Testing and Debugging

Tips

- Budget for testing and debugging, which usually take longer than coding.
- Find a reliable way to reproduce a bug (after restarting R).
- Shrink test case to a minimum via “binary search:”
  - cut data in half: e.g. `scan("numbersBug.txt", what=integer())`
  - cut code in half by, e.g., commenting it out (try it with `baby.dbinom.R`)
- Write code in small chunks. Save working versions (e.g. `hw3.R.22oct1315`, or learn git or other version-control software). Don’t write far past a working version.
- Test code in small chunks. “Test-driven development” calls for coding a function as follows:
  - Write tests of a function’s behavior first. Include
    * typical cases
    * boundary cases where the behavior changes
    * special cases, like vectors of length 0 or 1
  - Write a “stub” version of the function (e.g. that returns 0 or "" or NULL) and confirm that it fails the tests.
  - Implement function and debug until it passes tests.
  Don’t delete passed tests! They’ll be helpful for later bugs.
- Add an “assertion” to stop R if something is FALSE that you expect to be TRUE. Fail early!
  - In a test case for a function, you know the return value.
  - In a function’s first lines, confirm that arguments are legal.
  - In an "if ... elseif ... else" statement’s “else,” confirm the default condition.

`stopifnot(...)` stops unless each logical expression in ... is TRUE. e.g.

```r
stopifnot(x > 0) # did user give a positive argument as required?
stopifnot(isTRUE(all.equal(magnitude(3, 4, 5))))
... } else { stopifnot((0 <= score) & (score < 60)); grade = "F" ...}
```
- Use descriptive variable names to write “self-documenting” code. Typing now is easier than figuring out cryptic code later.
- Add comments to explain tricky code.
- Add print statements to display variables, especially function arguments. e.g.

  ```r
cat(sep="", " how.many(item=", item, ", n.max=", n.max, ")\n")
```

Retain your best print statements with a `debug=FALSE` or a `verbose=FALSE` parameter.

“The first 90% of the code accounts for the first 90% of the development time.
The remaining 10% of the code accounts for the other 90% of the development time.” –Tom Cargill, Bell Labs
• Simplify code. The only bug-free line of code is ________________________________.

• Don’t test for equality between two real numbers represented in a computer. Instead, use `isTRUE(all.equal(x, y, tolerance=(.Machine$double.eps ^ 0.5)))`, which tells whether the difference between x and y is small. e.g.

  49*(1/49) == 1
  49*(1/49) - 1
  isTRUE(all.equal(49*(1/49), 1))

• Demonstrate and explain your bug to a friend.

• Engage the problem, then get some sleep.

R’s debugging functions

• `traceback()` prints the call stack, or sequence of function calls, of the last uncaught error.

• A call to `browser()` in a function (or a click to the left of its line number in RStudio) stops its execution and starts a browser (“the debugger”) that allows line-by-line execution and inspection of the program state (e.g. try it with `baby.dbinom.R`):

  – `VARIABLE.NAME`: print value of variable (or look in Environment tab)
  – `c`: continue (note: RStudio has buttons for most of these commands)
  – `n`: next line (stepping over any function call)
  – `f`: finish current loop or function
  – `s`: step into function call
  – `where`: prints all active function calls
  – `Q`: quit browser and return to top-level prompt

In the browser, to see a variable with one of these names, use `print()` (or Environment tab).

• `debug(fun)` causes R to stop in a browser each time the function `fun` called:

  – `undebug(fun)` causes R to cease stopping in `fun`
  – `debugonce(fun)` causes R to stop on the next call only

• `trace(what, tracer)` inserts the code fragment `tracer` in the function `what`. e.g.

  – `trace(what=f, tracer=quote(expression))` runs `expression` when `f` is called (quote() prevents R from evaluating `expression` before passing it to `trace()` and `f()`.) e.g.

    ```
    mean(2 : 4+3)
    trace(what=mean, tracer=quote(cat(sep=" ", "x=" , x, "\n")))
    mean(2 : 4+3)
    ```

  – `untrace(what)` removes the tracing code