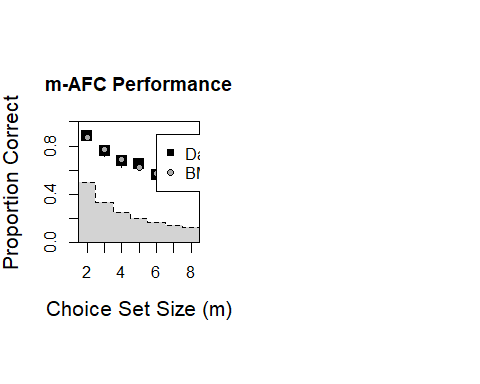
size <- c(sum(dd[1:2]),   
 sum(dd[3:4]),   
 sum(dd[5:6]),   
 sum(dd[7:8]),   
 sum(dd[9:10]),   
 sum(dd[11:12]),  
 sum(dd[13:14])  
 )  
size

## [1] 334 344 331 322 330 331 726

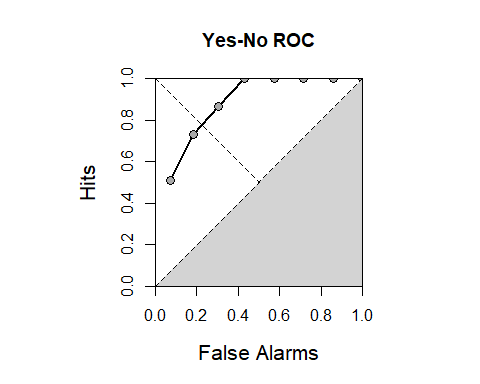
tpc <- dd[seq(1,13,2)]/size  
print(tpc)

## [1] 0.8862275 0.7616279 0.6737160 0.6490683 0.5606061 0.5498489 0.5027548

#pdf("Figure\_E1.pdf", 8, 4.5)  
  
par(pty="s")  
plot(2:8,tpc,pch=15,cex=1.5,xlim=c(1.5,8.5),ylim=c(0,1),  
 xlab="Choice Set Size (m)", ylab="Proportion Correct",  
 cex.lab=1.3, main="m-AFC Performance",xaxs="i",yaxs="i")  
  
cil <- tpc - 1.96\*sqrt((tpc\*(1-tpc))/sum(dd[1:2]))  
ciu <- tpc + 1.96\*sqrt((tpc\*(1-tpc))/sum(dd[1:2]))  
  
for(ii in 1:7) segments(ii+1,ciu[ii],ii+1,cil[ii])  
  
polygon(rep(seq(1.5,8.5,1),each=2),c(0,rep(1/(2:8),each=2),0),  
 col="lightgray",border = FALSE)  
lines(rep(seq(1.5,8.5,1),each=2),c(0,rep(1/(2:8),each=2),0),lty=2)  
box()  
points(2:8,c((qpfit\_BMML)),pch=21,cex=1,bg="darkgray")  
  
  
legend(6, 0.9, legend=c("Data","BMI-ML"),   
 pch=c(15,21),col=rep("black",2),   
 pt.bg=c(NULL,"darkgray"), cex=1)

 # ROC #

par(pty="s")  
plot(cumsum(rnkn),cumsum(rnko),typ="l",pch=16,xaxs="i",yaxs="i",  
xlab="False Alarms", ylab="Hits",cex.lab=1.3, main="Yes-No ROC",lwd=2,  
xlim=c(0,1), ylim=c(0,1),cex=1.3)  
points(cumsum(rnkn),cumsum(rnko),pch=21,bg="darkgray",cex=1.3)  
polygon(c(0,1,1),c(0,0,1),col="lightgray",border = FALSE)  
abline(0,1,lty=2)  
segments(0,1,0.5,0.5,lty=2)  
box()



#dev.off()