SISMID 2024 Spatial Statistics Waller Point Process 1: Monte Carlo Test of CSR

Lance A. Waller

7/14/24

- **What we have**
  - Event locations for a strand of 327 myrtle trees in a rectangular plot 170.5 $\times$ 213.0 meters.
  - 221 healthy trees.
  - 106 diseased trees.
  - Research question: Is the spatial pattern of diseased trees the same as the spatial pattern of health trees?

- **Reading in the data, basic R commands**

```
# Set my working directory (Lance's here for example)
#setwd("~/OneDrive - Emory University/meetings/SISMID.2021/SISMID.2021.Waller.Rcode")
myrtles.healthy = scan(here("data", "myrtles.healthy.d"),list(x=0,y=0))

####################################
# Let's see what we have.  Typing the
# name "myrtles.healthy" and hitting return
# prints out the values.
####################################

# Commented out so it doesn't list everything in the handout
#myrtles.healthy

####################################
# The "names" command just give the
# names of the variables inside the data frame.
####################################

names(myrtles.healthy)
```

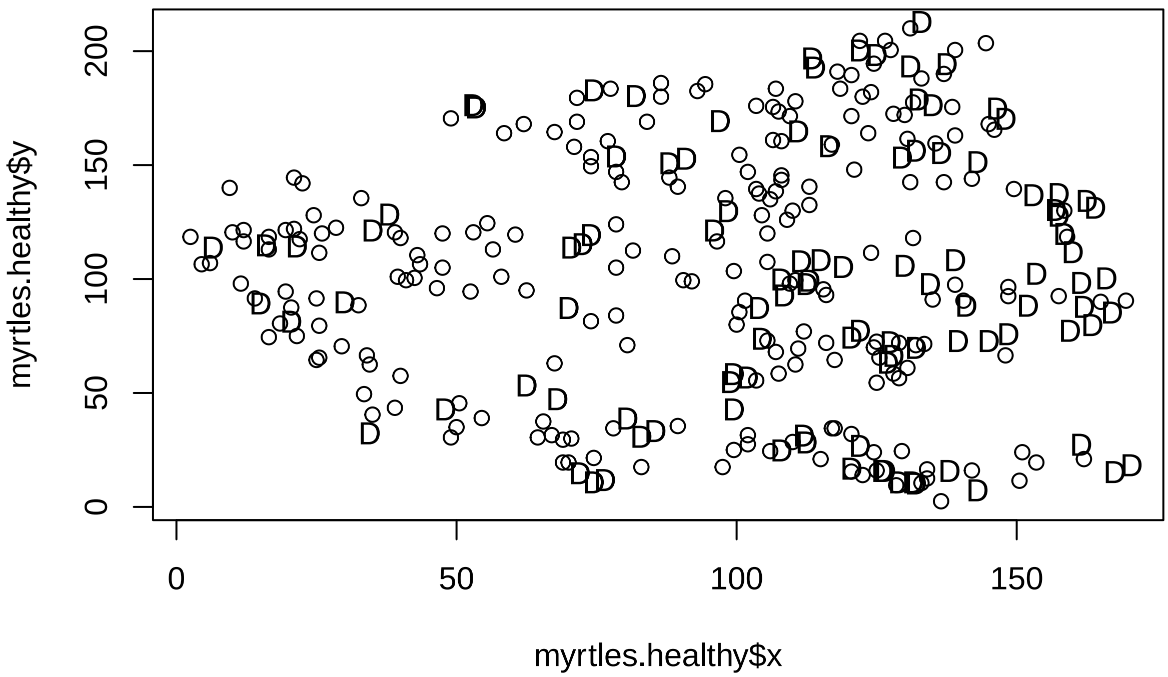
```
## [1] "x" "y"
```

```r
####################################
# To access the value within a data frame
# type the name of the frame, a dollar sign,
# then the name of the variable.
######################################

#Commented out so it doesn't list every value in the handout
#myrtles.healthy$x

#####################################
# To find out how many observations are
# in myrtles.healty$x, use the "length"
# command.
#####################################

length(myrtles.healthy$x)
```

```
## [1] 221
```

- **Plotting the data**
  - Read in data on diseased trees.
  - Plot patterns
  - Take care to make *square* plots, covering the *same* area. ***

```r
myrtles.all = scan(here("data", "myrtles.d"),list(x=0,y=0))
myrtles.diseased = scan(here("data", "myrtles.diseased.d"),list(x=0,y=0))

######################################
# Let's plot the data
######################################

plot(myrtles.healthy$x, myrtles.healthy$y)

######################################
# The "points" command adds points to a plot,
# and the "pch" option changes the "plot character".
# Let's add the diseased myrtle locations and plot
# them as "D"s.
######################################

points(myrtles.diseased$x,myrtles.diseased$y,pch="D")
```




```
######################################
# Notice that the plot is sort of square, but
# that the range of values for x is different
# from that for y.  Let's set the limits
# so that they are the same.  First, we find
# min and max of the x and y coordinates for
# ALL myrtle locations (healthy and diseased).
######################################

min(myrtles.all$x)
```

```
## [1] 2.5
```

```
max(myrtles.all$x)
```

```
## [1] 170.5
```

```
min(myrtles.all$y)
```

```
## [1] 2.5
```

```
max(myrtles.all$y)
```

```
## [1] 213
```

```r
######################################
# We can also use the "range" command
# to do this.
######################################

range(myrtles.all$x)
```

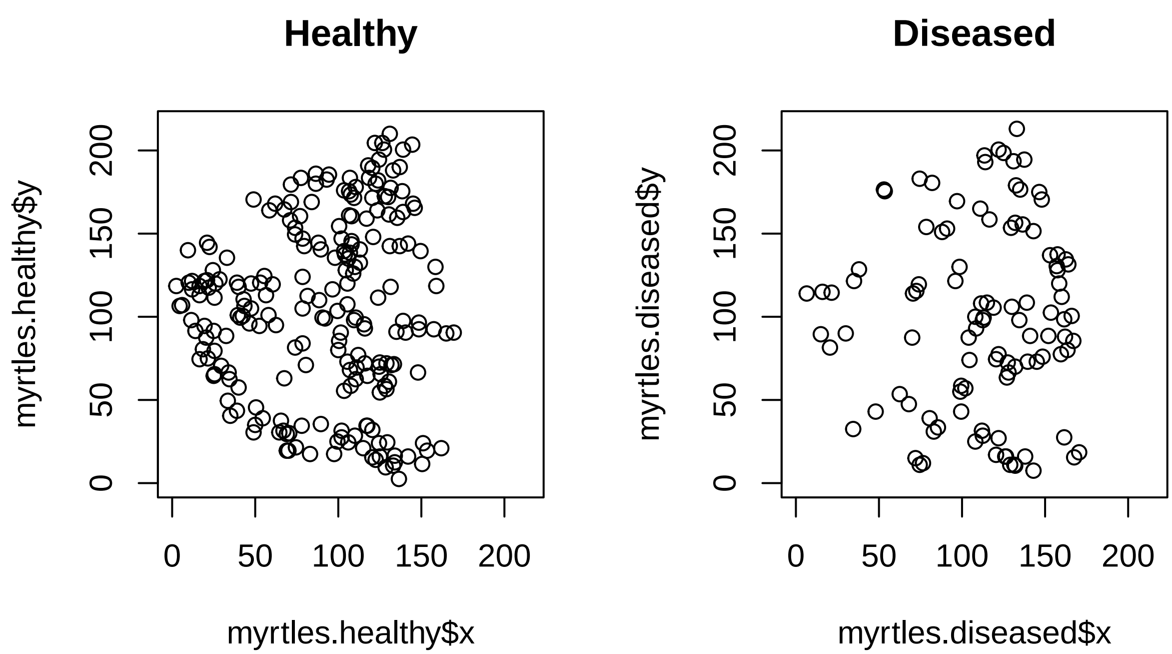
```
## [1]   2.5 170.5
```

```r
range(myrtles.all$y)
```

```
## [1]   2.5 213.0
```

```r
######################################
# Looks like if we set the plot boundaries
# for (0,215) for x and y, we'll catch all
# of the points.  We use the "xlim" and
# "ylim" parameters in the plot command.
# NOTE: we can continue a command onto the next
# line if we end with a comma and don't include
# a closing paranthesis until we are ready.
# ALSO NOTE: "c(0,215)" concatenates the values
# 0 and 215 into a vector.
######################################

plot(myrtles.healthy$x,myrtles.healthy$y,xlim=c(0,215),
      ylim=c(0,215))

####################################
# Finally, to make sure R draws the plotting area
# as a square, we introduce the "par" command.
# "par" sets plotting parameters and is a very,
# very, very, very, very, very, very important
# command with lots of uses.  You have to set
# "par" before plotting, but the settings stay until
# the next "par" command resets them.
# "pty" = "plot type" and "pty=s" means "set plot type
# to square".
#####################################

par(pty="s")
plot(myrtles.healthy$x,myrtles.healthy$y,xlim=c(0,215),
      ylim=c(0,215))

#####################################
# We can also use "par" to put put multiple plots
# in the same window.
# "mfrow" means "multiple figures by row".
# "mfrow=c(1,2)"  means "multiple figures, one row
# containing two figures".  Let's try it.
#####################################
```

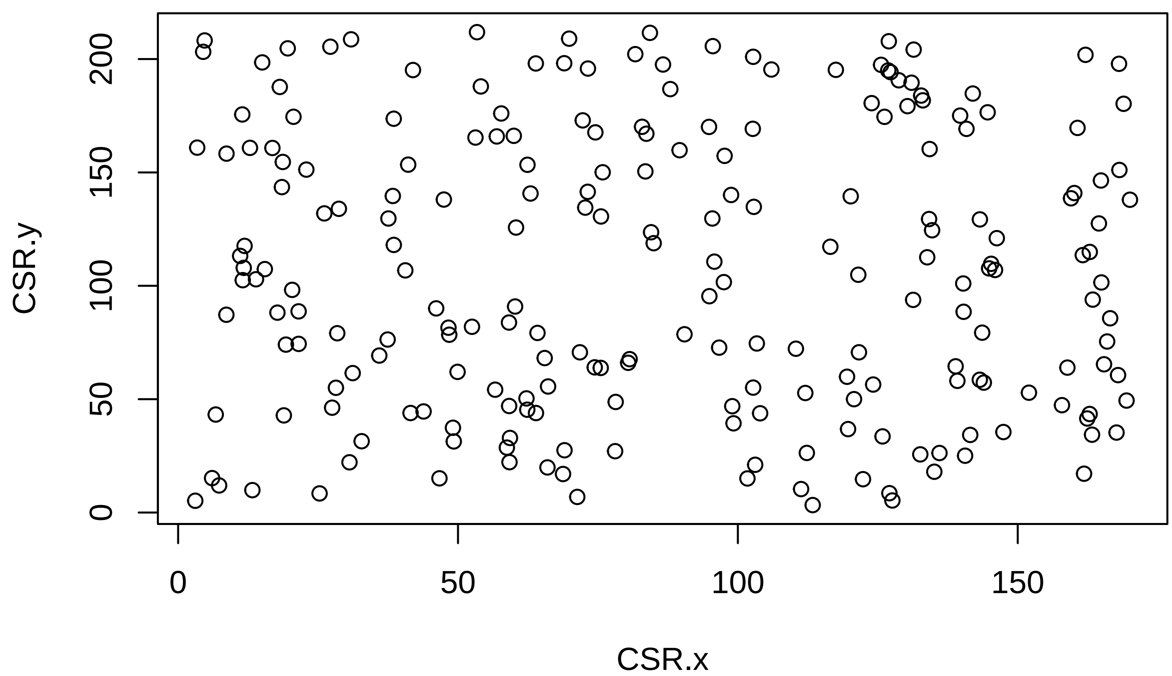
```r
par(pty="s",mfrow=c(1,2))

plot(myrtles.healthy$x,myrtles.healthy$y,xlim=c(0,215),
      ylim=c(0,215))
title("Healthy")

plot(myrtles.diseased$x,myrtles.diseased$y,xlim=c(0,215),
      ylim=c(0,215))
title("Diseased")
```



- **Let's test for CSR**
- Consider test statistic by Pielou (1959)
- Test statistic $P = \pi\lambda \sum X_i^2/n$
- X_i = distance from event i to its nearest neighbor
- Pielou (1959) suggests $P \overset{a}{\sim} N(1, 1/n)$.\ [link] (http://methodsblog.com/2017/03/10/ec-pielou/)
- Let's try a Monte Carlo test.

```r
##########################################
# To generate realizations from CSR in the range
# of values of the data we use "runif"
```

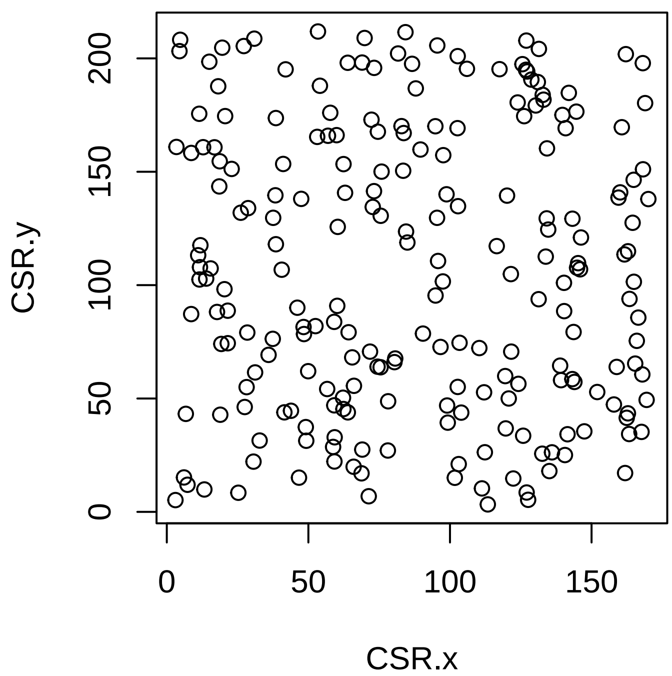
```r
# command that generates uniformly distributed
# random numbers.
# NOTE: We don't want to generate on the interval
# (0,215) since we want to limit the range of values
# to the range of the data.  We extended the region to
# get a square plot, but we want to limit simulations
# to the area with data.
##########################################
# Let's set the number of events to simulate to
# match the observed number of events.

num.events = length(myrtles.healthy$x)

CSR.x <- runif(num.events,min(myrtles.all$x),max(myrtles.all$x))
CSR.y <- runif(num.events,min(myrtles.all$y),max(myrtles.all$y))

plot(CSR.x,CSR.y)
```



***

- **Need to reset plot to be in a square**

```r
##########################################
# Oops need to reset "mfrow" and a square plot using "par"
##########################################

par(mfrow=c(1,1),pty="s")
plot(CSR.x,CSR.y)
```



For a Monte Carlo test, we will want to make a loop of CSR simulations and calculate the test statistic for each simulated realization.

Let's calculate a clustering statistic due to Pielou (1959). This statistic requires us to find the distance from each event to its nearest neighbor. (This also gives us a chance to try out some other R functions.)

First, for each event, calculate the distance to all other events.

We access individual x or y values by brackets with the index, i.e., x[1] is the first element of x.

We can find the distance between (x[1],y[1]) and all other observations by…

```r
dist1 = sqrt( (myrtles.all$x[1] - myrtles.all$x)^2 + (myrtles.all$y[1] - myrtles.all$y)^2 )
```

```r
##########################################
# This is a little tricky since
# "myrtle.all$x[1] - myrtle.all$x" is a number (x[1])
# minus a vector (x).  In R this results in subtracting the number from
# all elements of the vector.
##########################################
```

Now we want to find the minimum element of "dist1" that is NOT 0. We can use a nifty feature of R namely we can put logical expressions inside brackets and get only the elements where that expression is true. For instance,

```r
# Commented out so it doesn't list output
# dist1[dist1!=0]
```

gives the elements of dist1 that are not equal to zero. So

```r
 min(dist1[dist1!=0])
```

```
## [1] 4.472136
```

gives the nearest neighbor distance! Now we just need it for all values. Let's use a loop to get this.

```r
###################
# Set a vector of zeros with length equal to the number of locations in myrtles.all (length(myrtles.all

mindist.all <- 0*(1:length(myrtles.all$x))

for (i in 1:length(myrtles.all$x)) {
  dist = sqrt( (myrtles.all$x[i] - myrtles.all$x)^2 + (myrtles.all$y[i] - myrtles.all$y)^2 )
  mindist.all[i] = min(dist[dist!=0])
}
```

Now we need to calculate Pielou's statistic. ***

```r
n.all = length(myrtles.all$x)
area.all = (max(myrtles.all$x) - min(myrtles.all$x)) * (max(myrtles.all$y)-min(myrtles.all$y))
lambda.all = n.all/area.all
pielou.all = pi*lambda.all*sum(mindist.all^2)/n.all

############################################
# Print out the value using 'paste' to put text before it. (sep="" to have no
# separater between the text and the value)

print(paste("Pielou's statistic value = ",pielou.all,sep=""))
```

```
## [1] "Pielou's statistic value = 0.65680109594231"
```

Let's round the statistic to make the print output a little cleaner.

```
print(paste("Pielou's statistic value = ",round(pielou.all,4),sep=""))
```

```
## [1] "Pielou's statistic value = 0.6568"
```

Now to set up the Monte Carlo test*

To get a Monte Carlo p-value we need todo these same calculations to data generated under CSR. First define the number of simulations.

```
num.sim = 99
```

then define a vector to hold the values of Pielou's statistic for each simulated data set.

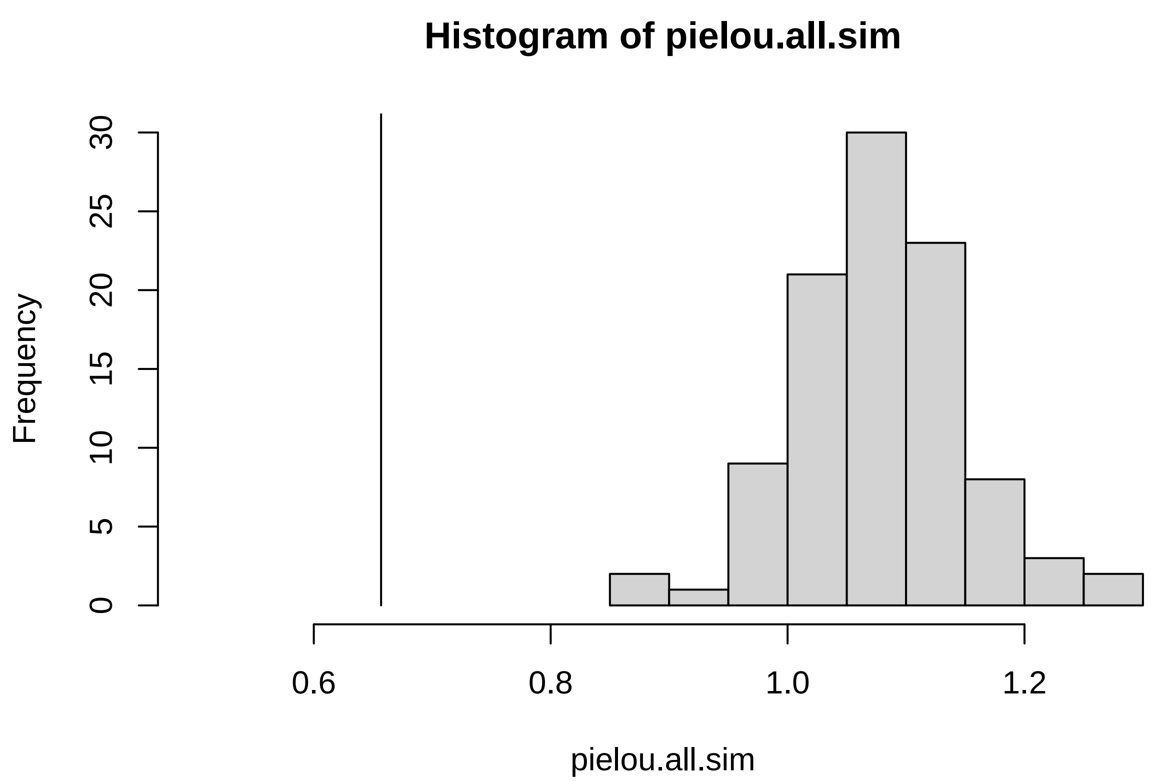
```
pielou.all.sim = 0*(1:num.sim)
```

Now set up the simulation loop

```
for (sim in 1:num.sim) {

  # define CSR data
  CSR.x = runif(num.events,min(myrtles.all$x),max(myrtles.all$x))
  CSR.y = runif(num.events,min(myrtles.all$y),max(myrtles.all$y))

  #define vector of min NN distances
  mindist.sim = 0*(1:length(myrtles.all$x))

  # find min distances (a loop within the simulation loop)

  for (i in 1:length(CSR.x)) {
     dist = sqrt( (CSR.x[i] - CSR.x)^2 + (CSR.y[i] - CSR.y)^2 )
     mindist.sim[i] = min(dist[dist!=0])
  }

  # calculate pielou.all.sim[sim] (Pielou's statistic for the "sim-th" CSR data set).
  pielou.all.sim[sim] = pi*lambda.all*sum(mindist.sim^2)/n.all


}
```

Make a histogram of the CSR values

```
par(pty="m")  # makes plot type "maximum" (rectangular in window).

hist(pielou.all.sim,xlim=c(0.5,max(pielou.all.sim)))

# add a vertical line showing Pielou's statistic from the observed data.
# ("segments(x1,y1,x2,y2)" draws a line segments between (x1,y1) and (x2,y2).

segments(pielou.all,0,pielou.all,100)
```



***

Now we can calculate Monte Carlo p-value (the number of statistics from simulated data that exceed the statistic from the observed data divided by the number of simulations + 1.

```
p.val = length(pielou.all.sim[pielou.all.sim>pielou.all])/(num.sim+1)

print(paste("Peilou's statistic: ",round(pielou.all,4)," p-val = ",round(p.val,4),sep=""))
```

```
## [1] "Peilou's statistic: 0.6568 p-val = 0.99"
```

** Now to do this for the healthy and diseased subsets

```
mindist.healthy = 0*(1:length(myrtles.healthy$x))

for (i in 1:length(myrtles.healthy$x)) {
  dist = sqrt( (myrtles.healthy$x[i] - myrtles.healthy$x)^2 + (myrtles.healthy$y[i] - myrtles.healthy$y
  mindist.healthy[i] = min(dist[dist!=0])
}

n.healthy = length(myrtles.healthy$x)
# NOTE: We still use area.all to cover the entire study area.
lambda.healthy = n.healthy/area.all
pielou.healthy = pi*lambda.healthy*sum(mindist.healthy^2)/n.healthy
print(paste("Peilou's statistic, healthy myrtles:",pielou.healthy))
```

```
## [1] "Peilou's statistic, healthy myrtles: 0.657911544652011"
```

Now for the diseased trees

```
mindist.diseased = 0*(1:length(myrtles.diseased$x))

for (i in 1:length(myrtles.diseased$x)) {
  dist = sqrt( (myrtles.diseased$x[i] - myrtles.diseased$x)^2 + (myrtles.diseased$y[i] - myrtles.diseas
  mindist.diseased[i] = min(dist[dist!=0])
}


n.diseased = length(myrtles.diseased$x)
# NOTE: We still use area.all to cover the entire study area.
lambda.diseased = n.diseased/area.all
pielou.diseased = pi*lambda.diseased*sum(mindist.diseased^2)/n.diseased
print(paste("Peilou's statistic, diseased myrtles:",pielou.diseased))
```

```
## [1] "Peilou's statistic, diseased myrtles: 0.507452851359244"
```

What we have: Separate tests for healthy trees and for diseased trees.

What we don't have: A comparison between healthy and diseased trees.