Intermediate Programming in R
Session 2: Loops

Olivia Lau, PhD
Outline

When to Use Loops
Measuring and Monitoring R’s Performance
Different Types of Loops
Fast Loops
When to Use Loops

• Loops repeat a chunk of code

• Loops are necessary in C, C++, Fortran, and other compiled languages

• In most cases, loops can be avoided in R
  – Most math operations and many basic functions are already vectorized
  – Runs fast!
  – Saves typing, increases readability for long chunks of code

• Rule of thumb: If loop iterations are independent and fixed in number, the operation is parallel rather than sequential.

• But sometimes, you need loops
  – Gibbs and other MCMC samplers that rely on draws \([i - 1]\) in iteration \(i\)
  – Complex functions on data frames and other types of ragged arrays
Examples from Last Week’s Assignment

• Example 1
  
  co2 <- matrix(NA, nrow = 214, ncol = 10,
  dimnames = list(NULL, as.character(2002:2011)))

  for (i in 2002:2011) {
    co2[, as.character(i)] <- worldbank[worldbank$Year == i, "CO2"]
  }

• Example 2
  
  co2.vars <- c("Country.Name", "CO2")
  co2 <- worldbank[worldbank$Year == 2002, co2.vars]

  for (i in 2003:2011) {
    co2 <- merge(co2, worldbank[worldbank$Year == i, co2.vars],
      all = TRUE, by = "Country.Name",
      suffixes = c("", paste0(".", i)))
  }
Measuring Performance

- **Runtime via** `system.time()`
  - **User**: CPU time spent in R
  - **System**: CPU time spent in the OS on behalf of R processes
  - **Elapsed**: How long it feels for you

- **Sidebar**: Memory usage can be evaluated via `object.size()`
  - **Common problems**
    - Error: cannot allocate vector of size ....
    - Error: cannot allocate vector of length ...
  - Large objects also lead to slow performance
Check In 1

• How long does it take to execute the following R code?

```r
for (i in 1:1000) {
  rnorm(100)
}
```

• Answer

```r
system.time(for (i in 1:1000) {
  rnorm(100)
})
```

• How long does it take to execute the following R code?

```r
replicate(1000, rnorm(100))
```

• Answer

```r
system.time(replicate(1000, rnorm(100)))
```

• Which is faster on your system?
Storing Output from Loops

- Initialize the data object of the correct size outside of the loop

- Simplify output as much as possible
  - Only save what you need
  - Avoid mixed-class output

- Choose the simplest data structure
  - If output is all of one class and always the same size, use a vector, matrix, or array
  - If output is of mixed class, or varies in size, use a list or data frame

- Don’t “grow” a data object to accommodate loop output
Check In 2

• Create an empty vector of class logical and length 10

  Answer
  ```r
  my.vec <- vector(mode = "logical", length = 10L)
  ```

• Create an empty matrix with 40 rows and 5 columns, pre-populated with zeros

  Answer
  ```r
  my.mat <- matrix(0, nrow = 40, ncol = 5)
  ```

• Create an empty list

  Answer
  ```r
  my.list <- list()
  ```
Types of Loops

• **For** loops iterate over a fixed number of iterations

• **While** loops iterate until a logical statement returns **FALSE**

• **Repeat** loops execute until told to break

• Methods to break a loop
  – By design, insert a `break` in the loop
  – Or hit **Ctrl-C** repeatedly
For Loops

```r
for (i in 1:100) {
    # [... do stuff here with i ...]
}
```

- **i** is the counter
  - Inside the `{}`, i takes each value defined in the vector
  - Usually a continuous integer vector

- Vector can be anything, including
  - Series of integers (not continuous)
  - Character vector
Check In 3

• Create an empty matrix with 100 rows and 20 columns

my.mat <- matrix(NA, nrow = 100, ncol = 20)

• Let \( i \) be the counter in your loop. Iterating over columns, take a random draw of 100 values from a Poisson distribution with mean \( i \) in column \( i \)

for (i in 1:ncol(my.mat)) {
  my.mat[, i] <- rpois(100, lambda = i)
}
Testing For Loops

• Write the loop, then comment out the first and last line, like this:
  
  # for (i in 1:100) {
  #   print(i^i)
  # }

• To test the loop, set \texttt{i = 2} (or any other value in the range) and step through the loop

• If one iteration works, uncomment the first and last lines and run the loop

• If your loop fails, the \texttt{i} will equal the iteration on which the loop failed (useful for debugging)
While Loops

counter <- 1                      # To keep track of total iterations
while (j < 100) {                # The controlling statement
  # [... do stuff here ...]
  if (condition) {              # The conditional statement
    # [ ... update value of j here ... ]
  }
  counter <- counter + 1        # Update the counter
}

• While loops execute while the controlling statement is TRUE (e.g., the loop exits when the controlling statement returns FALSE)
  – If there is no conditional statement in the while loop and the counter \( j \) increments on every run, then a while loop is identical to a for loop
  – Putting the update condition in a conditional statement makes while loops different

• Since while loops execute for a variable number of iterations, it’s a good idea to use a counter and update it at the end of each loop
Repeat Loops

repeat {
    # [ ... do stuff ...]
    if (condition) {
        break
    }
}

• Repeat loops execute until explicitly told to break

• Extremely dangerous since they will run forever if the break condition is not met

• More useful to put break into for and while loops to “fail-fast”
Faster Loops

• Not an oxymoron!

• Choice of counter in for loops

• Avoid nested loops at all costs
Assignment

• Review solution set to Assignment 1 (contains several examples of loops)

• Reading for this week
  – From the course text, Paul Teetor’s *R Cookbook*: Chapter 5
  – “Efficiency Tips for Basic R Loop” by Svetlana Eden
    (http://biostat.mc.vanderbilt.edu/wiki/pub/Main/SvetlanaEdenRFiles/handouts.pdf)
  – Review the R help pages for
    • `system.time`
    • `unique`
    • `intersect`
    • `aggregate` (for Question 3)

• Problem set as assigned