Profiling and Debugging

Rachel

2022-12-12

Debugging

General Advice

If a program or piece of code is not working as intended, there are a few approaches we can use to find the error:

- Understand what a particular error message is referring to. Error messages can often be cryptic and hard to interpret, so even if it does not reveal the source of the issue, it is helpful to understand the symptom of the error.
- Consider a minimal example which reproduces the result, which can greatly reduce the number of moving parts to consider. For this is it also to important to fix a seed if using any pseudo-random number generating processes.
- Attempt to pinpoint the source of the issue using a systematic process of eliminating potential causes for the problem.
- When the bug is fixed, include tests to find the issue again were it to re-appear

The `traceback()` Function

This is one of the easiest things to do when an error is first encountered. If we are using functions which have other functions nested inside them, the `traceback()` function will let us know in which layer the error is generated.

For example, we may have the following error:

```
check_n_value <- function(n) {
        if(n > 0) {
                stop("n should be <= 0")
        }
}
error_if_n_is_greater_than_zero <- function(n){
        check_n_value(n)
        n
}
error_if_n_is_greater_than_zero(5)
```

```
## Error in check_n_value(n): n should be <= 0
```

Then, we can call `traceback()` to see that the `stop()` function was called inside the `check_n_value()` function

```
traceback()
```

```
3: stop("n should be <= 0") at #3
2: check_n_value(n) at #2
1: error_if_n_is_greater_than_zero(5)
```

The `browser()` Function

The `browser()` function opens an interactive environment with the following commands

- `n`: execute next line and step over function calls
- `s`: execute next line and step into function calls
- `f`: finish evaluation of current loop/function
- `where` : print stack of all active function calls
- `c` : exit debugging environment and continue execution
- `Q` : exit debugging environment and execution, return to top level prompt

Returning to the previous example, we can insert the `browser()` function just before the point where we expect the error

```
check_n_value <- function(n) {
        if(n > 0) {
                browser()  ## Error occurs around here
                stop("n should be <= 0")
        }
}
```

Now when we run,

```
error_if_n_is_greater_than_zero(5)
```

```
## Error in check_n_value(n): n should be <= 0
```

The following window appears in RStudio

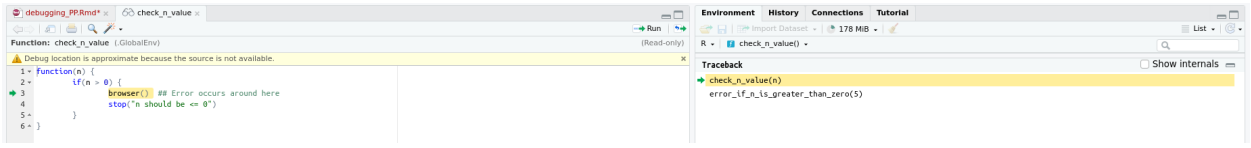

Figure 1: Result of Calling `browser()`