Shape Analysis Using Mixed Linear Models in R

Linear Models 2019—Diganta Bhattacharya

9 Approach as given

9.1 The Problem and its Components

To detect if a given test shape lies in the population of a category of star-shaped polygons up-to rotation and scaling using Fourier descriptor technique.

Figure 1: Mean Shape of population

Figure 2: Shape to test

9.2 The Basic Solution

We locate the Centroid of the polygon for each polygon in the given population or the random population generated as per true value. The distance $r(\theta)$ is calculated at regular intervals from $[0, 2\pi]$. This data is taken is taken as the characteristic of a particular population and fit into the Fourier descriptive model.Now the problem reduces to a Membership Test for this model.

- Only **OpenImageR** package is used for running this code.
- As we take at least 25 random images from the given shape it is safe to use normal approximation for the T-statistic obtained at the end.
- I am using *randpolygon* a function of my own to create random polygons from a mean shape instead of using *randt* in-built R function.

10 How to use the code

There are some things one should follow in order to get the correct response from the code.They are listed below.

- The user can either select the shapes from images saved in the working directory in PNG or JPEG format.
- Marking from a picture the user has to enter $\boxed{0}$ or otherwise $\boxed{1}$ to select from a blank plot.
- In case the user wants to select 0 , the user has to name the picture file first for the average population image and select the vertices of that polygon.
- Then he repeats the same for the picture containing the shape to test.
- For the code to work properly it is mandatory that the user selects the vertices in the correct order consecutively in anti-clockwise direction starting from any beginning vertex for both the images.

11 Other Details in the R code

```
Shape_Analysis<-function()
   {
      plot.locator<-function(num.points)
      {
       xs.list<-c(NULL)
```

```
 ys.list<-c(NULL)
 for( I in 1:num.points)
   {
    message("Select next point")
    location<-locator(1,type="p", par(pch=16,col="blue"))
    xs.list<-c(xs.list,location$'x')
    ys.list<-c(ys.list,location$'y')
   }
 locations<-list(x=xs.list, y =ys.list)
 return(locations)
}
randpolygon<-function(Vmatrix,N,J)
{
   V=nrow(Vmatrix)
   size=rnorm(N,mean=1,sd=0.2)
   ang=runif(N,0,2*pi)
   R=matrix(ncol=N,nrow=J)
   theta=1:J
   theta=(2*pi/J)*(theta-1)
   if(N==1)
   {
      size=1
      ang=0
   }
   for(i in 1:N)##checking for 1
   {
      e=rnorm(2*V,0,0.2)
      E=matrix(e,ncol=2)
      coord=(size[i])*(Vmatrix+E)
      X=coord[,1]
      Y=coord[,2]
      a=ang[i]
      Xcoord=cos(a)*X-sin(a)*Y
      Ycoord=sin(a)*X+cos(a)*Y
      xbar=mean(Xcoord)
      ybar=mean(Ycoord)
      d=((max(Ycoord-ybar)^2)+max((Xcoord-xbar)^2))^0.5+5
      P1=c(xbar,ybar)
      polyangle=1:V
      for(v in 1:V)
      {
         if(Xcoord[v]-xbar==0)
         {
            if(Ycoord[v]>ybar)
            {
               polyangle[v]=pi/2
            }else
            {
               polyangle[v]=3*pi/2
            }
         }else
         {
            m=(Ycoord[v]-ybar)/(Xcoord[v]-xbar)
            polyangle[v]=atan(m)
```

```
      if(atan(m)<0 & Xcoord[v]<xbar)
      {
        polyangle[v]=polyangle[v]+pi
      }
      if(atan(m)<0 & Xcoord[v]>xbar)
      {
        polyangle[v]=polyangle[v]+2*pi
      }
      if(atan(m)>0 & Xcoord[v]<xbar)
      {
        polyangle[v]=polyangle[v]+pi
      }
    }
  }
  for(j in 1:J)
  {
    v=1
    done=0
    t=theta[j]
    while(done==0 & v<=V)
    {
      if(v==V)
      {
        u=1
      }else
      {
        u=v+1
      }
      t1=polyangle[v]
      t2=polyangle[u]
      if(t1==t)
      {
        R[j,i]=((Xcoord[v]-xbar)^2)+((Ycoord[v]-ybar)^2)
        R[j,i]=R[j,i]^0.5
        done=1
      }else if(t2==t)
      {
        R[j,i]=((Xcoord[u]-xbar)^2)+((Ycoord[u]-ybar)^2)
        R[j,i]=R[j,i]^0.5
        done=1
      }else if(t1<t & t2>t)
      {
        if((Xcoord[u]-Xcoord[v])==0)
        {
          a=Xcoord[u]
          b=tan(t)*a+(ybar-tan(t)*xbar)
          r=c(a,b)
          R[j,i]=sum((r-P1)^2)
          R[j,i]=R[j,i]^0.5
          done=1
        }else if(cos(t)==0)
        {
          a=xbar
          m=(Ycoord[u]=Ycoord[v])/(Xcoord[u]-Xcoord[v])
```

```
            c=Ycoord[u]-m*Xcoord[u]
            b=m*a+c
            r=c(a,b)
            R[j,i]=sum((r-P1)^2)
            R[j,i]=R[j,i]^0.5
            done=1
         }else
         {
            m1=tan(t)
            m2=(Ycoord[u]=Ycoord[v])/(Xcoord[u]-Xcoord[v])
            c1=ybar-m1*xbar
            c2=Ycoord[u]-m2*Xcoord[u]
            a=(c1-c2)/(m2-m1)
            b=m1*a+c1
            r=c(a,b)
            R[j,i]=sum((r-P1)^2)
            R[j,i]=R[j,i]^0.5
            done=1
         }
      }
      v=v+1
   }
   if(v==V+1 & done==0)
   {
      v=which(polyangle==max(polyangle))
      u=which(polyangle==min(polyangle))
      if((Xcoord[u]-Xcoord[v])==0)
      {
         a=Xcoord[u]
         b=tan(t)*a+(ybar-tan(t)*xbar)
         r=c(a,b)
         R[j,i]=sum((r-P1)^2)
         R[j,i]=R[j,i]^0.5
      }else if(cos(t)==0)
      {
         a=xbar
         m=(Ycoord[u]=Ycoord[v])/(Xcoord[u]-Xcoord[v])
         c=Ycoord[u]-m*Xcoord[u]
         b=m*a+c
         r=c(a,b)
         R[j,i]=sum((r-P1)^2)
         R[j,i]=R[j,i]^0.5
      }else
      {
         m1=tan(t)
         m2=(Ycoord[u]=Ycoord[v])/(Xcoord[u]-Xcoord[v])
         c1=ybar-m1*xbar
         c2=Ycoord[u]-m2*Xcoord[u]
         a=(c1-c2)/(m2-m1)
         b=m1*a+c1
         r=c(a,b)
         R[j,i]=sum((r-P1)^2)
         R[j,i]=R[j,i]^0.5
      }
```

```
          done=1
        }
      }

    }
    return(R)
}
######################################
#######################################
Membershiptest<-function(R)
{
    N=ncol(R) ##Number of shapes considered in training set
    J=nrow(R) ##Number of values taken for each shape

    ####################################################################
    ##############Resizing assuming log of distances are stored#########
    ####################################################################
    for(j in 1:N)
    {
      R[,j]=R[,j]-mean(R[,j]-mean(R))
    }

    ############################################################
    ####################Adjusting Rotation######################
    ############################################################
    for(j in 2:N)
    {
      min=sum((R[,1]-R[,j])^2)
      minvector=R[,j]
      for(i in 2:J)
      {
        new=R[i:J,j]
        new=c(new,R[1:(i-1),j])
        temp=sum((R[,1]-new)^2)
        if(temp<min)
        {
          min=temp
          minvector=new
        }
      }
      R[,j]=minvector
    }

    ###########################################################
    ###############Forming the Information Matrix##############
    ###########################################################
    a=2*pi/J
    theta=1:J
    theta=a*(theta-1)
    K=3
    Z=NULL
    for(i in 1:N)
    {
      M=matrix(nrow=J,ncol=2*K+1)
```

```
    M[,1]=rep(1,times=J)
    for(k in 1:K)
    {
      M[,2*k]=sin(theta)
      M[,2*k+1]=cos(theta)
    }
    Z=rbind(Z,M)
  }
  y=NULL
  for(i in 1:N)
  {
    y=c(y,R[,i])
  }
  g=NULL
  for(i in 1:N)
  {
    g=c(g,rep(i,times=J))
  }
  g=as.factor(g)
  data=cbind(y,Z,g)
  naming=c("obs","(intercept)","a1","b1","a2","b2","a3","b3","shapeno")
  colnames(data)=naming
  data=as.data.frame(data)
  result=list()
  result[[1]]=Z
  result[[2]]=R
  result[[3]]=data
  return(result)
}
################################################
###############################################
###################################################
V=readline(prompt="Enter the number of vertices.")
class(V)="integer"
plot(0,0,type="n")
print("Enter the points indicating the population of polygons.")
coordinates1=plot.locator(V)
print("Enter the points indicating the test polygon.")
coordinates2=plot.locator(V)
truevalue=matrix(nrow=V,ncol=2)
testvalue=matrix(nrow=V,ncol=2)
truevalue[,1]=coordinates1$x
truevalue[,2]=coordinates1$y
testvalue[,1]=coordinates2$x
testvalue[,2]=coordinates2$y
A=truevalue-mean(truevalue)
test=testvalue-mean(testvalue)
J=readline(prompt="Enter the number of distances to be taken per shape.")
J=as.integer(J)
if(J<20)
{
  print("We will take at least 25 for precision.")
  J=25
}
```

```
N=readline(prompt="Enter the number of shapes to be randomly generated for the
    population.")
N=as.integer(N)
if(N<10)
{
   print("We will take at least 10 for precision.")
   N=10
}
R=randpolygon(A,N,J)
R=log(R)
f=Membershiptest(R)
Rnew=f[[2]]
r=1:J
for(i in 1:J)
{
   r[i]=mean(Rnew[i,])
}
X=matrix(nrow=J,ncol=7)
X[,1]=rep(0.7071,times=J)
theta=(2*pi/J)*((1:J)-1)
X[,2]=sin(theta)
X[,3]=cos(theta)
X[,4]=sin(2*theta)
X[,5]=cos(2*theta)
X[,6]=sin(3*theta)
X[,7]=cos(3*theta)
X=((2/J)^0.5)*X
b=t(X)%*%r
ss=0
for(i in 1:N)
{
   P=t(X)%*%R[,i]
   ss=ss+sum(R[,i]^2)-sum(P^2)
}
ms=ss/(N*(J-7))
sigma=ms^0.5
rhat=X%*%b
test_r=randpolygon(test,1,J)
test_r=test_r-mean(test_r)+mean(Rnew)
min=sum((test_r-Rnew[,1])^2)
minvector=test_r
for(i in 2:J)
{
   new=test_r[i:J]
   new=c(new,test_r[1:i-1])
   temp=sum((Rnew[,1]-new)^2)
   if(temp<min)
   {
      min=temp
      minvector=new
   }
}
test_r=minvector
t=mean((test_r-rhat)^2)/ms
```

```
    t=t^0.5
    if(t<1.96)
    {
      print("The given shape appears to be in the population.")
    }else
    {
      print("The given shape is not in the population.")
    }
    Ydir=NULL
    for(i in 1:N)
    {
      Ydir=c(Ydir,Rnew[,i])
    }
    Xdir=rep(1:J,times=N)
    plot(Xdir,Ydir)
    lines(1:J,test_r,col="red")
    lines(1:J,rhat,col="green")
    print("The green line shows the fitted value for r")
    print("The red line shows the value for r for the test shape.")
  }
```

11.1 Generating the Data

For the vertices marked by the user, we feed it into the function *randpolygon* to randomly generate polygons of the same population and return a matrix containing the values of $r(\theta)$ for further analysis. The number of iterations to be taken and the number of angles to be taken for each polygon has to be specified for using *randpolygon*.

12 Results which were accepted

Here are some of the results based on some randomly chosen n and some randomly chosen polygons to check the method

- For triangles,

Figure 3: Triangle Mean Shape

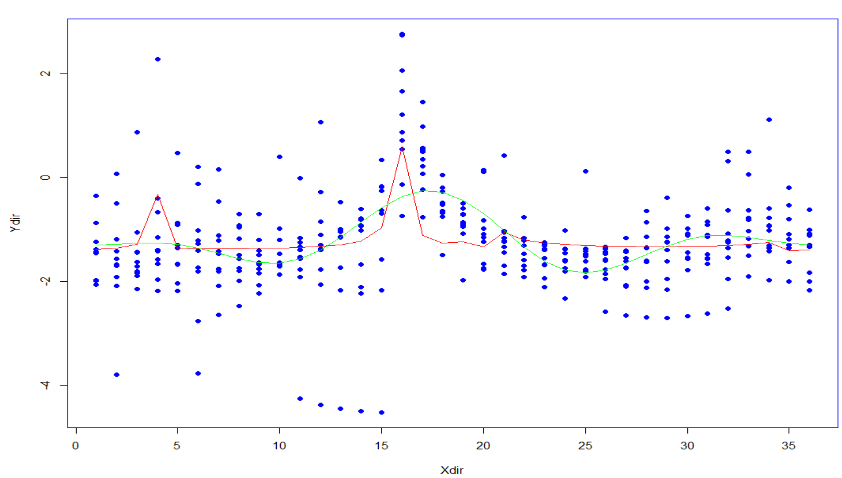

Figure 4: Result after test

- For quadrilaterals,

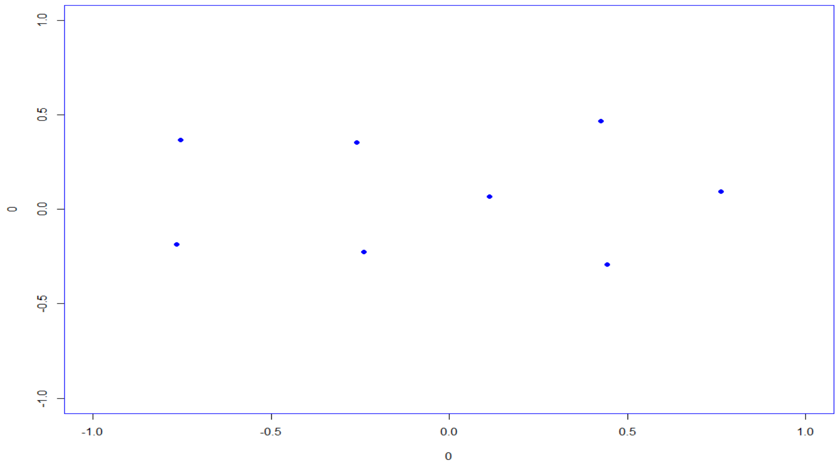

Figure 5: Quadrilateral Mean and Test Shape

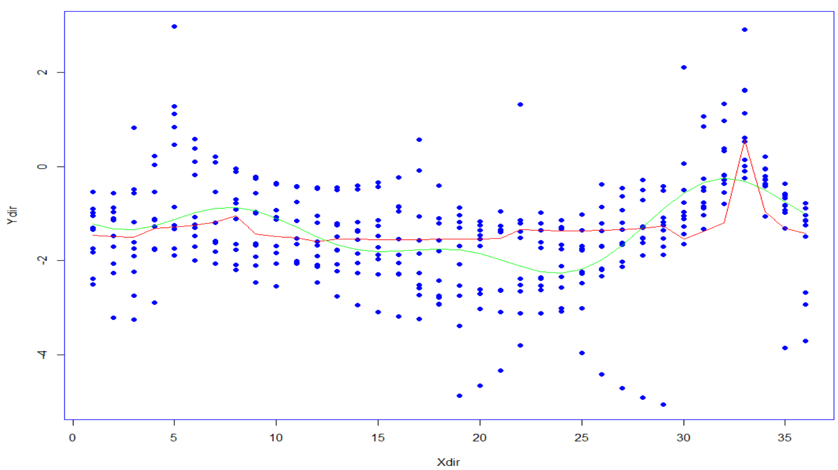

Figure 6: Result after test

- For pentagons,

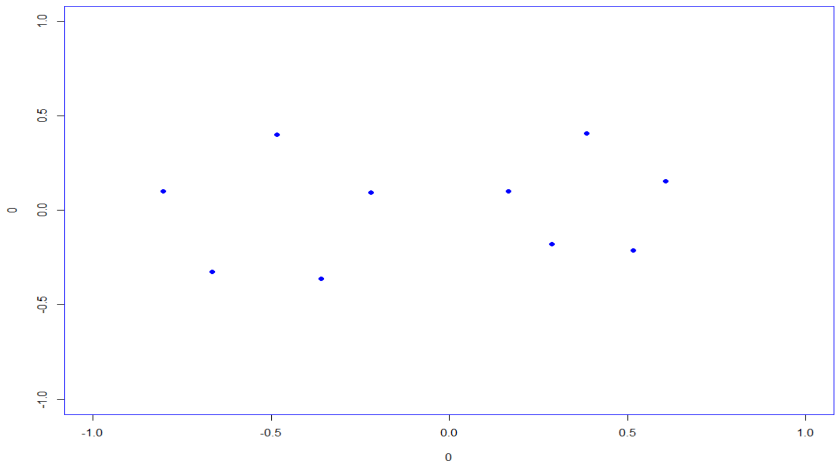

Figure 7: Pentagon Mean and Test Shape

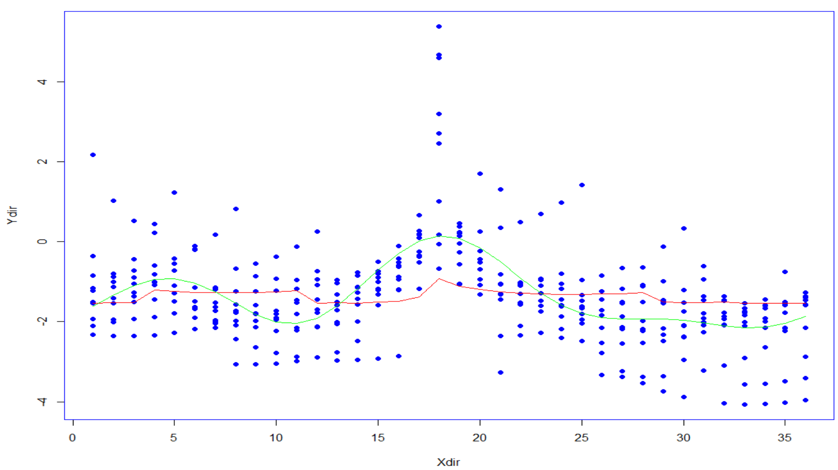

Figure 8: Result after test

13 Conclusion

- If the shapes are more ore less regular and we calculate $r(\theta)$ at small intervals then the tests are consistent.
- If we take number of angles greater than 50 then there are many out-liers in the data generated by *randpolygon* and fitted curve will match with the true curve in most of the places but some out-liers will be far apart from the curve. So we take the number of angles in between $25-36$.
- The test becomes less and less conservative as we increase the number of angles from 36.
- We have observed optimal result for number of angle between $30-36$ and 10 random shapes.