# DisperseR: Calculating Seed Dispersal In R

Samantha L. Davis

June 24, 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 $20m$ 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 $1m^2$ 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       1 163.0315 851.15697  NA
2       1 685.4428 445.75944  NA
3       1 483.9654  77.69053  NA
4       1 518.3450  42.93094  NA
```

```
5       1  52.4708 503.44690  NA
6       1 243.8687 530.70777  NA
```

```
> tail(myplot)
```

```
    species        x        y       dbh
745       5 692.0917 358.6456 41.121313
746       5  19.5082 123.7334 25.974873
747       5 243.3979 366.0807 59.885975
748       5 836.8184 891.9575  3.773943
749       5 264.6122 111.2575 70.373885
750       5 726.6019 419.5462 80.828688
```

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  163 685.4 484 518.3 52.5 ...
 $ y      : num  851.2 445.8 77.7 42.9 503.4 ...
 $ 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 15.99789  1.663235417
2       2       1            4 41.09464  9.103261647
3       3       1            6 45.03613 17.447672980
4       1       1            2 58.12180  0.009979131
5       2       1            4 93.34047  3.598748012
6       3       1            6 13.06455 11.963099437
```

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. There are two versions of getBufferedPoints, one with random sampling (default), and one with systematic. Both are featured below.

```
> randSubplots <- getBufferedPoints(x=myplot$x,
+                                   y=myplot$y,
+                                   buffer=20,
+                                   n=250)
> systSubplots <- getBufferedPoints(x=myplot$x,
+                                     y=myplot$y,
+                                     buffer=20,
+                                     systematic=TRUE,
+                                     by=15)
> head(randSubplots)

         x        y
1 427.5132 171.8657
2 850.4176 234.4082
3 644.5284 456.0196
```

```
4 937.4648 467.3260
5 410.7367 648.8677
6 430.9621 193.7751
```

```
> head(systSubplots)
```

```
         x        y
1 20.04768 21.03429
2 20.04768 36.03429
3 20.04768 51.03429
4 20.04768 66.03429
5 20.04768 81.03429
6 20.04768 96.03429
```

### 3.2 sampleSubplots

Of course, now that we have our 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 subplot locations and myplot dataframes, and samples appropriately with a subplot size of 25m.

```
> randSeedlingDensity <- sampleSubplots(randSubplots,
+                                       myplot,
+                                       subplotsize=25)
> head(randSeedlingDensity)
```

```
         x        y species numseedlings
1       NA       NA      NA           NA
2 794.6918 949.7023       3            1
3 959.8948 242.9892       1            1
4 894.7582 284.3698       2            1
5 493.0255 222.9763       3            1
6 901.4046 813.6106       5            1
```

```
> str(randSeedlingDensity)
```

```
'data.frame':        53 obs. of  4 variables:
 $ x           : num  NA 795 960 895 493 ...
 $ y           : num  NA 950 243 284 223 ...
 $ species     : num  NA 3 1 2 3 5 5 4 1 2 ...
 $ numseedlings: num  NA 1 1 1 1 1 1 1 1 1 ...
```

```
> systSeedlingDensity <- sampleSubplots(systSubplots,
+                                       myplot,
+                                       subplotsize=10)
> head(systSeedlingDensity)
```

```
         x        y species numseedlings
1       NA       NA      NA           NA
2 20.04768 411.0343       4            1
3 20.04768 441.0343       2            1
4 20.04768 771.0343       5            1
5 35.04768 156.0343       1            1
6 35.04768 726.0343       3            1
```

```
> str(systSeedlingDensity)
```

```
'data.frame':        178 obs. of  4 variables:
 $ x           : num  NA 20 20 20 35 ...
 $ y           : num  NA 411 441 771 156 ...
 $ species     : num  NA 4 2 5 1 3 3 2 1 5 ...
 $ numseedlings: num  NA 1 1 1 1 1 1 1 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.

```
> randParents <- getParentTrees(randSeedlingDensity, myplot)
> systParents <- getParentTrees(systSeedlingDensity, myplot)
> head(randParents)
```

```
  subplotx subploty species numseedlings        m       dbh    treex    treey
1 652.9239 370.0046       3            1 18.38679  1.897457 640.4509 383.5138
2 317.3713 639.3390       2            1 18.35288 77.954612 334.4264 632.5600
3 388.2775 296.4581       4            1 19.11338 79.763127 407.2773 294.3773
4 476.1521 739.4261       2            1 18.81404 13.982483 472.8608 757.9500
```

```
> nrow(randParents)
```

```
[1] 4
```

```
> head(systParents)
```

```
  subplotx  subploty species numseedlings        m       dbh    treex     treey
1 155.0477  66.03429       3            1 20.22774  5.174857 174.1650  59.42456
2 275.0477 231.03429       5            1 18.65769 80.592041 283.1845 247.82422
3 275.0477 951.03429       4            1 19.56523 17.313940 281.4726 932.55408
4 365.0477 756.03429       4            1 14.50301 63.364813 359.6779 769.50658
5 410.0477 456.03429       3            1  6.20754 98.933486 412.4319 450.30290
6 410.0477 951.03429       2            1 19.96672 29.692060 406.7480 970.72647
```

```
> nrow(systParents)

[1] 21
```

You can see pretty readily that in most cases, systematic sampling over a grid will be the way to extract the most information. Since we are using a randomly generated plot, there is no clumping of trees or seedlings, and most seedling plots should have low numbers of seedlings. In real life, however, seedlings are often clumped together, and that spatial structure would be accurately represented in the sampling scheme. If you recall, we set the subplot size to be much larger on the random sampling than on the systematic sampling. This is the only way to guarantee that *something* is found.

# 4 Calculating Parameters for the Ribbens Equation

So now that we have our parent tree table ready for modeling, how do we go about finding our parameters? For the sake of simplicity, we're going to ignore the "species" column in systParents and assume that every record is for the same species. We'll do this by setting species to 1 and rerunning the subplot sampling and parent finding.

```
> newplot <- myplot
> newplot$species <- 1
> newSeedlings <- sampleSubplots(systSubplots,
+                                newplot,
+                                subplotsize=10)
> newParents <- getParentTrees(newSeedlings, newplot)
> head(newParents)

  subplotx subploty species numseedlings         m      dbh     treex    treey
1 20.04768 411.0343       1            1 13.535268 38.09206 13.653186 399.1048
2 20.04768 441.0343       1            1 18.266665 98.93349  3.109436 434.1957
3 35.04768 156.0343       1            1  8.369560 45.21357 26.751014 157.1365
4 35.04768 726.0343       1            1 14.973856 80.41769 49.997870 725.1927
5 50.04768 606.0343       1            1  8.469415 45.21357 57.520642 602.0486
6 50.04768 606.0343       1            1 16.977848 25.68085 36.264704 596.1208

> unique(newParents$numseedlings)

[1] 1
```

The model that we're trying to run can be written into a formula in R like this:

```
> formula <- "numseedlings~(dbh/30)^2 * exp(-m^3)"
```

We will do the normalizer afterwards, because it should not affect the outcome of the model. Now that we have the data.frame and the model, it's a simple matter of running it. Because it is nonlinear, we need to use nls() with some start values.

# 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.