# DisperseR: Calculating Seed Dispersal In R

Samantha L. Davis

June 19, 2015

## Contents

## 1 Introduction

This is a small package intended to help users calculating seed dispersal in R. Although the R base machinery is capable of doing so, this package streamlines the process and enables you to focus more on the important aspects of data analysis instead of data generation or clean-up.

This code operates as follows. Ideally, you'll need a dataframe that contains the following data: (x,y) coordinates of each tree and seedling in a plot; and dbh measurements of any tree large enough. A tree is any individual that can be measured for diameter at breast height, and all trees are assumed to be reproductively active; a seedling is any individual that is new in the calendar year.

Spatial seed dispersal is characterized by a single equation,

$$R_i = STR * \sum_{k=1}^{T} \left( \frac{DBH_k}{30} \right)^2 e^{-Dm_{ik}^3} * \left( \frac{1}{n} \right) \tag{1}$$

where $n$ is a normalizer function that standardizes the equation to values between 0 and 1,

$$n = \int_0^\infty e^{-Dm_{ik}^3} \tag{2}$$

and where $STR$ is the standardized number of tree recruits, $DBH$ is the diameter at breast height, $D$ is a species-specific parameter estimated by this equation, and $m$ is the distance between the measured point $i$ and adult tree $k$, summed over each adult tree ($k$=1 to $T$ adult trees). These equations were originally established by Ribbens et al. (1994), in an experiment where seedling per $m^2$ along a belt transect were correlated to the number and size of any adults within a 20$m$ radius.

The first piece of the equation, containing STR, establishes the number of recruits produced for a tree of a standard DBH (30cm), and the second piece of the equation establishes the mean density of recruits found in a 1$m^2$ quadrat centered at $m$ distance away from the parent tree. Finally, $\frac{1}{n}$ serves as a normalizer to standardize the equation across species.

The parameters $STR$ and D are both needed by SORTIE-ND, an individual tree neighborhood dynamics forest gap model (say that five times fast!), to calculate seed dispersal for target species in its simulations. SORTIE-ND, unfortunately, does not come packaged with a magic bullet that offers species-specific parameters, and therefore, we must parameterize the model ourselves. This package is intended to help create estimates of both $STR$ and $D$ quickly, so that other parameters may be addressed.

What follows is a list of functions alongside example usage. To start, you must import or generate a plot map of all trees in a given area. This plot map must include a species identifier, an x coordinate, a y coordinate, and DBH (or NA) for each individual.

## 2 Generating Plot Map

### 2.1 generatePlotMap

We can generate a sample plot easily with generatePlotMap(). As you can see below, this function generates a plot map with NA's for seedlings and actual values of DBH for adult trees. See ?generatePlotMap() for information on how to customize your random plot map.

```
> library(disperseR)
> myplot <- generatePlotMap()
> head(myplot)

  species        x          y dbh
1       1 385.6154 985.785907  NA
2       1 851.7756 697.200094  NA
3       1 401.9323   4.610472  NA
4       1 836.0269 487.991343  NA
5       1 755.7098 917.864674  NA
6       1 306.3455 434.971478  NA

> tail(myplot)
```

```
    species         x        y      dbh
745       5 560.69002 292.8404 24.57768
746       5 106.19956 908.0030 84.65400
747       5 518.49203 498.0702 79.69312
748       5  52.77148 152.2081 37.59841
749       5  87.16742 837.4155 52.06587
750       5 943.00556 377.6981 67.90298
```

Now that we have a plotmap, we can focus on creating the spatial dispersal equations. Obviously, since this plot map is random, our end parameters will be useless, but this will at least demonstrate proof-of-concept, and you can apply it to real data later.

If you do have your own data, just make sure that it matches the column names of the plot map generated above, and also the data types. You can check the structure of a dataframe using str() and then as.numeric() or as.character() to adjust as needed. In our case, you need four columns: species, x, y, and DBH. x, y, and DBH should all be numeric. "species" can be a character vector or a numeric vector, as long as the species names are unique.

```
> ## exploring the structure of myplot
> str(myplot)

'data.frame':        750 obs. of  4 variables:
 $ species: int  1 1 1 1 1 1 1 1 1 1 ...
 $ x      : num  386 852 402 836 756 ...
 $ y      : num  985.79 697.2 4.61 487.99 917.86 ...
 $ dbh    : num  NA NA NA NA NA NA NA NA NA NA ...

> ## if we needed to convert a column
> myplot$species <- as.numeric(myplot$species)
```

## 3 Sampling The Plot Map

Now that we have a plot map ready, we need to be able to sample the plot. Ribbens et al. (1994) sampled using a belt transect, stopping every so often to count all of the seedlings in a $1m^2$ plot, and all adult trees within $20m$ of the seedling plot. So, for each seedling plot sampled, we need records of adult trees' DBH and their distance to the seedling subplot, up to 20m away. The end table to plug into the equation might look something like:

```
> spatialDisperseDf <- data.frame(subplot=rep(1:3,5),
+                                 species=1,
+                                 numseedlings=rep(c(2,4,6), 5),
+                                 DBH=runif(15, 0, 100),
+                                 m=runif(15, 0, 20))
> head(spatialDisperseDf)
```

```
  subplot species numseedlings       DBH          m
1       1       1            2 53.979248  9.5207819
2       2       1            4 20.727423 16.4417057
3       3       1            6  6.522029  6.0429382
4       1       1            2 61.213360 10.7982601
5       2       1            4 30.620536  4.1124465
6       3       1            6 16.957815  0.4305174
```

Of course, the key part of this package is to generate this dataframe and then use it for analysis. There are obviously several ways to sample that are as statstically valid as the belt transect method, and given that we have the benefit of an exhaustive map of a given plot, we should consider using other sampling methods that generate the same sort of information without the linear bias. For ease, this package picks the locations of seedling subplots from your plot randomly, with a buffer around the length and width to prevent trying to find adult trees outside of the plot area.

## 3.1 getRandomBufferedPoints

The first thing we need to do is select our subplots. We can do that with the function getRandomBufferedPoints(), which takes an x and a y vector, a buffer value, and "n", representing the number of samples that you need. This function then spits out "n" random x and y points within the buffered plot space. These locations can represent your seedling plots.

```
> subplotLocations <- getRandomBufferedPoints(x=myplot$x,
+                                             y=myplot$y,
+                                             buffer=20,
+                                             n=100)
> head(subplotLocations)
```

```
         x        y
1 563.9821 523.9435
2 875.9219 308.5930
3 276.4262 911.6807
4 339.7110 485.4485
5 374.0180 446.3255
6 256.2419 403.6310
```

## 3.2 sampleSubplots

Of course, now that we have our random seedling plots ready, we need to actually see if there are any seedlings inside of our randomly chosen subplot locations. We can use the sampleSubplots() function to do that.

The sampleSubplots function takes your x and y coordinates, builds a box around them, and then subsets your full plot dataframe to see if there are any seedlings present. This function takes our pre-existing subplotLocations and myplot dataframes, and samples appropriately with a subplot size of 25m.

```
> seedlingDensity <- sampleSubplots(subplotLocations, myplot, subplotsize=25)
> head(seedlingDensity)

         x        y species numseedlings
1       NA       NA      NA           NA
2 563.9821 523.9435       1            1
3 563.9821 523.9435       4            1
4 875.9219 308.5930       4            1
5 746.2678 513.5763       2            1
6 746.2678 513.5763       4            1

> str(seedlingDensity)

'data.frame':        27 obs. of  4 variables:
 $ x           : num  NA 564 564 876 746 ...
 $ y           : num  NA 524 524 309 514 ...
 $ species     : num  NA 1 4 4 2 4 4 2 5 3 ...
 $ numseedlings: num  NA 1 1 1 1 1 2 1 2 1 ...
```

Now that we have seedling density in our subplots, we need to figure out how many possible parent trees there are for each of the positive hits. We can do that using the getParentTrees() function.

## 3.3 getParentTrees

The getParentTrees() function works by searching a full plot for trees (where dbh is *not* NA) that fall within $20m$ of a seedling plot that contains that species. Of course, you can set that $20m$ buffer to some other value if you'd like.

```
> getParentTrees(seedlingDensity, myplot)

  subplotx subploty species numseedlings        m      dbh    treex    treey
1 746.2678 513.5763       2            1 17.50658 39.47245 752.6982 497.2935
2 159.0189 534.1358       3            1 15.11443 47.66104 148.0537 523.7335
3 159.0189 534.1358       3            1 13.84017 37.59841 149.0297 543.7152
```

## References

Ribbens, E., J. A. Silander, and S. W. Pacala, 1994. Seedling recruitment in forests : Calibrating models to predict patterns of tree seedling dispersion. *Ecology* **75**:1794–1806.