An Introduction to Developing R Packages

Women in Statistics and Data Science, 2017

October 20, 2017

Christina Knudson, Ph.D.	Lindsey Dietz, Ph.D.	Haema Nilakanta
University of St. Thomas	Federal Reserve of Minnesota	University of Minnesota

We have produced this guide and the R package wisdom as our contribution to the Women in Statistics and Data Science conference. We hope to help guide women through the basic steps of R package production so that women's R functions can be packaged neatly and shared easily. The electronic (pdf) version of this guide with hyperlinks can be found at `https://github.com/knudson1/WSDS2017`.

1 Before Starting

1.1 Design Document

Before you even touch your computer, develop a clear vision of your project. Put this vision and all the nitty gritty details into a "design document." The design document is the document you will refer to every time you sit down to program your package. Ideally, your design document will be so thorough that you will only think about programming according to that document and you will not need to think about statistics or data science anymore. The design document is essential for long-term projects so that you can communicate to your future self. A design document is also essential if you are working on a team: it will ensure that one function can call another seamlessly and with no mismatching arguments, even if the functions are written by different people.

A design document should have an overview of the purpose of the package as well as all the details. Decide exactly what you want your package to do. Then, carefully plan out each function that you will include in your package. For each function,

- write down all equations that function relies on.

- decide the inputs and outputs, as well as the properties of each. For example, will the input be a matrix, vector, or scalar?

- decide whether you will provide any defaults for the arguments. If you will, select the defaults.

- write pseudocode and try to imagine if there is anything tricky you need to consider.

- decide whether it will be available to the user or not. You might write helper functions that will be called by other functions. Users might not see these helper functions, and you will not need to write documentation for them later.

We also highly recommend you brainstorm tests and include these in the design document. You want to test your code every step of the way to ensure every command and function is doing what you think it is doing. Testing both small chunks of code and entire functions will help you isolate bugs or other mistakes. The specific tests you use will depend on the goals and functions of your package. As an example, if you are calculating the value of a function and its first and second derivatives, you can use the method of finite differences to check that the derivatives are in the right neighborhood.

1.2 Version Control

We highly recommend you use version control while you work. This will help you track your changes and revert to previous versions of your project. Some methods of version control also make collaboration smoother. Our preferred method of version control is git. Free public Github accounts are available for your remote repository.

2 Creating the Package

When you inspect the `wisdom` folder, you will find three folders (`tests`, `R`, and `man`) and two files (`DESCRIPTION` and `NAMESPACE`).

2.1 R Files

2.2 Documentation

Each function that you export (make available to users) must be documented. The package skeleton ...

2.3 Description

2.4 Namespace

In your namespace file, use `export` and list the function(s) in the package that you want to make available to users. This is necessary to make your function(s) available to users. The `wisdom` package has the most basic namespace possible, as it exports a single function, as shown below.

```
export(mPower)
```

If your package calls any other packages, you must `import` that package by listing it in the namespace. In the example namespace below, the function depends on R package `glmm` and so it imports it.

```
export(mPower)
```

```
import(glmm)
```

For a basic package, this is all you need to know about namespaces. If you are more advanced (e.g. writing S3 or S4 methods, calling C from R), you can find more details on the namespace file at Writing R Extensions.

2.5 Tests

Ensuring that your package does what you think it does is essential. You have written your functions generally, but you can test the functions by comparing them against specific examples. In the wisdom package, we wrote `mPower` to calculate x^m for any x and any m. To check that it is calculating these powers correctly, we choose a few specific examples and check the `mPower` results against the results we calculate ourselves.

When you write these tests, try to think creatively. It is easy to imagine testing the square of a positive number. What other values of x and m might someone enter? They can enter any real x, so we test $x < 0$. Additionally, the easiest power to imagine is integer m, but we do not require that m is an integer. The test in the wisdom package also checks the function for a few values of m between 0 and 1.

3   Checking the Package

When you are satisfied with your code, documentation, description file, and namespace file, it is time to build and check the R package. The first step is to create a "tarball," a compressed version of your package. To do this, navigate to the parent directory of your R package and type the following:

```
R CMD build wisdom
```

The time needed to build your package will increase as you add functions and tests to your package. Next, you will check your package by checking the tarball you have created. Type

```
R CMD check wisdom_1.0.tar.gz
```

to check your package. The number in the tarball name will depend on the version number of your package. The wisdom package is version 1.0, as the description file shows, so the tarball name includes 1.0.

4   Additional Considerations

4.1   Computational Stability

Before programming, it is wise to look at each equation in your design document and consider the computational stability. We cover a few items that we have used in our own R package production, but this list is by no means comprehensive.

4.1.1   Big Numbers and Close-to-Zero Numbers

If you are working with large numbers or numbers that are close to zero, you will need to take care because computers can keep track of only so many digits. Once a number is large enough, computers cannot

distinguish it from ∞. Similarly, once a number gets close enough to zero, computers cannot distinguish it from zero. A solution to this problem is to work with the log of a number rather than the number itself.

4.1.2   Log of Close-to-One Numbers

As you probably know, the log of 1 is 0. If you choose a number close enough to 1 and try to take the log using `log`, R will output 0. If you prefer a more accurate estimation of the log, use the function `log1p`, which uses a Taylor series expansion around 1.

4.1.3   Catastrophic Cancellation

Consider the variance calculations

$$\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

and

$$\left[\frac{1}{n}\sum_{i=1}^{n}x_i^2\right] - [\bar{x}]^2 \, .$$

If we consider the equations in a purely mathematical sense, we know these two calculations will yield identical results. However, when we program the variance calculation, the former is more stable due to the order of the summation and subtraction.

In the latter calculation we sum n items for one summation and n items for the other. Recall R can keep track of a limited number of digits for each of these sums. If the data set is large, each of these sums could be quite large. If variance is low, the first sum may be just barely larger than the second. If we have a large data set with low variability, we will lose most of our digits when we subtract one sum from the other. Suddenly losing many digits is called "catastrophic cancellation."

In the former calculation, we first calculate the mean and then we take n differences, square them, and sum them. If the data set is large but the variance is low, we will be taking many differences and many of these differences will be small. When we square them and sum them, we add each term's contribution one at a time. That is, the sum will inch up to produce the variance, rather than crash suddenly due to catastrophic cancellation.

4.2   Submitting to CRAN

If you intend to submit your package to CRAN, you will need to alter your check slightly to meet CRAN's specifications. Build your tarball as usual and check it with the following:

```
R CMD check wisdom.1.0.tar.gz --as-cran
```

When your package check returns no errors, check your package on other platforms. For example, you can use WinBuilder to check your package on the Windows platform. Once you are confident your package produces no errors, you may upload the tarball to CRAN. Do not upload a package that cannot pass `R CMD check`!

4.3   Licenses

You may choose from a variety of licenses. On one end of the spectrum is the "MIT" license. If you select this license, anyone can take any/all of your code to produce software, which they can then sell. The aster package uses this license.

Another license, the GPL-2 license, allows others to copy your code but requires any copy-cats to provide users with the same rights and freedoms the copy-cat received from your work. That is, the copy-cat cannot choose a license that is more restrictive than your own. The glmm package uses this license. The author selected the license because she wanted others to be able to use her code and ideas, but did not want others to profit off it.

4.4   Vignettes

Every R package produces a manual, but the helpfulness of the manual is limited due to the layout of the manual. Functions are listed in alphabetical order rather than the order a user might use them in. A vignette is a document that walks the user through one or more examples to demonstrate how to use the package. An example of a vignette can be found on `cknudson.com` or at `https://cran.r-project.org/web/packages/glmm/vignettes/intro.pdf`.