Notes on S3M sampling

10 June 2018

1. Warning message when reading boundary shapefiles

In `Read map data` node, you get the following warning messages:

```
st <- readShapeSpatial("Boundary_Shapefiles/02_StateBoundaries_Poly_2013.shp",
                       proj4string = CRS("+proj=longlat +dataum=WGS84"))
```

```
## Warning: use rgdal::readOGR or sf::st_read
```

```
## Warning: use rgdal::readOGR or sf::st_read
```

```
loc1 <- readShapeSpatial("Boundary_Shapefiles/03_Locality_Boundaries_Poly_2013.shp",
                         proj4string = CRS("+proj=longlat +dataum=WGS84"))
```

```
## Warning: use rgdal::readOGR or sf::st_read
```

```
## Warning: use rgdal::readOGR or sf::st_read
```

The `readShapeSpatial()` function in package `maptools` is scheduled for deprecation and has now been superseded by the `readOGR()` function in package `rgdal` (written by the same author). Since the first Sudan S3M, we have shifted our mapping libraries away from `maptools` and into `rgdal`. The main differentiating features of `readOGR()` is that it actually detects/reads the projection ascribed to a Shapefile without having to be specified (as what is needed for `readShapeSpatial()`) and that it can read about 8 other geographic formats (i.e., kml, gpx…). I would recommend that we do the same for this current workflow and for future S3M workflows.

To read the same files using `readOGR()`, we use the following syntax:

```
st <- readOGR(dsn = "Boundary_Shapefiles",            # directory containing SHP
              layer = "02_StateBoundaries_Poly_2013") # name of the SHP layer
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/ernest/Dropbox/R/S3M/new/s3mTRI/Boundary_Shapefiles", layer: "02_StateBound
## with 18 features
## It has 4 fields
```

You can remove the message output describing the geodata that has just been read by adding the `verbose` argument and setting it to `FALSE`:

```
st <- readOGR(dsn = "Boundary_Shapefiles",
              layer = "02_StateBoundaries_Poly_2013",
              verbose = FALSE)
```

You will notice that if you check for the project of the `SpatialPolygonsDataFrame` object created by `readOGR()`, it will show:

```
proj4string(st)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

The `SpatialPolygonsDataFrame` object `st` has been assigned the appropriate projection.

2. Consider other relevant geospatial libraries

I would suggesting a few relevant geospatial libraries that I think, based on experience, have really good functions that help with manipulating geospatial data.

a. `rgdal`

As commented above, I would suggest adding this and using its read and write functions for geospatial data;

b. `raster`

I looked at the node for the function libraries and `raster` was included there before. We included that before not becuase we are using raster data but mainly for two general features:

- `raster` package when installed alongside `rgdal` and `rgeos` allows for more efficient onloading and offloading of geospatial data. What I mean by this is that it faciliates "lazy" handling especially of big (in terms of memory size and of actual fields and features) geospatial data.
- `raster` package includes utility geospatial functions found in standard GIS packages that we might consider using for some functionalities that I think we might want consider for S3M based on my further comments below. Some examples of these functions are `intersect()`, `union()`, `difference()`.
- `geosphere` - I notice that we use `Imap` package mainly to use the `gdist()` function. I was wondering whether it would be better to use the `geosphere()` package which is a purely distance calculatur package and the functions are highly vectorised so they are very efficient (see comment below on `gdist()` warning). The other thinking to consider here will be whether we actually need to use a geodesic distance calculator for finding nearest village or any other distance calculation requirements down the line. It might be that we consider Euclidean approaches to distance calculation? They tend to be faster to implement, highly vectorised. I am thinking functions that include finidng nearest neighbour algorithm works well. The `gstat` package that we use for spatial interpolation uses `FNN` package for its nearest neighbour search for its `idw()` function. We also use the `FNN` package for our cross-validation functions for spatial interpolation.

3. Warning message when applying `nearestPoint()` function

In the node for `State Grid TRI` and `Locality Grid TRI`, there are multiple warnings that show up when the `nearestPoint()` function as applied:

```
selPS <- nearestPoint(SPs, vil)
```

```
## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
```

```
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used

## Warning in while (abs(lamda - lamda.old) > 1e-11) {: the condition has
## length > 1 and only the first element will be used
```

Despite the warning, the resulting object `selPS` contains the data frame that is expected.

```
selPS
```

```
##    id        x        y         village             locality
## 1 920 33.28083 12.61028  Galaa' AlBeid' AlDali and AlMazmoum
## 2  54 34.14706 12.64006  Wad Girif Awal            Abu Hijar
## 3 244 34.79008 12.82597  Um Bagara Garb            Al Dindir
## 4 939 33.67212 13.45171 Wad Hashim Mroa               Sennar
##                     d
## 1   8.39494441502051
## 2 0.0534743139366605
## 3   38.0782315663253
## 4  0.678767494050234
```

This warning is most likely coming from the application of the `gdist()` function within the `nearestPoint()` function.

`gdist()` requires single value inputs for `lon.1`, `lat.1`, `lon.2`, `lat.2` arguments. In the `nearestPoint()` function however, the `x` and `y` arguments are expected to be vectors and `gdist()` function is supplied a vector for `lon.2` and `lat.2` arguments. Despite this, the `gdist()` function is able to deal with this mainly because its application of the distance calculation is vectorised and R by default just addresses the problem by using the first values in the `lon.2` and `lat.2` vectors. So, as suggested in the workflow, it seems this warning can be safely ignored.

Ideally, the best solution to this would be for the `gdist()` function to be udpated and that the longitude and latitude coordinate inputs be vectorised as default rather than single values.

I wonder, however, whether we would want to update `nearestPoint()` so that it can deal with the limitations of the `gist()` function or that we change the geodesic distance calculator function that we are using. This is related to my comment earlier about the use of the functions in the `geosphere()` package instead or just using a Euclidean distance calculation from a basis nearest neighbour algorithm.

Below is an example of how we can update `nearestPoint()` by vectorising the application of the `gdist()`:

```
nearestPoint <- function(x, y) {
  near.point <- NULL
    for(i in 1:length(x)) {
      near.point1 <- mapply(FUN = gdist, lon.2 = y[ , 2], lat.2 = y[ , 3],
                            MoreArgs = list(x$x[i], x$y[i], units = "km"))

        near.point2 <- c(y$id[which(near.point1 == min(near.point1))],
                                      y$x[which(near.point1 == min(near.point1))],
                                    y$y[which(near.point1 == min(near.point1))],
                                    y$village[which(near.point1 == min(near.point1))],
                                    y$locality[which(near.point1 == min(near.point1))],
                                    min(near.point1))

      near.point <- rbind(near.point, near.point2)
    }
  near.point <- as.data.frame(near.point)
    names(near.point) <- c("id", "x", "y", "village", "locality", "d")
    rownames(near.point) <- NULL
    near.point[,1] <- as.numeric(near.point[,1])
    near.point[,2] <- as.numeric(near.point[,2])
    near.point[,3] <- as.numeric(near.point[,3])
    near.point <- near.point[!duplicated(near.point[,-6]),]
    return(near.point)
}
```

Here all I have done is to edit the line where we apply the `gdist()` function by simply using it inside `mapply()` so that it can be applied to every row of values in the vector of community location coordinates.

When we apply the udpate `nearestPoint()` function, we get no warnings and we get the same results.

```
selPS <- nearestPoint(SPs, vil)
selPS
```

```
##    id        x        y         village             locality
## 1 920 33.28083 12.61028  Galaa' AlBeid' AlDali and AlMazmoum
## 2  54 34.14706 12.64006  Wad Girif Awal            Abu Hijar
## 3 244 34.79008 12.82597  Um Bagara Garb            Al Dindir
## 4 939 33.67212 13.45171 Wad Hashim Mroa               Sennar
##                     d
## 1  8.39494441502051
## 2 0.053474313755245
## 3  38.0782315663253
## 4 0.678767472454879
```

Trying out the Euclidean distance calculations found in package `FNN`, we can use the following function:

```
get_nn <- function(data, x1, y1, query, x2, y2, n) {
  near.index <- get.knnx(data = data[ , c(x1, y1)],
                         query = query[ , c(x2, y2)],
```

```
                          k = n)
  near.point <- data[near.index$nn.index, ]
  near.point <- data.frame(near.point, d = c(near.index$nn.dist))
  near.point <- near.point[!duplicated(near.point[ , c(x1, y1)]), ]
  row.names(near.point) <- 1:nrow(near.point)
  return(near.point)
}
```

where:

data	an input data frame or matrix containing longitude and latitude coordinate of village locations
x1	a character value specifying the variable name in `data` holding the longitude and latitude coordinate of village locations
y1	a character value specifying the variable name in `data` holding the latitude coordinate values
query	an object of class `SpatialPoints` containing sampling point locations. This is usually the output from applying `spsample()` function from package `gstat` to create an even spatial across the entire sampling area
x2	a character value specifying the variable name in holding the longitude coordinate values
y2	a character value specifying the variable name in query holding the latitude coordinate values
n	number of nearest neighbours to search

Applying this function to find the nearest village to the sampling point, we get the same results in both structure and values with the exception of the distance values where the nearest neighbour algorithm outputs distance in coordinate units.

```
selPS <- get_nn(data = vil, x1 = "x", y1 = "y",
                query = SPs@coords, x2 = "x", y2 = "y",
                n = 1)
selPS
```

```
##    id        x        y         village             locality           d
## 1 920 33.28083 12.61028  Galaa' AlBeid' AlDali and AlMazmoum 0.0770641584
## 2  54 34.14706 12.64006  Wad Girif Awal            Abu Hijar 0.0004843054
## 3 244 34.79008 12.82597  Um Bagara Garb            Al Dindir 0.3487567809
## 4 939 33.67212 13.45171 Wad Hashim Mroa               Sennar 0.0062683062
```

Now trying out the package `geosphere`, we use the `distGeo()` function to calculate distances between sampling points and villages using this function:

```
get_nearest_point <- function(data, data.x, data.y, query, n = 1,
                              ellipsoid = c("AA", "AN", "??", "BR", "BN",
                                            "CC", "CD", "EB", "EA", "EC",
```

```
                                                "EF", "EE", "ED", "RF", "HE",
                                                "HO", "ID", "IN", "KA", "AM",
                                                "FA", "SA", "WD", "WE"),
                                  duplicate = TRUE) {
  dataSP <- SpatialPoints(coords = data[ , c(data.x, data.y)],
                          proj4string = crs(proj4string(query)))
  a <- refEllipsoids()[refEllipsoids()$code == "WE", "a"]
  f <- 1 / refEllipsoids()[refEllipsoids()$code == "WE", "invf"]
  if(length(ellipsoid) !=  24 & length(ellipsoid) == 1) {
    a <- refEllipsoids()[refEllipsoids()$code == ellipsoid, "a"]
    f <- 1 / refEllipsoids()[refEllipsoids()$code == ellipsoid, "invf"]
  }
  if(length(ellipsoid) > 1 & length(ellipsoid) != 24) {
    stop("More than one reference ellipsoid specified.
         Select only one. Try again", call. = TRUE)
  }
  if(class(data.x) != "character" | class(data.y) != "character") {
    stop("data.x and/or data.y is/are not character.
         Try again", call. = TRUE)
  }
  if(class(query) != "SpatialPoints") {
    stop("query should be class SpatialPoints object.
         Try again.", call. = TRUE)
  }
  near.point <- NULL
  for(i in 1:length(query)) {
      near.point1 <- distGeo(p1 = query[i, ], p2 = dataSP, a = a, f = f) / 1000
      near.point2 <- data[which(near.point1 %in% tail(sort(x = near.point1,
                                                           decreasing = TRUE),
                                                      n = n)), ]
      near.point2 <- data.frame("spid" = rep(i, n),
                                near.point2,
                                "d" = tail(sort(x = near.point1,
                                                decreasing = TRUE),
                                           n = n))
      near.point <- data.frame(rbind(near.point, near.point2))
  }
  if(duplicate == FALSE) {
    near.point <- near.point[!duplicated(near.point[ , c(data.x, data.y)]), ]
  }
  return(near.point)
}
```

where:

data	an input data frame or matrix containing longitude and latitude coordinate of village locations
data.x	a character value specifying the variable name in `data` holding the longitude and latitude coordinate of village locations
data.y	a character value specifying the variable name in `data` holding the latitude coordinate values

query	an object of class `SpatialPoints` containing sampling point locations. This is usually the output from applying `spsample()` function from package `gstat` to create an even spatial across the entire sampling area
n	number of nearest neighbours to search
ellipsoid	two letter character value specifying the reference ellipsoid to use for distance calculations
duplicate	if duplicate selected villages are to be kept or discarded

Applying this function to find the nearest village to the sampling point, we get the same results in both structure and values.

```
selPS <- get_nearest_point(data = vil, data.x = "x", data.y = "y",
                           query = SPs, ellipsoid = "WE",
                           n = 1, duplicate = FALSE)
selPS
```

```
##     spid  id        x        y         village             locality
## 920    1 920 33.28083 12.61028  Galaa' AlBeid' AlDali and AlMazmoum
## 54     2  54 34.14706 12.64006  Wad Girif Awal            Abu Hijar
## 244    3 244 34.79008 12.82597  Um Bagara Garb            Al Dindir
## 939    4 939 33.67212 13.45171 Wad Hashim Mroa               Sennar
##               d
## 920  8.39494442
## 54   0.05347431
## 244 38.07823157
## 939  0.67876749
```

4. `nearestPoint()` function currently selects only one nearest community

The `nearestPoint()` function only selects one nearest community to the sampling point. Based on earlier discussions, we might want to select more than one nearest village/community from the sampling point.

The `get_nn()` function and the `get_nearest_point()` function example/suggestion both allows for more than 1 nearest neighbour to be specified.

```
# Get 3 nearest villages using get_nn()
selPS <- get_nn(data = vil, x1 = "x", y1 = "y",
                query = SPs@coords, x2 = "x", y2 = "y",
                n = 3)
selPS
```

```
##     id        x        y          village             locality
## 1  920 33.28083 12.61028   Galaa' AlBeid' AlDali and AlMazmoum
## 2   54 34.14706 12.64006   Wad Girif Awal            Abu Hijar
## 3  244 34.79008 12.82597   Um Bagara Garb            Al Dindir
## 4  939 33.67212 13.45171  Wad Hashim Mroa               Sennar
## 5  791 33.13497 12.70617        algioytab AlDali and AlMazmoum
## 6  364 34.14936 12.63297          Gindeel              AlSooki
## 7  126 34.79900 12.84908 Om Bagara Sharig            Al Dindir
## 8  938 33.66769 13.44440  Wad Hashim Maly               Sennar
## 9  917 33.24297 12.53967       Wad A'Kara AlDali and AlMazmoum
## 10  49 34.13497 12.62322  Shamiyah Khalil            Abu Hijar
## 11 124 34.77686 12.85164         Om Sagit            Al Dindir
## 12 937 33.66446 13.44985  Wad Hashim Arab               Sennar
##               d
## 1  0.0770641584
## 2  0.0004843054
## 3  0.3487567809
## 4  0.0062683062
## 5  0.0999771165
## 6  0.0069649118
## 7  0.3544045868
## 8  0.0129061202
## 9  0.1053677149
## 10 0.0204462652
## 11 0.3739350563
## 12 0.0140465938
```

```
selPS <- get_nearest_point(data = vil, data.x = "x", data.y = "y",
                           query = SPs, ellipsoid = "WE",
                           n = 3, duplicate = FALSE)
selPS
```

```
##     spid  id        x        y          village             locality
## 791    1 791 33.13497 12.70617        algioytab AlDali and AlMazmoum
## 917    1 917 33.24297 12.53967       Wad A'Kara AlDali and AlMazmoum
## 920    1 920 33.28083 12.61028   Galaa' AlBeid' AlDali and AlMazmoum
## 48     2  48 34.12717 12.63519         Mishiraat            Abu Hijar
## 54     2  54 34.14706 12.64006   Wad Girif Awal             Abu Hijar
```

```
## 364    2 364 34.14936 12.63297          Gindeel              AlSooki
## 124    3 124 34.77686 12.85164         Om Sagit            Al Dindir
## 126    3 126 34.79900 12.84908 Om Bagara Sharig            Al Dindir
## 244    3 244 34.79008 12.82597   Um Bagara Garb            Al Dindir
## 937    4 937 33.66446 13.44985  Wad Hashim Arab               Sennar
## 938    4 938 33.66769 13.44440  Wad Hashim Maly               Sennar
## 939    4 939 33.67212 13.45171  Wad Hashim Mroa               Sennar
##               d
## 791 11.63585349
## 917 10.94923910
## 920  8.39494442
## 48   2.23184050
## 54   0.76920677
## 364  0.05347431
## 124 40.85328101
## 126 38.73978689
## 244 38.07823157
## 937  1.52157791
## 938  1.40712236
## 939  0.67876749
```

We might want to consider adding this functionality to the `nearestPoint()` function or choose to use one of the other functions.

5. Keeping nearest village/community search *local*

Currently, the `nearestPoint()` function searches nearest villages/communities among the full list of villages/communities in the survey area. In a few cases, this can lead to a sampling point having a nearest village/community that is not *local* - that is a village/community not within the sampling point's hexagonal grid area defined by `d` used to specify the size of the grid.

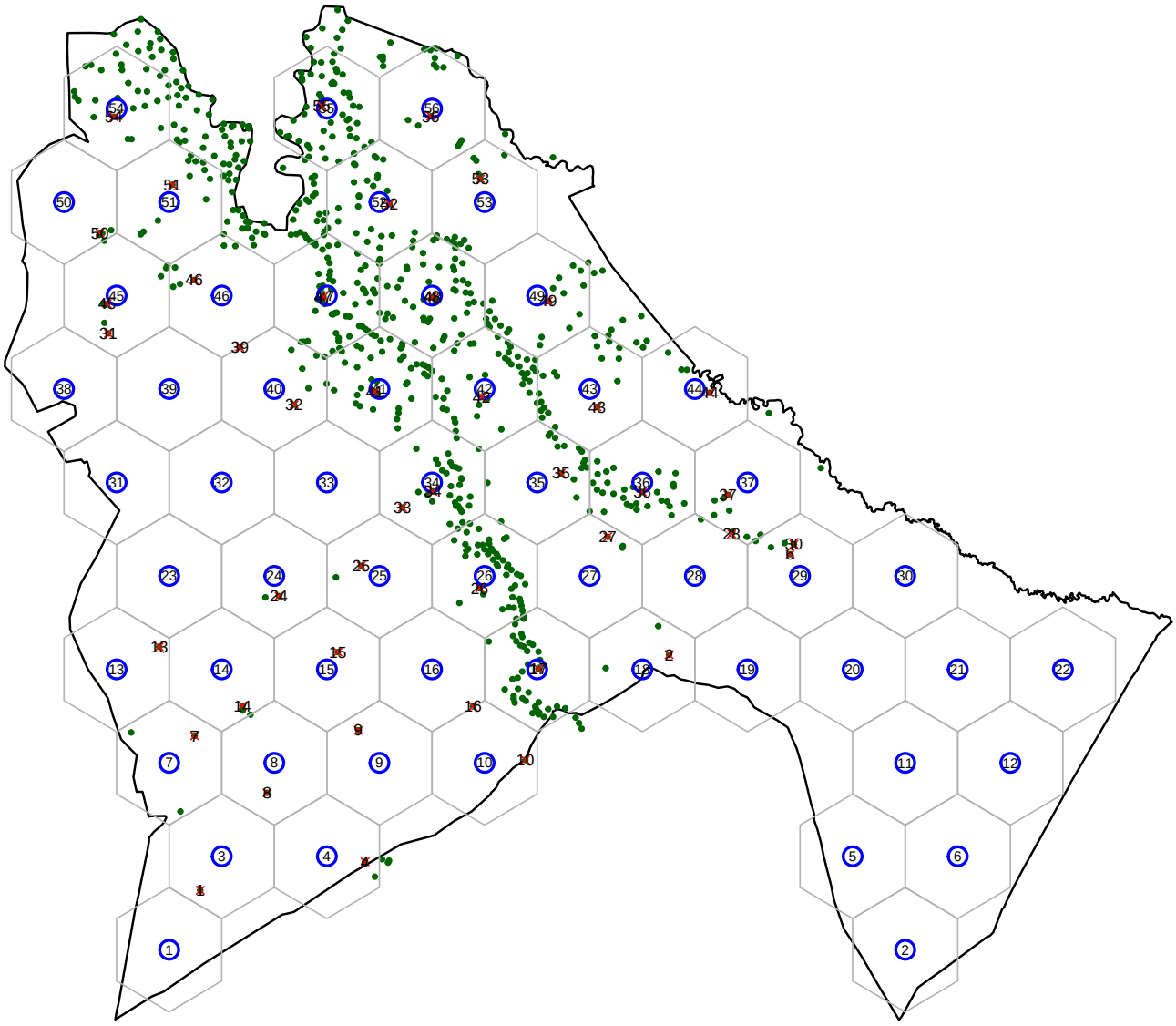

Figure 1: Sampling map of Sennar at d = 15 kms

In the map above, there are hexagons with more than one selected village with one of them not "local" to the sampling point it is associated with.

We might want to consider applying a more local search for the nearest village. We can potentially do this by limiting the search for the nearest village to within the hexagon of the sampling point. This will be a minor edit of the nearest point algorithms that we currently have.