Scientific Programming Assignment 3

303034908

January 2019

1 Simulating spatial point patterns

1.1 Part I: Build the dmin model [5 marks]

A function to simulate two dimensional point patterns was written. The points are added sequentially according to a "minimal distance" rule. If a new point comes within a minimal distance to any other existing point, it is discarded and its coordinates are redrawn from a uniform distribution between the $xlo - xhi$ and $ylo - yhi$ coordinates. The circular exclusion zone surrounding each point is taken from a normal distribution with mean m and standard deviation s, truncated at 0 as a lower bound.

Figure 1 shows the result with the following parameters: `res <- dmin2d(n=200, m=30, s=5, xlo=200, xhi=1000, ylo=100, yhi=900)`. Given the stochasticity of the simulator, a random seed was used to recreate the figure. The circular exclusion zone around each point is also shown. The function automatically plots the pattern unless stated by `plot=FALSE`.

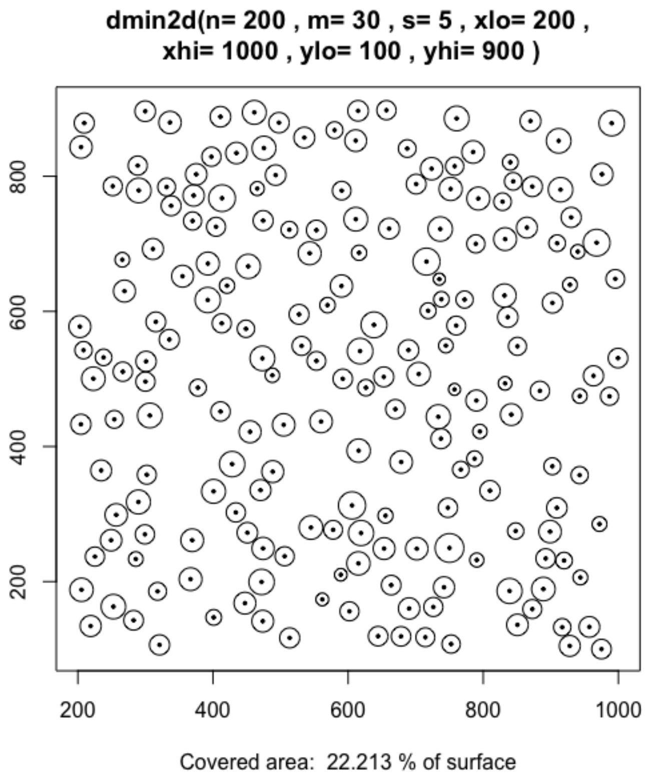

Figure 1: A point pattern with 200 points created using the sequential *dmin2d* model. Each point has a circular exclusion zone also shown around it.

1.2 Part II: Evaluate regularity index [10 marks]

It is visually very difficult to quantify the regularity of a spatial point pattern. The most popular method, the regularity index (RI), is calculated by taking the mean of the distances of each point to its nearest-neighbour, and dividing it by the standard deviation of this distribution (**WASSLEf1978-dw**).

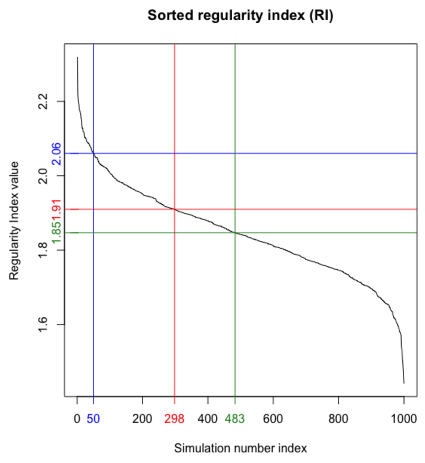

Figure 2: Regularity index of 1000 simulations: in red, the theoretical expectation for the RI for a set of points randomly positioned with no minimal distance constraint (=1.91). In blue, the 50th largest RI value (=2.05). In green, the mean (= 1.839).

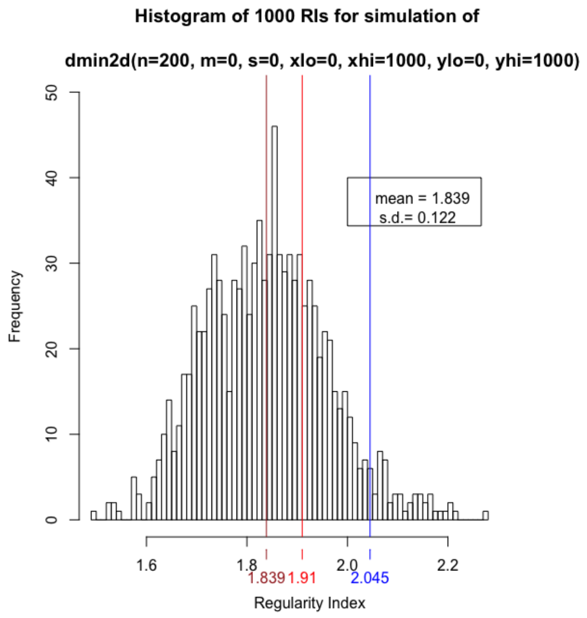

Figure 3: Histogram of 100 simulations of pattern. Theoretical expectation in red. Fiftieth value RI in blue. Mean in green.

The theoretical expectation for the RI for a set of points randomly positioned with no minimal distance constraint is around 1.91 (**Eglen2011-dk**).

Using the `dmin2d` model, 1000 simulations of 200 randomly positioned points with no minimal distance constraint were generated. The parameters were `dmin2d(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000)`.

The RI of each pattern was measured. This distribution is shown in figure 2 as the RI values sorted from largest to smallest and as a histogram in figure 3.

In this first example, the mean RI of 1.839 was slightly below the theoretical expectation of 1.91.

From figure 2 and figure 3, a few things are notable. Firstly, the mean of the 1000 RIs is around 0.07 below the expected RI for random points. A lower RI index suggests more irregularity, whilst a higher value suggests a more regular pattern. Whilst this is within 1 standard deviation of the distribution, it may suggest a systematic difference in the production of "random points" using the *dmin2d* model, in such a way that they are less regular than a random pattern.

Secondly,taking the 50th largest value is effectively calculating the 95th% point of the distribution of RIs. A use for this metric would be to test if a given distribution is actually random or not. If its RI is above 2.05, there is a 19 in 20 chance that it is not random, and a 1 in 20 chance that it still is. This is also shown by the fact that the mean of a normal distribution plus two of its standard deviations should include 97.5 % of all points. A Shapiro-Wilk normality test of the data above shows a p value of 0.2152, thus showing that the data is likely normal.

Next the effect of variation of the number of points n and variation of the geometry of the sample area on the regularity index was investigated. The number of points was varied from 200 to 50, 100, 400 and 800. The results are shown in figures 4, 5, 6, and 7.

From these figures we can see that neither the mean, nor the standard deviation nor the 50th largest vary more than a few decimal places as the number of points changes from 50 to 800.

A similar set of histograms of RI scores for different shapes are shown in figures 8, 9, 10, and 11. As these histograms show, the variation in mean, standard deviation and 50th value are slightly more marked when the differences occur in in the shape of the box instead of the number of points. A further way of showing this is in figures 12, and 13, as it shows the full distributions at the two most extreme ends of each change as well as the original plot.

1.3 Part III: Fit the model to some data [10 marks]

Real data, consisting of 238 points sampled in a region of size $400 \times 400 \mu m^2$ was provided. The task was given to find which possible values of m, s in the d_{min} model would generate patterns similar to the real data set.

To compare the model output for a given set of parameters with the provided data, the RI of the provided pattern was compared to the average RI of 99 simulations of the d_{min} model. If the RI of the real pattern is x_1

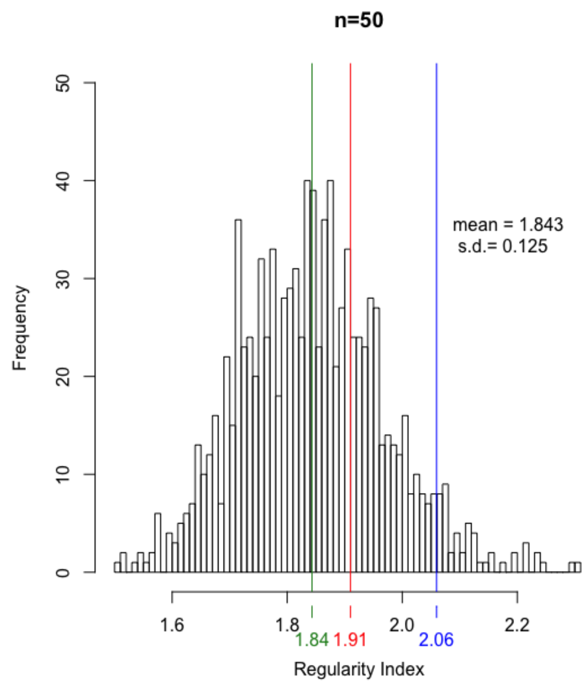

Figure 4:

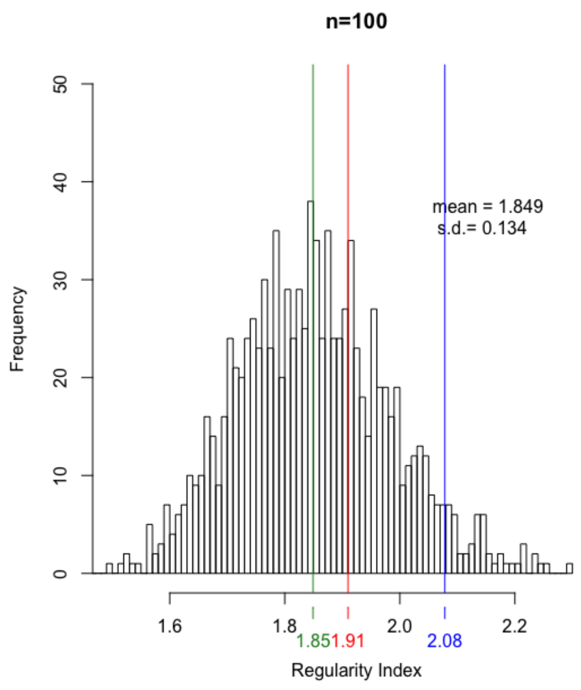

Figure 5:

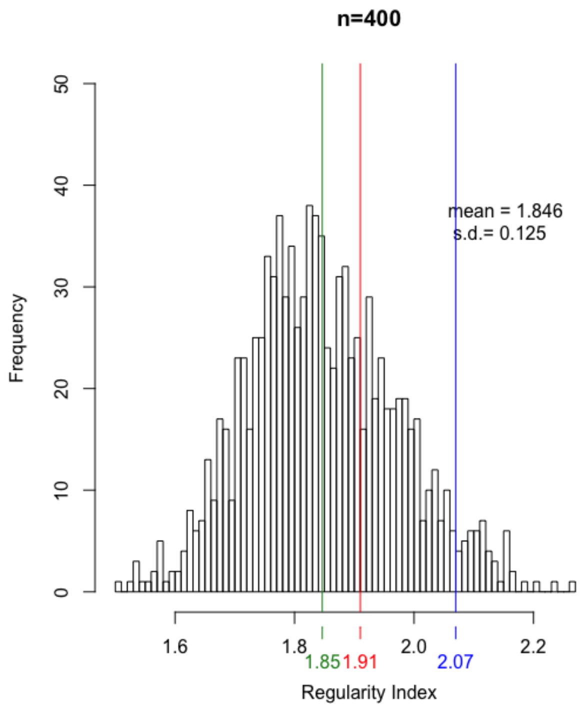

Figure 6:

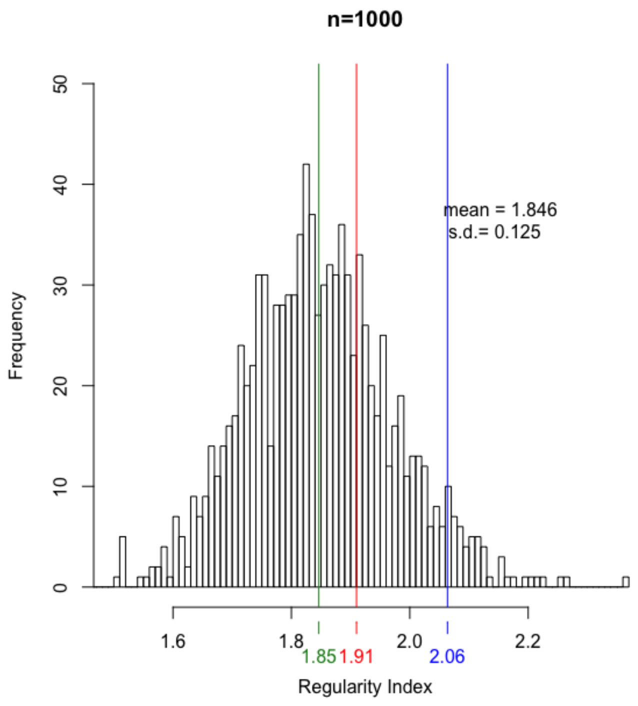

Figure 7:

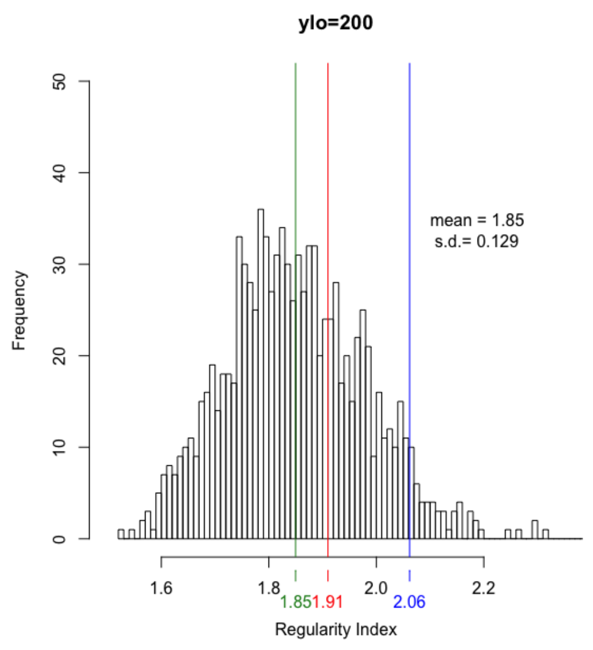

Figure 8:

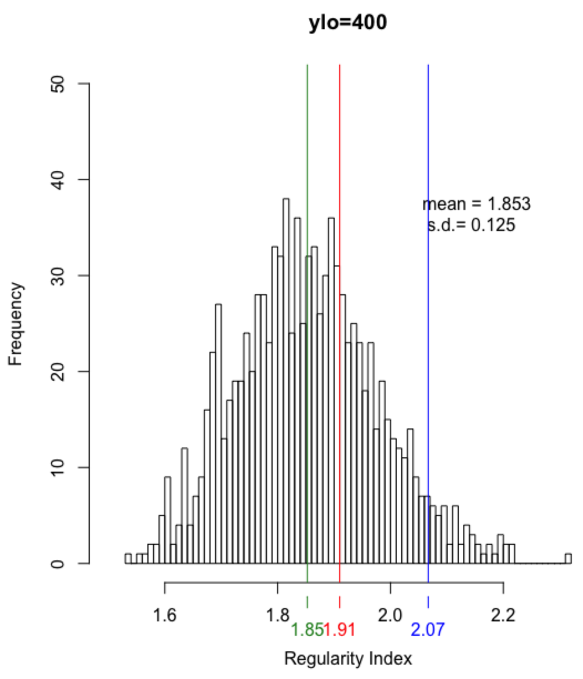

Figure 9:

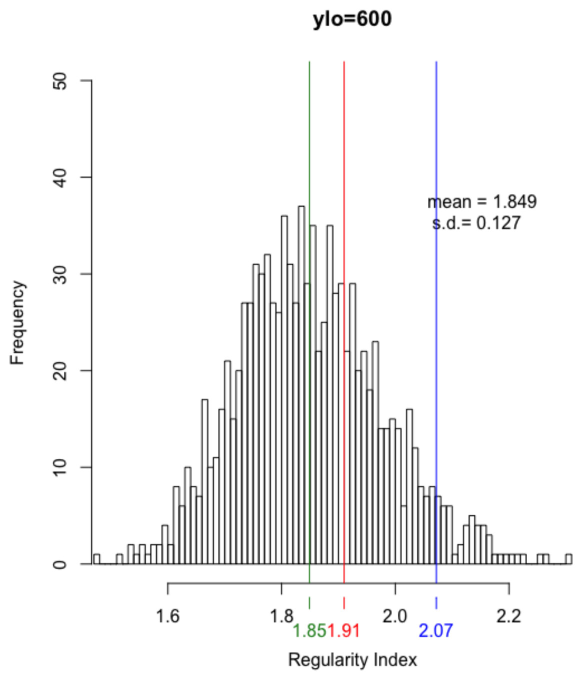

Figure 10:

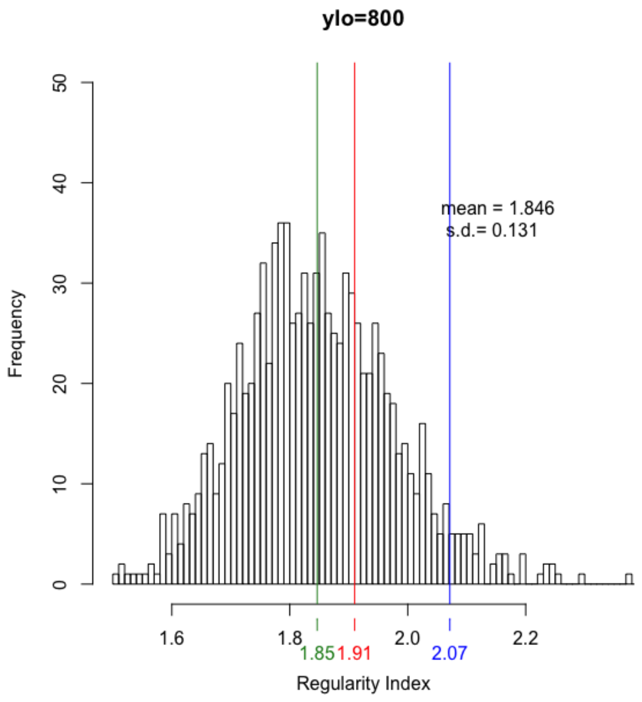

Figure 11:

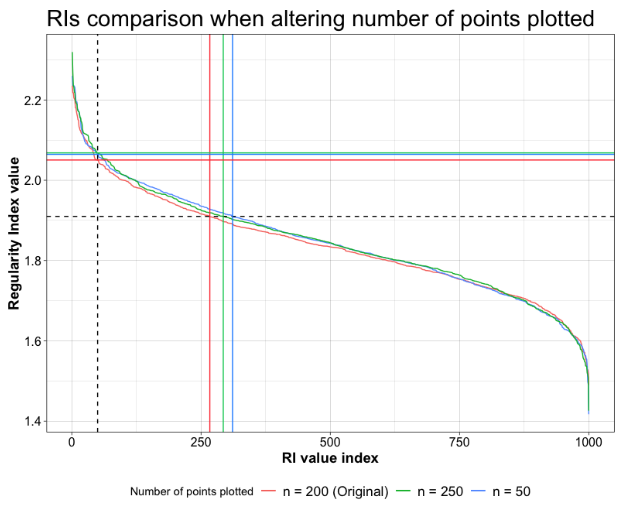

Figure 12:

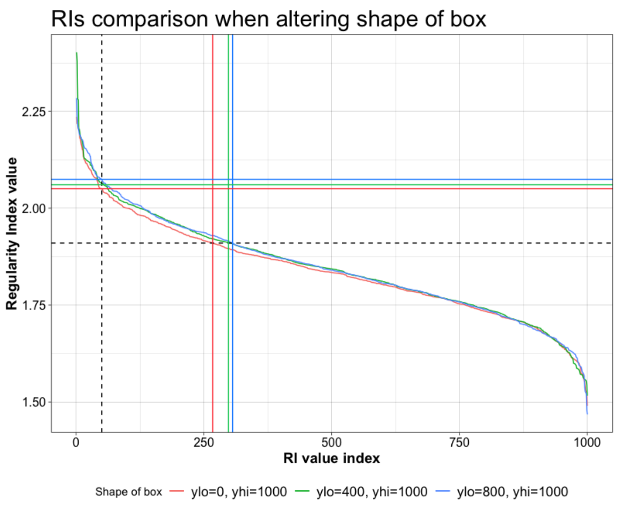

Figure 13:

and the RI of $n-1$ (n=100) simulated patterns are $x_2, ..., x_n$, then for each pattern i the u_i score was calculated by the following equation:

$$u_i = abs(x_i - \frac{1}{n-1}\sum_{j \neq i} x_j$$

The u-score is a method of assessing how close each value of a distribution is to the mean of that distribution. In this specific case, we would like the distribution of RIs created by our model to be as similar to the RI of the real data as possible. The expectation of a good model is two fold: the first U-score (the real data) should be approaching 0. Secondly, the other 99 simulations should also have a U-score close to that of the first. Of course there may be variation due to the stochastic nature of the *dmin*2*d* model.

Parameter finding can be approached by a coarser search followed by a more detailed search of the parameter space. For the coarse search, a sequence of m values from $10\mu m$ to $25\mu m$ going up by 1 was chosen. For each value of the mean diameter of the exclusion zone, a further sequence of standard deviations of 5 values from 0 to 8 was run. A heat map of this coarse search is shown in figure 14.

As we can see from the coarse heatmap in figure 14, the lowest $u1$ scores are achieved across a series of parameter matches. According to these results, the best match however was using mean 18 and standard deviation 6, giving a $u1$ score of 0.0112. The smallest ten u1 scores were all below 0.22.

A finer parameter search was conducted around the mean=18 μm sd=6 μm mark so as to give a finer model choice. The means were chosen between 17 and 19 μm by 0.2 μm, and the standard deviation was varied between 5 and 7 μm by 0.5 μm. This fine parameter search is shown in figure 15. Even with this fine search, the best result comes with the parameters mean=18 μm, standard deviation = 6 μm.

It may also be that the model parameters can be predicted by fitting a function to the results of the coarse parameter search. Three linear models were attempted, with increasing polynomial degrees. The results are shown in figures 17, 18, and 19. The 4 degree polynomial fits the data best, however, the third is nearly there and simpler. The model summary of for this prediction is shown in figure 16.

As shown by the summary, the intercept, first degree and second degree are the statistically significant belowt the p=0.05 level. However, even with the most confident of values, the intercept, there is a large difference between the model's 16.8 and the data's 13. More work would need to be done to successfully implement this linear model for prediction.

1.4 Part IV: Packing density [10 marks]

The *dmin*2*d* model is set to fail when it attempts 10,000 times to place a new circle into the existing pattern and fails all 10,000 times to match the minimum distance constraints. At this point, "no more points can be added. Max points = " and the number of existing circles is printed.

The "birth and death" model function was also created. At each epoch in this model, each circle is removed and, if possible re-added to the existing pattern based on the minimum distance constraints.

The percentage of the surface covered by the circle patterns produced by each model was calculated and compared to the theoretical maximum circle packing density: $\frac{\pi}{\sqrt{12}} = 0.90690$ (**Chang2010-xz**). This is achieved by packing circles hexagonally, as if it were a beehive.

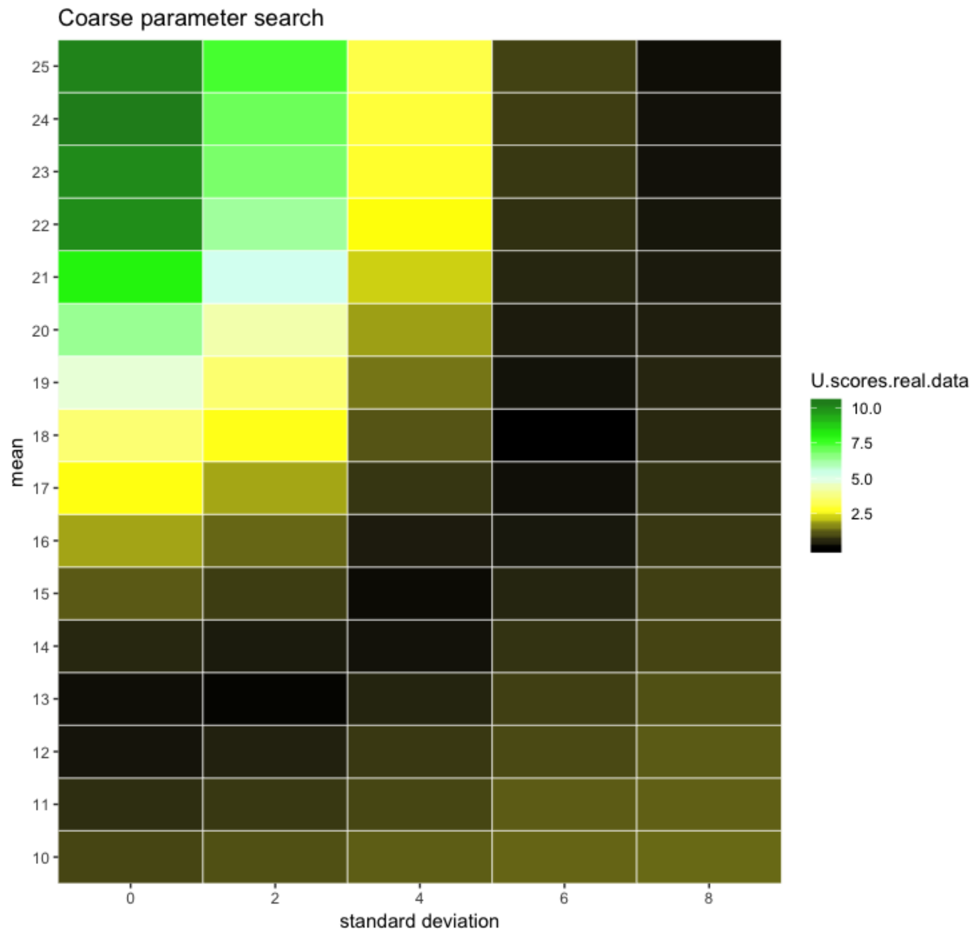

Figure 14: Heatmap of the u1 (U.scores.real.data) scores across different means and standard deviations. The more black, the smaller the u1 score, the better the fit.

The *dmin*2*d* model was run 100 times, and the max number of circles and max area covered calculated for each simulations. The mean maximum number of circles was 283.2, with standard deviation of 5.18. The mean area covered is 55.41% with standard deviation of 1.02 %.

The ”birth and death” model instead runs over 10 epochs to find a stable max number of circles. This max number is 332. The max area covered is 65.19 %.

To explain this difference, let us talk through an example. If a high n is set such that it is above the maximum packing density, the birth and death model will gradually remove each circle, decreasing n, until all points can be placed on the surface. At that point, the internal arrangement of the circles is gradually revisited and improved, such that the maximum packing density starts to increase.

The main weakness with the *dmin*2*d* model is that the earlier placed circles will a substantial effect on the ability of the later circles to be added. Moreover, these older circles cannot be altered. In the *birth.death* model, every point is revisited, allowing for a gradual reorganisation of the pattern until a higher packing density is achieved. The birth and death model can achieves, therefore, a higher circle packing density than then *dmin*2*d* model.

1.5 Part V: Moving points [15 marks]

The model of random disk packings from **Lubaehevsky˙undated-fd** was attempted. In this model, all the circles are placed together before the start of the algorithm. Each circle has an initial velocity and a diameter that increases with time. As time increases, the circles start to become larger and, therefore, closer to each other, until they are fully packed within the given area. At that point the simulation stops.

As a way to decrease the computational requirements of a simulation of this type, time is not continuous but discrete. The ”current time” is calculated as the minimum time for the next predicted collision. A collision can either be against another sphere or against a boundary.

The boundary conditions are also slightly altered, such that if a circle goes against a boundary, it will reappear on the opposite side of the box. As a way to simplify further this algorithm, if the new location on the

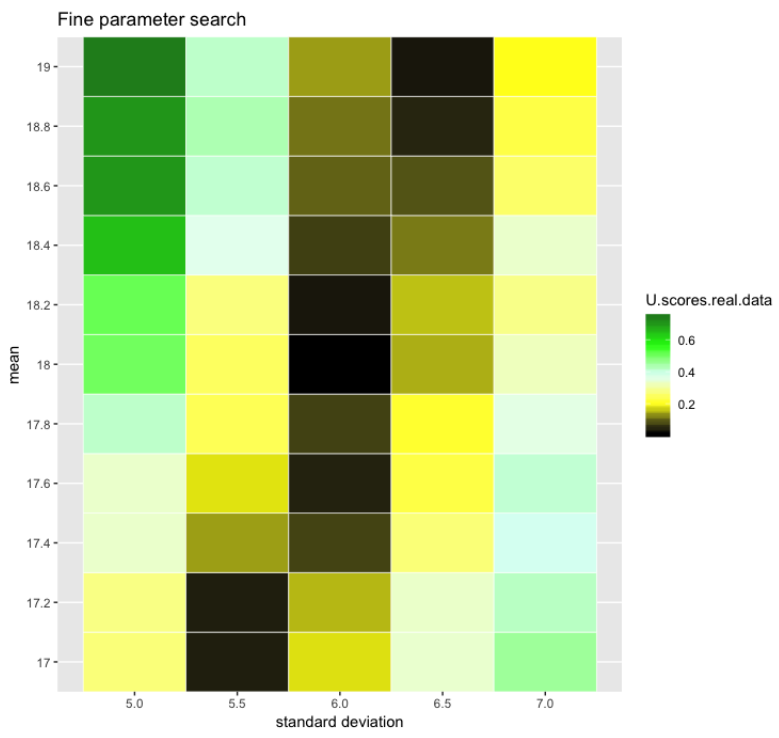

Figure 15: Heatmap of the u1 (U.scores.real.data) scores across different means and standard deviations. The more black, the smaller the u1 score, the better the fit.

```
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)            16.8000     0.2138  78.575   0.0081 **
poly(x, degree = 3)1    9.1706     0.4781  19.182   0.0332 *
poly(x, degree = 3)2    4.0089     0.4781   8.385   0.0756 .
poly(x, degree = 3)3    0.6325     0.4781   1.323   0.4121
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 16: Summary of the 3 degree linear model

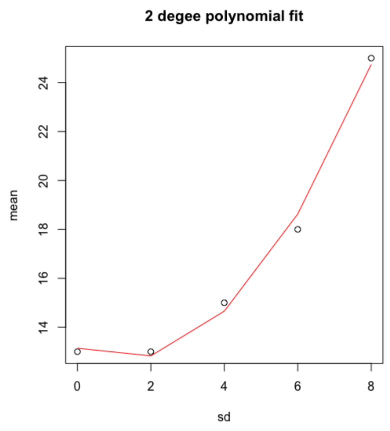

Figure 17: 2 degree polynomial linear model used to predict the best model parameters.

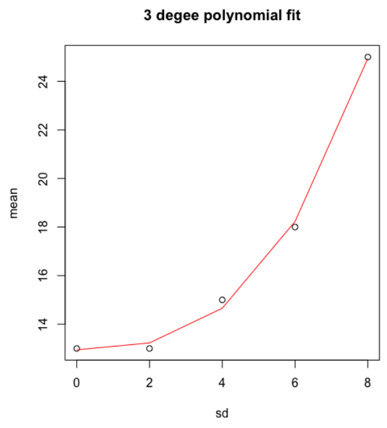

Figure 18: 3 degree polynomial linear model used to predict the best model parameters.

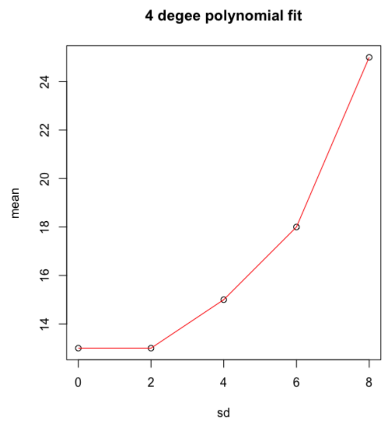

Figure 19: 4 degree polynomial linear model used to predict the best model parameters.

other side of the box was already taken by an existing circle, the original one bounces off the boundary instead of going through it.

To implement the algorithm, five major functions were written, of which one was described in psuedocode

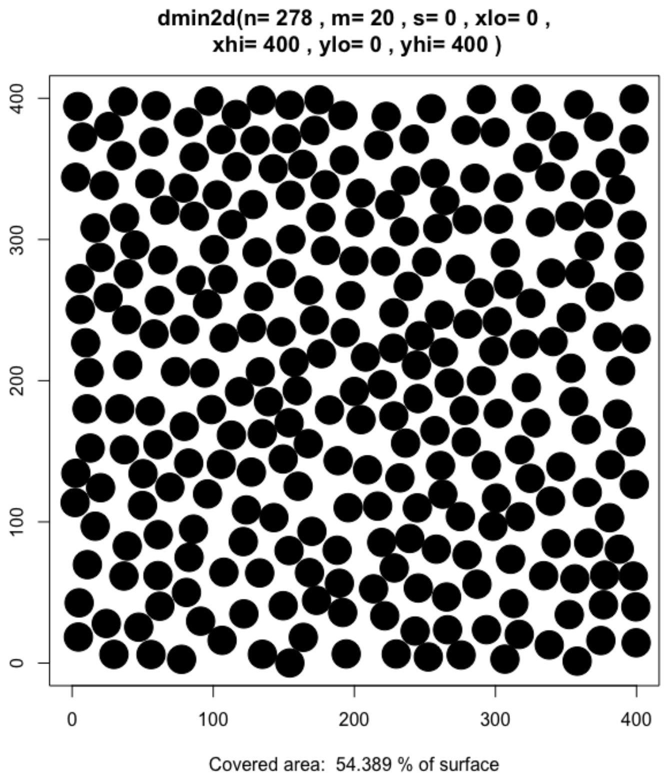

Figure 20: Example of max number of circles packed using the dmin2d algorithm

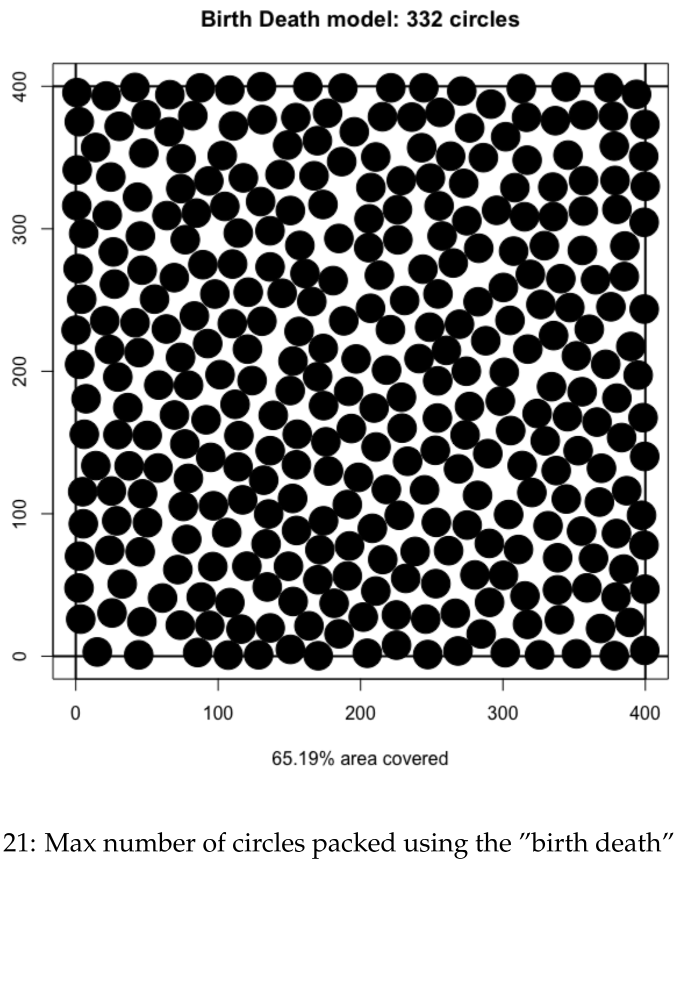

Figure 21: Max number of circles packed using the ”birth death” model

in the paper. These functions were `interaction.time.sphere`, `interaction.time.boundary`, `jump.sphere`, `jump.boundary` and `advance`.

The algorithm was able to run, however, sadly with a bug, which meant that the circles overlapped. A example of this can be seen in figure 22.

Whilst the algorithm was not successfully implemented, it is likely that it would have achieved an higher packing density than either the *dmin*2*d* or the "birth death" model.

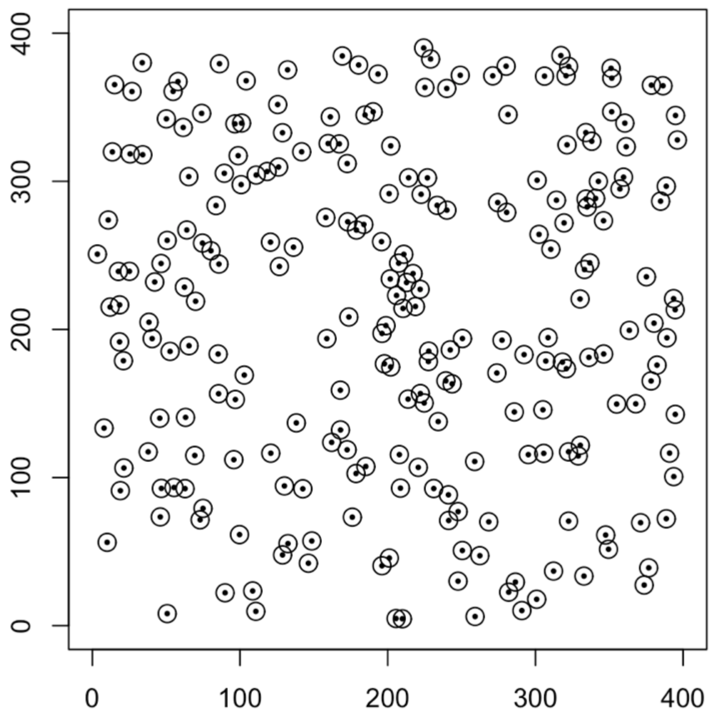

Figure 22: Example of an attempt at the Lubachevsky algorithm

A major difference between the Lubachevksy algorithm and those used above is the non-serial nature of the circle packings. As a way to model biological systems, this is a major advantage compared to the *dmin*2*d* model and the "birth and death" model. For example, the development of the retina into the a very regular pattern can be modelled by these circle packing simulations. However, retinal neurons do not suddenly appear serially and fully grown but rather start small and as they grow and move around and interact with other neurons, gradually order themselves into a final pattern. This is exactly this algorithm tries to achieve. One disadvantage, however, is its complexity and computational requirements.

Finally, all these models can do, even if very well, is to recreate a biological pattern. They do not, however, shine much light onto the biological mechanisms which actually lead to these patterns forming. They can be used as evidence to support a theoretical mechanism, but by themselves, cannot make discoveries.

```

setwd("~/Desktop/code/CompBio MPhil/Scientific Programming/Assignment3")

set.seed(4543)
require("plotrix")
library("ggplot2")
library("plyr")

positive.normal <- function(n,m, s){
  if (m>0){
    num <- rnorm(n, mean=m, sd = s)
    while(sum(num<0) != 0 ){
      num[num<0] <- rnorm(sum(num<0), mean=m, sd = s)
    }
    return(num)
  }else{
    return(0)
  }
}

dmin2d<- function(n, m, s, xlo, xhi, ylo, yhi, plot=TRUE) {
  ## n: number of points to simulate
  ## m: mean of Normal distribution
  ## s: s.d. of Normal distribution
  ## xlo, xhi: possible range of X values.
  ## ylo, yhi: possible range of Y values.
  coordinates <- matrix(0, nrow=n, ncol=3)
  stop = FALSE
  for (dot in 1:n){
    x_coord <- runif(1, min=xlo, max=xhi)
    y_coord <- runif(1, min=ylo, max=yhi)
    zone_dim <-positive.normal(1,m,s)
    k <- 0
    while (is.valid(x_coord, y_coord,zone_dim, coordinates) == FALSE){
      x_coord <- runif(1, min=xlo, max=xhi)
      y_coord <- runif(1, min=ylo, max=yhi)
      zone_dim <-positive.normal(1,m,s)
      k <- k+1
      #print(k)
      if (k > 10000){
        print("no more points can be added")
        stop = TRUE
        break
        }
      }
    if (stop==TRUE) {
      print(paste("max points =",dot))
      break}
    coordinates[dot, 1] <- x_coord
    coordinates[dot, 2] <- y_coord
    coordinates[dot, 3] <- zone_dim
  }
  final.coordinates <- coordinates[coordinates[,1]>0, ]
  if (plot){
    area.sum <- plot.find.area(final.coordinates, xlim=c(xlo, xhi), ylim=c(ylo,yhi),
                      main=paste("dmin2d(n=",dot,", m=", m, ", s=",s,", xlo=",
                                 xlo,",\n xhi=",xhi,", ylo=",ylo,", yhi=", yhi, ")"))
    covered.percentage <- round( (area.sum/((xhi-xlo)*(yhi-ylo))) * 100 , 3)
    title(xlab= paste("Covered area: ", covered.percentage , "% of surface"))
    print (paste("Covered area: ", covered.percentage , "% of surface"))
    ### ADD Covered area as SUB to plot
  }
  return(final.coordinates)
}

is.valid <- function(x_coord, y_coord, zone_dim, coordinates){
  if (sum(coordinates[,1]>0) == 0){
    return(TRUE)
  }else{
    distances <- rep(0, times = sum(coordinates[,1]>0))
    past.x <- coordinates[,1][coordinates[,1]>0]
    past.y <- coordinates[,2][coordinates[,2]>0]
    diff.x <- abs(x_coord - past.x )
    diff.y <- abs(y_coord - past.y)
    distances <- sqrt(diff.x^2 + diff.y^2)
    temp.rad <- zone_dim/2
    if  (any(distances < (temp.rad+coordinates[coordinates[,1]>0,3]/2))){return(FALSE)}
    else {return(TRUE)}
  }
}


plot.find.area <- function(coordinates, ...){
  par(pty="s")
  plot(1, type="n", xlab="", ylab="", ...)
  area_sum = 0.0
  for ( it in 1:nrow(coordinates) ){
    points(x=coordinates[it,1],y=coordinates[it,2],pch=20, cex=0.5)
    draw.circle(x=coordinates[it,1],y=coordinates[it,2],radius=coordinates[it,3]/2, col="black")
    area_sum = area_sum + ( pi * (coordinates[it,3]/2)^2 )
  }
  return(area_sum)
}



par(pty="s")
set.seed(4543)
res <- dmin2d(200, 30, 5, 200, 1000, 100, 900)

hist(res[,3], breaks=30, main="Histogram of diameters of circles", xlab='diameter')


##################################################################################################
# regularity index
##################################################################################################

#distance of point to its nearest neighbour
#mean / sd

regularity.index <- function(coordinates){
  closest.point.distances <- matrix(0, nrow=length(coordinates[,1]), ncol=2)
  for (point in 1:length(coordinates[,1])){
    x.cor <- coordinates[point,1]
    y.cor <- coordinates[point,2]
    diff.x <- x.cor - coordinates[,1]
    diff.y <-  y.cor - coordinates[,2]
    distances <- sqrt(diff.x^2 + diff.y^2)
    closest.point.value <- min(distances[distances>0])
    closest.point.distances[point,] <- c(closest.point.value, grep(closest.point.value, distances))
  }
  R.I.value <- mean(closest.point.distances[,1])/sd(closest.point.distances[,1])
  #print( grep(closest.point.value, distances))
  return(R.I.value)
}


random.data <-dmin2d(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000)
random.data <- random.data[[1]]
plot(x = random.data[,1], y=random.data[,2], pch=20, cex=0.5)
text(x = random.data[,1], y=random.data[,2]+10, labels=c(1:dim(random.data)[1]), cex=0.4)
regularity.index(random.data)


###############################################################################################
# Measure the RI of each pattern and report the 50th largest value. What is the utility of such
# a measure? How do your results vary as you vary the number of points (n) in a pattern, or the
# geometry of the sample area (i.e. square regions versus rectangular)?
#  (Hint: You may need to write your dmin2d function so that when the ???m??? argument is zero, the
#   minimal distance constraint is ignored.)
###############################################################################################


## FUNCTION TO COMPUTE 1000x DMIN2D MODEL
combined.f <- function(n, m, s, xlo, xhi, ylo, yhi){
  res2 <-dmin2d(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000, plot=FALSE)
  RI <- regularity.index(res2[[1]])
  return(RI)
}

thousandRIs.1 <- replicate(1000, combined.f(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000))

```

```r
thousandRIs.1.sorted <- sort(thousandRIs.1, decreasing = TRUE)
fiftieth <- thousandRIs.1.sorted[50]


## FUNCTION TO PLOT DISTRIBUTION OF 1000x DMIN2D MODEL
plot.RI.distribution <- function(coordinates, ...){
  sorted.coordinates <- sort(coordinates, decreasing = TRUE)
  fiftieth <- sorted.coordinates[50]
  plot(sort(coordinates, decreasing = TRUE), type='l', main="Sorted regularity index (RI)",
       ylab="Regularity Index value", xlab="Simulation number index", ...)

  abline(h = 1.91, v=which.min(abs(sorted.coordinates - 1.91)), col="red")
  axis(1, at=which.min(abs(sorted.coordinates - 1.91)),
       label=as.character(which.min(abs(sorted.coordinates - 1.91))), col="red", col.axis="red")
  axis(2, at=1.91, label="1.91", col="red", col.axis="red", pos=2)

  abline(h = fiftieth, v=50, col="blue")
  axis(1, at=50, label="50", col="blue", col.axis="blue")
  axis(2, at=fiftieth, label=as.character(round(fiftieth,3)), col="blue", col.axis="blue", pos=2)

  abline(h = mean(coordinates), v = which.min(abs(sorted.coordinates - mean(coordinates))),
         col="forestgreen")
  axis(1, at=which.min(abs(sorted.coordinates - mean(coordinates))),
       label=as.character(which.min(abs(sorted.coordinates - mean(coordinates)))),
       col="forestgreen", col.axis="forestgreen")
  axis(2, at=mean(coordinates), label=as.character(round(mean(coordinates),2)),
       col="forestgreen", col.axis="forestgreen", pos=2)
}

plot.RI.distribution(thousandRIs.2.1)


## FUNCTION TO PLOT HISTOGRAM OF 1000x DMIN2D MODEL
plot.RI.hist <- function(coordinates, ...){
  coordinates <- sort(coordinates, decreasing = TRUE)
  fiftieth <- coordinates[50]
  hist(coordinates, breaks=100, xlab="Regularity Index", xlim=c(1.5,2.35), ylim=c(0,50),... )
  legend(x=2.0,y=40, legend = paste("mean =", round(mean(coordinates),3),"\n",
                                    "s.d.=", round(sd(coordinates),3)), bty = "n")
  abline(v=1.91, col="red")
  axis(1, at=1.91, label="1.91", col="red", col.axis="red", pos=-3.5)
  abline(v = fiftieth, col="blue")
  axis(1, at=fiftieth, label=as.character(round(fiftieth,2)),  col="blue", col.axis="blue", pos
    =-3.5)
  abline(v = mean(coordinates), col="forestgreen")
  axis(1, at=mean(coordinates), label=as.character(round(mean(coordinates),2)),
       col="forestgreen", col.axis="forestgreen", pos=-3.5)
}

plot.RI.hist(thousandRIs.3.4, main="ylo=800")

shapiro.test(thousandRIs.1)

plot.RI.hist(thousandRIs.3.4)


## REPLICATE DMIN2D MODEL WITH DIFFERENT PARAMETERS
#if n changes:
thousandRIs.2.1 <- replicate(1000, combined.f(n=50, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000))
thousandRIs.2.2 <- replicate(1000, combined.f(n=100, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000))
thousandRIs.2.3 <- replicate(1000, combined.f(n=400, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000))
thousandRIs.2.4 <- replicate(1000, combined.f(n=800, m=0, s=0, xlo=0, xhi=1000, ylo=0, yhi=1000))

#if geometry changes:
thousandRIs.3.1 <- replicate(1000, combined.f(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=200, yhi=1000))
thousandRIs.3.2 <- replicate(1000, combined.f(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=400, yhi=1000))
thousandRIs.3.3 <- replicate(1000, combined.f(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=600, yhi=1000))
thousandRIs.3.4 <- replicate(1000, combined.f(n=200, m=0, s=0, xlo=0, xhi=1000, ylo=800, yhi=1000))

thousandRIs.tot <- data.frame(thousandRIs.1, thousandRIs.2.1, thousandRIs.2.2,thousandRIs.2.3,
    thousandRIs.2.4,
                              thousandRIs.3.1, thousandRIs.3.2, thousandRIs.3.3, thousandRIs.3.4)

saveRDS(thousandRIs.tot, "thousandRIs.tot")


```

```r
thousandRIs.tot.sorted <- apply(thousandRIs.tot,2,sort,decreasing=T)
thousandRIs.tot.sorted <- data.frame(thousandRIs.tot.sorted)
thousandRIs.tot.sorted["Index"] <- 1:nrow(thousandRIs.tot.sorted)

RIs.numbers <- data.frame("n" = c(50, 100, 200, 400, 800),
                 "50th RI" = c(thousandRIs.tot.sorted$thousandRIs.2.1[50],
                           thousandRIs.tot.sorted$thousandRIs.2.2[50],
                           thousandRIs.tot.sorted$thousandRIs.2.3[50],
                           thousandRIs.tot.sorted$thousandRIs.1[50],
                           thousandRIs.tot.sorted$thousandRIs.2.4[50]))

RIs.shape <- data.frame("ylo" = c(0, 200, 400, 600, 800),
                   "50th RI" = c(thousandRIs.tot.sorted$thousandRIs.1[50],
                              thousandRIs.tot.sorted$thousandRIs.3.1[50],
                              thousandRIs.tot.sorted$thousandRIs.3.2[50],
                              thousandRIs.tot.sorted$thousandRIs.3.3[50],
                              thousandRIs.tot.sorted$thousandRIs.3.4[50]))


thousandRIs.tot.index.at.1.91 <- apply(thousandRIs.tot.sorted,2,function(x) which.min(abs(x - 1.91))
    )


ggplot(thousandRIs.tot.sorted, aes(Index)) +
  theme_linedraw()+
  geom_line(aes(y = thousandRIs.1, colour = "n = 200 (Original)")) +
  geom_line(aes(y = thousandRIs.2.1, colour = "n = 50")) +
  geom_line(aes(y = thousandRIs.2.4, colour = "n = 800")) +
  geom_hline(yintercept = 1.91, linetype = "dashed", colour="black") +
  geom_vline(xintercept = thousandRIs.tot.index.at.1.91["thousandRIs.1"], colour="brown1") +
  geom_vline(xintercept = thousandRIs.tot.index.at.1.91["thousandRIs.2.1"], colour="dodgerblue1") +
  geom_vline(xintercept = thousandRIs.tot.index.at.1.91["thousandRIs.2.4"], colour="springgreen3") +
  geom_vline(xintercept = 50, linetype = "dashed", colour="black") +
  geom_hline(yintercept = thousandRIs.tot.sorted$thousandRIs.1[50], colour="brown1") +
  geom_hline(yintercept = thousandRIs.tot.sorted$thousandRIs.2.1[50], linetype = "solid", colour="
    dodgerblue1") +
  geom_hline(yintercept = thousandRIs.tot.sorted$thousandRIs.2.4[50], linetype = "solid", colour="
    springgreen3") +
  ggtitle("RIs comparison when altering number of points plotted") + xlab("RI value index") +
  ylab("Regularity Index value") + labs(colour="Number of points plotted")+
  theme(legend.position="bottom") +
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14,face="bold"),
        plot.title = element_text(size = rel(2)),
        legend.text = element_text(size = rel(1.2)))



ggplot(thousandRIs.tot.sorted, aes(Index)) +
  theme_linedraw() +
  geom_line(aes(y = thousandRIs.1, colour = "ylo=0, yhi=1000 (original)")) +
  geom_line(aes(y = thousandRIs.3.2, colour = "ylo=400, yhi=1000")) +
  geom_line(aes(y = thousandRIs.3.4, colour = "ylo=800, yhi=1000")) +
  geom_hline(yintercept = 1.91, linetype = "dashed", colour="black") +
  geom_vline(xintercept = thousandRIs.tot.index.at.1.91["thousandRIs.1"], colour="brown1") +
  geom_vline(xintercept = thousandRIs.tot.index.at.1.91["thousandRIs.3.1"], colour="springgreen3") +
  geom_vline(xintercept = thousandRIs.tot.index.at.1.91["thousandRIs.3.4"], colour="dodgerblue1") +
  geom_vline(xintercept = 50, linetype = "dashed", colour="black") +
  geom_hline(yintercept = thousandRIs.tot.sorted$thousandRIs.1[50], colour="brown1") +
  geom_hline(yintercept = thousandRIs.tot.sorted$thousandRIs.3.1[50], colour="springgreen3") +
  geom_hline(yintercept = thousandRIs.tot.sorted$thousandRIs.3.4[50], colour="dodgerblue1") +
  ggtitle("RIs comparison when altering shape of box") + xlab("RI value index") +
  ylab("Regularity Index value") + labs(colour="Shape of box")+
  theme(legend.position="bottom") +
  theme(legend.position="bottom") +
  theme(axis.text=element_text(size=12),
        axis.title=element_text(size=14,face="bold"),
        plot.title = element_text(size = rel(2)),
        legend.text = element_text(size = rel(1.2)))




##############################################################################################
#Fit the model to some data [10 marks]
##############################################################################################
```

```r

## LOAD REAL DATA
real.data <- read.delim("spa3_real.dat.txt", header=F, sep=" ")
real.data.mat <- as.matrix(real.data)
plot(x = real.data[,1], y=real.data[,2], pch=20, cex=1, main="Real data: 238 points",
     xlab="", ylab="" )
#text(x = real.data[,1], y=real.data[,2]+10, labels=c(1:dim(real.data)[1]), cex=0.4)
#regularity.index(real.data)


## FUNCTIONS FOR PARAMETER SEARCHING:
## CALCULATE.U.SCORE & FIND.M.S.

calculate.u.score <- function(n.min1.models, real.data){
  ## n.min1.models: list of n repetitions of specific model
  ## real.data: matrix of point pattern coordinates provided
  #calculate regularity index of each repeated pattern
  n.min1.RIs <- lapply(n.min1.models, regularity.index)
  #add RI of real data as first in vector
  n.RIs <- c(regularity.index(real.data), unlist(n.min1.RIs ))
  n.U <- list()
  k <- 1
  for (ii in 1:length(n.RIs)){
    #print(paste("U scores for iteration" ,k,"out of", length(n.RIs)))
    n.U[[k]] <- abs(n.RIs[ii] - 1/length(n.RIs-1) * sum(n.RIs[-ii]))
    k <- k+1
  }
  n.U <- as.matrix(n.U)
  return(n.U)
}


find.m.s <- function(m.min, m.max, m.by, s.min, s.max, real.data.mat){
  ## create 99 repetitions for each parameter couple.
  ## save the 99 repetitions for each model in a coordinates.full.
  ## apply the calculate.u.score to each item of coordinates.full.
  ## create dataframe with parameters (m,s) and first U score of
  ## the real data.
  m.scores <- seq(from=m.min, to=m.max, by= m.by)
  coordinates.full <- list()
  hundred.U.full <- list()
  parameters <- list()
  k <- 1
  for (i in m.scores){
    for (j in round(seq(from=s.min, to=s.max, length.out=5), 2)){
      parameters[[k]] <- c(i, j)
      print(paste('mean =',i,',' ,'sd =',j))
      coordinates.full[[k]] <- rlply(99, dmin2d(n=238, m=i, s=j,
                                                xlo=0, xhi=400, ylo=0, yhi=400, plot=FALSE),
                                    .progress = "text")
      #print(k)
      k <- k+1
    }
  }
  hundred.U.full <- lapply(X = coordinates.full, FUN = calculate.u.score, real.data = real.data.mat)
  U.scores.real.data <- lapply(hundred.U.full, `[[`, 1)
  U.scores.real.data <- unlist(U.scores.real.data)
  parameters <- unlist(parameters)
  means <- parameters[c(TRUE, FALSE)]
  s.d.s <- parameters[c(FALSE, TRUE)]
  df.result <- data.frame(means,s.d.s, U.scores.real.data)
  return(df.result)
}

## RUNNNING FIND.M.S. FUNCTION
test.ms <- find.m.s(m.min=10, m.max=25, m.by=1, s.min=0, s.max=8, real.data.mat=real.data.mat)
saveRDS(test.ms, "coarse.parameter.search")

fine.search <- find.m.s(m.min=17, m.max=19, m.by=0.2, s.min=5, s.max=7, real.data.mat=real.data.mat)


test.ms

test.ms[which.min(test.ms$U.scores.real.data),]

U1s <- test.ms$U.scores.real.data
```

```r
sort(U1s, decreasing=FALSE)[1:10]

min.0 <- which.min( test.ms[grep(0, test.ms$s.d.s), 3] ) + 9
min.2 <- which.min( test.ms[grep(2, test.ms$s.d.s), 3] ) + 9
min.4 <- which.min( test.ms[grep(4, test.ms$s.d.s), 3] ) + 9
min.6 <- which.min( test.ms[grep(6, test.ms$s.d.s), 3] ) + 9
min.8 <- which.min( test.ms[grep(8, test.ms$s.d.s), 3] ) + 9

y <- c(min.0, min.2,min.4, min.6,  min.8)
x <- c(0,2,4,6,8)

model=lm(y~poly(x,degree = 3))
y2=predict(model)
plot(x,y, ylab="mean", xlab="sd", main="4 degee polynomial fit", ylim=c(0,25))
lines(x,y2,col="red")
summary(model)

model.df <- data.frame(x=x, y=y2)

x.test <- seq(from=0, to=10, by=0.1)
y.test <- 16.8 + 9.17*x + 4*x^2 + 0.6*x^3 + 0.47*x^4
lines(x.test, y.test)




#CAORSE SEARCH HEATMAP
mycol <- c("black","yellow","lightcyan","green", "forestgreen")

ggplot(fine.search,aes(x=s.d.s,y=means,fill=U.scores.real.data))+
  geom_tile()+
  #redrawing tiles to remove cross lines from legend
  geom_tile(colour="white",size=0.25, show.legend=FALSE)+
  #remove axis labels, add title
  labs(x="standard deviation",y="mean",title="Fine parameter search")+
  #add colours
  scale_fill_gradientn(colours = mycol) +
  #remove extra space
  scale_y_discrete(expand = c(0,0), breaks= unique(fine.search$means),
                   labels=as.character(unique(fine.search$means)),
                   limits=unique(fine.search$means)) +
  #custom breaks on x-axis
  scale_x_discrete(expand = c(0,0), breaks = unique(fine.search$s.d.s),
                   labels = as.character(unique(fine.search$s.d.s)),
                   limits = as.character(unique(fine.search$s.d.s)))


unique(fine.search$s.d.s)


#
    ####################################################################################################
##              Packing density [10]
#
    ####################################################################################################


## BIRTH DEATH FUNCTION
birthdeath <- function(n, m, s, xlo, xhi, ylo, yhi){
  coordinates <- matrix(0, nrow=n, ncol=3)
  coordinates[, 1] <-runif(n, min=xlo, max=xhi)
  coordinates[, 2] <- runif(n, min=ylo, max=yhi)
  coordinates[,3] <- positive.normal(n,m,s)
  nepochs <- 10
  for (epoch in 1:nepochs) {
    ## One round of birth and death.
    print(paste("Epoch", epoch))
    sequence <- sample(n)
    for (i in sequence) {
      stop=FALSE
      ## Point i must now be killed, and a new point
      ## positioned (born) randomly subject to satisfying
      ## the minimal distance constraint.
      k <- 0
      coordinates[i,] <- 0
      x_coord <-  runif(1, min=xlo, max=xhi)
      y_coord <- runif(1, min=ylo, max=yhi)
      zone_dim <- positive.normal(1,m,s)
      while (is.valid(x_coord, y_coord,zone_dim, coordinates) == FALSE){
        x_coord <- runif(1, min=xlo, max=xhi)
        y_coord <- runif(1, min=ylo, max=yhi)
        zone_dim <-positive.normal(1,m,s)
        k <- k+1
        if (k >= 10000){
          #print(paste(k, "times I tried but no more points can I add."))
          print(paste(sum(coordinates[,1]>0), "points left"))
          stop = TRUE
          break
        }
      }
      if (stop==TRUE) {next}
      else{
        coordinates[i, 1] <- x_coord
        coordinates[i, 2] <- y_coord
        coordinates[i, 3] <- zone_dim
      }
    }
    print(paste(sum(coordinates[,1]>0), "points left"))
  }
  full.coordinates <- coordinates[coordinates[,1]>0 , ]

  return(full.coordinates)
}

## RUNNING BIRTH.DEATH + PLOT
birth.death.max <- birthdeath(n=400, m = 20, s = 0, xlo = 0, xhi = 400, ylo = 0, yhi = 400)
par(pty = 's')
plot(x = birth.death.max[,1], y=birth.death.max[,2], xlim=c(0,400), ylim=c(0, 400), pch=20, cex=0.2,
     main="Birth Death model: 332 circles",
     xlab ="65.19% area covered", ylab="")
abline(v=c(0, 400), h=c(0,400), lwd=2)
symbols(x = birth.death.max[,1], y=birth.death.max[,2], circles = birth.death.max[,3]/2, add=T,
    inches=F , bg="black")


dmin2d(n=400, m = 20, s = 0, xlo = 0, xhi = 400, ylo = 0, yhi = 400)

## 100 REPETITIONS OF DMIN2D FOR DISTRIBUTION OF MAX CIRCLES
combine.dmin2d.plot.area <- function(n, m, s, xlo, xhi, ylo, yhi){
  res <- dmin2d(n=n, m=m, s=s, xlo=xlo, xhi=xhi, ylo=ylo, yhi=yhi)
  area <- plot.find.area(res[[1]])
  return(c(res[[2]], area))
}

max.dmin2d.100 <- rlply(100, combine.dmin2d.plot.area(n=300, m=20, s=0, xlo=0, xhi=400, ylo=0, yhi
    =400),.progress = "text")

max.dmin2d.100 <- unlist(max.dmin2d.100)
max.circles.dmin2d <- max.dmin2d.100[c(TRUE, FALSE)]
max.area.dmin2d <- max.dmin2d.100[c(FALSE, TRUE)]



#
    ##################################################################################################
##             Lubachevsky and Stillinger (1990) algorithm
#
    ##################################################################################################


hundred.seconds <- Lubachevksy.algo(238)


Lubachevksy.algo <- function(N){
  current.time <- 0
  a0 <- 5 #### based on area?
  K <- matrix(0, nrow=1, ncol=4)
  K[1,] <- c(0, 400, 0, 400)
```

```r
boundaries <- c(0, 400, 0, 400)
#event$time[,1] <- rnorm(10,4,1)
new <- rep(1, length=N)
old <- rep(2, length=N)
event <- list('time' = matrix(0, nrow=N, ncol=2),
              'state' = list('position' = list("xy" = matrix(c(0,0), nrow=N, ncol=2),
                                               'xxyy' = matrix(c(0,0), nrow=N, ncol=2)),
                             'velocity' = list("xy" = matrix(c(0,0), nrow=N, ncol=2),
                                               'xxyy' = matrix(c(0,0), nrow=N, ncol=2))),
              'partner'=matrix(nrow=N, ncol=2))
initial.pos <- dmin2d(238, a0, 0, a0, 400-a0, a0, 400-a0, plot=FALSE )
event[['state']][['position']][[old[1]]] <- initial.pos[[1]][,c(1,2)]
event[['state']][['velocity']][[old[1]]] <- apply(event[['state']][['velocity']][[old[1]]],c(1,2),
                                                  function(x) runif(1,min=-1, max=1))
event[['state']][['position']][[new[1]]] <- event[['state']][['position']][[old[1]]]
event[['state']][['velocity']][[new[1]]] <- event[['state']][['velocity']][[old[1]]]

while (current.time < 10){
  current.time <- min(sapply(1:N, function(i) event$time[i, new[i]]))
  i.star <- which.min(sapply(1:N, function(i) event$time[i, new[i]]))
  print(current.time)
  xs <- sapply(1:N, function(i) event[["state"]][["position"]][[new[i]]][i,1])
  ys <- sapply(1:N, function(i) event[["state"]][["position"]][[new[i]]][i,2])
  plot(x = xs, y=ys, xlim=c(K[1,1],K[1,2]), ylim=c(K[1,3],K[1,4]), pch=20, cex=0.5)
  length(xs); length(ys)
  for (i in 1:length(xs)){
    draw.circle(x=xs[i], y=ys[i], radius=(a0*current.time)/2)
  }
  Sys.sleep(.1)
  new[i.star] <- old[i.star]
  old[i.star] <- 3-new[i.star]
  P <- calc.P(N, i.star, event, new, old, a0)
  if (P[1] < Inf){
    j.star <- P[2]
  }
  Q <- calc.Q(K, event,i.star, old, a0)
  timeQ <- as.numeric(Q[1])
  if (timeQ < Inf){
    k.star <- Q[2]
  }
  R <- as.numeric(min(P[1],timeQ))
  event$time[i.star, new[i.star]] <- R
  if (R < Inf){
     state1 <- advance(c(event[['state']][['position']][[old[i.star]]][i.star,],
                         event[['state']][['velocity']][[old[i.star]]][i.star,]),
                       event$time[i.star, old[i.star]], R)
     event[['state']][['position']][[new[i.star]]][i.star,] <- state1[1:2]
     event[['state']][['velocity']][[old[i.star]]][i.star,] <- state1[3:4]

    if (timeQ < P[1]){
      state1 <- jump.boundaries(state1, k.star, a0, event, R, i.star, K)
      event[['state']][['position']][[new[i.star]]][i.star,] <- state1[1:2]
      event[['state']][['velocity']][[new[i.star]]][i.star,] <- state1[3:4]
      event$partner[i.star, new[i.star]] <- NA
    }else{
      event$time[j.star, new[j.star]] <- R
      state2 <- advance(c(event[['state']][['position']][[old[j.star]]][j.star,],
                          event[['state']][['velocity']][[old[j.star]]][j.star,]),
                        event$time[j.star, old[j.star]], R)
      new.states <- jump.spheres(state1, state2)
      event[['state']][['position']][[new[i.star]]][i.star,] <- new.states[1:2]
      event[['state']][['velocity']][[new[i.star]]][i.star,] <- new.states[3:4]
      event[['state']][['position']][[new[j.star]]][j.star,] <- new.states[5:6]
      event[['state']][['velocity']][[old[j.star]]][j.star,] <- new.states[7:8]
      m.star <- event$partner[j.star, new[j.star]]
      event$partner[i.star, new[i.star]] <- j.star
      event$partner[j.star, new[j.star]] <- i.star
      if ((is.na(m.star) == FALSE) & (isTRUE(all.equal(m.star,i.star)) == FALSE) ){
        state.m <- advance(c(event[['state']][['position']][[old[m.star]]][m.star,],
                             event[['state']][['velocity']][[old[m.star]]][m.star,]),
                           event$time[m.star, old[m.star]], event$time[m.star, new[m.star]])
        event[['state']][['position']][[new[m.star]]][m.star,] <- state.m[1:2]
        event[['state']][['velocity']][[new[m.star]]][m.star,] <- state.m[3:4]
      }
    }
  }
  }
  return(event)
}


calc.P <- function(N, i.star, event, new, old, a0){
  P.istar.j <- rep(0, N)
  for (j in 1:N){
    if (j != i.star){
      P.istar.j[j] <- interaction.time.sphere(c(event[['state']][['position']][[old[i.star]]][i.star
      ,],
                                              event[['state']][['velocity']][[old[i.star]]][i.star
      ,]),
                                            event$time[i.star,old[i.star]],
                                            c(event[['state']][['position']][[old[j]]][j,],
                                              event[['state']][['velocity']][[old[j]]][j,]),
                                            event$time[j, old[j]], a0)
    }
  }
  m <- min(P.istar.j[P.istar.j>0])
  j.star <- grep(m,P.istar.j )
  return(c(m, j.star))
}

calc.Q <- function(K, event,i.star, old, a0){
  Q.i.k <- matrix(NA, nrow=dim(K)[1], ncol=2)
  for (k in 1:dim(K)[1]){
    Q.i.k[k,] <- interaction.time.boundary(c(event[['state']][['position']][[old[i.star]]][i.star,],
                                             event[['state']][['velocity']][[old[i.star]]][i.star,]),
                                           event$time[i.star,old[i.star]], K[k,], a0)
  }
 return(Q.i.k)
}


dot <- function(vector1, vector2){
  p <- vector1[1]*vector2[1] + vector1[2]*vector2[2]
  return(p)
}

interaction.time.sphere <- function(state1, time1, state2, time2, a0){
  ## given state1 time1 state2, time2, compute the time of the next potential
  ## interaction of sphere 1 with sphere 2 while ignoring presence of other
  ## spheres and boundaries
  ## state: c(position.x, position.y, velocity.x, velocity.y)
  ## time: single number
  t.star <- max(time1, time2)
  r.10 <- state1[1:2]+state1[3:4]*(t.star-time1)
  r.20 <- state2[1:2]+state2[3:4]*(t.star-time2)
  r <- r.20 - r.10
  v <- state2[3:4] - state1[3:4]
  v.abs <- sqrt(v[1]^2 + v[2]^2)
  r.abs <- sqrt(r[1]^2 + r[2]^2)
  A <- v.abs^2 - (a0)^2
  B <- dot(r,v) - (a0*t.star)
  C <- r.abs^2 - (a0*t.star)^2
  if ((B <= 0 | A<0) & (B^2- A*C >=0)){
    t <- (-B-(B^2 - A*C)^0.5)/A
  } else if ((B>0 & A>=0) | (B^2 - A*C < 0)){
    t <- Inf
  }
  time.tot <- t.star + t
  return(time.tot)
}


deg2rad = function(deg) {
  return((pi * deg) / 180)
}

rad2deg = function(rad) {
  return((180 * rad) / pi)
}

interaction.time.boundary <- function(state1, time1, boundaries, a0){
  ## To express boundary crossings, k is index for the set of K boundaries
  xlo <- boundaries[1]
  xhi <- boundaries[2]
  ylo <- boundaries[3]
  yhi <- boundaries[4]
  x <- state1[1]
  y <- state1[2]
  Vx <- state1[3]
  Vy <- state1[4]
  theta.1 <- rad2deg(atan((xhi-x)/(yhi-y)))
  theta.2 <- rad2deg(atan((yhi-y)/(xhi-x)))
  theta.3 <- rad2deg(atan((y-ylo)/(xhi-x)))
  theta.4 <- rad2deg(atan((xhi-x)/(y-ylo)))
  theta.5 <- rad2deg(atan((x-xlo)/(y-ylo)))
  theta.6 <- rad2deg(atan((y-ylo)/(x-xlo)))
  theta.7 <- rad2deg(atan((yhi-y)/(x-xlo)))
  theta.8 <- rad2deg(atan((x-xlo)/(yhi-y)))
  if (Vx>0 & Vy>0){
    phi <- 90 - abs(rad2deg(atan(Vy/Vx)))
  } else if (Vx>0 & Vy<0){
    phi <- 90 + abs(rad2deg(atan(Vy/Vx)))
  } else if (Vx<0 & Vy >0){
    phi <- 270 + abs(rad2deg(atan(Vy/Vx)))
  } else if (Vx<0 & Vy<0){
    phi <- 270 - abs(rad2deg(atan(Vy/Vx)))
  }
  psi <- abs(rad2deg(atan(Vy/Vx)))
  Vmag <- sqrt(Vx^2 + Vy^2)
  if(theta.1 < phi & phi < (theta.1+theta.2+theta.3)){
    d <- (xhi-x)/cos(deg2rad(psi)) - (a0*time1)/2
    t <- d/Vmag
    E <- "R"
  } else if ((theta.1 +theta.2 +  theta.3) < phi & phi < (theta.1 +theta.2 +  theta.3 + theta.4 +
    theta.5)){
    d <- (y-ylo)/cos(deg2rad(psi))- (a0*time1)/2
    t <- d/Vmag
    E <- "B"
  } else if ((theta.1 +theta.2 +  theta.3 + theta.4 + theta.5) < phi &
                 phi < (theta.1 +theta.2 +  theta.3 + theta.4 + theta.5 + theta.6 + theta.7)){
    d <- (x-xlo)/cos(deg2rad(psi)) - (a0*time1)/2
    t <- d/Vmag
    E <- "L"
  } else if (phi > 180){
    if ((theta.1 +theta.2 +  theta.3 + theta.4 + theta.5 + theta.6 + theta.7)< phi & phi < (theta.1
    + 360)){
      d <- (yhi-y)/cos(deg2rad(psi)) - (a0*time1)/2
      t <- d/Vmag
      E <- "T"
    }
    } else if (phi < 180){
      if ((theta.1 +theta.2 +  theta.3 + theta.4 + theta.5 + theta.6 + theta.7) < (phi + 360) & phi
    < theta.1){
        d <- (yhi-y)/cos(deg2rad(psi)) - (a0*time1)/2
        t <- d/Vmag
        E <- "T"
      }
    }
  return(c(time1+t, E))
}



advance <- function(state0, time1, time2){
  ## given state0 at time 0 and time1, compute state2 this sphere
  ## would have at time1 ignoring possible collusions with the other
  ## spheres or boundary corissings on the interval [time0,time1]
  ## state <= c(position.x, position,y, velocity.x, velocity.y)
  state1 <- rep(NA, length=4)
  diff.time <- time1 - time2
  diff.x <- diff.time*state0[3] #x component of velocity
  diff.y <- diff.time*state0[4] #y component of velocity
  state1[1] <- state0[1] + diff.x #new x of location
  state1[2] <- state0[2] + diff.y #new y of location
  state1[3:4] <- state0[3:4] #equalise velocities of state0 and state1
  return(state1)
}
```

```r


jump.spheres <- function(state1, state2){
  ## given state1 and state2 of colliding spheres 1 and 2,
  ## return new_state1 and new_state2 of these spheres immediately
  ## after the interaction
  #conservatino of momementum -> but massess are always equal
  # sum of momementum on y before collision equal to y momentum after collision
  diff.x <- max(state1[1], state2[1]) - min(state1[1], state2[1])
  diff.y <- max(state1[2], state2[2]) - min(state1[2], state2[2])
  joining.vect <- c(diff.x,diff.y)
  distance <- sqrt(diff.x^2 + diff.y^2)
  v.1.old <- state1[3:4]
  v.2.old <- state2[3:4]
  v.1.norm <- (dot(joining.vect,v.1.old)/(distance^2))*joining.vect
  v.1.tang <- v.1.old - v.1.norm
  v.2.norm <- (dot(joining.vect,v.2.old)/(distance^2))*joining.vect
  v.2.tang <- v.2.old - v.2.norm
  v.1.new <- round((v.1.tang + v.2.norm), digits=3)
  v.2.new <- round((v.2.tang + v.1.norm), digits=3)
  new.state1 <- c(state1[1:2], v.1.new)
  new.state2 <- c(state2[1:2], v.2.new)
  return(c(new.state1,new.state2))
}

jump.boundaries <- function(state1, k.star, a0, event, R, i.star, K){
  ##
  boundaries <- K[1,] #c(xlo, xhi, ylo, yhi)
  xlo <- boundaries[1]
  xhi <- boundaries[2]
  ylo <- boundaries[3]
  yhi <- boundaries[4]
  x <- state1[1]
  y <- state1[2]
  Vx <- state1[3]
  Vy <- state1[4]
  if (k.star == "L"){
    new.state1 <- c(xhi- (a0*R), y, Vx, Vy)
  } else if (k.star== "B"){
    new.state1 <- c(x, yhi-(a0*R), Vx, Vy)
  } else if (k.star == "R"){
    new.state1 <- c(x+(a0*R), y, Vx, Vy)
  } else if (k.star == "T"){
    new.state1 <- c(x, y + (a0*R), Vx, Vy)
  }
  sphere.present = check.sphere.presence(new.state1, event, a0, R)
  if ( sphere.present == TRUE){
    new.state1 <- bounce.off(state1, k.star)
    return(new.state1)
  } else if (sphere.present == FALSE){
    return(new.state1)
  }
}


check.sphere.presence <- function(new.state1, event, a0, R){
  distances <- rep(0, times = dim(event$state$position$xy)[1] )
  past.x <- event[['state']][['position']][[old[i.star]]][,1]
  past.y <- event[['state']][['position']][[old[i.star]]][,2]
  diff.x <- abs(new.state1[1] - past.x )
  diff.y <- abs(new.state1[2] - past.y)
  distances <- sqrt(diff.x^2 + diff.y^2)
  if (any(distances < a0*R)){
    j.star <- which(distances < a0*R)
    return(c(TRUE))
  } else {return(FALSE)}
}


bounce.off <- function(state1, k.star){
  Vx <- state1[3]
  Vy <- state1[4]
  if (k.star == "T"){
    new.state1 <- c(state1[1:2],Vx, -Vy )
  } else if (k.star == "R"){
    new.state1 <- c(state1[1:2], -Vx, Vy )
  } else if (k.star =="B"){
    new.state1 <- c(state1[1:2], Vx, -Vy )
  } else if (k.star == "L"){
    new.state1 <- c(state1[1:2], Vx, -Vy )
  }
  return(new.state1)
}





##########################
```