The goal of this project is to simulate a dispersal algorithm taken from Chapter 3 of "Spread of an Invading Population" by Mark Lewis.

1 background

R has many built-in random number generators that you can sample from. For example, this list is taken from the "Introduction to R" manual accompanying the R program

Distribution	R name	additional arguments
beta	`beta`	shape1, shape2, ncp
binomial	`binom`	size, prob
Cauchy	`cauchy`	location, scale
chi-squared	`chisq`	df, ncp
exponential	`exp`	rate
F	`f`	df1, df1, ncp
gamma	`gamma`	shape, scale
geometric	`geom`	prob
hypergeometric	`hyper`	m, n, k
log-normal	`lnorm`	meanlog, sdlog
logistic	`logis`	location, scale
negative	`binomial`	nbinom size, prob
normal	`norm`	mean, sd
Poisson	`pois`	lambda
Student's t	`t`	df, ncp
uniform	`unif`	min, max
Weibull	`weibull`	shape, scale
Wilcoxon	`wilcox`	m, n

These are the "stem" for the "d,r,p and q" functions.

The "r" functions allow you to sample. To take a sample from a normal you use `norm` as

```
> rnorm(10)                       % for mean=0,sd=1
 [1]  0.3958 -0.7842 -0.9949 -0.1029 -1.1473  1.6541  1.6763
 [8]  1.0555 -1.2926  0.5646
> rnorm(5,mean=10,sd=5)
[1] 16.849 15.048  9.178 10.930  9.565
```

The "d" functions give a density. For example, this will plot the density of a normal, make a sample and then plot the sample and a density estimate

```
> mu = 10;sigma = 5
> n = 35
> curve(dnorm(x,mu,sigma),mu-3*sigma,mu+3*sigma)
> x = rnorm(n,mu,sigma)
> rug(x)
> lines(density(x))
```

1.1 Those are not enough?

In our simulation, we want to be able to use distributions that are not named. We'll need some way of generating random samples from them.

1.2 mixtures

A mixture of distributions is when more than one population is present. To sample from a mixture is straightforward: choose which population and then sample from that one. This two-step approach requires to samples – one to pick the population and one from the population.

For example, this will select from one of 3 normal populations.

```
> which.one = sample(1:3,1,p=c(.1,.2,.7))
> switch(which.one,rnorm(1),rnorm(1,10),rnorm(1,100))
[1] 9.461
```

The above example chooses from a mean of 1, 10 or 100 with probability .1, .2 and .7. In the example, it clearly chose from the second one.

This can be simplified (no `switch()`) if you have only 2 things to choose from. For example

```
> p = sample(0:1,1,p=c(.3,.7))
> p*rnorm(1) + (1-p)*rnorm(1,10)
[1] -0.01223
```

Choosing `p` to be 0 or 1, make the selection easy.

If our sampling comes from some arbitrary function f, then we need to work a little harder. Suppose, we have $f(x)$ a density on $[a,b]$ with maximum value M. A simple way of generating random samples from $f(x)$ is to choose two random variables U and V. U is uniform on $[a,b]$ and V is uniform on $[0,M]$. If $f(U) < V$, then we use U, if not, we try again (and again,...).

Why does this work? When we actually select a value, the probability that is less than x is

$$P(selection < x) = P(U < x, f(U) > V) = \int_a^x \int_0^f (u)dvdu = \int_a^x f(x)dx = F(x).$$

To implement this we can use a while loop. For example. We can define an f and g as follows

```
f = function(x) 1 - abs(x)
g = function() {
    M = 1; a= -1; b= 1
    U = runif(1,a,b);V = runif(1,0,M)
    while(f(U) < V) {
      U = runif(1,a,b);V = runif(1,0,M)
    }
    cat("U,f(U),V are",U,f(U),V,"\n")
    U
  }
```

Then, `g()` will return a random sample.

2 the algorithm

The dispersal algorithm is a "branching process" with dispersal. Initially there is a collection of "particles" (plants, birds, ...). At the end of each year, each particle breeds and produces N_j offspring *and then die*. Then each of these offspring move off during the year. The motion is assumed (for now) to be radial. That is, you specify independent directions and distances and the direction is assumed to be uniform. Depending on the dispersal function, you can get different effects.

To model this we have the following steps

- Start with an initial set of particles at $(x_1, y_1), \ldots, (x_n, y_n)$.
- For each particle randomly choose the number of offspring according to some distribution. (Poisson with rate λ.)
- Choose a random angle, θ (Uniform on $[0, 2\pi]$.
- Choose a random distance, r (from some dispersal function)
- Move the new born $r\cos(\theta)$ in the x direction and $r\sin(\theta)$ in the y direction.
- Repeat n steps

Then look at the resulting spatial distribution.

The algorithm is easily implemented in `R`. You can download it from `http://www.math.csi.cuny.edu/verzani/classes/MTH804/computer/invading.R`, save it locally, and then source it in with `File > Source`.

Once you have your copied stored locally, you can make changes to it as desired and re-source it in.

Depending on the branching rate λ and n the number of life cycles you may or may not have any offspring left. Depending on the length of the tails of the dispersal function, you may or may not see "clumping".

Here are some things to try once you get the simulation working.

- Change the branching rate. $\lambda = 1$ is a critical value. Try it with a number much less than 1 to see what happens.
- n and λ are related. If λ is much bigger than 1, then n better be small or the simulation will take a **long** time. If λ is much smaller than 1, then extinction will happen relatively soon. So if n is large, your simulation will stop with no offspring.
- The dispersal function can be varied to make different types of simulations. In general, these branching processes will cause clumping, but if the dispersal isn't too great you can't see the clumping too well with the scatterplot. Changing the dispersal to have occasional large clumps makes these apparent. For example, try a mixture of normals with mean of 0 and a mean of 100. Make the one of 100 relatively rare.
- Distances from the origin are given by

```
> sqrt(x^2 + y^2)
```

Make a histogram of the distances. Does it relate to the dispersal function somehow?

- The farthest away point can be found from `res` as follows

```
> max(sqrt(x^2 + y^2))
```

Keep track of the farthest point away as you step through n. Plot it to see how it grows in n.

- For fixed n, do several simulations and store the value of the farthest point. Can you describe it's distribution?