New York City R-Meetup

Columbia University
March 4, 2010

Jay Emerson
Yale University
http://www.stat.yale.edu/~jay/
Talk resources: http://www.stat.yale.edu/~jay/Rmeetup/

Abstract

I’ll give an introduction to R’s package system and the process of creating, checking, and building a simple package from scratch. I’ll touch upon the inclusion of C/C++ code and the use of basic S3 and S4 in packages. I hope to conclude by showing off Steve Weston’s brilliant foreach package, which provides a wonderfully flexible, portable, and easy-to-use framework for parallel computing. The examples and latter topics relate to package design in interesting ways.

Contents

1 About this document 2

2 Getting started: building a build environment 2

2.1 Linux .......................... 3
2.2 Mac .................................. 3
2.3 Windows .................................... 3

3 A first package 6

3.1 Creating the package skeleton ............................................. 6
3.2 From a package skeleton to a package ..................................... 7

4 Getting into C with .C() 9

5 Getting into C with .Call() and Sexp 12

6 And much, much more 14

6.1 S3 and bcp ......................................................... 14
6.2 S4 and bigmemory ..................................................... 14
6.3 foreach and a revision of bcp ........................................... 14
6.4 Final comment: foreach with bigmemory ............................ 15

References 16
1 About this document

This document is intended to complement (and not completely duplicate) the talk, and was created using Sweave (which is included with R). For more information, see Friedrich Leisch’s web page:

http://www.stat.uni-muenchen.de/~leisch/Sweave/

The short story: you use R to flush your “master copy” of the document (a .Rnw file) through Sweave, producing a .tex file which is then processed using L\TeX. In this way, you can include R code in your document and, automatically, the results of the code. Even plots are easy to integrate. The file (Rmeetup.Rnw) used to produce this document (Rmeetup.pdf) is available along with other materials related to the Meetup, at

http://www.stat.yale.edu/~jay/Rmeetup/

If you are interested in Sweave and look at Rmeetup.Rnw, please note that I’ve done a few unusual things because of the nature of this particular presentation. Please read the special comments at the top of the file.

2 Getting started: building a build environment

Everyone should know enough to start here [8]:

http://www.r-project.org/

After a click or two, I generally wind up at the mirror site hosted at Carnegie Mellon. Most of us have downloaded and installed a precompiled binary version of R at some point, but that won’t be sufficient for our purposes. We want a full build environment. Although this doesn’t require building R itself from source, I think this is a good way to ensure that the build environment is up and ready to go.

To begin with, everyone needs the R source tarball, R-2.10.1.tar.gz (or whatever newest and greatest release is available). This is available under “Source Code for All Platforms” on a page like

http://lib.stat.cmu.edu/R/CRAN/

You should also be able to reference R Installation and Administration [9], found in the “Manuals” area of CRAN, or directly at

http://lib.stat.cmu.edu/R/CRAN/doc/manuals/R-admin.html

Sections 2-4 of R Installation and Administration cover installation for Linux, Windows, and Mac respectively. In a nutshell (“help, help, I’m in a nutshell!”), you’ll do something like the following in some appropriate location:

me@mycomputer:$ ./configure
(lots of output)
me@mycomputer:$ make
(even more output)
Yes, certain things need to be in certain places, and the process has dependencies that can be platform specific (see below). It probably won’t work the first time, but a little common sense, attention to error messages, and careful reference to available documentation generally suffices.

2.1 Linux

Historically, users falling into this category wouldn’t need any advice. However, the Linux distributions are so friendly these days, I don’t want to completely take this for granted. Section 2 of *R Installation and Administration* may be helpful. You may need to add some flavor of \LaTeX{} to your system if it wasn’t installed by default. I don’t remember having any serious problems.

2.2 Mac

Section 4 of the *R Installation and Administration* is the place to start. There are two additional places to look for Mac help: the Mac FAQ, and a page hosted by Simon Urbanek down at AT&T:

http://lib.stat.cmu.edu/R/CRAN/bin/macosx/RMacOSX-FAQ.html
http://r.research.att.com/

There are some extra tools you’ll need (like Xcode Tools, sometimes installed by default, sometimes not). For others, please pay special attention to advice from the experts on versions – newer is not always better and may not work. I’m not a Mac guy, so I don’t have anything else to add to this section.

2.3 Windows

This is probably the “toughest” of the three platforms for this stuff. So although my other laptop runs Linux, I thought I’d make myself do the Meetup from Windows. I apologize in advance for typos with regards to the use of `\` and `/`. Section 3 of *R Installation and Administration* is the place to start, and Section 8 of the Windows FAQ is also relevant:

http://lib.stat.cmu.edu/R/CRAN/bin/windows/base/rw-FAQ.html

These will quickly lead you to a site hosted by Duncan Murdoch:

http://www.murdoch-sutherland.com/Rtools/

You’ll need two things besides the R source tarball: Rtools (I used *Rtools211.exe* from Duncan’s page) and \LaTeX{} (specifically, MiKTeX from http://www.miktex.org/). You can take the basic MiKTeX installer (mine was basic-miktex-2.8.3582.exe). I’ll skip the Inno Setup installer, because I’m not interested in creating a stand-alone installer.

I’m using a new installation of Windows Vista 64-bit Ultimate. I made the following notes while doing the build of R from scratch, using only the three things listed above.
1. I started with MiKTeX, installing to C:\MiKTeX2.8 because I don’t like spaces in paths. I chose “letter” rather than “A4” along the way.

2. Next I installed Rtools, taking the default C:\Rtools directory. I added the extras to build R: TCL/TK, bitmap code, internationalization. I took the R source home C:\R, and will need to relocate things from here a bit later. I checked the box to allow it to edit the environment variable PATH for me, saving me the trouble.

3. I confirmed that my PATH variable seemed to be consistent with what is recommended (go to “System Properties: Advanced: Environment Variables”). I didn’t make any changes at this point (but remember this, later).

4. I copied R-2.10.1.tar.gz into C:\.

5. I opened a command window (or command prompt), with administrator rights, and this by default had me in C:\Users\jay. I typed cd C:\ to get where I wanted to be, the top level directory where I just put R-2.10.1.tar.gz. Unsure about this? Try dir and see if R-2.10.1.tar.gz is there.

6. tar xf R-2.10.1.tar.gz unpacks the package source to a directory R-2.10.1. This, by the way, is Rtools at work, getting Windows to behave like Linux.

7. In step 2 above, recall that I used C:\R for the “extras”? Maybe this was a mistake and there could be an easier way to do this. So I went in by hand and merged the contents of C:\R into my C:\R-2-10.1 tree. Perhaps if I’d unpacked tar xf R-2.10.1.tar.gz first, things would have worked with the location C:\R-2-10.1 back in step 2. Oh well.

8. At this point, I note that the INSTALL file in the top level directory recommends what I’m used to in Linux: configure, make, and so on, while the Section 3 of R Installation and Administration is telling me to do things in various subdirectories. I’m going to follow INSTALL, partly to make a point: this doesn’t always work the first time you try it.

9. Inside R-2.10.1 in the command prompt, I tried configure. Nope, didn’t work. If first you don’t succeed, try and try again.

10. Go into srv/gnuwin32 as instructed. I open MkRules with Windows Notepad. Disaster, no proper end-of-line characters. Opened with WordPad. Ok. Looked for stuff to edit. Unsure about the “overrides for making bitmapdll” so I left this. I didn’t make any changes at all, although this may have been a mistake (see step 16).

11. make all recommended.

12. Went to get a beer.

13. Didn’t work. The error: cannot stat /tmp/R644. Seems like a permissions problem. Interestingly, there is a comment about this in the Linux Section 2.1 of R Installation and Administration, but the same comment seems to apply here. So I created a folder C:\tmp.
14. Tried again *(make all recommended)*, after restarting the command prompt with administrator rights (just in case I’d forgotten to do that earlier).

15. Cool, seemed to work.

16. Tried *make bitmapdll* and it failed. Looks like I needed to work with those lines in *MkRules* after all (and I had *jpeg-8* and *libpng* instead of the defaults). Tried again but still failed, with a different error – it was not clear what the problem was. I quit. Hopefully not a problem, and everything else seemed to work.

17. Tried *make check-recommended*. Takes a while. Results kind of puzzling, I’ve never really understood this. There are some differences, perhaps in rounding, perhaps in precision. I’ve always seen this, on all platforms. So why bother checking? What is a “good” check? I suppose if there were a serious problem there might be a helpful notice.

18. I should note that the check triggered some extra *MiKTeX* installations. Fine.

19. Skipped *devel* and *all* checks. Tried *make manuals*, with no problem; *MiKTeX* did another download along the way, and it worked.

20. Went into the folder *R-2.10.1/bin* (using the mouse) and created a shortcut on the Desktop to the *Rgui* executable. Voila!

21. One more thing: I went back into “System Properties: Advanced: Environment Variables,” and added *C:/R-2.10.1/bin* to the end of the system *PATH*. This will be necessary later so that we can call *R* from the command prompt.

That did it for me. I’ve gone through this process 2-3 times in Windows. It never works the first time, but it has never been that difficult to recover and fix up the problem. Problems have *always* been my fault (usually because I didn’t read the documentation carefully enough).

One final note: my *R* isn’t displaying the HTML help files with working links. I only did a quick scan of some of the documentation (late last night), it appears that I should have built *R* with a *--html* flag. The default seems to be *--no-html*. I didn’t want to risk a problem for today by repeating the exercise. See Section 2.2 of *R Installation and Administration*. 
## 3 A first package

Please see *Writing R Extensions* [10] for the authoritative guide to package authoring,

http://lib.stat.cmu.edu/R/CRAN/doc/manuals/R-exts.html

Fire up R. Make sure you’re working in some reasonable folder, preferably with no spaces in the path (this used to be a deal-breaker – not sure now). Try the following:

```r
> getwd()
[1] "/home/jay/Desktop/Teaching/R-Meetup"
> ls()
character(0)
```

At this point, I’m doing little more than show off Sweave. Your working directory will differ, but the result of `ls()` should be the same; we need to have a completely empty environment at this point. In general, I recommend *never saving the environment when quitting R*. This causes more problems than you could possibly believe, and encourages poor programming and data analysis habits.

### 3.1 Creating the package skeleton

My first package is going to contain a single function, and this function should be the only thing in my environment. I then simply use the function `package.skeleton()` to get started:

```r
> babywhatis <- function(x) {
+       if (!is.data.frame(x)) {
+           x <- data.frame(x)
+           warning("Object coerced to a data frame.\n")
+       }
+       return(unlist(lapply(x, class)))
+  }
> ls()
[1] "babywhatis"
> package.skeleton("MyToolkit")
Creating directories ...
Creating DESCRIPTION ...
Creating Read-and-delete-me ...
Saving functions and data ...
Making help files ...
Done.
Further steps are described in './MyToolkit/Read-and-delete-me'.
```

That’s all for now: quit R.
3.2 From a package skeleton to a package

Get into the new MyToolkit folder and browse around. Obviously, read (and perhaps delete) Read-and-delete-me. I think one extra piece of advice is needed: edit DESCRIPTION. After all, you want to receive proper credit for this work of art, don’t you?

Section 1.1 of Writing R Extensions describes the package structure. We won’t need many of the gory details today. We need to:

1. Edit DESCRIPTION.

2. Edit the help files in the man folder (short for manuals or man pages).

3. Check and build the package.

Instead of editing on the fly, I’ll pull my edited files from Extras. Those of you with internet connections might be able to grab these from the web and work along,

http://www.stat.yale.edu/~jay/Rmeetup/Extras/

Next, we need to check the package. From your shell or command prompt, in the folder containing MyToolkit (and not from inside the MyToolkit folder itself), invoke R with the following arguments to check the package:

me@mycomputer:$ R CMD check MyToolkit
* checking for working pdflatex ... OK
* using log directory '/home/.../MyToolkit.Rcheck'
* using R version 2.10.0 (2009-10-26)
* using session charset: UTF-8
(lots of other stuff, omitted)

Pay attention to any warnings or errors. By far the most common problems seem to be simple syntax errors in the help pages. Try to fix any problems, and repeat the check. Iterate. Once the check has succeeded, you can build the package:

me@mycomputer:$ R CMD build MyToolkit
* checking for file 'MyToolkit/DESCRIPTION' ... OK
* preparing 'MyToolkit':
* checking DESCRIPTION meta-information ... OK
* removing junk files
* checking for LF line-endings in source and make files
* checking for empty or unneeded directories
* building 'MyToolkit_1.0.tar.gz'

That’s it! In Windows, we generally distribute package binaries, and an additional step is required (in Linux and on the Mac – I think – packages are usually built from source upon installation):

C:\Users\Jay\Desktop\Rmeetup> R CMD build --binary MyToolkit
(stuff omitted)
At this point, you can fire up R again and install the package (what follows is for Linux; work with the MyToolkit.zip “binary” in Windows):

```r
> install.packages("MyToolkit_1.0.tar.gz", repos=NULL)
```

Warning in install.packages("MyToolkit_1.0.tar.gz", repos = NULL) :
argument 'lib' is missing: using ...
* installing *source* package 'MyToolkit' ...
** R
** preparing package for lazy loading
** help
*** installing help indices
** building package indices ...
* DONE (MyToolkit)

The repos=NULL argument may be necessary to force the installation from the local package (rather than trying to find the package on CRAN). Finally, we can try it out:

```r
> library(MyToolkit)
> data(CO2)
> babywhatis(CO2)
```

```
Plant1  Plant2  Type  Treatment  conc  uptake
"ordered" "factor" "factor" "factor" "numeric" "numeric"
```

For comparison purposes, from YaleToolkit [2], with apologies for the wrapping of the output here:

```r
> library(YaleToolkit)
> data(CO2)
> whatis(CO2)
```

```
variable.name type missing distinct.values precision
1 Plant ordered factor 0 12 NA
2 Type pure factor 0 2 NA
3 Treatment pure factor 0 2 NA
4 conc numeric 0 7 1.0
5 uptake numeric 0 76 0.1
```

```
min      max
1  Mc1     Qn3
2 Mississippi Quebec
3 chilled nonchilled
4 95       1000
5 7.7      45.5
```
4 Getting into C with .C()

Let’s build an R function that takes a matrix and returns a vector of column minima, using C to do the real work. Yes, it’s a toy example, but I think it provides a reasonable foundation without being overwhelming. I’m willing to let the first C function ignore NA values at this point. I’ll start with the R code (trying to handle the NA values here):

```r
mymin1 <- function(x, verbose = FALSE) {
  if (!is.matrix(x)) stop("x is not a matrix.")

  # Question: why do the following need to be here, before
  # the next little block of code?
  numCols <- ncol(x)
  numRows <- nrow(x)
  ret <- rep(0, numCols) # Living dangerously?

  if (!is.numeric(x)) {
    x <- as.numeric(x) # Very interesting. Really.
    warning("Coercing x to numeric matrix.")
  }

  # My personal decision is to ignore NAs, and this does it... almost.
  # Why isn't it a bullet-proof solution?
  if (any(is.na(x))) {
    warning("Matrix contains one or more NA values.")
    x[is.na(x)] <- max(x, na.rm=TRUE)
  }

  if (verbose) cat("Before the C work, x[1,1] =", x[1], "\n")

  ans <- .C("Cmin1",
            x, as.integer(numRows), as.integer(numCols),
            as.integer(verbose), ret=ret)

  if (verbose) {
    cat("Done with C work... printing the result, a list:\n")
    print(ans)
    cat("Because I'm curious: x[1,1] =", x[1], "\n")
    cat("Returning the vector of minima only...\n")
  }

  return(ans$ret)
}
```
The C code

#include <stdio.h>

double VecMin(double *v, int n)
{
    int i;
    double ans = v[0];

    for (i=1; i<n; i++) {
        if (v[i] < ans) { ans = v[i]; }
    }

    return(ans);
}

void Cmin1(double *pMat, int *numRows, int *numCols, int *verb, double *pAns)
{
    int col, row;
    int nc = *numCols;
    int nr = *numRows;

    if (*verb==1) {
        printf("Hello, world, number of columns is %d.\n", nc);
        for (row=0; row<nr; row++) {
            for (col=0; col<nc; col++) {
                printf("Row %d, col %d, value %8.5f.\n", row+1, col+1, pMat[nr*col+row]);
            }
        }
    }

    for (col=0; col<nc; col++) {
        pAns[col] = VecMin(&pMat[nr*col], nr);
    }

    // This doesn't matter, it's just helping me make a point:
    pMat[0] -= 100;
}
OTHER STUFF

zzz.R

.onLoad=function(libname, pkgname)
{
    library.dynam("MyToolkitWithC", pkgname, libname)
}

.onUnload=function(libpath)
{
    library.dynam.unload("MyToolkitWithC", libpath)
}

NAMESPACE

export(mymin1, mymin2, babywhatis)
useDynLib(MyToolkitWithC)

After adding the previous code to the package (and renaming it MyToolkitWithC), I tried to check the package. It worked, more or less, but with two warnings:

...  
* checking if this is a source package ... WARNING
Subdirectory 'MyToolkitWithC/src' contains object files.
...
* checking for missing documentation entries ... WARNING
Undocumented code objects:
    mymin1
All user-level objects in a package should have documentation entries. See the chapter 'Writing R documentation files' in manual 'Writing R Extensions'.

The first warning I’ve seen for years, and it rarely causes problems; but watch out if you’ve edited one of your .h files, for example. The second is simply letting me know that I forgot to provide documentation for the new function mymin1. I point this out to have an excuse to demonstrate one of the lesser-known (but useful) features of the package management system:

> prompt(mymin1)

This generates a skeleton of the help page, mymin1.Rd, which then needs to be edited (I make an edited copy available in Extras/).
5 Getting into C with .Call() and Sexp

The R code

```r
mymin2 <- function(x, verbose = FALSE)
{
  if (!is.matrix(x)) stop("x is not a matrix.")

  # Note the change in the following... why?
  if (!is.numeric(x)) {
    x <- matrix(as.numeric(x), nrow(x), ncol(x))
  }

  if (verbose) cat("Before the C work, x[1,1] =", x[1,1], "\n")

  ans <- .Call("Cmin2", x, as.logical(verbose))

  if (verbose) {
    cat("Done with C work... printing the result, a vector here:\n")
    print(ans)
    cat("Because I'm curious: x[1,1] =", x[1,1], "\n")
    cat("Returning the vector of minima only...\n")
  }

  return(ans)
}
```

The C code

```c
#include <stdio.h>
#include <R.h>
#include <Rdefines.h>

double VecMinNA(double *v, int n)
{
  int i;
  double ans = v[0];

  for (i=1; i<n; i++) {
    if (!ISNAN(v[i])) {
      if (ISNAN(ans) || (!ISNAN(ans) && (v[i] < ans))) { ans = v[i]; }
    }
  }

  return(ans);
}
```
SEXP Cmin2(SEXP Rmat, SEXP verb)
{
    // Remember the following:
    // Rmat is a matrix of doubles in R
    // verb was an integer before; now a boolean R object

    int col, row;
    int nc = ncols(Rmat);
    int nr = nrows(Rmat);

    double *mat = NUMERIC_DATA(Rmat);

    SEXP Rret;
    PROTECT(Rret = NEW_NUMERIC(nc));
    // Rret[0]; // This will not work.

    // NUMERIC_DATA and NUMERIC_POINTER are both REAL(x) in Rdefines.h
    double *ret = NUMERIC_DATA(Rret);

    if (LOGICAL_DATA(verb)[0]==TRUE) {
        printf("Hello, world, number of columns is %d.\n", nc);
        for (row=0; row<nr; row++) {
            for (col=0; col<nc; col++) {
                printf("Row %d, col %d, value %8.5f.\n", row+1, col+1,
                        mat[nr*col+row]);
            }
        }
    }

    for (col=0; col<nc; col++) {
        ret[col] = VecMinNA(&mat[nr*col], nr);
    }

    // This now actually matters:
    mat[0] -= 100;

    UNPROTECT(1);
    return(Rret);
}

There is a Pandora’s box of topics we could discuss, including things you see above (ISNAN(), NUMERIC_DATA()) and other things you don’t see (ISNA(), NUMERIC_POINTER(), R_FINITE(), other data structures, and so on).
6 And much, much more

I like Robert Gentleman’s book *R Programming for Bioinformatics* [7], even though I think the name is unfortunate; I think it’s generally useful far beyond the Bioinformatics. His brief take on S3 and S4 is a good place for us to start in our brief discussion of object-oriented programming (from [7], page 67):

[In R there are essentially] two masters - reliability and ease of use. S3 is indeed easy to use, but can be made unreliable through nothing other than bad luck, or a poor choice of names, and hence is not a suitable paradigm for constructing large systems. S4, on the other hand, is better suited for developing large software projects but has an increased complexity of use.

6.1 S3 and bcp

I’m going to show a basic S3 example in the context of package bcp [3, 4, 5], something I did with my first grad student, Chandra Erdman. Frankly, a revision is overdue, as we may see later, but the use of S3 provides a nice example. The package implements the Barry and Hartigan (1993) product partition model for the classical change point problem [1] using Markov Chain Monte Carlo (MCMC).

6.2 S4 and bigmemory

We’ll look at the package bigmemory [6] (co-author Michael Kane), which plays to the strengths of S4. And you might find it useful for other reasons.

Multi-gigabyte data sets challenge and frustrate R users, even on well-equipped hardware. Use of C/C++ can provide efficiencies, but is cumbersome for interactive data analysis and lacks the flexibility and power of R’s rich statistical programming environment. The package bigmemory and sister packages biganalytics, synchronicity, and bigalgebra bridge this gap, implementing massive matrices and supporting their manipulation and exploration. The data structures may be allocated to shared memory, allowing separate processes on the same computer to share access to a single copy of the data set. The data structures may also be file-backed, allowing users to easily manage and analyze data sets larger than available RAM and share them across nodes of a cluster. These features of the Bigmemory Project (http://www.bigmemory.org/) open the door for powerful and memory-efficient parallel analyses and data mining of massive data sets.

6.3 foreach and a revision of bcp


If you go back and look at bcp.R, you’ll see the relative clunkiness of our optional parallel processing. IF someone can get NetworkSpaces running (not trivial, see http://www.stat.yale.edu/~jay/nws/ for example), then the bcp() runs the MCMC in parallel...
and combines the results. It only works with NetWorkSpaces; support for snow or Rmpi would require completely different code. It wasn’t bad at the time; now, it’s a dinosaur.

In contrast, I’ll show you the flexibility of foreach. The code becomes cleaner I think, and is completely portable. The same code will work in serial (no parallel processing required) or in parallel on whatever “parallel backend” is registered. In other words, you and I could have two completely different parallel environments (maybe I use multicore on my Linux laptop, and you use snow on your simple network of workstations), and the same code will work for each of us. The only difference: a single line or two up front registering the parallel backend.

Here’s a simple example of foreach():

```r
> library(foreach)
> ans <- foreach(i = 1:3) %dopar% {
+     return(c(i, i^2, sqrt(i)))
+ }
> ans

[[1]]
[1] 1 1 1

[[2]]
[1] 2.000000 4.000000 1.414214

[[3]]
```

By default, this is run sequentially on a single processor. However, if a “parallel backend” is registered, foreach spreads the work around the resources provided by the parallel environment. The resources could be available cores in an SMP environment (e.g. most of our laptops or desktops), or nodes of a cluster. If you really want to force sequential execution, the %do% operator is available.

Currently available parallel backends include: doMC (Linux and Mac only), doSNOW, and doMPI. My revision of package bcp will demonstrate the use of foreach, and we’ll test it out using doSNOW. I would argue that a wide range of current and future packages should take advantage of foreach() as a painless alternative to many loops or apply-like statements.

### 6.4 Final comment: foreach with bigmemory

Package bigmemory supports matrices in shared memory and works brilliantly with foreach. Imagine a time-intensive analysis (suitable for operations done in parallel) of a large data set (perhaps even larger than RAM). With bigmemory, massive matrices (perhaps in RAM, perhaps filebacked) can be shared across processes without creating costly duplicates of the data; foreach makes the code development and portability a snap. For more information about any of these topics, please feel free to drop me an email (john.emerson AT yale.edu); papers (and a vignette) are in the pipeline, and feedback is always appreciated.
References


