10. Interfacing R

Thomas Lumley
Ken Rice

Universities of Washington and Auckland

Seattle, June 2011
Interfacing R

With Bioconductor, R can do a huge proportion of the analyses you’ll want – but not everything

- Intensive (or anachronistic) C++, FORTRAN work, e.g. for pedigrees

- ‘Speciality’ analyses; some need different computing architecture

- Fancy interactive graphics

R can be used to ‘manage’ other software. Today we’ll illustrate some favorite examples
Starting other software

NB these commands are for Windows only; see help files for e.g. Unix versions

- `system` is the equivalent of a DOS-style command

- `system("notepad")` starts the Notepad editor

- **If** the command takes arguments, put them in the same string;
  `system("notepad myfile.txt")`

The *shell* command does much the same thing.
Starting other software

Some more options for system:

- wait; R ‘hangs’ until completion
- minimized; new program only appears in TaskBar
- show.output.on.console

Paths for files can be a little messy; system starts in your working directory (getwd). Outside of this, give the full pathway.

paste is useful, if you need to do a lot of this sort of thing. source may also help
Examples

Code for a really mundane job;

```r
for(i in 1:100){
    infile <- paste("gene",i,"data.txt", sep="")
    outfile <- paste("gene",i,"phase.out", sep="")
    system(paste("PHASE",infile,outfile))
}
```

... this will churn away for hours, although with no error-control. (With a few more `paste()` commands, and `cat()`, it’s possible to automating writing R scripts, and then set them running on multiple processors)

Why did we use `wait=TRUE` here? (the default)
Examples

- WinBUGS implements Bayesian analyse; it’s not super-fast but it is very flexible - and easier to code than writing your own MCMC.

- It needs special (& clever) architecture to achieve this

- WinBUGS’ input, output, and graphics are all rather clunky

- R’s are better; so R2WinBUGS calls WinBUGS for the difficult bits, and does all the ‘translation’ itself

- It’s all done with (repeated) use of system()
Many programs already exist to do useful analyses. It is more convenient to call them from R than to rewrite them in R.

Sometimes this involves calling the C code directly, sometimes just involves using R to write input files for another program.

Examples:

- Graphviz: drawing networks
- PMF: input files for ancient Fortran software
- Google Earth: displaying outliers in context.
**Drawing networks**

GraphViz ([http://www.graphviz.org](http://www.graphviz.org)) is a free program for drawing networks, written by AT&T researchers.

Its input format looks like

```
"15" [shape= box,regular=1 ,height= 0.5 ,width= 0.75 ,style=filled,color= grey ] ;
"16" [shape= circle ,height= 0.5 ,width= 0.75 ,style=filled,color= grey ] ;
"2x3" [shape=diamond,style=filled,label="",height=.1,width=.1] ;
"2" -> "2x3" [dir=none,weight=1] ;
"3" -> "2x3" [dir=none,weight=1] ;
"2x3" -> "1" [dir=none,weight=2] ;
"2x3" -> "4" [dir=none,weight=2] ;
"2x3" -> "5" [dir=none,weight=2] ;
"2x3" -> "6" [dir=none,weight=2] ;
```

The `sem` package uses GraphViz to display path diagrams for structural equation models and the `gap` package uses it to draw pedigrees.
### Drawing networks

In *gap* the `pedtodot()` function writes a GraphViz input file from a pedigree in GAS or LINKAGE format.

<table>
<thead>
<tr>
<th>pid</th>
<th>id</th>
<th>fid</th>
<th>mid</th>
<th>sex</th>
<th>aff</th>
<th>GABRB1</th>
<th>D4S1645</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10081</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7/7</td>
<td>7/10</td>
</tr>
<tr>
<td>2</td>
<td>10081</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1/-</td>
<td>1/-</td>
</tr>
<tr>
<td>3</td>
<td>10081</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2/7</td>
<td>7/9</td>
</tr>
<tr>
<td>4</td>
<td>10081</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7/9</td>
<td>3/7</td>
</tr>
<tr>
<td>5</td>
<td>10081</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1/7</td>
<td>7/7</td>
</tr>
<tr>
<td>6</td>
<td>10081</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1/7</td>
<td>7/7</td>
</tr>
<tr>
<td>7</td>
<td>10081</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2/7</td>
<td>7/7</td>
</tr>
<tr>
<td>8</td>
<td>10081</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1/-</td>
<td>1/-</td>
</tr>
<tr>
<td>9</td>
<td>10081</td>
<td>9</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>1/7</td>
<td>7/9</td>
</tr>
<tr>
<td>10</td>
<td>10081</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1/-</td>
<td>1/-</td>
</tr>
<tr>
<td>11</td>
<td>10081</td>
<td>11</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>1/7</td>
<td>7/7</td>
</tr>
<tr>
<td>12</td>
<td>10081</td>
<td>12</td>
<td>2</td>
<td>10</td>
<td>2</td>
<td>2/6</td>
<td>7/7</td>
</tr>
<tr>
<td>13</td>
<td>10081</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1/-</td>
<td>1/-</td>
</tr>
<tr>
<td>14</td>
<td>10081</td>
<td>14</td>
<td>13</td>
<td>11</td>
<td>1</td>
<td>1/7</td>
<td>7/8</td>
</tr>
<tr>
<td>15</td>
<td>10081</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1/-</td>
<td>1/-</td>
</tr>
<tr>
<td>16</td>
<td>10081</td>
<td>16</td>
<td>15</td>
<td>12</td>
<td>2</td>
<td>1/6</td>
<td>7/7</td>
</tr>
</tbody>
</table>
Drawing networks

First the code prints nodes for each individual, with sex and affectedness information

```r
for (s in 1:n) cat(paste("\"", id.j[s], ", [shape=", sep = ""), shape.j[s], ",height=", height, ",width=", width, ",style=filled,color="", shade.j[s], "] ;\n")
```

giving output like

"16" [shape= circle ,height= 0.5 ,width= 0.75 ,style=filled,color= grey ] ;

It then works out all the matings and creates small nodes for each mating and lines connecting the parents to these nodes

```r
mating <- paste("\"", s1, "x", s2, "\"", sep = "")
cat(mating, "[shape=diamond,style=filled,label="",height=.1,width=.1] ;\n")
cat(paste("\"", s1, "\"", sep = ""), " -> ", mating,
    paste(" [dir="", dir, ",weight=1]", sep = ""),
    " ;\n")
cat(paste("\"", s2, "\"", sep = ""), " -> ", mating,
    paste(" [dir="", dir, ",weight=1]", sep = ""),
    " ;\n")
```
Drawing networks

giving output like

"2x3" [shape=diamond, style=filled, label="", height=.1, width=.1] ;
"2"  ->  "2x3"  [dir=none, weight=1] ;
"3"  ->  "2x3"  [dir=none, weight=1] ;

and then connects children to parents.
[Bioconductor also has GraphViz more integrated with R in the RGraphViz package]
PMF: factor analysis

PMF is a program for constrained factor analysis in analytic chemistry. It is controlled by an ugly text input file:

```r
pmfini<-c(" ##PMF2 .ini file for: Simulations from R",
" ## Monitor code M: if M>1, PMF2 writes output every Mth step",
" ## For finding errors, use M<1 to output debug information",
" ## M PMF2 version number",
" 1 4.2",
" ## Dimensions: Rows, Columns, Factors. Number of "Repeats"",
" @nt@ @ns@ @sources@ 1",
" ## "FPEAK" (>0.0 for large values and zeroes on F side)",
" @FPEAK@",
" ## Mode(T:robust, F:non-robust) Outlier-distance (T=True F=False)",
" @isrobust@ @outlier@",
" ## Codes C1 C2 C3 for X_std-dev, Errormodel EM=[-10 ... -14]",
" 0.0100 0.0000 0.0100 -12",
" ## G Background fit: Components Pullup_strength",
" 0 0.0000",
" ## Pseudorandom numbers: Seed Initially skipped",
" @seed@ 0",
" ## Iteration control table for 3 levels of limit repulsion "lims"")
```

The @value@ are places where we want to substitute in a value.
PMF: factor analysis

R code for the substitutions looks like

temp<-gsub("@FPEAK@",formatC(fpeak,digits=4,format="f"),pmfini)
temp<-gsub("@isrobust@",isrobust,temp)
seed<-as.character(as.integer(seed))
temp<-gsub("@seed@", seed, temp)

gsub() looks finds matches in character strings, and replaces them with specified text.
PMF: factor analysis

We can write data files needed by PMF, and then write the control file, then call PMF with the `system()` function. After PMF finishes we read in the results.

```r
write.table(cX, file=xfile, quote=FALSE, col.names=FALSE, row.names=FALSE)
write.table(cU, file=efile, quote=FALSE, col.names=FALSE, row.names=FALSE)
if (!debug)
  on.exit(unlink(c(xfile, efile, inifile)))
writeLines(temp, inifile)

sysval<-system(paste(pmf, inifile), intern=TRUE, invisible=!debug)

ffactor<-read.table(outfiles$f, row.names=sourcenames, col.names=species)
gfactor<-read.table(outfiles$g, row.names=times, col.names=sourcenames)
```

From the user’s viewpoint it looks as though everything was done in R.
SVG+tooltips

SVG (Scalable Vector Graphics) is a non-bitmap graphics format for the web.

The RSvgDevice and RSVGTipsDevice packages allow R output to SVG format.

We can use this to create graphs with links and tooltips. For example, a funnelplot showing associations between a large number of SNPs and VTE.

Point at a dot to see the SNP it represents, and click to go to information about the gene.
for(i in 1:length(or)) {
    setSVGShapeToolTip(title=gene[i],
        desc1=snp[i],
        desc2=if(abs(lor[i]/se[i])>qnorm(0.5/n,lower.tail=FALSE))
            qvals[i] else NULL
    )

    setSVGShapeURL(paste("http://pga.gs.washington.edu/data",
        tolower(gene[i]),
        sep="/"))

    points(prec[i],lor[i], cex=1, pch=19, col='grey')
}
Google Earth

Google Earth is controlled by KML files specifying locations. KML is another plain text format.

We can write a KML file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Placemark>
    <name>1</name>
    <Point>
      <coordinates>-118.0256,34.11619,400</coordinates>
    </Point>
  </Placemark>
</kml>
```

and then send it to Google Earth with the `shell.exec(filename)` function, which opens a file using whatever is the appropriate program.
Google Earth

The `identify()` function lets the user select a point on a scatterplot.

In this example the points are locations where air pollution was measured, and we call Google Earth to quickly look at the location.