SpatialTests

Paul M

11/17/2020

Playing around with cluster detection on networks

Load some packages

```r
library("RColorBrewer")
library("png")
library("ggraph")
```

```
## Loading required package: ggplot2
```

```r
library("networkD3")
library("animation")
library("maps")
library("ggplot2")
library("geosphere")
library("RColorBrewer")
```

Globals…

```r
set.seed(593)
SizeOfOurNetwork <- 40
ProbOfAVertex <- 0.075
```

Build a random network

```r
RandomEdges <- function (x){
  x <- runif(1)<ProbOfAVertex
  return (x)
}

RemoveSelfLoops <- function(x){
  if (x[1]==x[2])  x[3] <- 0
  return (x)
}

BuildRandomNetwork <- function(NetSize){
  MyNodes <- seq(1,SizeOfOurNetwork)

  x <- seq(1, SizeOfOurNetwork)
  y <- x

  MyEdges <- expand.grid(x = x, y = y)
  # make it undirected
  MyEdges2 <- MyEdges[MyEdges[,1]<MyEdges[,2],]

  EdgePresent <- rep(0,length(MyEdges2[,1]))
  EdgePresent <- apply(as.matrix(EdgePresent),MARGIN=1,FUN=RandomEdges)

  MyEdges3 <- cbind(MyEdges2,EdgePresent)

  MyEdges4 <- MyEdges3[MyEdges3[,3]==1,]

  ThisNetwork <- graph_from_data_frame(d=as.data.frame(MyEdges4),vertices=as.data.frame(MyNodes), direc

  return (ThisNetwork)
}

# make it undirected
#MyEdges2 <- MyEdges[MyEdges[,1]<MyEdges[,2],]
# remove self-self edges
#MyEdges3 <- t(apply(MyEdges,MARGIN=1,FUN=RemoveSelfLoops))
# remove missing edges
```

Plot the network using the igraph library

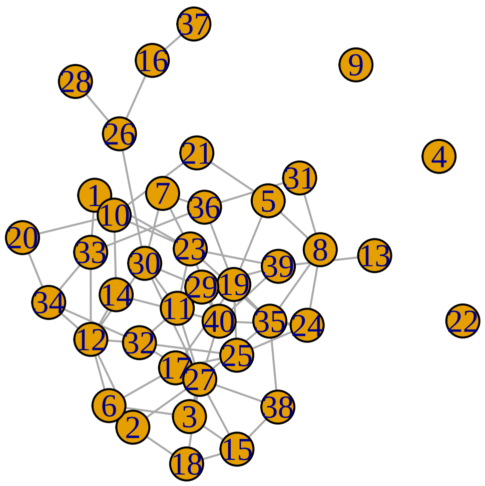
```
library("igraph")
```

```
##
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

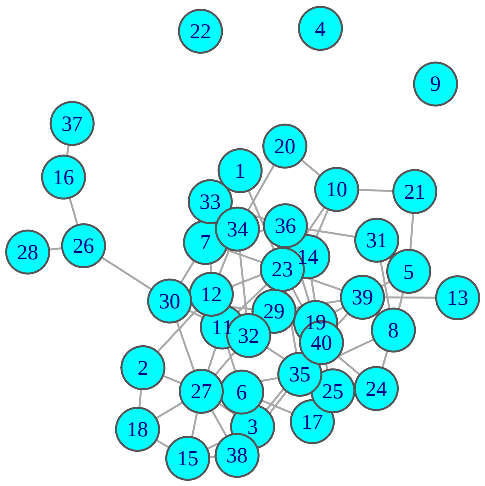
```
#net <- graph_from_data_frame(d=as.data.frame(MyEdges4),vertices=as.data.frame(MyNodes), directed=F)

net <- BuildRandomNetwork(SizeOfOurNetwork)
class(net)
```

```
## [1] "igraph"
```

```
#net

plot(net)
```



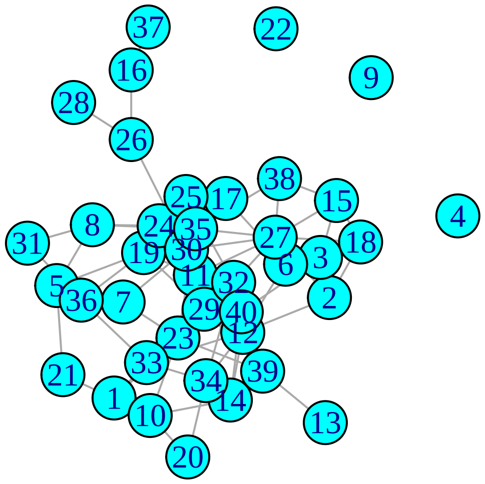
```
# there are all sorts of pretty options
plot(net, edge.arrow.size=.2, edge.curved=0,
     vertex.color="cyan", vertex.frame.color="#555555",
     vertex.label.cex=.7,vertex.size=20)
```



```
# Compute node degrees (#links) and use that to set node size:
#deg <- degree(net)
sum(E(net)==1)
```

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

```
V(net)$size <- 20 #deg*3
l <- layout_with_fr(net)
plot(net, layout=l, vertex.color="cyan",)
```

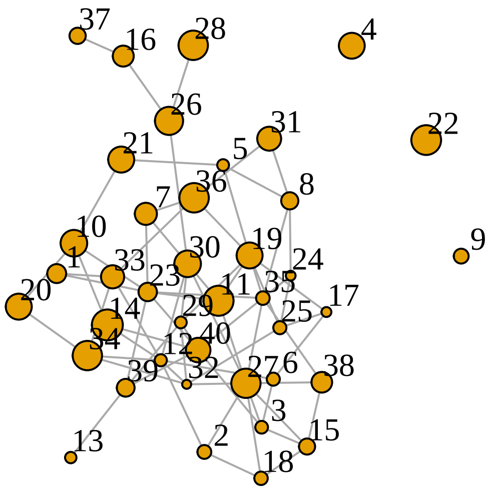


Edges, vertices and entire mx can be accessed as follows: (nice tutorial at https://kateto.net/wp-content/uploads/2016/01/NetSciX_2016_Workshop.pdf)

```
E(net)
V(net)
net[]
```

Add attributes to the network, vertices, or edges as follows

```
V(net)$MyAttribute <- runif(length(V(net)),0,1)
#vertex_attr(net)
plot(net, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
vertex.size=4+10*V(net)$MyAttribute)
```



A network diameter is the longest geodesic distance (length of the shortest path between two nodes) in the network. In igraph, diameter() returns the distance, while get_diameter() returns the nodes along the first found path of that distance. Note that edge weights are used by default, unless set to NA.

```
diameter(net, directed=F, weights=NA)
```

```
## [1] 7
```

```
diameter(net, directed=F)
```

```
## [1] 7
```

```
diam <- get_diameter(net, directed=F)
diam
```

```
## + 8/40 vertices, named, from 1c0d607:
## [1] 21 5  19 11 30 26 16 37
```

```
#plot(net, layout=l, vertex.color="cyan",)
```

Distances between nodes:

```
#distances(net,v=V(net)[1],to=V(net)[2])
#distances(net,v=(V(net)==9),to=(V(net)==18))
#distances(net,v=(V(net)==2),to=(V(net)==17))
#distances(net,v=V(net),to=V(net))

DM <- distances(net,v=V(net),to=V(net))
```

Calcluating Moran's (global) I

```
MoransI <- function(DistanceMx, NodeAttributes){
  NoOfNodes <- length(DistanceMx[1,])
  WeightSum <- 0
  MoranSum <- 0
  DenomSum <- 0
  AttributeMean <- mean(NodeAttributes)
  #cat("\nAttribute mean= ",AttributeMean)
  for (i in 1:NoOfNodes){
    DenomSum <- DenomSum + (NodeAttributes[i]-AttributeMean) * (NodeAttributes[i]-AttributeMean)
    for (j in 1:NoOfNodes){
      if ( i != j){
        ThisDist <- DistanceMx[i,j]
        if (ThisDist == 1) # neighbors only
        {
          WeightSum <- WeightSum + ThisDist
          MoranSum <- ThisDist * (NodeAttributes[i]-AttributeMean) * (NodeAttributes[j]-AttributeMean)
        }
        # V(net)$MyAttribute
      }
    }
  }
  MoransI <- NoOfNodes * MoranSum / ( DenomSum * WeightSum)
}

cat("\nMorans-I: ",MoransI(DM,V(net)$MyAttribute),"    expectation= ",-1/(length(V(net)$MyAttribute)-1)
```

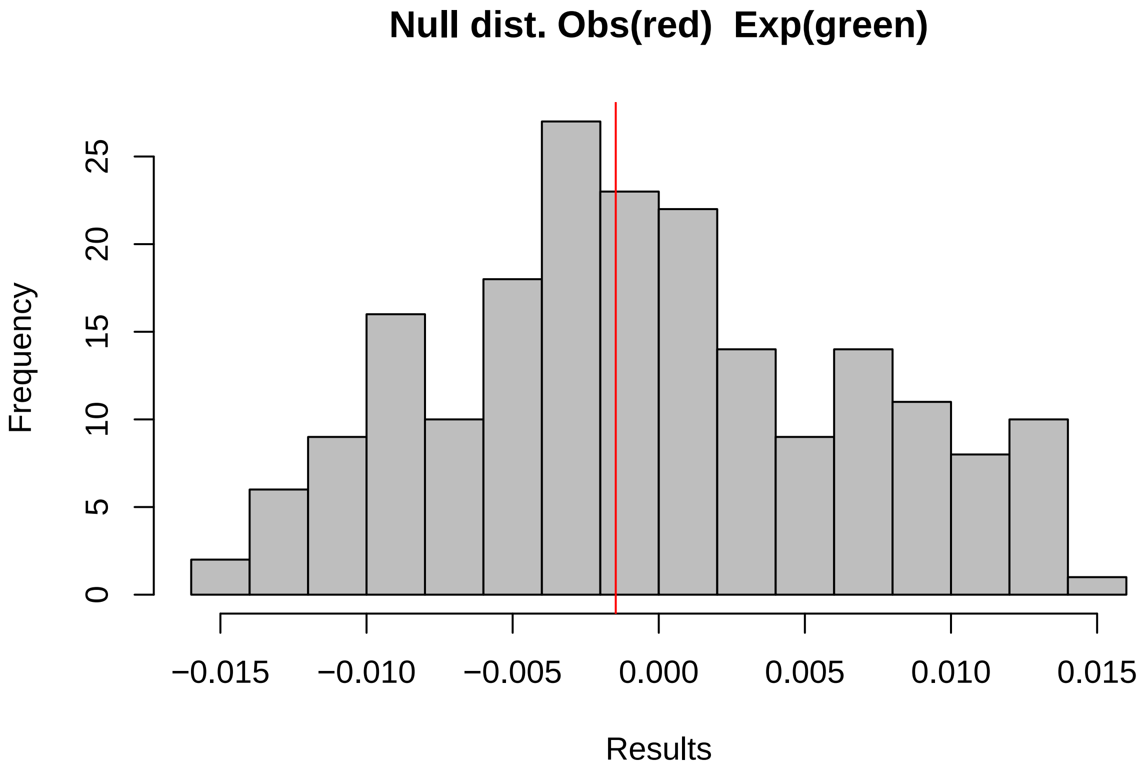
```
##
## Morans-I:  -0.001472167     expectation=  -0.02564103
```

Test it out…

```
for (k in 1:10){
  net <- BuildRandomNetwork(SizeOfOurNetwork)
  for (i in 1:length(V(net)))
    V(net)$MyAttribute[i] <- runif(1)
  vertex_attr(net)
  DM <- distances(net,v=V(net),to=V(net))
  plot(net, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
  vertex.size=4+10*V(net)$MyAttribute)

  cat("\nMorans-I: ",MoransI(DM,V(net)$MyAttribute),"    expectation= ",-1/(length(V(net)$MyAttribute)-

}
```

Assessing null distribution for Moran-s I via permutation tests

```
PermutationTest <- function(ntwk,HowManyPermutations,WhichMeasure)
{
  Results <- rep(-9,HowManyPermutations)
  for (i in 1:HowManyPermutations){
    V(ntwk)$MyAttribute <- sample(V(ntwk)$MyAttribute,size=length(V(ntwk)$MyAttribute),replace=FALSE)

    if (WhichMeasure == 1)  # global Moran's-I
    {
      Results[i] <-  MoransI(DM,V(ntwk)$MyAttribute)
    }else{
      cat("\nUndefined measure for permuation test. Exit.")
      break;
    }
  }
  return (Results)
}

Results <-PermutationTest(net,200,1)
hist(Results,breaks=20,col="grey",main="Null dist. Obs(red)  Exp(green)")
abline(v=MoransI(DM,V(net)$MyAttribute),col="red")
abline(v=-1/(length(V(net)$MyAttribute)-1),col="seagreen")  # the expected value for Morans-I
```



Find immediate neighbors of a vertex

```
FindNeighbors <- function(ntwk,focnode)+{
  nbrs<-NULL
  for (i in 1:length(V(ntwk))){
    if ((distances(ntwk,v=(V(ntwk)==focnode),to=(V(ntwk)==i))==1) & (i!=focnode)){
      nbrs <- c(nbrs,i)
    }
  }
  return(nbrs)
}
(FindNeighbors(net,5))
```

```
## [1]  8 19 21
```

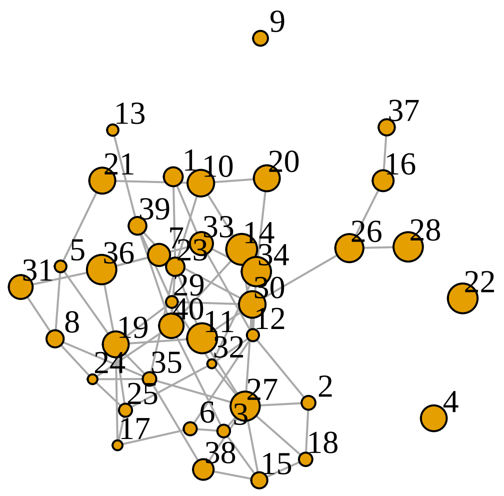
```
(FindNeighbors(net,1))
```

```
## [1] 23 33
```

Smoothing the labels to make them correlated. Here we use a simple proof of principle scheme in ewhich we generate the labels independently and then jsmooth them by taking a weight average f each label and the label of its neighboriong vertices. Later on, we will try something more formal.

```
Smoother <- function(ntwk,weight){
  NewLabels<-rep(0,length(V(ntwk)))
  for (i in 1:length(V(ntwk))){
    naybrs <- FindNeighbors(ntwk,i)
    NewL <- V(ntwk)$MyAttribute[i]
    for (j in naybrs){
      NewL <- NewL + weight * V(ntwk)$MyAttribute[j]
    }
    NewLabels[i]= NewL/(1+weight*length(naybrs))
  }
  return (NewLabels)
}

plot(net, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
vertex.size=4+10*V(net)$MyAttribute, main="Uniform attributes")
```

Uniform attributes



```
(V(net)$MyAttribute)
```

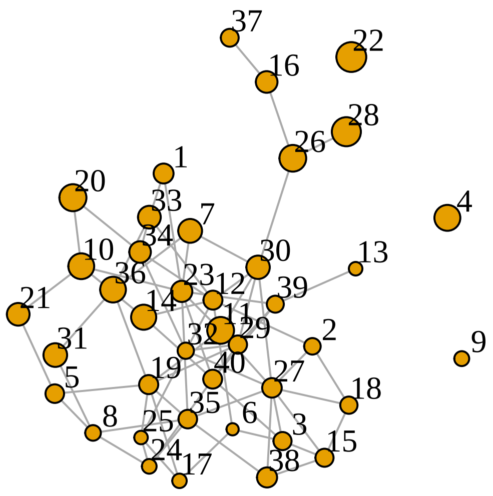
```
##  [1] 0.455997068 0.226238777 0.172442198 0.768400676 0.132057823 0.190961041
##  [7] 0.599613849 0.375970033 0.271802699 0.803639327 0.956975113 0.148821557
## [13] 0.113118909 0.997517304 0.327354304 0.547106159 0.048619037 0.206662221
## [19] 0.773564051 0.774183847 0.775094645 0.945890882 0.432763028 0.040646400
## [25] 0.192295964 0.871435822 0.921677486 0.929114026 0.134019463 0.802796356
## [31] 0.680940919 0.006164699 0.645605933 0.926459406 0.213765195 0.930609303
## [37] 0.333754934 0.532404060 0.403454849 0.707701667
```

```
V(net)$MyAttribute <- Smoother(net,0.5)
(V(net)$MyAttribute)
```

```
##  [1] 0.4975908 0.3459278 0.4154298 0.7684007 0.4377489 0.1503610 0.6730793
##  [8] 0.3032251 0.2718027 0.7644729 0.8074585 0.4548207 0.2098976 0.7310394
## [15] 0.4146491 0.5748508 0.2508118 0.3777190 0.4617189 0.8196166 0.6214716
## [22] 0.9458909 0.5562970 0.2618376 0.2089310 0.8043776 0.4807416 0.9098880
## [29] 0.4231029 0.6547670 0.6671153 0.3336576 0.6255165 0.5712825 0.4380694
## [36] 0.7601572 0.4048720 0.5055210 0.3657521 0.4495308
```

```
plot(net, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
vertex.size=4+10*V(net)$MyAttribute,main="smoothed attributes")
```

smoothed attributes



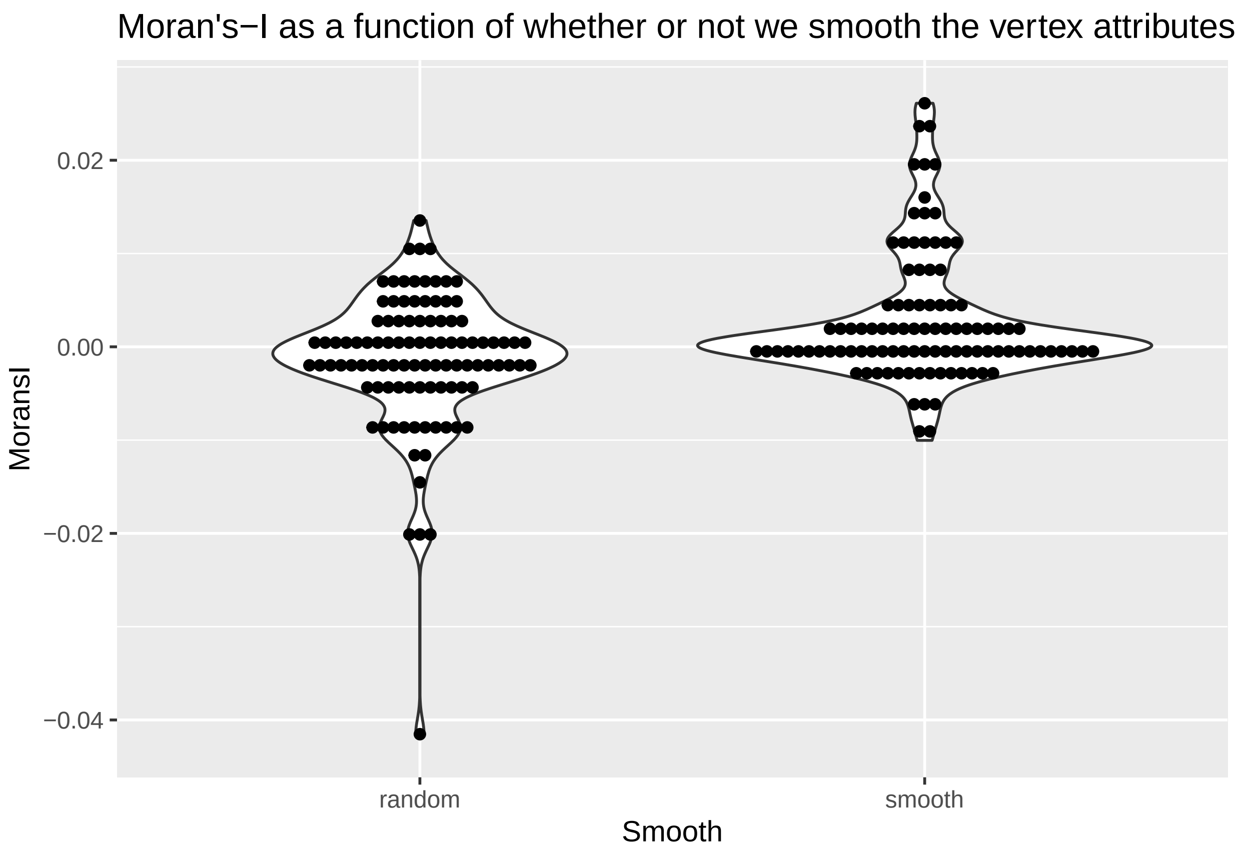
So let's compare Moran's-I for random labels and smoother labels…

```r
NTimes <- 100
RandomMoransI <- rep(0,NTimes)
SmoothMoransI <- rep(0,NTimes)
for (i in 1:NTimes){
  # generate a random graph with random labels
  net <- BuildRandomNetwork(SizeOfOurNetwork)
  #for (j in 1:length(V(net)))
    V(net)$MyAttribute <- rnorm(length(V(net)),0,1)

  # calculate distances between nodes
  DM <- distances(net,v=V(net),to=V(net))

  # calculate Moran's-I for this graph
  RandomMoransI[i] <- MoransI(DM,V(net)$MyAttribute)

  # now smooth it and recalculate Moran's-I
  V(net)$MyAttribute <- Smoother(net,1)
  SmoothMoransI[i] <- MoransI(DM,V(net)$MyAttribute)
}

# compare via a violin plot
RandomMoransI <- cbind(rep("random",NTimes),RandomMoransI)
SmoothMoransI <- cbind(rep("smooth",NTimes),SmoothMoransI)
I <- rbind(RandomMoransI,SmoothMoransI)
dfsm <- data.frame( "Smooth" = I[,1], "MoransI" = as.numeric(I[,2]))

hw_p <- ggplot(dfsm, aes(x = Smooth, y = MoransI)) +
    geom_violin() +
    geom_dotplot(binaxis='y', stackdir='center', dotsize=0.5, bin.width=60) +
    ggtitle("Moran's-I as a function of whether or not we smooth the vertex attributes")
```

```
## Warning: Ignoring unknown parameters: bin.width
```

```
(hw_p)
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



And now for a more formal addition of spatial correlation, using correlated normals, where the degree of correlation depends upon the network structure (as originally proposed by George VY). So, we generate data $y = (I_n - \rho W)^{-1} \times \epsilon$, where $\epsilon \sim MVN(0, I_n)$. Since $\epsilon \sim MVN(0, I_n)$ the independent version has attrributes with $Normal(0, 1)$ distribution.

```
rho <- 0.5  # The degree of correlation
SARsmoother <- function(ntwk,rho)
{
#rho <- .5 # Spatial autocorrelation
# form edge matrix
  W <- as_adjacency_matrix(net, type = c("both"), names=FALSE, sparse=FALSE)
  n <- length(W[1,])
  #diag(W) <- 0
  W2 <- W/rowSums(W, na.rm = TRUE)

  # if nodes are disconnected, we will get NaNs, so set those to 0.
  W2[!is.finite(W2)] <- 0

  # generate the attributes
  y <- solve(diag(n) - rho * W2) %*% rnorm(n)


  # Check whether we generate spatial autocorrelation?
  #library(ape)
```

```r
  #sc <- Moran.I(as.vector(y), W)
  #cat("\np=",sc$p.value)

  return (y)
}

attribs <- SARsmoother(net,0.5)
net$MyAttributes <- SARsmoother(net,0.5)
```

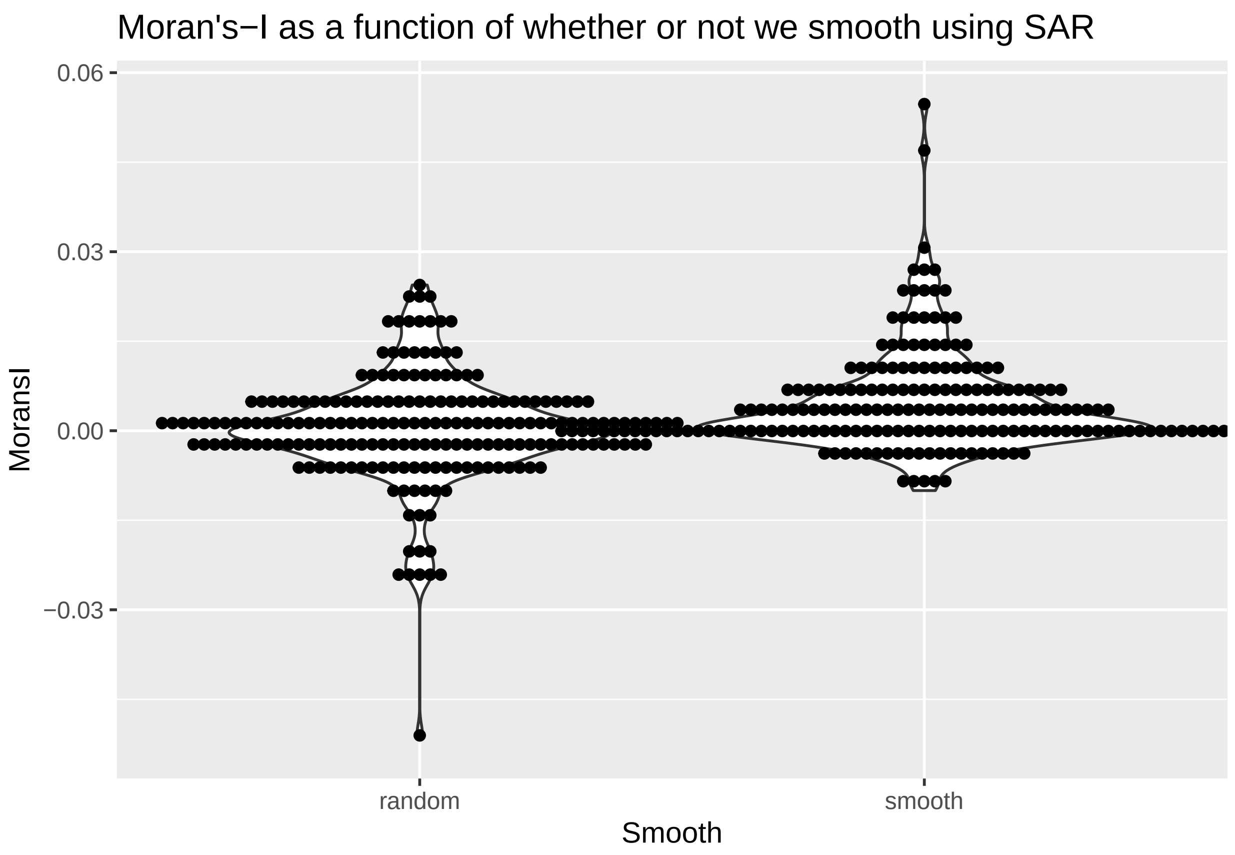
Now write some tests for the above, comparing them to models in which the attrivbutes are Normal(0,1).

```r
NT <- 200
RandomMoransI <- rep(0,NT)
SARsmoothMoransI <- rep(0,NT)
set.seed(49)
for (i in 1:NT){
  # generate a random graph with random labels
  net <- BuildRandomNetwork(SizeOfOurNetwork)

  # calculate distances between nodes
  DM <- distances(net,v=V(net),to=V(net))

  # generate indepen dent vertex attributes from a MVNormal(0,1)
  V(net)$MyAttribute <- rnorm(length(V(net)),0,1)
  #V(net)$MyAttribute <- SARsmoother(net,0)
  # calculate Moran's-I for this graph
  RandomMoransI[i] <- MoransI(DM,V(net)$MyAttribute)

  # now generate spatial correlated vertex labels and recalculate Moran's-I
  V(net)$MyAttribute <- SARsmoother(net,0.8)
  SARsmoothMoransI[i] <- MoransI(DM,V(net)$MyAttribute)
}

# compare via a violin plot
RandomMoransI <- cbind(rep("random",NTimes),RandomMoransI)
SARsmoothMoransI <- cbind(rep("smooth",NTimes),SARsmoothMoransI)
I <- rbind(RandomMoransI,SARsmoothMoransI)
dfsm <- data.frame( "Smooth" = I[,1], "MoransI" = as.numeric(I[,2]))
#df$dataN <- as.factor(df$dataN)

hw_p <- ggplot(dfsm, aes(x = Smooth, y = MoransI)) +
    geom_violin() +
    geom_dotplot(binaxis='y', stackdir='center', dotsize=0.5, bin.width=60) +
    ggtitle("Moran's-I as a function of whether or not we smooth using SAR")
```

```
## Warning: Ignoring unknown parameters: bin.width
```

```r
(hw_p)
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```

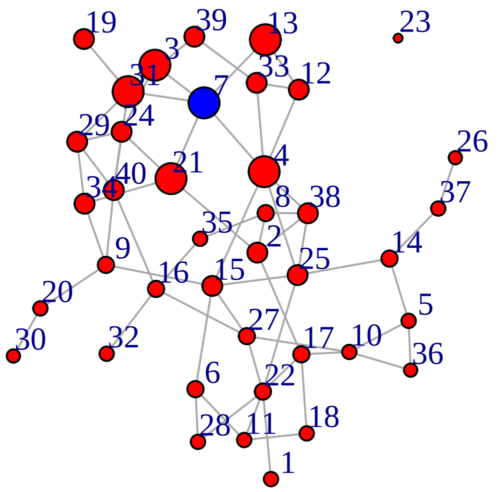


Assigning spatially correlated node labels

```
SpatiallyCorrelatedLabels1 <- function(ntwk,CentralNode){

  # label it and all its neighbors 1, and everything else 0.
  V(ntwk)$MyAttribute <- 0
  V(ntwk)$MyAttribute[CentralNode] <- 1
  for (i in 1:SizeOfOurNetwork){
    #if (distances(ntwk,CentralNode,i) == 1){
    #  V(ntwk)$MyAttribute[i] <- 1
    #}
    if (i != CentralNode){
      V(ntwk)$MyAttribute[i] <- 1/distances(ntwk,CentralNode,i)
    }
  }
  return (ntwk)
}

  # Pick a focal node at random
  FocalNode <- sample(1:length(net),1)
  cat("\nFocalNode: ",FocalNode)
```

```
##
## FocalNode:  7
```

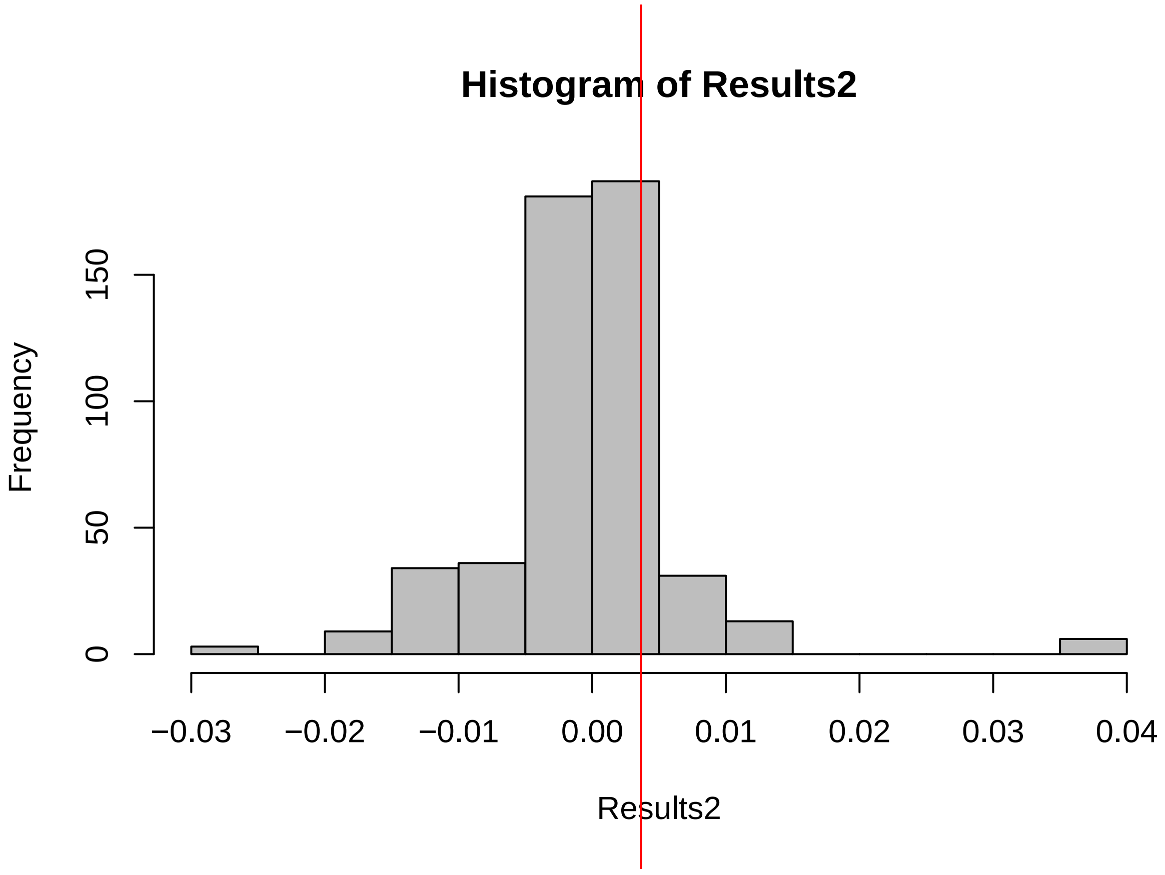
```
  newnet <- SpatiallyCorrelatedLabels1(net,FocalNode)
  plot(newnet, edge.arrow.size=.5, vertex.color=ifelse(V(newnet)==FocalNode,"blue","red"), vertex.label
  vertex.size=4+10*V(newnet)$MyAttribute)
```



```
  cat("\nMorans-I: ",MoransI(distances(newnet,v=V(newnet),to=V(newnet)),V(newnet)$MyAttribute),"     expe
```

```
##
## Morans-I:  0.003650274     expectation=  -0.02564103
```

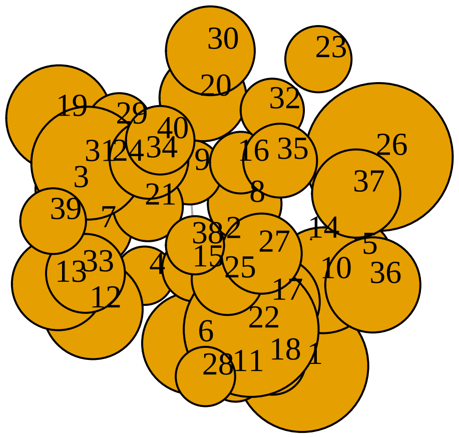
```
  # Does it look significant?
  Results2 <-PermutationTest(newnet,500,1)
hist(Results2,breaks=20,col="grey")
abline(v=MoransI(DM,V(newnet)$MyAttribute),col="red")
```



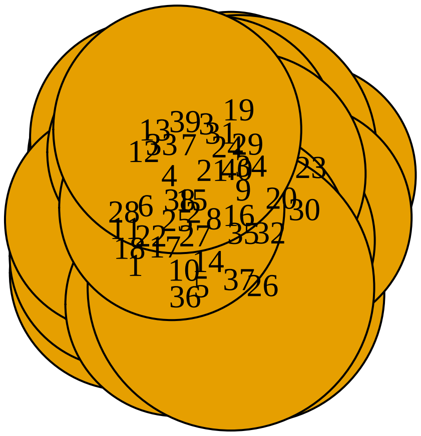
Local Moran's-I (LISA). We suppose that each node has some (binary or continuous) annotationxi, and standardize those values by setting $z_i = x_i - \bar{x}$. The LISA measure of local clustering for each node, i, is then defined as $I_i = z_i \sum_{j \in J_i} w_{ij} z_j$.

Here, J_i is the set of neighbors of node i (although the definition can be generalized in an obvious way), w_{ij} is a weight that is used to characterize the distance between nodes. For example, the weight might measure the number of edges on the shortest path between nodes i and j.

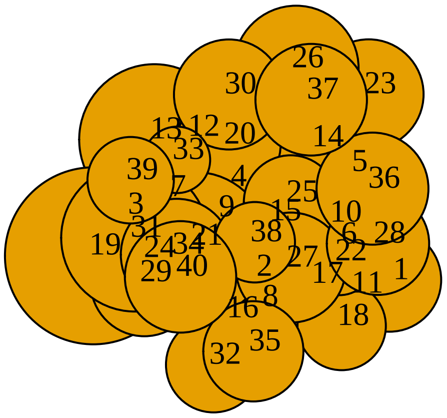
```
MyLISA <- function(ntwk){
  V(ntwk)$LISAstat <- rep(-9,length(V(ntwk)))
  V(ntwk)$StandardizedAttribute <- rep(-9,length(V(ntwk)))
  AttributeMean <- mean(V(ntwk)$MyAttribute)
  # Standardize node labels
  V(ntwk)$StandardizedAttribute <- V(ntwk)$MyAttribute - AttributeMean
  for (i in 1:length(V(ntwk))){
    L <- 0
    for (j in 1:length(V(ntwk))){
      if (distances(ntwk,v=V(ntwk)[i],to=V(ntwk)[j]) == 1){
        # they are neighbors
        L <- L + V(ntwk)$StandardizedAttribute[j]
      }
    }
    L <- L * V(ntwk)$StandardizedAttribute[i]
    V(ntwk)$LISAstat[i] <- L
  }
  return (ntwk)
}
LISAnet <- MyLISA(net)
plot(LISAnet, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
vertex.size=10*(0.01-min(V(LISAnet)$LISAstat)+V(LISAnet)$LISAstat))
```



```
LISAval <- V(LISAnet)$LISAstat
index <- seq(from=1, to=length(LISAval))
dataN <- rep(1,length(LISAval))
z1 <- cbind(dataN,index,LISAval)

# permute the attributes and repeat
PermNtwk <- net
V(PermNtwk)$MyAttribute <- sample(V(PermNtwk)$MyAttribute,size=length(V(PermNtwk)$MyAttribute),replace=
LISAnetPerm <- MyLISA(PermNtwk)
plot(LISAnetPerm, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
vertex.size=10*(0.01-min(V(LISAnetPerm)$LISAstat)+V(LISAnetPerm)$LISAstat))
```



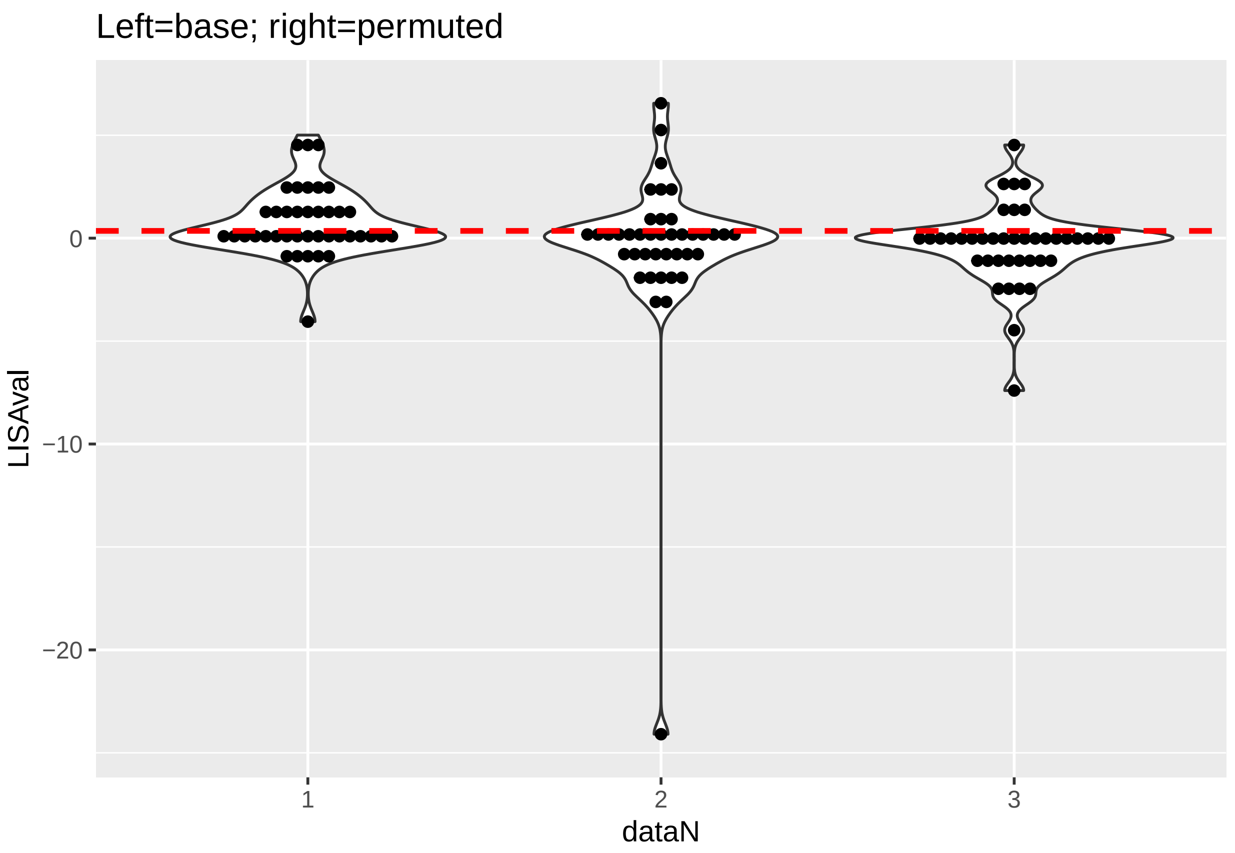
```
LISAval<- V(LISAnetPerm)$LISAstat
index <- seq(from=1, to=length(LISAval))
dataN <- rep(2,length(LISAval))
z2 <- cbind(dataN,index,LISAval)

PermNtwk2 <- net
V(PermNtwk2)$MyAttribute <- sample(V(PermNtwk2)$MyAttribute,size=length(V(PermNtwk2)$MyAttribute),replac
LISAnetPerm <- MyLISA(PermNtwk2)
plot(LISAnetPerm, edge.arrow.size=.5, vertex.label.color="black", vertex.label.dist=1.5,
vertex.size=10*(0.01-min(V(LISAnetPerm)$LISAstat)+V(LISAnetPerm)$LISAstat))
```



```
LISAval <- V(LISAnetPerm)$LISAstat
index <- seq(from=1, to=length(LISAval))
dataN <- rep(3,length(LISAval))
z3 <- cbind(dataN,index,LISAval)


## violin plot
zz <- rbind(z1,z2,z3)
df <- as.data.frame(zz)
df$dataN <- as.factor(df$dataN)

hw_p <- ggplot(df, aes(x = dataN, y = LISAval))
hw_p +
  geom_violin() +
  geom_dotplot(binaxis='y', stackdir='center', dotsize=0.5) +
  ggtitle("Left=base; right=permuted") +
```

```r
  geom_hline(yintercept=V(LISAnet)$LISAstat[FocalNode], linetype=2, color="red", size=1)
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



```r
  #scale_fill_brewer(palette="Dark2")
#violinplot(data=df$LISAval)
```

Read sbml files. . .

```r
# if (!requireNamespace("BiocManager", quietly = TRUE))
#     install.packages("BiocManager")
# BiocManager::install("SBMLR")
library(SBMLR)    # from Bioconductor
#readSBML("Apoptosis_signaling_pathway.xml")
arach=readSBML("2-arachidonoylglycerol_biosynthesis.xml")
#Apop=readSBML(file.path(system.file(package="SBMLR"), "Apoptosis_signaling_pathway.xml"))

if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install(version = "3.12")
#source("https://bioconductor.org/biocLite.R")
##biocLite("rsbml")
#BiocManager::install("rsbml")
#library(rsbml)
#file <- system.file("sbml", "GlycolysisLayout.xml", package = "rsbml")
#  doc <- rsbml_read(file)

#install.packages("remotes")
```

```r
#remotes::install_github("ahmohamed/NetPathMiner")
# docs at https://rdrr.io/github/ahmohamed/NetPathMiner/man/SBML2igraph.html
library(NetPathMiner)
# the following is supposed to read an SBML object and parse it as an igraph
SBML2igraph("2-arachidonoylglycerol_biosynthesis.xml", parse.as = c("metabolic", "signaling"),
  miriam.attr = "all", gene.attr, expand.complexes, verbose = TRUE)
```

- Random Other things

Node degrees The function degree() has a mode of in for in-degree, out for out-degree, and all or total for total degree.

```r
deg <- degree(net, mode="all")
plot(net, vertex.size=deg*2)
hist(deg, breaks=1:vcount(net)-1, main="Histogram of node degree")
```

Erdos-Renyi random graph model (???n??? is number of nodes, ???m??? is the number of edges).

```r
er <- sample_gnm(n=100, m=200)
plot(er, vertex.size=6, vertex.label=NA)
```

Barabasi-Albert preferential attachment model for scale-free graphs (n is number of nodes, power is the power of attachment (1 is linear); m is the number of edges added on each time step)

```r
ba <- sample_pa(n=100, power=1, m=1, directed=F)
plot(ba, vertex.size=6, vertex.label=NA)
```

Rewiring a graph each_edge() is a rewiring method that changes the edge endpoints uniformly randomly with a probability prob.

```r
set.seed((876))
ba.rewired <- rewire(ba, each_edge(prob=0.1))
lay <- layout_with_fr(ba.rewired)
plot(ba.rewired, vertex.size=10, vertex.label=NA, vertex.color="blue")
plot(ba.rewired, vertex.size=10, vertex.label=NA, vertex.color="blue",layout=lay)
```