Testing Testing!

Obviously we all test our code 😊 but can we do it more efficiently? This talk is a brief introduction (I’m no expert!) on how to write formal, automated tests.

Helen Kettle
Mathmo Meeting sept 2021
Testing...

• We all know why we need to test our code – especially for mathematical models which may be 1000s of lines of code

• How do we go about this?
  • The results look “about right”
    • Two wrongs can make a right!
  • Compare with data
    • this tests the whole thing but not the individual parts
    • Some data sets will not trigger various parts of a model
  • We often test a function when we write it but these tests are informal and easily lost
Unit tests – what?!

These can test one function at a time in a very simple way

A unit test tests the expected output from a function, i.e. we expect certain inputs to give a certain answers.

Example (paired down!)

\[
f(x) = function(x)\{2*x\}
\]

\[\text{expect_equal}(f(3),6)\]

\[\text{expect_true}(f(1)>1)\]

\[\text{expect_length}(f(1),1)\]

\[\text{expect_gt}(\text{length}(f(c(1,2,3))),1)\]

\[\text{expect_error}(f(‘x’))\]
Unit tests – why?

• Unit tests are stored formally and run in automated way – this stops you endlessly repeating stuff you’ve already done

• They can all be run at once or separately

• They can be tested for extremes i.e. when the inputs are not what you expect them to be – writing unit tests can be combined with adding error-catching to your code

• If we come to add a feature or reorganise/restructure (i.e. refactor) our code we can run our unit tests and be safe in the knowledge we haven’t broken it all!

• Writing tests often helps you to massively improve your code as it forces you to break your code down into testable chunks.

• When you come to share your code (hello, Open Science!) you can feel more confident about it!
Unit tests – how?

• I will focus on unit testing in R but the principles will apply to other languages

• Resource: https://r-pkgs.org/tests.html

• In R there is a library called testthat – install.packages(‘testthat’)
• This allows you to write and run your unit tests

• I think creating the folders etc can all be automated (use package called “usethis”) but I do it by hand as I’m a control freak
  - you need a folder called ‘tests’ and within that a folder called ‘testthat’ and then you put in any number of files which contain any number of unit tests.
Organising

Project1

Code
- R
- Shellscripts
  - main.R
  - setOfFunctions1.R
  - setOfFunctions2.R
  - setOfFunctions3.R
  - setOfFunctions4.R
  (for running in batch mode)
- tests
  - testthat
  - testFunctions1.R
  - testFunctions2.R
  - testFunctions3.R
  - testFunctions4.R
  (for analysing/plotting model output)

Data

Results

Papers
- Text
- Figures
testFunctions1.R

library(testthat)
source('./../R/setOfFunctions1.R')

X=1
Y=2
test_that('function1',{
    expect_equal(f1(X,Y),-2)
    expect_length(f1(X,Y),1)
})

Z=3
test_that('function2',{
    expect_equal(f1(X,Y,Z),6)
    expect_true(f1(X,Y,Z)>0)
})

You can then simply run this script i.e.

setwd('testthat')
source('testFunctions1.R')

When you run the tests you get a message and the details for any failed tests
Examples of expectations you can test

https://testthat.r-lib.org/reference/index.html

**Objects**

- `expect_equal()`
- `expect_identical()`
- `expect_type()`
- `expect_s3_class()`
- `expect_s4_class()`

**Vectors**

- `expect_length()`
- `expect_lt()`
- `expect_lte()`
- `expect_gt()`
- `expect_gte()`
- `expect_named()`
- `expect_setequal()`
- `expect_mapequal()`
- `expect_true()`
- `expect_false()`
- `expect_vector()`

**Side-effects**

- `expect_error()`
- `expect_warning()`
- `expect_message()`
- `expect_condition()`

Does code return the expected value?

Does code return an object inheriting from the expected base type, S3 class, or S4 class?

Does code return a vector with the specified length?

Does code return a number greater/less than the expected value?

Does code return a vector with (given) names?

Does code return a vector containing the expected values?

Does code return TRUE or FALSE?

Does code return a vector with the expected size and/or prototype?
Functions for running tests

Run tests

**auto_test()**
Watches code and tests for changes, rerunning tests as appropriate.

**auto_test_package()**
Watches a package for changes, rerunning tests as appropriate.

**describe()**
describe: a BDD testing language

**test_file()**
Run all tests in a single file

**test_package()**
Run all tests in a package

**test_check()**
Locate file in testing directory.

**test_local()**

**test_path()**

**test_that()**
Run a test

**use_catch()**
Use Catch for C++ Unit Testing
Summary

• You don’t need to be building an R package to formally test your code

• You can write as many or as few tests as you like

• You can run them whenever you like

• You can automate all this using testthat, devtools, usethis
  • This will make it easier than I have shown but it’s good to understand the fundamentals I think!☺️
  • I often make plots in my test files too but would comment these out for packaging

• There are a lot of online resources – google: ‘unit tests R’, ‘testthat’, ‘creating packages in R’ etc.

• Caveat – I am not an expert (just a self-taught hacker!) – ask David and Bram for better info!

• Happy testing!