Testing

Naomi Tague

January, 2023

Testing

Top 12 Reasons to Write Unit Tests - Burke and Coyner (Java programmers)

- Tests reduce bugs in new features
- Tests reduce bugs in existing features
- Tests defend agains other programmers
- Testing forces you to slow down and think
- Testing makes development faster
- Tests reduce fear

Model developers now often use software to help them automate the testing process Re-uses tests - makes it efficient to repeat many tests as you develop and modify the code

Particularly helpful when you have multiple modules (as in our mangrove example)

This type of software is available for R, Python, C etc.

Testing in R

In R, *testthat* library

Install and add this library

- need devtools and "testthat" libraries
- test_dir("name") :runs all tests in the "name" subdirectory (all files beginning with the word "test"
- test_file("name"): runs all the tests in a file called "name"

What is a test?

Testing in R uses expectation

Expectation

functions that you can use to make sure your code is working the way you think it should be working

- expect_true(condition)
- expect_equal(value, value)
- expect_match(string1, string2)
- expect_null(value)
- expect_length(object, n)

```
# runs a test on a statement and checks to see if it is correct
# this one should work
expect_equal(2+2, 4)
# this one should 'throw and error'
expect_equal(2+5, 4)
```

```
## Error: 2 + 5 not equal to 4.
## 1/1 mismatches
## [1] 7 - 4 == 3
```

```
# we can also use expect_true
# works with variables two
a = 200
expect_true(a > 2 )
a = 0
expect_true(a > 2 )
```

```
## Error: a > 2 is not TRUE
##
## `actual`:   FALSE
## `expected`: TRUE
```

```
# matches
fish1 = "salmon"
fish2 = "trout"
expect_match(fish1, fish2)
```

```
## Error: `fish1` does not match "trout".
## Actual value: "salmon"
```

```
# length
a = seq(from=1, to=10)
```

```
expect_length(a, 10)
expect_length(a, 11)
```

```
## Error: `a` has length 10, not length 11.
```

Expectation

We can use expectation with functions

Quickly and repeatedly test that they are working as expected as you develop code

```
source("../R/compute_NPV.R")

futurevalue = 100
expect_true(compute_NPV(value=futurevalue, time=10, discount=0.01) > 0)
futurevalue = -100
expect_true(compute_NPV(value=futurevalue, time=10, discount=0.01) > 0)
```

```
## Error: ... > 0 is not TRUE
##
## `actual`:   FALSE
## `expected`: TRUE
```

```
# what if function returns a list?
# we can use the buit in summary function to see how this works
testdata = rnorm(mean=25, sd=3, n=100)
summary(testdata)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   17.61   23.01   24.73   24.94   26.82   33.32
```

```
# it is actually returning a list, but what are the names?
names(summary(testdata))
```

```
## [1] "Min."    "1st Qu." "Median"  "Mean"    "3rd Qu." "Max."
```

```
# now we can specify them
summary(testdata)["Max."]
```

```
##     Max.
## 33.32251
```

```
# and use in a test to make sure this function works -...hmm
# well the max should be greater than the min right?

expect_true((summary(testdata)["Max."]-summary(testdata)["Min."]) > 0)
```

More on testing functions

Lets create a function with an error

We want a function that computes total rainfall and total recharge from a daily time series of rainfall

Daily recharge of water to soil after rain is:

recharge = daily rainfall - interception

inputs: time series of daily rainfall (mm), single value interception (default 1mm)

outputs: totalrainfall (mm), totalrecharge (mm)

```
# Implement function
source("../R/compute_recharge.R")

# lets do some simple tests
# if it never rains for 730 days then both total rainfall and recharage # should be zero

raindata = rep(0, times=730)
expect_that(effective_rain(raindata)$totalrain, equals(0))
# so far so good, now for recharge

raindata = rep(0, times=730)
expect_that(effective_rain(raindata)$totalrecharge, equals(0))
```

```
## Error: `x` not equal to `expected`.
## 1/1 mismatches
## [1] -730 - 0 == -730
```

```
# what went wrong???
```

The art of testing

It is good practice to create tests for functions (especially useful if you are going to have multiple functions and/or multiple developers)

It make sure that the function returns what you think

Recall when we designed contracts at the beginning of class, testing is a way to make sure the program is honoring the contract

You need to be creative here - test for things you know will work (quick way to find bugs)

Usually we want to run multiple tests..

Testing in R

A **test** runs a bunch of **expectation** functions and returns a message

Format

test_that(description, series of expectations)

We usually store these in a subdirectory called **tests**

Example *test_compute_NPV.R*

Using tests

What we really want is to have tests that we **always** run, everytime we change something

Store multiple test in a *name.R* file store these in these in the *tests/testthat* directory

Two options

To run all tests in a file, use *test_file*

To run all tests in a directory,use *test_dir* (you can also do this from Build menu)

Goal: Make tests for each function in your package We will talk more about making packages next week

```
source("../R/compute_NPV.R")
# put your test in a file
test_file("../tests/test_compute_NPV.R")


# run all tests
test_dir("../tests")
```

Spring Summary Function

A more complex function examples - a function that extracts information about spring temperature from a long time series of daily climate data

Review the tests and *spring_summary* function for next class These are in

ESM_262_Examples

PLEASE pull most recent version - I had to make some changes to *spring_summary*

More on Testing

A good source for more expectation code and ways to do testing

testing

Assignment

For this assignment work in *pairs* - and submit as a group

- Design a function - you can pick any subject and you can even make up the equations as long as they conceptually make sense
  - *Make sure it has at least 2 inputs and 1 parameter (ideally more) and at least 2 outputs*
  - *Code your function in R, save as a R file in subdirectory called R*
  - *Make sure you include documentation (both at the top as we've shown in past examples, and inline)*
  - *Include some error checking*
- In an Rmarkdown file, generate some data for 2 of the function inputs
  - *use a for loop to run the function for the data*
  - *repeat the "looping" using something from the "purrr" package*
  - *Graph results (you can decide what the most interesting way to graph - you just need to make one graph)*
- Write at least 2 tests for your function; store in a separate test file
- Put this in a git repo that includes an R subdirectory a tests subsdirectory and an Rmarkdown file and submit the link on Gauchospace (or you can put all of this in a new git repo or a subdirectory of an existing git repo)

Submit the link to repo (and name of subdirectory if part of a larger git project) as a group with your partner,

You will have to fill out the group select on Gauchospace first before you submit

Grading Rubric

- Created a function
  - *at least 2 inputs, 1 parameter and 2 outputs (it can have more) (5pts)*
  - *includes error checking (5 pts)*
- Rmarkdown that generates test data and runs the function
  - *data generation (5pts)*
  - *run function multiple times with a loop (5 pts)*
  - *run function multiple times with something from purr (pmap) (5pts)*
  - *Graphs results (5 pts)*
- Test file
  - *a file with two tests (10pts)*

Good programming practices (10 pts) Total 50pts

Also review spring_summary.R and tests of that function from ESM_262_Examples