```
### INFERRING INEQUALITY Examples and RESULTS

# The first part of this file contains examples to illustrate the functions specified in the Inferring Inequality Functions file.
# Run all of the code in that file before running these experiments.

# The second part produces the results with the results published in Abul Naga, Stapenhurst and Yalonetzky "Inferring Inequality: Testing for Median Preserving Spreads in Ordinal Data":
# Happiness Inequality in the United States (table 5)
# Inequality in Self-assessed Health in Europe (figure 6)
# Size-boundary and power-locus curves (figures 3 and 5)

# Please send any questions or comments to Christopher Stapenhurst, cpstapenhurst.academic.ws

##### PART 1 EXAMPLES

### Input data (from section 3.1 worked example)
# Self-reported happiness response counts 2002 (x) and 1972 (y)
x <- c(154, 765, 480)
y <- c(218, 789, 599)

### Descriptive analysis
## sample sizes
nx<-sum(x)
ny<-sum(y)
c(nx,ny)

# empirical mass functions
f<- x/nx
g<- y/ny
rbind(f,g)

# empirical distribution functions
F<-cumsum(f)
G<-cumsum(g)
L<-cumsum(x*y)/(nx*ny)
rbind(F,G)

#MPS plot
mps.plot(F,G)

# median categories
find.m(F)
find.m(G)

# is F and mps of G?
is.mps(G,F)
# is G and mps of F?
is.mps(F,G)

### Null hypothesis: G is not an MPS of F.

### Test statistics
z.stat(x,y)
LR.stat(x,y)

#z stat worked example
signal<-sqrt(L*(1-L)/(nx*ny))
((G-F)/signal/(nx*ny)^sqrt(nx*ny))[1]

#LR stat worked example
cmle<-find.CMLE(x,y)
ftilde<-cmle[1:3]
gtilde<-cmle[1:3+3]
2*log(dmultinom(x,prob=x)+dmultinom(y,prob=y)/dmultinom(x,prob=ftilde)/dmultinom(y,prob=gtilde))

##### Test functions
asymp.test(x,y, stat.fun="LR.stat")
asymp.test(x,y, stat.fun="z.stat")
boot.test(x,y, stat.fun="LR.stat",B=9999)
boot.test(x,y, stat.fun="z.stat",B=9999)



### PART 2 RESULTS

# create a new folder to store results and change directory.
setwd("Inference with ordered states/results")

#### Happiness Inequality in the United States
# This code replicates the results of Inferring inequality table 5:
# i.e. the bootstrap p-value of the orderings displayed in Dutta and Foster (2013) table 2.

# Input data from Dutta and Foster (2013) table 1.
mat<-matrix(
  c(
    1972, 13.600, 49.100, 37.300, 1606,
    1973, 12.286, 50.932, 36.782, 1500,
    1974, 12.515, 49.184, 38.291, 1480,
    1975, 12.973, 53.630, 33.397, 1485,
    1976, 12.249, 52.920, 34.831, 1499,
    1977, 11.013, 53.242, 35.745, 1527,
    1978, 8.369, 56.189, 35.442, 1517,
    1980, 11.600, 52.000, 36.400, 1462,
    1982, 11.700, 53.499, 34.801, 1505,
    1983, 12.092, 56.239, 31.668, 1573,
    1984, 11.614, 52.091, 36.295, 1445,
    1985, 8.600, 58.400 , 33.100, 1530,
    1986, 9.200, 55.800 , 35.000, 1449,
    1987, 9.700, 53.300 , 37.000, 1437,
    1988, 8.241, 55.696 , 36.065, 1466,
    1989, 8.793, 56.737 , 34.470, 1826,
    1990, 7.740, 56.527 , 35.733, 1361,
    1991, 9.485, 58.004 , 32.511, 1504,
    1993, 9.736, 56.866 , 33.399, 1601,
    1994, 11.287, 58.216, 30.497, 2977,
    1996, 10.902, 57.357, 32.141, 2885,
    1998, 10.896, 55.851, 33.353, 2806,
    2000, 9.644, 56.436 , 33.921, 2777,
    2002, 11.282, 55.846, 32.872, 1369,
    2004, 11.696, 54.718, 33.586, 1337,
    2006, 10.552, 55.901, 33.547, 2828,
    2008, 13.289, 54.749, 31.962, 1942,
    2010, 14.192, 57.010, 28.798, 2039)
  ,ncol=5,byrow=TRUE)

# format data to give samples x and y
mat1<-round(mat[,2:4]*mat[,5])/100
mat2<-cbind(matrix(rep(t(mat1),nrow(mat1)),ncol=3,byrow=TRUE),matrix(rep(mat1,each =nrow(mat1)),ncol=3))
# set the seed
set.seed(1)
# apply bootstrap LR test with B=499 bootstrap samples
res<-round(apply(mat2,1,function(z) boot.test(z[1:3],z[4:6], B=499)),2)
res1<-round(matrix(res,nrow=nrow(mat1)),2)
colnames(res1)<-mat[,1]
rownames(res1)<-mat[,1]

#display results in table 5
res1

#calculate the number of significant results at 1, 5 and 10% levels.
sum(res1<=0.01)
sum(res1<=0.05)
sum(res1<=0.1)
sum(res1<1)


#### Inequality in Self-assessed Health in Europe

x<-round(c( 0.01,0.04,0.19,0.54,0.22) * 13328)  # netherlands
y<-round(c(0.03,0.06,0.21,0.45,0.25) * 5906) # denmark

nx<-sum(x)
ny<-sum(y)
k<-length(x)
f<-x/sum(x)
F<-cumsum(f)
g<-y/sum(y)
G<-cumsum(g)

# bar graph of mass functions f and g
pdf("EUbarplot.pdf",height=7,width=7)
barplot(rbind(f,g), main="Self reported health",
        legend = c("Netherlands","Denmark"), xlab="Health category", col=c("darkblue","red"),
        names.arg=c("(1) very bad","(2) bad","(3) average","(4) good","(5) very good"),
        beside=TRUE)
dev.off()

# MPS plot of F and G.
pdf("EUMPSplot.pdf",height=3,width=3)
mps.plot(F,G)
dev.off()

# find closest null distribution to (f,g)
cmle<-find.CMLE(f*nx,g*ny)
f0<-cmle[1:k]
g0<-cmle[k+1:k]
F0<-cumsum(f0)
G0<-cumsum(g0)

# PLot closest null distribution
pdf("EUCMPSplot.pdf",height=3,width=3)
mps.plot(F0,g0)
dev.off()

# initialise data frame and size and power graph objects
df<-data.frame(i=0:1000/1000)
rng<-0.1 # plot range
grob0<- ggplot()+ylab("Actual size")+xlab("Nominal size")+geom_line(data=df,aes(i,i))+coord_cartesian(ylim=c(0, rng),xlim=c(0, rng))
grob40<- ggplot()+ylab("Power")+xlab("Actual size")+geom_line(data=df,aes(i,i))+coord_cartesian(ylim=c(0, rng),xlim=c(0, rng))

# set the seed
set.seed(2)
# 10000 monte carlo simulations
M<-10000
# Specify B=499 Bootstrap samples
B<-499

df$LR0<-ecdf.fun(monte.carlo(asymp.test,f=f0,g=g0,nx=nx,ny=ny,M=M,stat.fun="LR.stat"))
df$Z0<-ecdf.fun(monte.carlo(asymp.test,f=f0,g=g0,nx=nx,ny=ny,M=M,stat.fun="z.stat"))
df$LRA<-ecdf.fun(monte.carlo(asymp.test,f=f,g=g,nx=nx,ny=ny,M=M,stat.fun="LR.stat"))
df$ZA<-ecdf.fun(monte.carlo(asymp.test,f=f,g=g,nx=nx,ny=ny,M=M,stat.fun="z.stat"))
df$LR0<-ecdf.fun(monte.carlo(boot.test,f=f0,g=g0,nx=nx,ny=ny,M=M,B=B,stat.fun="LR.stat"))
df$Z0<-ecdf.fun(monte.carlo(boot.test,f=f0,g=g0,nx=nx,ny=ny,M=M,B=B,stat.fun="z.stat"))
df$LRA<-ecdf.fun(monte.carlo(boot.test,f=f,g=g,nx=nx,ny=ny,M=M,B=B,stat.fun="LR.stat"))
df$ZA<-ecdf.fun(monte.carlo(boot.test,f=f,g=g,nx=nx,ny=ny,M=M,B=B,stat.fun="z.stat"))

# add results to graph objects
grob0<-grob0+geom_line(data=df,aes(i,bLR0),color="blue")+geom_line(data=df,aes(i,aLR0),color="red")+
  geom_line(data=df,aes(i,i),color="black")+geom_line(data=df,aes(i,aZ0),color="red")
grob40<-grob40+geom_line(data=df,aes(bLRA,bLRA),color="blue")+geom_line(data=df,aes(aLR0,aLRA),color="red")+
  geom_line(data=df,aes(bZ0,bZA),color="black")+geom_line(data=df,aes(aZ0,aZA),color="red")

# produce and save size and power curves curve graphics
pdf("EUsize.pdf",height=3,width=3)
grob0
dev.off()
pdf("EUpower.pdf",height=3,width=3)
grob40
dev.off()

# arrange size and power curves in panel
nparr<-parse(text=paste("f=",paste(round(f,2), collapse = ","), ",  g=(",paste(round(g,2), collapse = ","),"); ","nx=", nx,", ny=",ny), collapse = "")
pdf("EU2pan.pdf",height=3,width=6)
suppressWarnings(grid.arrange(grob0,grob40,nrow=1,top="Inequality in self-assessed health"))
dev.off()




#### Size-boundary and power-locus curves
set.seed(1)
M<-10000
B<-999

# Conduct experiments
data.10.10<-pcFun(10,10,M=M,B=B)
data.10.100<-pcFun(10,100,M=M,B=B)
data.10.1000<-pcFun(10,1000,M=M,B=B)
data.100.10<-pcFun(100,10,M=M,B=B)
data.1000.10<-pcFun(1000,10,M=M,B=B)
data.100.100<-pcFun(100,100,M=M,B=B)
data.1000.1000<-pcFun(1000,1000,M=M,B=B)
data.1000.100<-pcFun(1000,100,M=M,B=B)
data.100.1000<-pcFun(100,1000,M=M,B=B)

# Plot results

# size-boundary curves at 5% nominal level (figure 3)
size<-0.05
pdf("size05.pdf",height=7,width=7)
suppressWarnings(
  grid.arrange(
    size.plot.fun(data.10.10,size=size),
    size.plot.fun(data.10.100,size=size),
    size.plot.fun(data.10.1000,size=size),
    size.plot.fun(data.100.10,size=size),
    size.plot.fun(data.100.100,size=size),
    size.plot.fun(data.100.1000,size=size),
    size.plot.fun(data.1000.10,size=size),
    size.plot.fun(data.1000.100,size=size),
    size.plot.fun(data.1000.1000,size=size),
    ncol=3,nrow=3)
)
dev.off()

# size-boundary curves at 1% nominal level
size<-0.01
pdf("size01.pdf",height=7,width=7)
suppressWarnings(
  grid.arrange(
    size.plot.fun(data.10.10,size=size),
    size.plot.fun(data.10.100,size=size),
    size.plot.fun(data.10.1000,size=size),
    size.plot.fun(data.100.10,size=size),
    size.plot.fun(data.100.100,size=size),
    size.plot.fun(data.100.1000,size=size),
    size.plot.fun(data.1000.10,size=size),
    size.plot.fun(data.1000.100,size=size),
    size.plot.fun(data.1000.1000,size=size),
    ncol=3,nrow=3)
)
dev.off()

# size-boundary curves at 10% nominal level
size<-0.1
pdf("size1.pdf",height=7,width=7)
suppressWarnings(
  grid.arrange(
    size.plot.fun(data.10.10,size=size),
    size.plot.fun(data.10.100,size=size),
    size.plot.fun(data.10.1000,size=size),
    size.plot.fun(data.100.10,size=size),
    size.plot.fun(data.100.100,size=size),
    size.plot.fun(data.100.1000,size=size),
    size.plot.fun(data.1000.10,size=size),
    size.plot.fun(data.1000.100,size=size),
    size.plot.fun(data.1000.1000,size=size),
    ncol=3,nrow=3)
)
dev.off()

# power-locus curves at 5% nominal level (figure 5)
size<-0.05
pdf("power05.pdf",height=7,width=7)
suppressWarnings(
  grid.arrange(
    power.plot.fun(data.10.10,size=size),
    power.plot.fun(data.10.100,size=size),
    power.plot.fun(data.10.1000,size=size),
    power.plot.fun(data.100.10,size=size),
    power.plot.fun(data.100.100,size=size),
    power.plot.fun(data.100.1000,size=size),
    power.plot.fun(data.1000.10,size=size),
    power.plot.fun(data.1000.100,size=size),
    power.plot.fun(data.1000.1000,size=size),
    ncol=3,nrow=3)
)
dev.off()

# power-locus curves at 1% nominal level
size<-0.01
pdf("power01.pdf",height=7,width=7)
suppressWarnings(
  grid.arrange(
    power.plot.fun(data.10.10,size=size),
    power.plot.fun(data.10.100,size=size),
    power.plot.fun(data.10.1000,size=size),
    power.plot.fun(data.100.10,size=size),
    power.plot.fun(data.100.100,size=size),
    power.plot.fun(data.100.1000,size=size),
    power.plot.fun(data.1000.10,size=size),
    power.plot.fun(data.1000.100,size=size),
    power.plot.fun(data.1000.1000,size=size),
    ncol=3,nrow=3))
dev.off()

# power-locus curves at 10% nominal level
size<-0.1
pdf("power1.pdf",height=7,width=7)
suppressWarnings(
  grid.arrange(
    power.plot.fun(data.10.10,size=size),
    power.plot.fun(data.10.100,size=size),
    power.plot.fun(data.10.1000,size=size),
    power.plot.fun(data.100.10,size=size),
    power.plot.fun(data.100.100,size=size),
    power.plot.fun(data.100.1000,size=size),
    power.plot.fun(data.1000.10,size=size),
    power.plot.fun(data.1000.100,size=size),
    power.plot.fun(data.1000.1000,size=size),
    ncol=3,nrow=3))
dev.off()

#### END ####
```