Making a package from base R files

John C. Nash

11/06/2021

Background

This article tries to explain an approach to developing alternative versions of functions which are in the distributed base of R. Our interest was in developing improvements to the nls() function and related features in R as part of a Google Summer of Code project for which Arkajyoti Bhattacharjee is the funded student. However, nls() has many tentacles involving a number of files and functions that may or may not be called as `nls()` is executed.

Part of the difficulty in carrying out such development of alternative versions is that one needs to be able to execute the new variants in parallel with the existing ones. A heavy-effort approach would be to have separate full sets of R code and build each system and run them separately. That is, we could have two versions of R.

Attempts to provide R functions to switch between the sets of working functions – e.g., `ReplaceNLS.R` and `RestoreNLS.R` – were not promising, and not pursued.

Duncan Murdoch suggested, and generously provided an almost-complete prototype of, a package of interlocked functions that provide the set of `nls()` capabilities. This package `nlspkg` has been cloned to `nlsalt` which is being modified to provide the new capabilities or new expression of the code. Moreover, these capabilities, such as a function `Anyfn()` can be tested side by side by use of the double colon syntax, namely,

```
nlspkg::Anyfn() # existing functionality or code
# and
nlsalt::Anyfn() # New functionality of code
```

Packaging from base R files

Duncan Murdoch described his process of developing the prototype of `nlspkg`:

```
Identify the main R functions I wanted in the package, and copy that
code into ./R.

Identify the C/Fortran functions that are referenced, and copy the
source for that into ./src.

Try to build, and get lots of errors about missing functions. Find
those and copy as above.

There was one case (the `%||%` operator) where I thought the code used
an unexported object from utils; the base-internals.R function just used
::: to get it, but a better solution would be to copy the source (which
is pretty trivial), and it turns out, is what stats did (the duplicate
source is in AIC.R).
```

```
Some files contained functions or methods that weren't relevant to us
(e.g. confint.lm).  I commented those out so I didn't need to worry
about the functions they used, and imported the generic (i.e. confint)
from stats.

In the C code, we were lucky because the nls code only made use of one C
function that's not in the API:  R_NewEnv. (That was called from
numeric_deriv.) There's an R function to do that, so I called it and
passed the result in.

Finally, once it would build, I tried running R CMD check.  This
identified all the missing help pages, which I copied over.  They didn't
need any modifications.

Because nls isn't really base functionality, all of this was a lot
easier than if I had tried to move something that works more with the
internals.  That's probably true for most functions in stats, but would
not be true for other base packages (except datasets).
```

After I downloaded the ZIP file of Duncan's package from his Github repository, I built and checked the package. There were two WARNING messages:

```
* checking DESCRIPTION meta-information ... WARNING
Non-standard license specification:
  GPL-2, GPL-3
Standardizable: FALSE
* checking for missing documentation entries ... WARNING
Undocumented code objects:
  'C_bvalus' 'C_lowesp' 'C_lowesw' 'C_nls_iter' 'C_numeric_deriv'
  'C_port_ivset' 'C_port_nlminb' 'C_port_nlsb' 'C_pppred' 'C_rbart'
  'C_setppr' 'C_setsmu' 'C_smart' 'C_supsmu' 'anova.nls'
  'anovalist.nls' 'coef.nls' 'confint.nls' 'deviance.nls'
  'df.residual.nls' 'fitted.nls' 'formula.nls' 'logLik.nls' 'nlsModel'
  'nlsModel.plinear' 'nls_port_fit' 'nobs.nls' 'plot.profile.nls'
  'port_cpos' 'port_get_named_v' 'port_msg' 'port_v_nms' 'predict.nls'
  'print.nls' 'print.summary.nls' 'profiler' 'profiler.nls'
  'residuals.nls' 'summary.nls' 'vcov.nls' 'weights.nls'
All user-level objects in a package should have documentation entries.
See chapter 'Writing R documentation files' in the 'Writing R
Extensions' manual.
```

The first WARNING can be overcome by changing the License line in the package DESCRIPTION file from

```
License: GPL-2, GPL-3
```

to

```
License: GPL-2 | GPL-3
```

as noted in https://www.r-project.org/Licenses/.

The warnings about documentation come from two sources:

- many of the `nls()` features parallel generic functions such as `residuals`, `predict`, `coef`, and so on, so are generically documented in R.
- the rest of the "undocumented" functions are C functions (identifiable as their names have underscore characters in the names). It is these functions that our "Improvements to nls()" project aims to replace where sensible to do so with R functions. They may then be documented as needed.

It is worth mentioning that the NAMESPACE file provided by Duncan is an important part of the packaging. It could be worthwhile to document the elements in that file needed for a package such as `nlspkg`.

Developing an alternative set of functions and alternative package

The alternative package is being developed by replacing calls to C functions with R work-alike functions. A preliminary example that already is set up and has passed preliminary checks is the function `numericDeriv()` from the file `R/nls.R` with called elements from the file `src/nls.c`. Note that some code needs to be commented out or deleted from the C file(s) and headers to avoid errors when building and checking the alternative.

To prepare the `nlsalt` package a directory of that name was created and the contents of the `nlsalt` directory copied over. The DESCRIPTION file was edited to provide the `nlsalt` name and the result checked with build and CHECK.

At this stage, we can begin to substitute for calls to C code and remove that code once replacement R material is present. Frequent build and CHECK cycles are helpful.

For Rstudio users, note that each package directory needs its own (name).Rproj file, created using Rstudio's "File/New Project" etc. dialog. However, the package directories with their Rproj files can be within the umbrella version control project `improvenls` for the Git version control. (We are using a Gitlab repository.)

Windows issue

While all the above worked well in a Linux environment, when we tried running the build and CHECK in a Windows environment, we found we got errors. (I normally run in Linux Mint 20.1, but Arkajyoti Bhattacharjee uses Windows 10. Each of us has a VirtualBox VM for the alternative OS to allow for checking of our work.) There is a load error for functions from the BLAS (Basic Linear Algebra Subroutines) collection in a 32 bit architecture. This is revealed by the Rstudio "Build / Clean and rebuild" feature. Using the `CMD.exe` terminal window showed that `R CMD build` worked, but that `R CMD INSTALL` gave the load errors.

After some exchange of emails with others and with the R-package-devel mailing list, a Google search using terms suggested in the emails found the posting

https://stackoverflow.com/questions/42118561/error-in-r-cmd-shlib-compiling-c-code

This suggested that a file `Makevars.win` is needed in the `src/` directory of the package(s), and that this file should contain just one line

```
PKG_LIBS = $(LAPACK_LIBS) $(BLAS_LIBS) $(FLIBS)
```

I did, however, note that it is not clearly indicated that the file needs to be in `src/`, nor (until revealed by `R CMD check`) that the file must be `Makevars.win` and not `makevars.win`. There is a lot of material on `Makevars` in the Writing R Extensions manual, but these points do not seem to be presented.

Downstream benefit

The approach to replacing functionality in base R suggested here has a follow-on benefit should some or all of the improvements we hope to be able to make are eventually included in R's distribution files. That benefit is that the initial package can be renamed to, for example, `nlslegacy` so that R users can, should they need, use legacy capabilities. Indeed, from preliminary timings of some functions, it appears that pure R code runs 2-3 times slower than the mixed R and C versions. On the other hand, the pure R code in our early cases is much shorter and, we suspect, much easier to maintain. Furthermore, it simplifies the packaging of base R for alternative interpreter or compiler versions of the R language.

Acknowledgements

It is a pleasure to note the help and encouragement of Duncan Murdoch, Arkajyoti Bhattacharjee, Dirk Eddelbuettel and Brett Klamer.