---

title: 'ESM 244 Lab 8 Part 1: Spatial point pattern analysis'

author: "Allison Horst, Casey O'Hara"

date: "2/24/2022"

output: html\_document

---

See: - CRS & proj4 components breakdown: https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/reproject-vector-data/

```{r setup, include=TRUE}

knitr::opts\_chunk$set(echo = TRUE)

library(spatstat)

library(maptools)

# library(sp)

library(raster) ### BEFORE tidyverse! b/c select()

library(tidyverse)

library(here)

library(sf)

library(tmap)

```

This is an example of point pattern analysis with a density plot, and the G- & L- function (distance methods) to compare our observed points with simulated complete spatial randomness.

```{r}

# Read in the tree vole data

voles <- read\_sf(dsn = here("redtreevoledata"),

layer = "ds033") %>%

janitor::clean\_names() %>%

select(county) %>% # Only select the county attribute

filter(county == "HUM") %>% # Only keep observations in Humboldt County

st\_transform(crs = 32610) # Update CRS to UTM zone 10 N

# Plot it (exploratory)

plot(voles)

# Get Humboldt County outline

humboldt <- read\_sf(dsn = here("redtreevoledata"),

layer = "california\_county\_shape\_file") %>%

janitor::clean\_names() %>%

filter(name == "Humboldt") %>% # Isolate Humboldt County

select(name) %>% # Only keep one attribute (name) to simplify

st\_set\_crs(4326) %>%

st\_transform(crs = 32610)

# plot them together

ggplot() +

geom\_sf(data = humboldt,

color = "darkorchid",

fill = "darkorchid4",

size = 1) +

geom\_sf(data = voles,

color = "orange",

alpha = 0.7,

size = 2) +

theme\_minimal()

```

These need to be combined into spatial point pattern data (points + window combo), and for point pattern analysis this \*\*requires a 2D projection\*\* (in this case, UTM), which is why we set the CRS to 32610 above. This looks quite a bit different from what we've done so far - it uses functions in `spatstat` to create point patterns that play nicely with other functions for data viz & point pattern analysis.

```{r}

voles\_sp <- as(voles,"Spatial") # Convert to object 'Spatial'

voles\_ppp <- as(voles\_sp, "ppp") # Convert to spatial point pattern

humboldt\_sp <- as(humboldt, "Spatial") # Convert to object 'Spatial'

humboldt\_win <- as(humboldt\_sp, "owin") # Convert to spatial point pattern from spatstat

# Combine as a point pattern object (points + window):

voles\_full <- ppp(voles\_ppp$x, voles\_ppp$y, window = humboldt\_win)

plot(voles\_full) # Illegal point (outside window) shows up as the plus sign

```

## Make a kernel density plot:

### Density

Run to see vole "hotspots" by kernel density, then see what happens when you change sigma here!

```{r}

voles\_density <- density(voles\_full, sigma = 5000)

plot(voles\_density)

```

Pretty clear that there are "hotspots" where voles are observed - both in the originally plotted data and in the density plot. How can we compare this to complete spatial randomness?

```{r}

# Can you start viewing this in tmap? Yes, rasterize it:

vole\_raster <- raster(voles\_density)

crs(vole\_raster) <- crs(voles)

# Then plot:

tmap\_mode("view")

tm\_shape(vole\_raster) +

tm\_raster(midpoint = NA,

palette = "Reds",

legend.show = FALSE)

```

## Nearest neighbor (G-function)

In last week's lecture, we learned about distance methods to compare our point pattern to a scenario of complete spatial randomness. Here, we'll use both the G- and L-functions (L function is the K-function, standardized...interpretation is the same) to compare our observed point pattern to a simulated CSR scenario, to help us determine if it is \*more clustered\* or \*more uniform\* than CSR.

What is going on in this code?

- `r`: a sequence of distances (in the spatial units of the data) over which we'll calculate the proportion of points with nearest neighbor within that range

- `gfunction`: This uses the `envelope()` function within which we run simulations for CSR, \*and\* calculate the G-function value at distances \*r\* for each simulation. So this will calculate the G-function for \*our\* actual data, and also for simulations of CSR if we had the same number of observations in the window but they were independent. The `nsim = 100` here means there will be 100 simulations of CSR. The `nrank = 2` means that the second highest and second lowest values from simulations are shown as the "hi" and "lo" value envelopes, with the "theo" being the "theoretical value of the summary function under CSR (Complete Spatial Randomness, a uniform Poisson point process) if the simulations were generated according to CSR." So we're really comparing our "observed" data to the "theoretical CSR" here, and those "hi" and "lo" envelope bounds give us an idea of spread for the simulations.

```{r}

r\_vec <- seq(0, 10000, by = 100) # Make a sequence of distances over which you'll calculate G(r)

gfunction <- envelope(voles\_full, fun = Gest, r = r\_vec, nsim = 100, nrank = 2) # Calculate the actual and theoretical G(r) values, using 100 simulations of CRS for the "theoretical" outcome

gfunction # << Check the output of gfunction, then...

# Gather this to plot series in ggplot:

gfunction\_long <- gfunction %>%

as.data.frame() %>%

pivot\_longer(cols = obs:hi, names\_to = "model", values\_to = "g\_val")

# Then make a graph in ggplot:

ggplot(data = gfunction\_long, aes(x = r, y = g\_val, group = model)) +

geom\_line(aes(color = model))

```

This again confirms clustering - our data (model = obs) has a greater proportion of events with nearest neighbor at \*smaller distances\* compared to a theoretical CSR scenario (model = theo). But remember, the G-function only considers the single nearest neighbor.

Let's similarly look at the L-function (standardized K-function) which considers densities of observations within some distance R (expanding circles around each point) for comparison. This is using very similar code, but now the function is `Lest` for "L estimate", which calculates the density of events within growing circles around \*each point\*. That is much more intensive than just the single nearest neighbor, so I run `nsim = 10` here instead (you can do 100 or more again, you'll just notice that creating the simulations takes longer).

```{r}

r\_vec2 <- seq(0, 100000, by = 5000)

lfunction <- envelope(voles\_full, fun = Lest, r = r\_vec2, nsim = 10, rank = 2, global = TRUE)

# Gather this to plot series in ggplot:

lfunction\_long <- lfunction %>%

as.data.frame() %>%

pivot\_longer(cols = obs:hi, names\_to = "model", values\_to = "k\_val")

ggplot(data = lfunction\_long, aes(x = r, y = k\_val, group = model)) +

geom\_line(aes(color = model))

```

We again see that at lower distances, our data overall has a higher density of nearest neighbors compared to a simulated CSR scenario. Again, evidence of clustering.

## End Lab 8 part 1

---

title: 'Part 2: tsibble basics for time series exploration'

author: "Allison Horst"

output: html\_document

---

```{r setup, include=TRUE}

knitr::opts\_chunk$set(echo = TRUE, message = FALSE, warning = FALSE)

library(tidyverse)

library(here)

library(lubridate)

library(tsibble)

library(feasts)

library(slider)

```

## 1. Always look at your data

Toolik Station (LTER) meteorological data (Source: Source: Shaver, G. 2019. A multi-year DAILY file for the Toolik Field Station at Toolik Lake, AK starting 1988 to present. ver 4. Environmental Data Initiative.)

Notice that the date parsed (assumed class) as \*character.\* That limits the nice time series features we can use, so we'll quickly convert it into a tsibble (a time series data frame) so that we can use functions in `feasts` and `fable` to explore & analyze it.

### Read in data:

```{r}

toolik <- read\_csv(here("data","toolikweather.csv"))

```

### Convert the data frame to a tsibble

Go ahead and try plotting the data as imported.

```{r, eval = FALSE}

ggplot(data = toolik, aes(x = date, y = mean\_airtemp)) +

geom\_line()

# Booo we get a warning (only one observation per series)

```

Notice that it doesn't work - because R doesn't understand the date is a \*date\* until we tell it.

Let's go ahead and convert it to a tsibble using the `as\_tsibble()` function. First, we'll need to convert the date to a `date` class, \*then\* convert to a tsibble:

```{r}

toolik\_ts <- toolik %>%

mutate(date = lubridate::mdy(date)) %>%

as\_tsibble(key = NULL, index = date)

```

Now let's plot it:

```{r}

ggplot(data = toolik\_ts, aes(x = date, y = mean\_airtemp)) +

geom\_line() +

labs(x = "Date",

y = "Mean daily air temperature (Celsius)\n at Toolik Station")

```

We need to ask some big picture questions at this point, like:

- Does there appear to be an overall trend? No.

- Does there appear to be seasonality? Yes.

- Does there appear to be cyclicality? Unsure.

- Any notable outliers or additional patterns? No noted.

## 2. Use `index\_by()` to aggregate time series by increments

We will use `index\_by()` instead of `group\_by()` to do the trick. See `?index\_by()` to group by a time index, then `summarize()` to specify what to calulate & return for each interval.

```{r}

toolik\_month <- toolik\_ts %>%

index\_by(yr\_mo = ~yearmonth(.)) %>%

summarize(monthly\_mean\_temp = mean(mean\_airtemp, na.rm = TRUE))

```

Now let's take a look:

```{r}

ggplot(data = toolik\_month, aes(x = yr\_mo, y = monthly\_mean\_temp)) +

geom\_line()

# Or break it up by month:

toolik\_month %>%

ggplot(aes(x = year(yr\_mo), y = monthly\_mean\_temp)) +

geom\_line() +

facet\_wrap(~month(yr\_mo, label = TRUE)) +

labs(x = "Year",

y = "Annual mean air temperature (Celsius)",

title = "Toolik Station mean annual air temperature",

subtitle = "1988 - 2018",

caption = "Source: Shaver, G. 2019. A multi-year DAILY \nweather file for the Toolik Field Station at Toolik Lake, AK\n starting 1988 to present. ver 4. Environmental Data Initiative.")

```

Can you do other increments with `index\_by()`? Absolutely! \*\*See `?index\_by()` for grouping options!\*\*

Let's find the yearly average for 2000:

```{r}

toolik\_annual <- toolik\_ts %>%

index\_by(yearly = ~year(.)) %>%

summarize(annual\_airtemp = mean(mean\_airtemp, na.rm = TRUE))

ggplot(data = toolik\_annual, aes(x = yearly, y = annual\_airtemp)) +

geom\_line()

```

And how about a weekly average?

```{r}

toolik\_weekly <- toolik\_ts %>%

index\_by(weekly = ~yearweek(.)) %>%

summarize(weekly\_airtemp = mean(mean\_airtemp, na.rm = TRUE))

ggplot(data = toolik\_weekly, aes(x = weekly, y = weekly\_airtemp)) +

geom\_line()

```

## 3. Use `filter\_index()` to filter by date-times!

We can use `filter\_index()` specifically to help us filter data by time spans. See `?filter\_index()` for more information.

\*\*Example 1:\*\* Filter from June 2000 through October 2001

```{r}

toolik\_ts %>%

filter\_index("2000-06" ~ "2001-10")

```

\*\*Example 2:\*\* Filter from April 10, 2006 to May 15, 2006

```{r}

toolik\_ts %>%

filter\_index("2006-04-10" ~ "2006-05-15")

```

\*\*Example 3:\*\* Filter from December 20, 2017 to the end of the dataset

```{r}

toolik\_ts %>%

filter\_index("2017-12-20" ~ .)

```

## 4. Explore changes in seasonality with seasonplots

Let's look at seasonality over the years with a seasonplot, using the `feasts::gg\_season()` function. Notice that we can still do wrangling on a tsibble like we would with a normal data frame:

```{r}

toolik\_ts %>%

filter(year(date) > 2014) %>%

gg\_season(y = mean\_airtemp)

```

Daily measurements seems a bit excessive to return in this visualization, right? Maybe it makes more sense to use the monthly averages in ``.

```{r}

# Now a season plot:

toolik\_month %>%

gg\_season(y = monthly\_mean\_temp) +

theme\_minimal() +

labs(x = "Year",

y = "Mean monthly air temperature (Celsius)",

title = "Toolik Station air temperature")

```

## 5. Seasonal subseries plots

Sometimes it can be useful to explore how values within one season/month/etc. change over time (e.g. across years).

We can use `gg\_subseries()` to explore how values change within a specified window over time.

Do you notice any trends that differ across the months?

```{r}

toolik\_month %>%

gg\_subseries(monthly\_mean\_temp)

```

## 6. Moving averages in tsibbles

We'll use the `slider` package to find moving (or rolling) averages for different window sizes.

The general structure will tend to be something like:

`df %>% slide(variable, function, .before = , .after = )`

Let's make a test vector just so we can see how this works:

```{r}

set.seed(2021)

test<- rnorm(100, mean = 40, sd = 10)

# Show the series based on values +2 and -2 from each observation

# Use ~.x to show the windows

slide(test, ~.x, .before = 2, .after = 2)

# Change that to a function name to actually calculate something for each window

# Note that I add `as.numeric` here, since the outcome is otherwise a list

w5 <- as.numeric(slide(test, mean, .before = 2, .after = 2))

w5

# Find the mean value of a window with n = 11, centered:

w11 <- as.numeric(slide(test, mean, .before = 5, .after = 5))

w11

# Find the mean value of a window with n = 19, centered:

w19 <- as.numeric(slide(test, mean, .before = 9, .after = 9))

w19

# Plot these together:

combo <- data\_frame(time = seq(1:100), test, w5, w11, w19)

ggplot(data = combo) +

geom\_line(aes(x = time, y = test)) +

geom\_line(aes(x = time, y = w5), color = "red") +

geom\_line(aes(x = time, y = w11), color = "blue") +

geom\_line(aes(x = time, y = w19), color = "orange")

```

Now for an example with our Toolik Station data, let's say we want to find the \*average\* value at each observation, with a window that extends forward and backward n days from the observation:

```{r}

roll\_toolik\_15 <- toolik\_ts %>%

mutate(ma\_15d = as.numeric(slide(toolik\_ts$mean\_airtemp, mean, .before = 7, .after = 7)))

roll\_toolik\_61 <- toolik\_ts %>%

mutate(ma\_61d = as.numeric(slide(toolik\_ts$mean\_airtemp, mean, .before = 30, .after = 30)))

ggplot() +

geom\_line(data = toolik\_ts, aes(x = date, y = mean\_airtemp), size = 0.2, color = "gray") +

geom\_line(data = roll\_toolik\_15, aes(x = date, y = ma\_15d), color = "orange") +

geom\_line(data = roll\_toolik\_61, aes(x = date, y = ma\_61d), color = "blue") +

theme\_minimal()

```

## 7. Autocorrelation function

We'll look at outcomes for both daily lags (yikes) and monthly lags (cool).

```{r}

toolik\_ts %>%

ACF(mean\_airtemp) %>%

autoplot()

toolik\_month %>%

ACF(monthly\_mean\_temp) %>%

autoplot()

```

##8. Decomposition

Here we will use STL decomposition (Seasonal, Trend, and Loess) decomposition. You can read about the advantages of STL decomposition here: https://otexts.com/fpp2/stl.html.

```{r}

toolik\_dec <- toolik\_month %>%

model(STL(monthly\_mean\_temp ~ season(window = Inf)))

components(toolik\_dec) %>% autoplot()

```

## END Part 2