from the tree). The use of handlers allows us to build quite simple filters for extracting data. For example, suppose we have measurements of height taken at monthly intervals for different subjects in a study, but that potentially different numbers of measurements were taken for each subject. Part of the dataset might look like

```
<record><id>193</id>
  <height unit="cm">
    <real>140.5</real>
    <real>143.4</real>
    <real>144.8</real>
  </height>
</record>
<record>
  <id>200</id>
  <height>
    <real>138.4</real>
  </height>
</record>
```

Now, suppose we want to get the records for the subjects which have more than one observation. We do this by specifying a handler for the `<record>` tags and counting the number of sub-nodes of the `<height>` element.

```
xmlTreeParse("data",
  handlers=list(record=function(x,...)
    ifelse(xmlSize(x[["height"]]) > 1, x, NULL)))
```

This will discard the second record in our example above, but keep the first. We should note we can also use XSL to filter these documents before reading the results of the filtering into S. More sophisticated things can be done in S, but XSL can be simpler at times and avoids a dependency on S. The good thing is that we have the luxury of choosing which approach to use.

The package also provides other facilities for reading XML documents using what is called event or SAX parsing which is useful for very large documents. Also, users can create XML documents or trees within S. They can be written to a file or a string by incrementally adding XML tags and content to different types of "connections".

The future

XML provides an opportunity to provide a universal, easy-to-use and standard format for exchanging data of all types. This has been a very brief introduction to some of the ways that we as statisticians may use XML. There is little doubt that there are many more. Also, regardless of how we can exploit XML for our own purposes, we will undoubtedly encounter it more as we interact with other disciplines and will have to deal with it in our daily work. Therefore, we should help to define formats for representing data types with which we work frequently and integrate the XML tools with ours.

Resources

The surge in popularity of XML has rivalled that of Java and hence the number of books on all aspects of XML has increased dramatically over the past few months. There are books on XML, XSL, XML with Java, etc. As with any dynamic and emerging technology, and especially one with so many subdomains with special interests, the Web is probably the best place to find information. The following are some general links that may help you find out more about XML:

- W3's XML page (http://www.w3.org/XML/)
- OASIS (http://www.oasis-open.org/)
- Apache's XML page (http://xml.apache.org/)

Duncan Temple Lang
Bell Labs, Murray Hill, NJ, U.S.A.
duncan@research.bell-labs.com

Programmer's Niche

by Bill Venables

Welcome and invitation

Welcome to the first installment of *Programmer's Niche*.

This is intended to be the first of a regular series of columns discussing R issues of particular interest to programmers. The original name was to have been *Programming Pearls* with the idea of specialising in short, incisive pearls of R programming wisdom, but we have decided to broaden it to include any matters of interest to programmers. The gems are still included, though, so if you have any favourites, let's hear about them.

That brings me to the real purpose of this preamble: to invite readers to contribute. I have offered to edit this column and occasionally to present an article but for the column to serve its purpose properly the bulk of the contributions must come from readers. If you have a particularly fine example of R

programming and you can isolate the message succinctly, then let's hear about it. If you have a short expository idea for something you think is important and not well enough understood by other R programmers, then let's hear about that, too. On the other hand, if you have a question about R programming that has you stumped and you are just dying to get the views of the experts—then do send it along to `r-help`, there's a sport.

Please send contributions directly to me at Bill.Venables@cmis.csiro.au.

R profiling

One of the small miracles to come with R 1.2.0, at least for Unix and Linux platforms, is the ability to *profile* R functions. If you are a serious R programmer this is a tool you will come to use often and if you don't, you should.

The facility is expounded with an example in *"Writing R Extensions"* and I will not repeat it all here. Rather I will try to illustrate the idea with another, smaller example that I will take a bit further down the track.

Well, what is a profile? Any computation in R will involve calling at least one function. Under normal circumstances that function will itself call other functions and so on. The profile of the computation is a split up of the total time taken into components detailing how much time was spent inside each function called. Note that if function A calls function B then the time spent inside A will include the time spent inside B so under this interpretation the individual time components will not add up to the total time. It might be more useful to know the time spent in A excluding time spent in functions that are called from within A, so that the components do add to the total. In fact the profile provides both: the first is called the "total time" for function A and the second the "self time".

A convenient example is the well-known `subsets` function first used in MASS to illustrate the idea of a recursive function. This function finds all distinct subsets of size r from a set of size n. The set is defined by a vector, v. The results are presented as the rows of an $\binom{n}{r} \times r$ matrix. Here is a version of the function that has just a little more bulletproofing than the original version:

```
subsets0 <- function(n, r, v = 1:n) {
  if(r < 0 || r > n)
    stop("invalid r for this n")
  if(r == 0) vector(mode(v), 0) else
  if(r == n) matrix(v[1:n], 1, n) else
  rbind(cbind(v[1],
              Recall(n-1, r-1, v[-1])),
              Recall(n-1, r,   v[-1]))
}
```

The computation seems to come from nowhere but it is based on the simple premise that the subsets of size r are of two distinct types, namely those which contain `v[1]` and those that do not. This provides the essential divide-and-conquer step to allow recursion to take over and do the rest.

OK, let's give it a reasonably hefty example to chew up a bit of time and then use profiling to see how it has been spent. Before we do that, though, we need to make two small preliminary points.

Firstly note that not only is profiling at the moment restricted to Unix and Linux (because the current code uses Unix system calls), but to have it available at all you must install R on your machine with profiling enabled (which is the default); *"Writing R Extensions"* gives the details.

Secondly profiling is done in two steps, one within the R session and one outside. Inside R you turn profiling on and off with calls to the `Rprof` inbuilt function, for example:

```
> Rprof("Rsubs0.out")
> X <- subsets0(20, 6, letters)
> Rprof(NULL)
```

The two calls to `Rprof` start and stop a process where the computation is inspected at regular intervals (by default every 20 milliseconds) and the names of the functions currently on the evaluation stack are dumped onto an output file, in this example 'Rsubs0.out'. For long calculations this file can become quite large, of course.

Outside R there is a Perl script that can be used to summarise this information by giving the "total" and "self" times, and percentages, for each function called. It is called through R as shown in Figure 5.

The information is actually given twice: sorted by total time and by self time. I have only shown the second form here.

What really surprised me about this is the amount of self time taken by `vector`, `mode` and the ancillaries they use such as `switch` and `typeof`. I had thought this would be entirely negligible.

Given that clue let's recast the function so that the call to `vector` is done once and the value held in an enclosing environment. All the recursive hard work will then be done by a very lean internally-defined function, `sub`, that lives in the same environment. A version is given in Figure 6.

We have in fact added a few frills ("Do you want `v` to define a true set, so that repeated items are to be removed, or are you not fussed about repeats?") and removed all time consuming bulletproofing from the inner function, `sub`. I will not present the full profiling output, but the header says it all, really:

```
Each sample represents 0.02 seconds.
Total run time: 25.96 seconds.
....
```

That's a reduction of about 14 seconds in 40, at essentially no cost. It comes from realising that what

```
$ R CMD Rprof Rsubs0.out

Each sample represents 0.02 seconds.
Total run time: 39.48 seconds.

Total seconds: time spent in function and callees.
Self seconds: time spent in function alone.
....
   %       self        %       total
 self     seconds    total    seconds    name
 17.73      7.00     99.39     39.24     "cbind"
 11.65      4.60    100.00     39.48     "rbind"
  9.68      3.82     99.75     39.38     "Recall"
  9.47      3.74    100.00     39.48     "subsets0"
  6.64      2.62      6.64      2.62     "-"
  6.03      2.38      9.17      3.62     "matrix"
  5.02      1.98     17.43      6.88     "mode"
  4.81      1.90      7.50      2.96     "switch"
  4.51      1.78     21.94      8.66     "vector"
  3.85      1.52      4.51      1.78     "is.expression"
  3.70      1.46     14.13      5.58     "|"
  3.14      1.24      3.14      1.24     "=="
  2.74      1.08      3.14      1.24     "as.vector"
  2.74      1.08      6.69      2.64     ">"
  2.68      1.06      2.68      1.06     "typeof"
  2.48      0.98      3.75      1.48     "<"
  2.43      0.96      2.43      0.96     "!="
  0.30      0.12      0.30      0.12     ":"
  0.20      0.08      0.20      0.08     "is.call"
  0.20      0.08      0.20      0.08     "is.name"
```

Figure 5: Output of `R CMD RProf`

```
subsets1 <- function(n, r, v = 1:n, set = TRUE) {
  if(r < 0 || r > n) stop("invalid r for this n")
  if(set) {
    v <- unique(sort(v))
    if (length(v) < n) stop("too few different elements")
  }
  v0 <- vector(mode(v), 0)
  sub <- function(n, r, v) {  ## Inner workhorse
    if(r == 0) v0 else
    if(r == n) matrix(v, 1, n) else
      rbind(cbind(v[1], Recall(n-1, r-1, v[-1])),
                        Recall(n-1, r,   v[-1]))
  }
  sub(n, r, v[1:n])
}
```

Figure 6: Definition of function `subsets1`

seemed to be a computation done only rarely is in fact done very frequently and chomps away at the time.

That clue clearly indicates that if we can stop the recursion at an even earlier stage before it gets to the null sets it may pay off even more handsomely. The sets of size 1 are trivial to enumerate so let's take advantage of that with one extra line:

```
...
    if(r == 0) v0 else
    if(r == 1) matrix(v, n, 1) else
    ## the extra line
    if(r == n) matrix(v, 1, n) else
...
```

Now according to profiling on my machine the time for the same computation drops to just 12.32 seconds, less than one-third the original.

The story does not end there, of course. It seemed to me you could really make this computation zing (at the expense of memory, but hey, this is the 21st century) if you found a way to cache and re-use partial results as you went along. I did find a way to do this in S but using frame 0 or frame 1. Then Doug Bates neatly ported it to R making very astute use of the R scoping rules, function closures and environments, but that is another story for another time.

Bill Venables
CSIRO Marine Labs, Cleveland, Qld, Australia
Bill.Venables@cmis.csiro.au

Writing Articles for R News

or how (not) to ask for Christmas presents.

by Friedrich Leisch

Preface

When I wrote the call for articles for this first edition of R News on the evening of December 20, 2000 on my home laptop I shortly thought about which formats to accept. The decision that the newsletter itself would be produced in LaTeX had long been made, in fact we almost never use something different for text processing. I do a lot of interdisciplinary research with people from the management sciences where MS Word is the predominant text processor and hence am often confronted with conversion between '.doc' and '.tex' files when writing joint papers.

If the text uses only trivial markup (section headers, ...), then conversion is not too hard, but everybody can easily learn the little LaTeX that is involved in that, see the examples below. However, once mathematical equations or figures are involved, I know of no conversion that does not need considerable manual fixing to get a decent result (and we have tried a lot of routes). I considered including some words on Word, but then I thought: "Well, the email goes to `r-devel`, people that may be used to writing '.Rd' files—the format of R help files which is not unlike LaTeX—probably everybody knows LaTeX anyway, let's simply see if anybody really wants to write in Word.", sent the email and went to bed. Bad mistake...

The next day I had meetings in the morning and my first chance to read email was in the afternoon. To keep it short: I had started one of the perhaps most emotional (and longest) threads on `r-devel` so far, and as there is no such thing as a universal *best* word processing paradigm, there was of course also no "winner" in the discussion. So all I could add to the pro-LaTeX arguments 18 hours after my first email is that this editorial decision had already been made. I want to use this article to apologize to all for not being more detailed in my email that started the thread and explain the decision.

As all of R, R News is a volunteer project. We have no staff to do the editing or layouting etc., hence the editors have to "outsource" as much as possible to you, the prospective authors. This will only work if all use the same format, because—as explained above—automatic conversion simply does not work in practice. I know that the majority of R developers use LaTeX, hence the decision was not too hard which paradigm to choose.

The structure of R News articles

Figure 7 shows parts of the source code for this article. LaTeX is a markup language like, e.g., HTML and mixes layout commands and contents in a single text file (which you can write in any editor you like, or even Word). Most commands start with a backslash, arguments to the commands are usually given in curly brackets. We first specify the title, subtitle and author of the article and then issue the command `\maketitle` to actually typeset it, update the table of contents and PDF bookmarks. The command `\section*{}` starts a new section (`\section*{}` starts sections without numbering them, `\section{}` without the star would add numbers), and then we can simply enter the main text, separating paragraphs by blank lines.

The command `\file{}` has been defined by us to typeset file names in a different font and enclose them in single quotes. Finally we specify the au-