SUPPLEMENTARY MATERIAL

The R code for the simulation described in the paper: How Robust Are Multi-Rater Inter-Rater Reliability Indices to Changes in Frequency Distribution?

```
#################################################################
# How Robust Are Multi-Rater Inter-Rater Reliability Indices    #
# to Changes in Frequency Distribution?                         #
#################################################################
#
# A Monte Carlo simulation exploring the effects of level changes
# and frequency distribution changes on 5 IR indices.

###### GLOBAL VARIABLES ########
numFDs = 6
numADs = 6
allIRs = array(0,c(5,11,numFDs,numADs))
#################################

# Returns a 6 x L matrix where (a,b) is probability of a scorer 1 giving rating b
# assuming frequency distribution a.  L is the number of levels in the scale.
# All 6 FDs are special cases of the beta-binomial distribution.
buildFDMatrix = function(L = 9)
{
  # need dbetabinom.ab function
  library(VGAM)

  shapes = list(c(0.25,0.25), c(1,1), c(2,2), c(50,50), c(25,50), c(5,50))

  probs = array(0,c(numFDs,L))
  for (i in 1:numFDs) probs[i,] = dbetabinom.ab(x=0:(L-1),size=L-1,shape1=shapes[[i]][1],
                                                shape2=shapes[[i]][2])

  return(probs)
}


# Create a discrete uniform distribution with finite support on {a,a+1,...,b}
# x is a vector of indices containing the range a:b, 0s are placed outside a:b
discUnif = function(x,a,b)
{
  prob = 1/(b-a+1)
  probs = rep(0,length(x))
  probs[a:b-x[1]+1] = prob
  return(probs)
}


# Create a discrete triangluar distribution with support on {a,a+1,...,b}; size of support
# should be odd.  If so, the high point will be at (b-a)/2; values at a and b are 0
discTriang = function(x,a,b)
{
  if ((b-a) %% 2 != 0) stop("Size of finite support must be odd")
  c = (b+a)/2
  h = 2/(b-a)
  slope = h/(c-a)
  ups = slope * (a:c - a)
  downs = -slope * ((c+1):b - b)
  triang = c(ups,downs)
  triang = triang/sum(triang)
  probs = rep(0,length(x))
  probs[a:b-x[1]+1] = triang
  return(probs)
}


# Returns a 6 x L x L matrix where (a,b,c) is probability of a scorer giving rating c
# when rater 1 gives rating b assuming an agreement distribution a.  L is the number of
# levels in the scale.
buildADMatrix = function(L = 9)
{
  # even L values are a problem for binom distributions since we want a mode
  # in our ADs; add 2 to move 0 endpoints of triangular distribution outside support
  B = ifelse(L %% 2 == 0, L,L-1)+2
```

```
  ADs = list(function(B) dbinom(x=(2-B):(2+B),prob=1/2,size=4),
             function(B) dbinom(x=(-B/2):(3*B/2),prob=1/2,size=B),
             function(B) discTriang(x=-B:B,-2,2),
             function(B) discTriang(x=-B:B,-floor(B/2),floor(B/2)),
             function(B) discUnif(x=-B:B,-floor(B/4),floor(B/4)),
             function(B) discUnif(x=-B:B,-B,B))

  # now move the support window around all possible R1 values
  probs = array(0,c(numADs,L,L))
  for (a in 1:numADs) {
    for (r1 in 1:L) {
      lb = (B+2-r1)
      ub = lb + L - 1
      # shift!
      unscaled = ADs[[a]](B)[lb:ub]
      # rescale probabilities when things fall outside 1:L
      unscaled[is.na(unscaled)] = 0
      probs[a,r1,] = unscaled/sum(unscaled)
    }
  }

  return(probs)
}


# This function returns an array where the rows represent different frequency distributions
# and the columns are different agreement distributions.  The entries are the IRs unders
# those conditions, averaged across numSims simulations.  By repeatedly calling this
# method with various inputs, we can explore how all the parameters interrate, and hence,
# what factors most powerfully influence IR.
buildIRs = function(numLevels = 9, numUnits = 20, numRaters = 8, numSims = 5,
                    IRfunc = krippen.alpha.raw)
{
  fdmat = buildFDMatrix(numLevels)
  admat = buildADMatrix(numLevels)


  ratings = array(0,c(numFDs,numADs,numRaters,numUnits,numSims))

  # build rater 1's scores using fdmat
  for (fd in 1:numFDs)
    ratings[fd, ,1, , ] = sample(x=c(1:numLevels),
                                 size=numADs*numUnits*numSims,
                                 prob=fdmat[fd,],
                                 replace=TRUE)


  # build scores for raters 2 through numRaters using admat
  for (fd in 1:numFDs)
    for (ad in 1:numADs)
      for (u in 1:numUnits)
        for (s in 1:numSims) {
          r1 = ratings[fd,ad,1,u,s]
          ratings[fd,ad,2:numRaters,u,s] = sample(x=c(1:numLevels),
                                                  size=numRaters-1,
                                                  prob=admat[ad,r1,],
                                                  replace=TRUE)
        }

  # build IRs for each sim
  IRVals = array(0,c(numFDs,numADs,numSims))
  for (fd in 1:numFDs)
    for (ad in 1:numADs)
      for (s in 1:numSims)
        IRVals[fd,ad,s] = IRfunc(t(ratings[fd,ad,,s]),weights="quadratic",print=FALSE)[3]


  return(apply(IRVals,c(1,2),mean))
}


# Builds all the IRs used in the paper.  Approx run time: 1 hour, 40 minutes
# Alters the global allIRs matrix where (ir,L,fd,ad) is found by:
# 1) Take a fixed fd and ad.  Build a ratings table with 8 raters and 100 units
#    assuming a scale with L levels.
# 2) Calculate the IR using IR function ir.
# 3) Repeat the process for 500 iterations and average the 500 IR results.
# This value goes in spot (ir,L,fd,ad).
#
# Possible levels (values for L): 4 to 11
# Possible IR indices (values for ir): fleiss, conger, krippendorff,
#                                      brennan-prediger, gwet AC2 (called ac1 below)
buildAllIRs = function()
{
  irfuncs = list(fleiss.kappa.raw, conger.kappa.raw, krippen.alpha.raw,
                 bp.coeff.raw, gwet.ac1.raw)
  for (ir in 1:5)
    for (L in 4:11) {
      allIRs[ir,L,] <<- buildIRs(numLevels=L, numUnits=100, numRaters = 8,
                                 numSims = 500, IRfunc = irfuncs[[ir]])
```

```
      #code to watch progress
      print(paste("done with", ir, " ",L))
    }
}


# Creates visualizations for the allIRs global variable which is passed as the
# parameter data; prints a black and white figure if color is false
# generates 5 total plots; these are send to PDFs if printToFile is true
createLevelPlots = function(data, color = TRUE, printToFile = F)
{
  library(lattice)

  IRNames = c("Fleiss's kappa","Conger's kappa", "Krippendorff's alpha",
              "Brennan-Prediger coefficient","Gwet's AC2")

  path = "C://Users//dquarfoot//Desktop//GradSchool//Dissertation//AmerStat Paper//FinalVersion"
  fileNames = c("fleissplot.pdf","congerplot.pdf","krippplot.pdf",
                "bpplot.pdf","gwetplot.pdf")
  version = ifelse(color,"col","bw")
  fileNames = paste(version,fileNames,sep="")
  fileNames = paste(path,fileNames,sep="//")


  myPanel = function(x, y, z, ...)
  {
    panel.levelplot(x,y,z,...)
    panel.text(x, y, round(z,2))
  }

  if (color) { # use color for online printing
    pal = cm.colors(40)
  } else # use white to gray for journal printing
    pal = gray(100:61/100)


  for (ir in 1:5) {
    X = c(numADs:1)
    Y = c(1:numFDs)
    grid = expand.grid(X=X,Y=Y)
    grid$Z = as.vector(data[ir,9,,])  # use 9-level Likert scale
    if (printToFile) pdf(file=fileNames[ir],width = 6, height = 4)

    plot(levelplot(Z ~ Y*X, grid, panel = myPanel,
                   col.regions = pal,
                   scales=list(x=list(at=c(1:numADs), labels=paste("AD",c(1:numADs))),
                               y=list(at=c(numFDs:1), labels=paste("FD",c(1:numFDs)))),
                   xlab="",ylab="",main=paste("Average IRs using",IRNames[ir])))

    if (printToFile) dev.off()
  }
}


# Makes the plots studying the effect of the number of levels, L
# generates 2 total plots; again, these are sent to a PDF if printToFile is true
createViz2 = function(data,color=TRUE,printToFile=F)
{
  library(latticeExtra)

  path = "C://Users//dquarfoot//Desktop//GradSchool//Dissertation//AmerStat Paper//FinalVersion"
  fileNames = c("levels13.pdf","levels46.pdf")
  version = ifelse(color,"col","bw")
  fileNames = paste(version,fileNames,sep="")
  fileNames = paste(path,fileNames,sep="//")

  colorModel = ifelse(color,"srgb","gray")

  ind = which(!is.na(data),arr.ind=TRUE)
  df = as.data.frame(cbind(data,ind))
  names(df) = c("val","IR","levels","FD","AD")

  # not enough space for 6x6 plot, so we break it into 2 3x6 plots
  plotgroup = list(c(1:3),c(4:6))

  for (i in 1:2) {
    ddf = subset(df,(FD %in% plotgroup[[i]]))

    # append FD or AD to number to make clearer in plot
    ddf$FD = factor(paste("FD",ddf$FD))
    ddf$AD = factor(paste("AD",ddf$AD))
    ddf$FD = factor(ddf$FD,levels = rev(paste("FD",plotgroup[[i]]))) # fix plot order

    if (printToFile) pdf(file=fileNames[i],paper='a4r',width=11,
                         height = 8,colormodel=colorModel)

    # now plot
    plot(useOuterStrips(
      xyplot(val ~ levels | factor(AD) + factor(FD),group = factor(IR),
             data=ddf, main="Influence of Levels on IR",
             xlab = "Number of Levels", ylab = "IR Value", type=c("l","p"),
```

```
            scales=list(x= list(at=1:8,labels = 4:11, cex = 0.7)),
            par.settings = list(superpose.symbol = list(pch = 1:5),
                                superpose.line=list(lty=1:5)),
            auto.key=list(text = c("Fleiss","Conger","Krippendorff",
                                   "Brennan-Prediger","Gwet AC1/2"),
                          space="top", columns=5,  points=T, lines=T, cex.title=1),
            strip = strip.custom(factor.levels = factor(ddf$AD)),
            strip.left=strip.custom(factor.levels = factor(ddf$FD))))) + theme_bw()

    if (printToFile) dev.off()
  }
}


# Draws the 6 FDs in a grid layout of row rows and col columns
drawFDs = function(row=3,col=2,L = 9)
{
  names = c("FD 1: Extremes","FD 2: Uniform","FD 3: Central","FD 4: Binomial",
            "FD 5: Skewed","FD 6: Very Skewed")
  par(mar=c(4.5,2,1,1))
  par(mfrow=c(row,col))
  fdmat = buildFDMatrix(L)
  for (i in 1:numFDs)
    barplot(fdmat[i,], #ylim=c(0,max(fdmat)+0.05),
            xlab=names[i])
}


# Draws the 6 ADs in a grid layout of row rows and col columns
drawADs = function(row=3,col=2,L = 9,r1 = floor((L+1)/2))
{
  names = c("AD 1: Fixed-Width Binomial","AD 2: Fully-Scaling Binomial",
            "AD 3: Fixed-Width Triangular","AD 4: Fully-Scaling Triangular",
            "AD 5: Partially-Scaling Uniform","AD 6: Fully-Scaling Uniform (Random)")
  par(mar=c(4.5,2,1,1))
  par(mfrow=c(row,col))
  admat = buildADMatrix(L)
  for (i in 1:numFDs) {
    barplot(admat[i,r1,],xlab=names[i])
    axis(1,at=c(r1+0.5),labels=c("*"))
  }
}


# Shows a faceted bar chart for AD2 assuming L = 9 to give a feel for how
# the AD depends on R1's score
drawViz = function(L = 9, ADNum = 2)
{
  library(ggplot2)

  data =buildADMatrix(L)[ADNum,,]
  dataf = as.data.frame(which(array(T,c(L,L)),arr.ind = T))
  dataf$val = as.vector(data)
  names(dataf) = c("R1","OtherRater","prob")
  dataf$R1 = as.factor(dataf$R1)
  dataf$OtherRater = as.factor(dataf$OtherRater)

  temp = ggplot(dataf, aes(x=OtherRater,y=prob)) +
         geom_bar(stat="identity") +
         facet_wrap( ~ R1, ncol = 3) +
         xlab("Other Rater's Score")
  plot(temp)
}

# Creates the histograms for the Geometry problems study
# Uses data from the variable raters (from elsewhere)
geomHistograms = function(printToFile=F)
{
  path = "C://Users//dquarfoot//Desktop//GradSchool//Dissertation//AmerStat Paper//FinalVersion"
  fileNames = c("diff.pdf","nov.pdf","pd.pdf","auth.pdf")
  fileNames = paste(path,fileNames,sep="//")

  whichOnes = c(1,3,12,14)
  names = c("Difficulty","Novelty","Productive Dispositions","Authenticity")

  for (i in 1:4) {
    scores = NULL
    for (j in 1:8) {  # the expert panel
      scores = c(scores,raters[[j]]$pscores[,whichOnes[i]])
    }
    if (printToFile) pdf(file=fileNames[i],width = 6, height = 4)

    barplot(table(factor(scores,levels=1:9)),
            main = paste("Histogram of",names[i]),
            xlab = "Rating", ylab = "Count", space = 0)

    if (printToFile) dev.off()
  }
}
```