Geometric Growth

Dick Gomulkiewicz

The purpose of this document is to introduce a few basics of R and its use as a programming language. This will be done using simple deterministic geometric populaiton growth.

Imagine a population that grows deterministically by a factor $\lambda = 1 + r$ each time step. (λ is called the 'finite growth rate' and r is the 'intrinsic growth rate.') That is, if the abundance at time step t is N_t, then the abundance at the next time step is given by the recursion

$$N_{t+1} = \lambda N_t = (1 + r)N_t.$$

It is easy to see that, if the initial size is N_0, then $N_t = \lambda^t N_0 = (1 + r)^t N_0$. Nonetheless, we will use the recursion to compute population sizes at different times to illustrate how programming works.

Store parameters and initial values

First, create variables to store the initial population size (N_0), the number of time steps we wish project, and the intrinsic rate of growth or decline r.

```
#This is a comment
initial.size <- 10
steps <- 50
r <- 0.01
```

Define a function

Next, we create a function called `next.size` that computes the population size at the next time step given the current abundance `n` and intrinsic growth rate `r`:

```
next.size <- function(n, r) {n*(1+r)}
```

Create a list to store computations

We now create a list (aka "vector") to store the computed population sizes. To allocate computer memory for this list, we create a column vector (i.e., a table or `matrix` with `steps + 1` rows and 1 column) filled with zeros for temporary placeholders:

```
n <- matrix(0,steps+1,1)
```

Note that we need a list with `steps + 1` entries to include the initial value (N_0).

Initialize, project, and plot the results

Store the initial population size `initial.size` in the first position of the list.

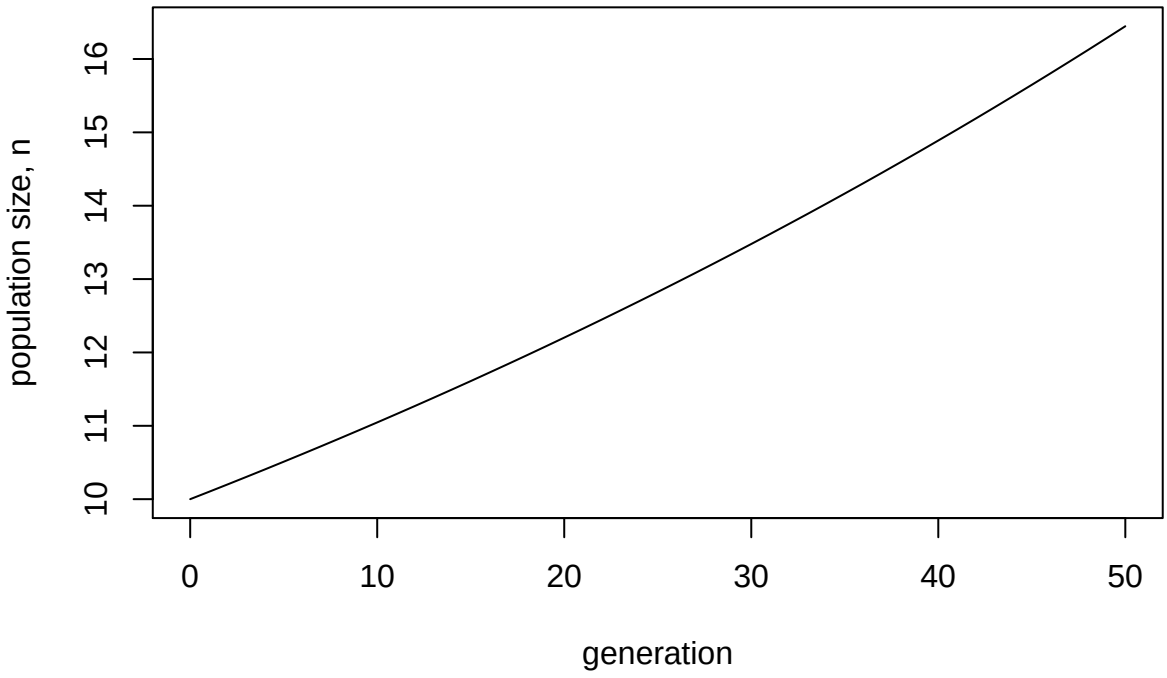
```
n[1] <- initial.size
```

Next, create a "`for`" loop to compute the sizes that projects the population sizes for `steps` time steps.

```
for(i in 1:steps){ n[i+1] = next.size(n[i],r)}
```

Finally, plot the results.

```
plot(0:steps,n,type = "l",xlab = "generation",ylab = "population size, n")
```



Stochastic Population Growth

We will create function that computes the population size at the next time step assuming each individual has a binomially-distributed number of descendants, with mean $1 + r$ and max number of offspring = `max.off`. This is a binomial distribution with parameters $N =$ `max.off` and $p = (1 + r)/$`max.off`. The function `rbinom(n,N,p)` is a built-in R function that generates a list of `n` binomial random variables with parameters `N` and `p`. The function `sum(l)` is a built-in R function that sums the elements in a list `l`.

```
next.size.rand <- function(n, max.off,r) {sum(rbinom(n,max.off,(1+r)/max.off))}

r <- 0.01 #expected intrinsic rate of growth or decline (a fixed parameter)
max.off <- 2  #this is the maximum number of descendants per individual
reps <- 5 # number of replicates
```

We next create a *table*—aka *matrix*—with (`steps`+1) rows and `reps` columns to hold the population sizes for `reps` replicates. Each column contains the population sizes, including the starting size, for each replicate.

```
nstoch <- matrix(0,steps+1,reps)
```

This code stores the initial population size in the first position (row) of each column.

```
nstoch[1,] <- initial.size
```

Now execute a double `for` loop that projects the population sizes for the `reps` replicates starting from `initial.size`. The results for the `j`th replicate will be stored in the `j`th column.

```
for(j in 1:reps){
for(i in 1:steps){ nstoch[i+1,j] = next.size.rand(nstoch[i,j],max.off,r)}
}
```

Finally, plot all the replicate trajectories together.

```
matplot(0:steps,nstoch,type="l",xlab="generation",ylab="population size")
```