Olek's Function Work

Olek Wojcik

4/22/2021

```r
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(sf)
```

```
## Linking to GEOS 3.7.1, GDAL 2.4.0, PROJ 5.2.0
```

```r
library(pdxTrees)
library(lwgeom)
```

```
## Linking to liblwgeom 3.0.0beta1 r16016, GEOS 3.7.1, PROJ 5.2.0
```

```r
set.seed(13)

pdx <- get_pdxTrees_parks() %>%
  sample_n(100)

pdx_sf <- st_as_sf(pdx,
                   coords = c("Longitude", "Latitude"),
                   crs = 4326)

#I don't think we need these helper functions, I used a simpler method below, lmk if I missed something
# one_nn_distance <- function(sf_object, one_row, ...){
#   sf_object %>%
#     mutate(distance_temporary_column = as.numeric(st_distance(one_row, sf_object, ...)))%>%
#     #shouldnt make a new column like this
#     filter(distance_temporary_column != 0) %>%
#     #what do we do if multiple points are on the same spot?
#     summarize(m = min(distance_temporary_column)) %>%
#     as.data.frame() %>%
#     dplyr::select(m) %>%
#     as.numeric()
# }
#
# pdx_one <- slice_head(pdx_sf, n = 1)
```

```
#
#
# sf_nn_distances <- function(sf_object, ...){
#   sf_object %>%
#     rowwise() %>%
#     mutate(distance = one_nn_distance(sf_object = sf_object,
#                                       one_row = geometry, ...))
#   }
```

##G Function

```
g_function <- function(sf_object, ...){
  g <- st_distance(sf_object)
gm <- as.matrix(g)
diag(gm) <- NA
distances <- apply(gm, 1, min, na.rm=TRUE)
  max_dist <- max(distances)
  distances_df <- data.frame(distances = distances)
  d <- seq(from = 0, to = max_dist, by = max_dist/100)
  props <- d %>%
    map_dbl(.f = function(.){
      #print(distances_df$distances > .)
      mutate(distances_df,
             true = case_when(distances < . ~ 1,
                              distances >= . ~ 0)) %>%
        summarize(prop = mean(true)) %>%
        as.double()
    })
g_df <- data.frame(distance = d,
             prop = props)
ggplot(data = g_df,
       mapping = aes(x = distance,
                     y = prop)) +
  geom_line(color = "steelblue") +
  theme_minimal() +
  labs(title = "G-Function")
}

g <- g_function(pdx_sf)
```

##F Function

```
set.seed(22)
f_function <- function(sf_object, ...){
  #First generating random points
   bb_studyregion = st_bbox(sf_object) # the study region's bounds
  random_df = tibble(
  x = runif(n = length(sf_object), min = bb_studyregion[1], max = bb_studyregion[3]),
  y = runif(n = length(sf_object), min = bb_studyregion[2], max = bb_studyregion[4])
  )
  random_points = random_df %>%
  st_as_sf(coords = c("x", "y")) %>% # set coordinates
  st_set_crs(st_crs(sf_object)) # set geographic CRS
 #prepping for the F-function formula
  dt <- st_distance(random_points, sf_object$geometry)
  dm <- as.matrix(dt)
  distances <- apply(dm, 1, min, na.rm=TRUE)
  max_dist <- max(distances)
  distances_df <- data.frame(distances = distances)
  d <- seq(from = 0, to = max_dist, by = max_dist/100)
  props <- d %>%
    map_dbl(.f = function(.){
      #print(distances_df$distances > .)
      mutate(distances_df,
             true = case_when(distances < . ~ 1,
                              distances >= . ~ 0)) %>%
        summarize(prop = mean(true)) %>%
        as.double()
    })
 f_df <- data.frame(distance = d,
             prop = props)
ggplot(data = f_df,
       mapping = aes(x = distance,
                     y = prop)) +
  geom_line(color = "steelblue") +
  theme_minimal() +
  labs(title = "F-Function")
}
f <- f_function(pdx_sf)
```

##K Function

```
library(geosphere)
k_function <- function(sf_object, ...){
pol <-  st_as_sfc(st_bbox(sf_object))
sfarea <- as.numeric(st_area(pol))
sfdens <- as.numeric(sfarea/nrow(sf_object))

#distance
d <- distm(st_coordinates(sf_object),st_coordinates(sf_object), fun=distHaversine)
#Applying formula from class
dist <- seq(1, 25000, 100)
Kd <- sapply(dist, function(x) sum(d < x)) # takes a while
Kd <- Kd / (length(Kd) * sfdens)
K_df <- data.frame(distance = dist, Kd=Kd)
ggplot(data = K_df,
       mapping = aes(x = distance,
                     y = Kd)) +
  geom_line(color = "steelblue") +
  theme_minimal() +
  labs(title = "K-Function")
}
 k <- k_function(pdx_sf)
```

```
library(spatstat)
```

```
## Loading required package: spatstat.data
```

```
## Loading required package: nlme
```

```
##
```

```
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## Loading required package: rpart
```

```
## Registered S3 method overwritten by 'spatstat':
##   method     from
##   print.boxx cli
```

```
##
## spatstat 1.64-1       (nickname: 'Help you I can, yes!')
## For an introduction to spatstat, type 'beginner'
```

```
##
## Note: spatstat version 1.64-1 is out of date by more than 11 months; we recommend upgrading to the la
```

```
##
## Attaching package: 'spatstat'
```

```
## The following object is masked from 'package:geosphere':
##
##     perimeter
```

```
library(gridExtra)
```

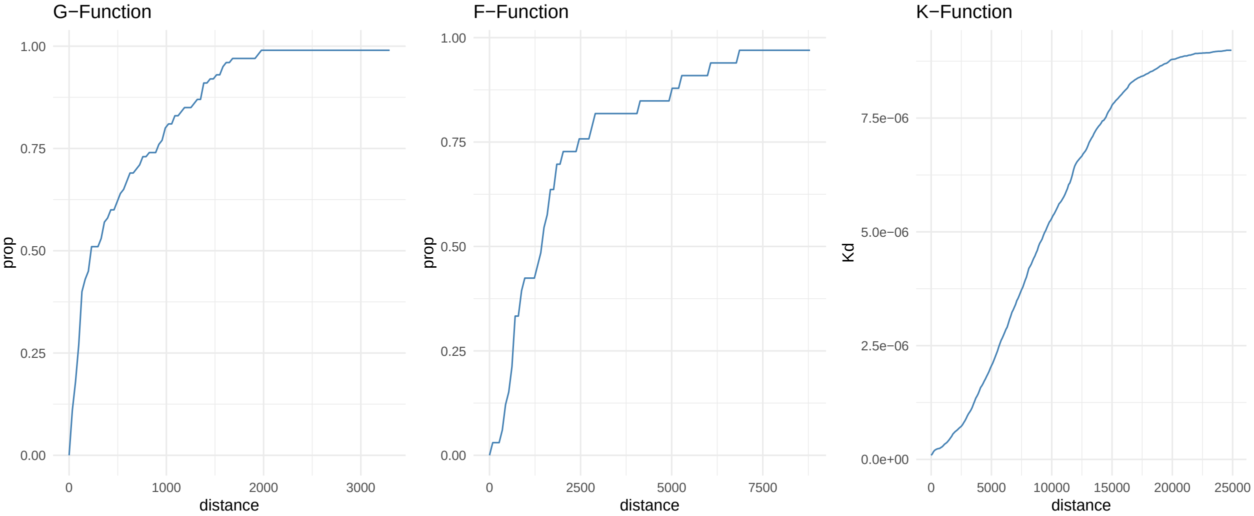
```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

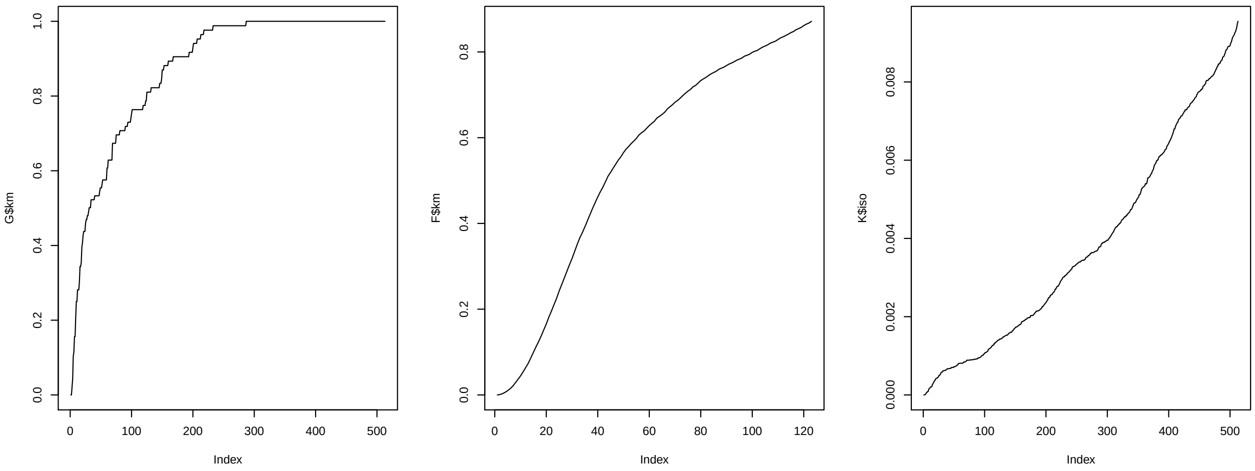
```
pts.owin <- owin(c(-122.76421,-122.48729), c(45.46413,45.64891))
coords <- sf::st_coordinates(pdx_sf)
pdx_ppp <- ppp(x = coords[,1], y = coords[,2],window =  pts.owin)
G <- Gest(pdx_ppp)
F <- Fest(pdx_ppp)
K <- Kest(pdx_ppp)
```

#Comparing our plots vs. spatstat plots

```
grid.arrange(g,f,k, ncol=3)
```



```
par(mfrow=c(1,3))
plot(G$km, type="l")
plot(F$km, type=’l’)
plot(K$iso, type=’l’)
```



```
#trash experimental functions
one_nn <- function(one_row, sf_object){
  sf_object %>%
    mutate(distance_temporary_column = as.numeric(st_distance(one_row, sf_object)))%>%
    #shouldnt make a new column like this
    filter(distance_temporary_column != 0) %>%
    #what do we do if multiple points are on the same spot?
    filter(distance_temporary_column == min(distance_temporary_column)) %>%
    select(-distance_temporary_column)
}

sf_nn <- function(sf_object){
  sf_object %>%
    rowwise() %>%
    map(.f = function(.){
      #one_geom <- st_transform(., crs = 4326)
      one_nn(., sf_object)
    })
}
```