10. Interfacing R

Thomas Lumley
Ken Rice

Universities of Washington and Auckland

Lausanne, September 2014

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

- `shell()` does the equivalent of a DOS-style command
- `shell("notepad")` starts the Notepad editor
- **If** the command takes arguments, put them in the same string;
  `shell("notepad myfile.txt")`

The `system()` and `shell.exec()` commands do much the same thing.

Starting other software

Some more options for `shell()`;

- `wait`; `R` ‘hangs’ until completion
- `translate`; makes forward and backslashes work properly
- `intern`; return the output as an R object

For other options see the `system()` help page, for example `minimized=TRUE`.

Paths for files can be a little messy; `shell()` starts in your working directory (find it using `getwd()`). For files outside of this, give the full pathway.

`paste()` is useful, if you need to do a lot of this sort of thing.

Examples

Code for a really mundane job;

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

... this will churn away for hours, although with no error-control.

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

Examples

- WinBUGS implements Bayesian analyse; it's not super-fast but is very flexible

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

- WinBUGS' input, output, graphics are all rather clunky

- `R` is better; so `R2WinBUGS` calls WinBUGS for the difficult bits, and does all the 'translation' itself

- This is done with (repeated) use of `system()`

Outline

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) 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.

```
     pid id fid mid sex aff GABRB1 D4S1645
1  10081  1   2   3   2   2    7/7    7/10
2  10081  2   0   0   1   1    -/-     -/-
3  10081  3   0   0   2   2    7/9    3/10
4  10081  4   2   3   2   2    7/9     3/7
5  10081  5   2   3   2   1    7/7    7/10
6  10081  6   2   3   1   1    7/7    7/10
7  10081  7   2   3   2   1    7/7    7/10
8  10081  8   0   0   1   1    -/-     -/-
9  10081  9   8   4   1   1    7/9    3/10
10 10081 10   0   0   2   1    -/-     -/-
11 10081 11   2  10   2   1    7/7     7/7
12 10081 12   2  10   2   2    6/7     7/7
13 10081 13   0   0   1   1    -/-     -/-
14 10081 14  13  11   1   1    7/8     7/8
15 10081 15   0   0   1   1    -/-     -/-
16 10081 16  15  12   2   1    6/6     7/7
```

Drawing networks

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

```
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

```
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")
```

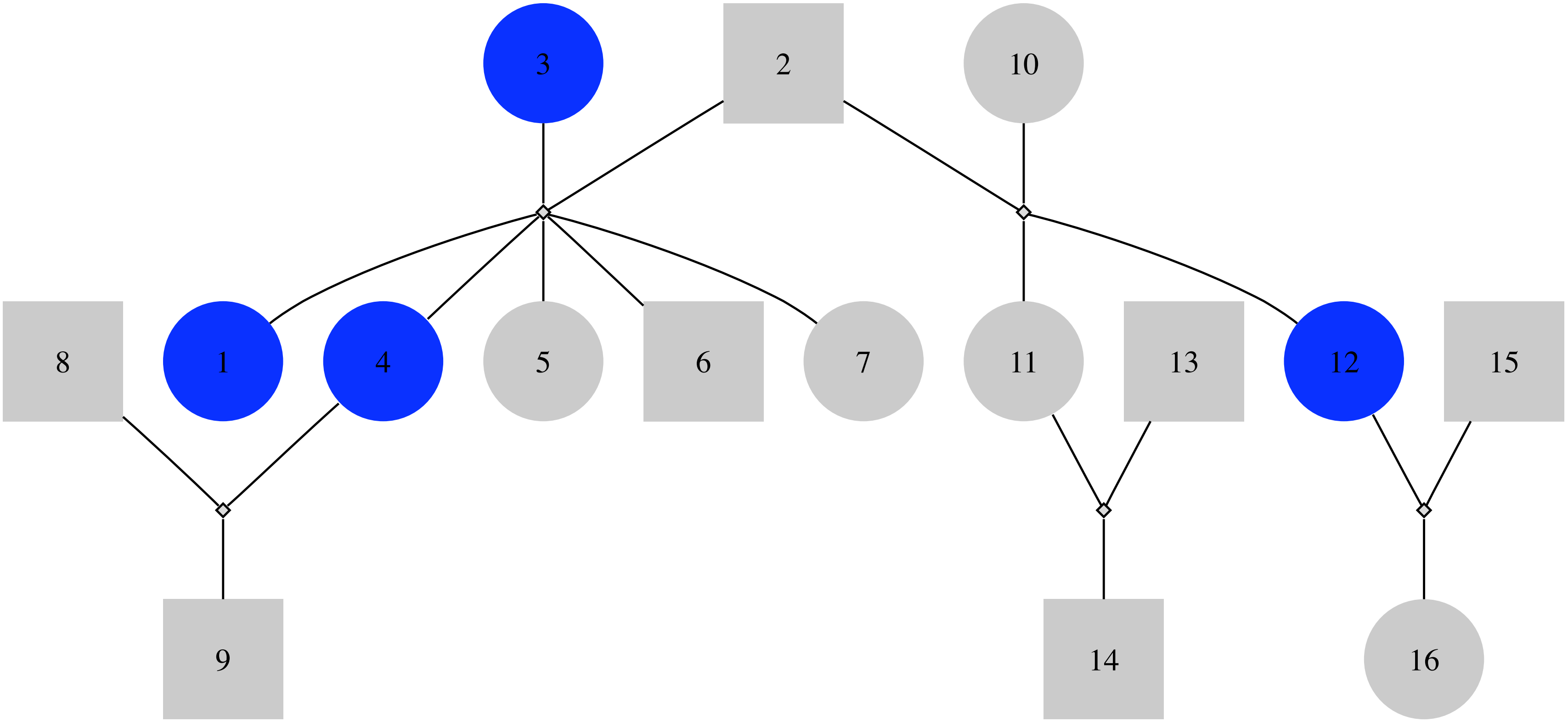
Drawing networks

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.

Drawing networks



pedigree 10081

[Bioconductor also has GraphViz more integrated with R in the `RGraphViz` package]

Chromosome simulation

MaCS, the Markov Coalescent Simulation (Chen et al, 2008) simulates realistic genotypes using an approximation to the coalescent.

It's a command-line program written in C++, with output:

```
/Users/tlumley/macs/macs 2000 15000 -t .001 -r .001 .001
1390319964

//
segsites: 134
positions: 0.000421536259 0.0100671644 0.0111485623 0.0230004096 0.0332452159...
100000000000000000000000000000100100010001000000000000000000001010000...
000000000000000000000000000000000000000000000000000000000000000000000...
000000000000000000000000000000000000000000000000000000000000000000000...
```

We want the same sort of simulation functions as in earlier sessions, so we need an R function that calls `MaCS` and returns a data matrix.

Chromosome simulation

Tasks

- Call MaCS
- Read in the lines of haploid genotypes as character strings
- Split into numbers
- Recode so 1 is the minor allele
- Combine pairs of haploids into a diploid

Chromosome simulation

```
makemacsdata<-function(N,length=15000,filter=0.05){
  f<-tempfile()
  system(paste("˜/macs/macs",2*N,length,
      " -t .001 -r .001 2>/dev/null | ˜/macs/msformatter >", f))
  input<-readLines(f)[-(1:6)]
  unlink(f)
  haplo<-do.call(rbind,lapply(strsplit(input,""),as.integer))
  diplo<-haplo[1:N,]+haplo[(N+1):(2*N),]
  af<-colMeans(diplo)/2
  diplo[,af>0.5]<- 2-diplo[,af>0.5,drop=FALSE]
  maf<-colMeans(diplo)/2
  diplo[,af<=filter,drop=FALSE]
}
```

Chromosome simulation

From the user's viewpoint it looks as though everything was done in R.

```
> d<-makemacsdata(1000)
> str(d)
 num [1:1000, 1:80] 0 0 0 0 0 0 0 0 0 0 ...
> summary(colMeans(d)/2) #maf
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.00050 0.00100 0.00450 0.01142 0.01512 0.05000
```

Chromosome simulation

Now simulate a new version of the SKaT rare-variant test

```
one.sim<-function(thresholds=c(Inf,1/2,1/3,1/4),
          sqrtweights=wuweights,
          N=4000, n=200, length=15000, filter=0.02)
{
  G <- makemacsdata(N,length,filter=filter)
  y <- sample(rep(0:1,c(N-n,n)))
  sapply(thresholds,
      function(c) winskat(G,y,threshold=c,sqrtweights))
}
```

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.

SVG+tooltips

```
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.

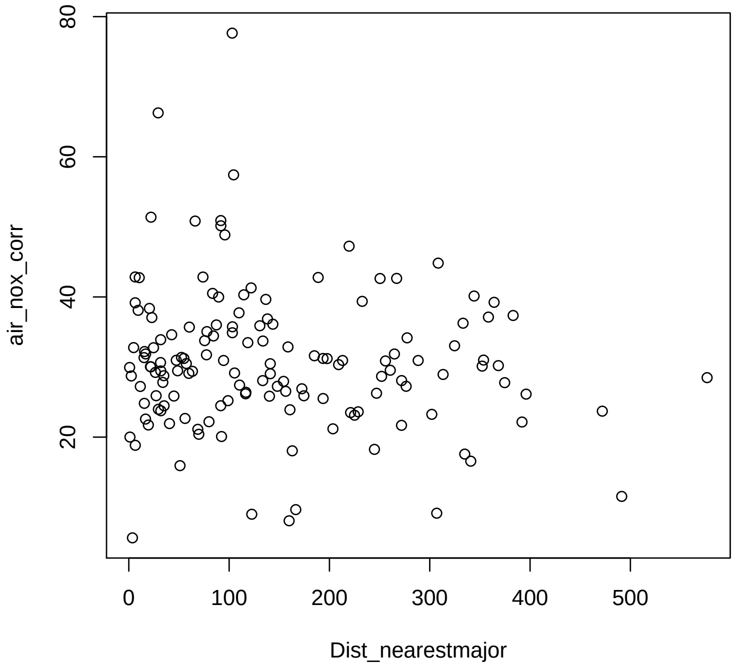
We can write a KML file

```
<?xml 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 can call Google Earth to look at the location.