DisperseR: Calculating Seed Dispersal In R

Samantha L. Davis

June 22, 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,

$$n = \int_0^\infty e^{-Dm_{ik}^3} \qquad (2)$$

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       1 807.4247 507.8046  NA
2       1 716.7755 250.1089  NA
3       1 663.9457 754.5766  NA
4       1 142.9948 374.6370  NA
```

```
5       1 530.3830 221.6209  NA
6       1 609.4712 460.2917  NA

> tail(myplot)

    species        x        y      dbh
745       5 966.5782 529.7618 48.87171
746       5 358.1308 784.8376 54.50437
747       5 187.9779 648.3317 62.67000
748       5 942.6674 496.2957 29.26230
749       5 122.2823 142.3175 57.63271
750       5 533.8458 553.2040 65.55217
```

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  807 717 664 143 530 ...
 $ y      : num  508 250 755 375 222 ...
 $ 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 22.60360 10.4383481
2       2       1            4 89.85045  9.6430821
3       3       1            6 57.95143  5.4454423
4       1       1            2 69.76575  3.4763289
5       2       1            4 91.61190 11.8251530
6       3       1            6 72.63399  0.9457134
```

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
1 701.5746 492.9402
2 662.9241 192.8330
3 409.8576 454.9313
```

```
4 959.7612 936.4005
5 976.9659 611.2381
6 761.6715 703.7339
```

```
> head(systSubplots)
```

```
         x        y
1 22.44809 21.07131
2 22.44809 36.07131
3 22.44809 51.07131
4 22.44809 66.07131
5 22.44809 81.07131
6 22.44809 96.07131
```

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
2 959.7612 936.4005       1            1
3 672.8342 517.4681       4            1
4 672.8342 517.4681       5            1
5 777.3006 146.1837       1            1
6 777.3006 146.1837       3            1
```

```
> str(randSeedlingDensity)
```

```
'data.frame':       61 obs. of  4 variables:
 $ x           : num  NA 960 673 673 777 ...
 $ y           : num  NA 936 517 517 146 ...
 $ species     : num  NA 1 4 5 1 3 3 1 5 4 ...
 $ 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 22.44809 156.0713       1            1
3 22.44809 471.0713       2            1
4 22.44809 681.0713       4            1
5 22.44809 906.0713       5            1
6 37.44809 126.0713       2            1
```

```
> str(systSeedlingDensity)
```

```
'data.frame':        153 obs. of  4 variables:
 $ x           : num  NA 22.4 22.4 22.4 22.4 ...
 $ y           : num  NA 156 471 681 906 ...
 $ species     : num  NA 1 2 4 5 2 1 3 4 2 ...
 $ numseedlings: num  NA 1 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
1 807.7493 684.10911       5            1  6.524443 48.88866 802.3040 680.51515
2 607.8512 451.26102       1            2 11.790139 87.79125 615.4009 460.31695
3 608.9193  26.13498       5            1 13.228046 20.79004 606.8196  13.07463
4 952.0113 420.01434       5            1  4.948387 58.38689 951.1664 415.13862
5 367.6224 625.12015       3            1  8.126055 13.57113 369.8776 617.31330
```

```
> nrow(randParents)
```

```
[1] 5
```

```
> head(systParents)
```

```
  subplotx  subploty species numseedlings         m       dbh    treex
1 127.4481 321.07131       2            1  8.308819 36.394576 135.4600
2 142.4481 756.07131       3            1 17.848016 65.058796 138.8851
3 142.4481 756.07131       3            1 15.363776 27.505605 155.5424
4 217.4481 306.07131       5            1  9.633771 66.686525 215.2592
```

```
5 487.4481  66.07131       2            1 19.915035 90.798106 469.1691
6 532.4481 891.07131       3            1 19.560899  8.282892 551.6884
      treey
1 323.27252
2 773.56007
3 748.03490
4 296.68951
5  73.97624
6 894.59822
```

```
> nrow(systParents)
```

```
[1] 16
```

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 22.44809 471.0713       1            1 16.079451 92.94126 38.504673 471.9286
2 22.44809 471.0713       1            1  3.740320 73.46949 24.010826 474.4695
3 22.44809 681.0713       1            1 15.596639 39.57474  9.993568 671.6830
4 22.44809 906.0713       1            1  7.469008 11.21596 24.667640 913.2029
5 22.44809 906.0713       1            1  5.815216 88.91823 19.713086 900.9394
6 22.44809 906.0713       1            1 16.669561 65.55217  7.284792 912.9958
```

```
> 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.