Supplementary materials for "A class of goodness-of-fit tests for circular distributions based on trigonometric moments"

S. Rao Jammalamadaka[1], M. Dolores Jiménez-Gamero[2] and Simos G. Meintanis[3,4]

This supplementary material contains the R code for the test proposed in the paper. The first function is for a fixed value (or vector of values) of λ, while the second function allows data-dependent choice of λ.

December 2019

The material contained herein is supplementary to the article named in the title and published in SORT-Statistics and Operations Research Transactions Volume 43(2).

[1] Department of Statistics and Applied Probability, University of California Santa Barbara, USA
[2] Department of Statistics and Operations Research, University of Sevilla, Spain
[3] Department of Economics, National and Kapodistrian University of Athens, Greece
[4] Unit for Business Mathematics and Informatics, North-West University, South Africa

```
#-----------------------------------------------
# TEST for a fixed vale/values of lambda
#-----------------------------------------------
# Entries:
#-----------------------------------------------
# x=vector of circular data taking values in (0, 2*\pi]
# r=number ok summands for the practical calculation of the test statistic
# lambda=parameter of the Poisson. It can be a vector
# B=number of bootstrap replications
#-----------------------------------------------
# Outputs:
#-----------------------------------------------
# hat.mu=MLE of mu
# hat.kappa=MLE of kappa
# p.value=bootstrap p-value
#-----------------------------------------------
# Warning: it requires the package CircStats
#-----------------------------------------------
library(CircStats)
TEST.CF=function(x,r,lambda,B)
{mi.CF=function(x,hat.mu,hat.kappa,lambda,r){
 x=x-hat.mu; n=length(x); n.l=length(lambda)
 M=r; S1=rep(0,n.l); S2=rep(0,n.l); S3=rep(0,n.l); aux=outer(x,x,"-")
 for (i in 1:n.l){
 lambda.0=lambda[i]
 eps.fun=function(theta){
 sal=cos(lambda.0*sin(theta))*exp(lambda.0*(cos(theta)-1))
 return(sal)} #end of eps.fun
 S1[i]=sum(eps.fun(aux))/n}
 aux1=besselI(hat.kappa,0:M); I.0=aux1[1]
 for (i in 1:n.l){
 lambda.0=lambda[i]
 aux2=dpois(0:M,lambda.0)
 S2[i]=sum((aux1^2)*aux2)
 E.fun=function(theta){aux3=cos(theta*(0:M)); sal=sum(aux1*aux2*aux3); return(sal)}
 S3[i]=sum(unlist(lapply(x,E.fun)))}
 sal=S1+n*S2/(I.0^2)-2*S3/I.0
 return(sal)}# end of mi.CF
 d.lambda=length(lambda)
 est.boot=matrix(0,ncol=d.lambda,nrow=B)
 estim=as.numeric(vm.ml(x))
 hat.mu=estim[1]; hat.kappa=estim[2]
 est.values=mi.CF(x,hat.mu,hat.kappa,lambda,r);
 for (i.boot in 1:B){ #print(c("Bootstrap iteration", i.boot), quote=FALSE)
   x.b=rvm(n,hat.mu,hat.kappa)
   estim.b=as.numeric(vm.ml(x.b));
   hat.mu.b=estim.b[1]; hat.kappa.b=estim.b[2]
   est.boot[i.boot,]=mi.CF(x.b,hat.mu.b,hat.kappa.b,lambda,r)}#end de i.boot
 sal=c()
 for (j in 1:d.lambda) {
  aux=mean(est.boot[,j] >= est.values[j]); sal=c(sal,aux)}
 nada=c()
 if (d.lambda>1) nada=rep(" ",(d.lambda-1))
 sss=rbind(c("hat.mu",hat.mu,nada),c("hat.kappa",hat.kappa,nada),c("lambda", lambda))
 sss=c(sss,c("bootstrap p.values", sal)); print(sss, quote=FALSE)}
```

```
#-----------------------------------------------
# TEST with data-dependent choice of lambda
#-----------------------------------------------
# Entries:
#-----------------------------------------------
# x=vector of circular data taking values in (0, 2*\pi]
# r=number ok summands for the practical calculation of the test statistic
# lambda=vector containing parameter of the Poisson.
# n.boot1=number of bootstrap replications for calculation of critical values
# n.boot2=number of bootstrap replications for calibration
# alpha=probability of type I error
#-----------------------------------------------
# Outputs:
#-----------------------------------------------
# hat.mu=MLE of mu
# hat.kappa=MLE of kappa
# decision= reject or not reject
#-----------------------------------------------
# Warning: it requires the package CircStats
#-----------------------------------------------
library(CircStats)
TEST.CF.DATA=function(x,r,lambda,B1,B2,alpha)
{mi.CF=function(x,hat.mu,hat.kappa,lambda,r){
 x=x-hat.mu; n=length(x); n.l=length(lambda)
 M=r; S1=rep(0,n.l); S2=rep(0,n.l); S3=rep(0,n.l); aux=outer(x,x,"-")
 for (i in 1:n.l){
 lambda.0=lambda[i]
 eps.fun=function(theta){
 sal=cos(lambda.0*sin(theta))*exp(lambda.0*(cos(theta)-1))
 return(sal)} #end of eps.fun
 S1[i]=sum(eps.fun(aux))/n}
 aux1=besselI(hat.kappa,0:M); I.0=aux1[1]
 for (i in 1:n.l){
 lambda.0=lambda[i]
 aux2=dpois(0:M,lambda.0)
 S2[i]=sum((aux1^2)*aux2)
 E.fun=function(theta){aux3=cos(theta*(0:M)); sal=sum(aux1*aux2*aux3); return(sal)}
 S3[i]=sum(unlist(lapply(x,E.fun)))}
 sal=S1+n*S2/(I.0^2)-2*S3/I.0
 return(sal)}# end of mi.CF
 estim=as.numeric(vm.ml(x));
 hat.mu=estim[1]; hat.kappa=estim[2]
 est.values=mi.CF(x,hat.mu,hat.kappa,lambda,r)
 d.lambda=length(lambda); pp=rep(0,n.boot1)
est.boot1=matrix(0,ncol=d.lambda,nrow=n.boot1); est.boot2=matrix(0,ncol=d.lambda,nrow=n.boot2)
 for (i.boot in 1:n.boot1){
   x.b=rvm(n,hat.mu,hat.kappa)
   estim.b=as.numeric(vm.ml(x.b));
   hat.mu.b=estim.b[1]; hat.kappa.b=estim.b[2]
   est.boot1[i.boot, ]=mi.CF(x.b,hat.mu.b,hat.kappa.b,lambda,r);
 }#end of for (i.boot in 1:n.boot1)
 for (j in 1:d.lambda) {est.boot1[,j]=sort(est.boot1[,j],decreasing=TRUE)}
 for (i.boot in 1:n.boot2){
   x.b=rvm(n,hat.mu,hat.kappa)
   estim.b=as.numeric(vm.ml(x.b));
   hat.mu.b=estim.b[1]; hat.kappa.b=estim.b[2]
   est.boot2[i.boot, ]=mi.CF(x.b,hat.mu.b,hat.kappa.b,lambda,r);
```

```
}#end of for (i.boot in 1:n.boot2)
for (r in 1:n.boot1)
 {aux1=t(matrix(rep(est.boot1[r,],n.boot2),ncol=n.boot2))
  aux2=est.boot2-aux1; aux3=apply(aux2,1,max); pp[r]=mean(aux3>0)
 } #end of for (r in 1:n.boot1)
pp.alpha=length(pp[pp<=alpha])
sal=sum(max(est.values-est.boot1[pp.alpha,])>0)
dec="Reject"; if (sal==0) dec="Not reject"
sss=rbind(c("hat.mu",hat.mu),c("hat.kappa",hat.kappa),c("Decision", dec)); print(sss, quote=FALSE)}
```