Introduction to R
Lecture 2

EEB C119/C219 (Winter 2012)

Christopher C. Strelioff

Where are we going?

- Today
  - **for** loops
  - If else
  - Program flow
  - More scripts - discrete time logistic
  - Functions (next lecture)
- Last time
  - Variables and assignment
  - Vectors
  - Matrices
  - Plots
  - Workspace
  - Intro to scripts, **source** command

Programming basics

for loop

- **for** loops are a basic building block of all programming languages
- Allows for simple execution of repetitive calculations
- In R, can (sometimes) use vectorized statements instead
- (I think) a **for** loop can be easier to understand
- I will avoid **while** loops – watch out for ∞-loops!

for loop

A first example

- for loop strucure

```
> for (i in 1:4) {
+   cat("This time i is:", i , "\n")
+ }
```

```
This time i is: 1
This time i is: 2
This time i is: 3
This time i is: 4
```

- `i` is set to each element in vector `1:4` in turn
- Notice brackets: `(` and `)`, also `{` and `}`

for loop

A silly, but different example

- This is very general

```
> letters <- c("H","e","l","l","o","!")
> for (letter in letters) {
+   cat(letter)
+ }
```

```
Hello!
```

for loop

Geometric model

- Last class we used a vectorized equation to evaluate the geometric model: $n_{t+1} = Rn_t$

```
> R <- 1.2;                # set R
> n0 <- 10                 # set IC
> time <- 1:5              # I will do t=1 to t=5
> (nn <- (R^time)*n0)      # vectorized version
[1] 12.0000 14.4000 17.2800 20.7360 24.8832
```

- How do we do this with a for loop?

for loop

Geometric model – think about the process

- Start with n_0 and iterate $n_{t+1} = Rn_t$

$$
\begin{aligned}
n_1 &= Rn_0 \\
n_2 &= Rn_1 \\
n_3 &= Rn_2 \\
&\dots
\end{aligned}
$$

- This looks like the output of our example **for** loops.

for loop

Geometric model with for loop

- Use equation: $n_{t+1} = Rn_t$

```
> R <- 1.2                    # reproductive num.
> nn <- rep(0,6)              # initialize vector
> nn[1] <- 10                 # put IC at index 1
> # loop through t=1:5 to generate n(t)
> for (t in 1:5) {
+   nn[t+1] <- R*nn[t]
+ }
> nn
```

```
[1] 10.0000 12.0000 14.4000 17.2800 20.7360 24.8832
```

- Notice, the result is the same as the vectorized version, except $n_0 = 10$ is included in the vector

for loop

Varying reproductive number

- What if repropductive number depends on t: R_t?
- Use equation: $n_{t+1} = R_t n_t$

$$
\begin{aligned}
n_1 &= R_0 n_0 \\
n_2 &= R_1 n_1 \\
n_3 &= R_2 n_2 \\
&\dots
\end{aligned}
$$

- We can evaluate this with a **for** loop

for loop

Varying reproductive number

- Use equation: $n_{t+1} = R_t n_t$

```
> # vector of reproductive nums
> R <- c(1.0,0.9,1.1,0.75,1.5)
> nn <- rep(0,6)          # initialize vector
> nn[1] <- 10             # put IC at index 1
> # loop through t=1:5 to generate n(t)
> for (t in 1:5) {
+   nn[t+1] <- R[t]*nn[t]
+ }
> nn
[1] 10.0000 10.0000  9.0000  9.9000  7.4250 11.1375
```

Programming basics

if and **else**

- **if** and **else** let us do conditional execution
- If a specified *condition* is met $\rightarrow$ *execute appropriate code*
- A basic flow control in all programming languages
- Let's learn by example

if and else

- A first example

```
> R <- 1.2
> if (R > 1.0) {
+   cat('This was a good year, R>1 \n')
+ } else {
+   cat('This was a bad year, R<1 \n')
+ }
This was a good year, R>1
```

if and **else**

- A second example

```
> R <- 1.0
> if (R > 1.0) {
+   cat('This was a good year, R>1 \n')
+ } else if (R == 1.0) {
+   cat('This was an OK year, we survived (R=1). \n')
+ } else {
+   cat('This was a bad year, R<1 \n')
+ }
```

```
This was an OK year, we survived (R=1).
```

if and **else**

also a **for** loop

- What years were good from our vector of R's?

```
> # vector of reproductive nums
> R <- c(1.0,0.9,1.1,0.75,1.5)
> # loop through vector of R's
> for (Rval in R) {
+   if (Rval < 1.) { cat('* bad yr ') }
+   else if (Rval == 1.0) { cat('* ok yr ') }
+   else{ cat('* good yr ') }
+ }
```

```
* ok yr * bad yr * good yr * bad yr * good yr
```

Pseudocode

Write out the flow of your script

- Pseudocode is a useful tool for organizing programming
- Plan out the things your code needs to do
  - What variables need to be assigned?
  - What operations need to be done to variables, vectors?
  - What **order** do these things need to be done?
- Use comments to do this before coding
- Writing pseudocode is helpful in the learning stages

Pseudocode

Example: discrete time logistic model

```
# clear workspace

# set model parameters

# set number of timesteps

# initialize population vector, set IC

# set the plotflag to TRUE or FALSE

# use a for loop to iterate

# plot the results if PLOTFLAG is TRUE
```

Script

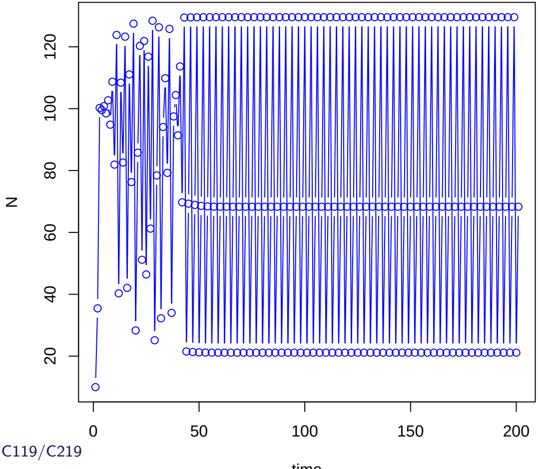
discrete time logistic model

```
## clear workspace
cat('\n','* Clearing Workspace','\n'); rm(list=ls())

# set model parameters
rd <- 2.83;K <- 100

# set number of timesteps
timesteps <- 200

# initialize population vector, set IC
N <- rep(0,timesteps+1); N[1] <- 10

# set the plotflag to TRUE or FALSE
PLOTFLAG <- TRUE

# use a for loop to iterate
for (t in 1:timesteps) N[t+1] <- N[t]*(1 + rd*(1 - N[t]/K))

# plot the results if PLOTFLAG is TRUE
if (PLOTFLAG) plot(1:(timesteps+1), N,
                   xlab="time", ylab="N",type="b", col='blue')
```

Script output

discrete time logistic model

```
* Clearing Workspace
```



Logistic dynamics

Some lessons from the examples

- Start of time series looked chaotic (random)
- However, dynamics settled into period-3 behavior
  - Careful of transient behavior
  - Are you interested in short- or long-term behavior?
  - This is an issue with **both** discrete time and continuous time models
  - Vary initial conditions, parameters – make lots of plots!

A pause

We have covered a lot of material ...

- You can do a lot now
- Practice and play to gain experience
  - Homework provides practice
  - Also, start thinking about your project
- Use discrete time logistic model
  - Change r_d, K, n_0
  - Is this too much work? Next week – functions!