Preliminaries for intermediate course, Boston, March 2008 Revised Final version

VJ Carey

March 1, 2008

Contents

1 Discussion **1**

2 Illustrations **4**
2.1 Typography 4
2.2 Dialogue 4

3 Review **12**
3.1 Essential R repertory 12
3.2 Essential Bioconductor repertory 16

4 Exercises **21**

1 Discussion

1. You will be using R 2.7.0 (devel version). A distribution will be supplied for all windows users that will be independent of any installations resident on the laptop.

2. You are expected to have a reasonable facility with the R programming language. Please see www.r-project.org/Manuals. Read An Introduction to R and work through the sample session.

3. R is a language for data analysis and visualization. It is also provides vehicles for developing and disseminating software tools. These vehicles are a basis for, and have been extended by, the Bioconductor project.

(a) R is persistently *extended* by *packages*. Examples are provided at cran.r-project.org. Packages contain software and documentation, and often data for illustration or as a central component. Packages at CRAN and Bioconductor are subjected to standard quality control, often on a nightly basis, as they evolve and as R evolves.

(b) Documentation for R and Bioconductor exists at three levels.

   i. At the highest level are monograph-like documents such as Modern Applied Statistics with S (Venables and Ripley) or Introductory Statistics with R (Dalgaard). The manuals at www.r-project.org are at a similar level, covering programming processes and analysis methods over a wide variety of packages. A monograph on Bioconductor is also available (Bioinformatics and Computational Biology Solutions with R and Bioconductor, eds. Gentleman, Carey et al.) Access: Bookstores.

   ii. At the next level are vignettes. Typically a vignette is associated with an R package. A vignette will describe a workflow, typically using several functions in one package. Vignettes are 'computable documents'. We will show how to create such documents at the end of the course. Access: CRAN/bioconductor pages devoted to packages; R function openVignette().

   iii. At the lowest level are manual pages for package functions. These documents describe details of function calls. Executable examples are typically provided with each manual page. Access: help([topicname]), example([topicname]), help.search(), help.start().

(c) Acquiring packages will not be a major part of the course. All software will be supplied at the start of the course. But it should be noted that package acquisition is supported in the R language with function install.packages(). at http://www.bioconductor.org, a script called biocLite.R can be 'sourced', so that biocLite("[packagename]") will acquire a package for installation. A given package may depend upon other packages and biocLite will arrange to resolve all the dependencies and install all needed packages.

4. Bioconductor is a project devoted to the creation of data structures and algorithms supporting modern computational biology. The primary products are R packages, available through the Bioconductor portal www.bioconductor.org.

   (a) The Biobase package defines most of the primary data structures for genome-scale data.

   (b) Various biological metadata packages, such as hgu133plus2, illuminaHumanv2, or org.Hs.eg.db, define mappings between identifiers, and packages GO.db and KEGG.db define metadata structures relating to Gene Ontology and the KEGG pathway catalog.

(c) Analytical packages such as limma, MLInterfaces, GOstats, Category, graph, RBGL, address various approaches to interpretation of genome scale data.

(d) Preprocessing packages such as oligo, arrayQualityMetrics, affyPLM, lumi, beadarray, beadarraySNP, help to transform raw assay outputs into interpretable measures and to assess assay quality.

(e) Experimental data packages such as MAQCsubset, yeastCC, bronchialIL13 illustrate how experiments can be organized for convenient analysis and evaluation.

5. This 'preliminaries' document attempts to cover relevant mechanical practices needed to work effectively with microarray and allied data. The review section following illustrations covers essential repertory

   (a) in R: calling functions, creating numerical data, attaching software packages, running documentation examples, loading stored data, capturing data from files, creating or rewriting complex objects, use indexing on vectors and matrices to extract information, understanding the recycling rule, basic visualization, statistical procedure invocation and report generation, creation of simple functions, and use of applicative programming

   (b) in Bioconductor: obtaining Bioconductor packages for procedures and data, understanding introspective capacities of Bioconductor containers for high-throughput datasets, investigating values of expression assays, understanding resolution of identifiers using annotation maps

   The preliminaries are thus weighted towards building facilities in R, so that we can readily exploit high-level facilities in Bioconductor to answer the biological questions that interest us.

   You should feel comfortable giving examples of or writing brief paragraphs on *almost every item* in the laundry list of repertory topics given above before coming to the course.

2 Illustrations

The following operations and their results should be completely straightforward after light study. You can set up a version of R that supports the dialogue below by running

```
> source("http://www.bioconductor.org/biocLite.R")
> biocLite(c("Biobase", "graph", "Rgraphviz", "MAQCsubset", "hgu133plus2.db",
+     "GO.db", "annotate"))
```

2.1 Typography

The typography above works as follows. What you type into R is rendered thusly:

```
> x = rnorm(20)
```

What R does is rendered thusly:

```
 [1] -1.415931610  0.317440244  0.728134168 -1.899549457 -1.144503647
 [6]  0.576936010  0.198325980  1.074562398  0.625656592 -0.699694826
[11] -0.396495342  0.004635531 -0.748074420 -0.989724856  1.360047389
[16] -2.769526125  0.701093017  1.144115444  0.626119072  1.426245190
```

When you see plus-signs in italics on the left, you are seeing continuation symbols that are not to be typed in – they show that input to R is spanning several lines.

2.2 Dialogue

Class inspection

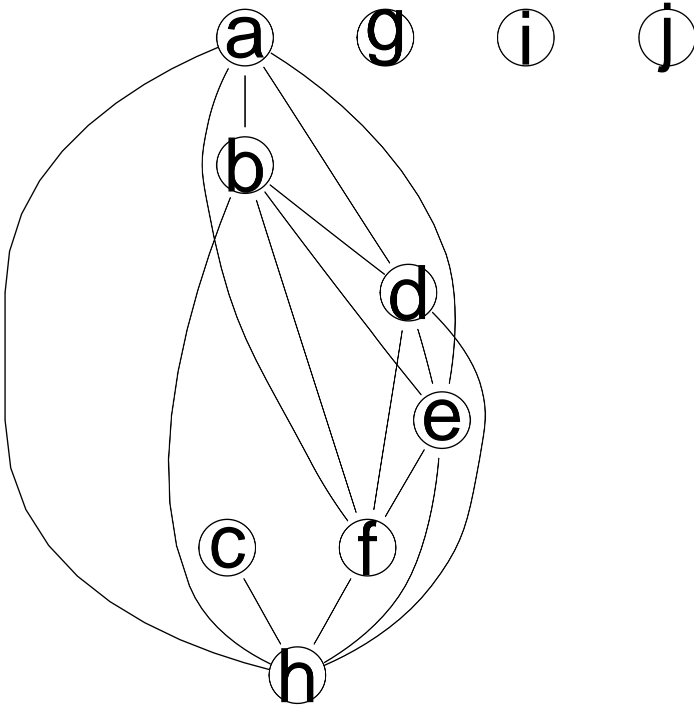
```
> #
> # attach the base package for Bioconductor
> # and check the structure of eSet class
> #
> library(Biobase)
> getClass("eSet")
```

```
Virtual Class

Slots:

Name:           assayData           phenoData         featureData
Class:          AssayData AnnotatedDataFrame AnnotatedDataFrame

Name:      experimentData          annotation  .__classVersion__
Class:              MIAME           character            Versions
```

```
Extends:
Class "VersionedBiobase", directly
Class "Versioned", by class "VersionedBiobase", distance 2

Known Subclasses: "ExpressionSet", "NChannelSet", "MultiSet", "SnpSet"
```

Network structures

```
> #
> # get packages that define network structures and
> #  support their visualiation; plot a random graph
> #
> library(graph)
> example(randomGraph)

rndmGr>  set.seed(123)

rndmGr>  V <- letters[1:10]

rndmGr>  M <- 1:4

rndmGr>  g1 <- randomGraph(V, M, 0.2)

rndmGr>  numEdges(g1) # 16, in this case
[1] 16

rndmGr>  edgeNames(g1)# "<from> ~ <to>"  since undirected
 [1] "a~b" "a~d" "a~e" "a~f" "a~h" "b~f" "b~d" "b~e" "b~h" "c~h" "d~e" "d~f"
[13] "d~h" "e~f" "e~h" "f~h"

> library(Rgraphviz)
> plot(g1)
```



MAQC data inspection

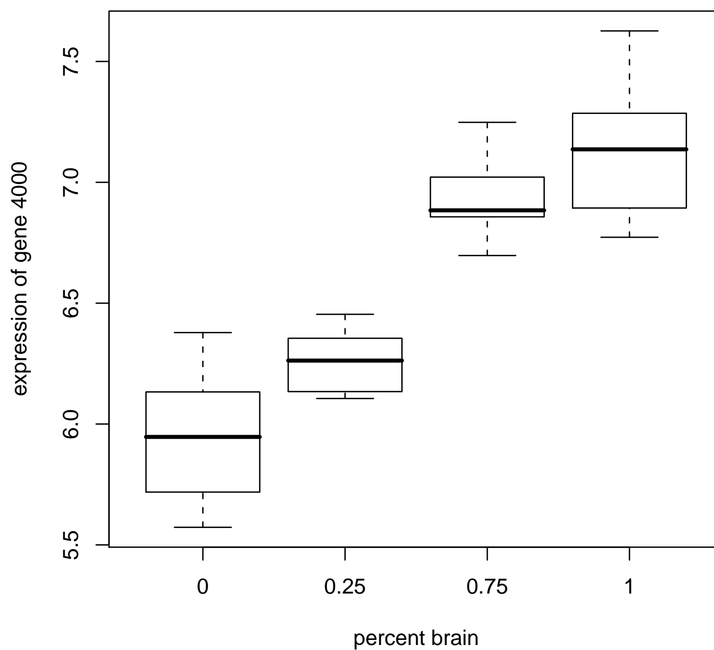
```
> #
> # get data from MAQC
> #
> library(MAQCsubset)
> data(afxsubRMAES)
> experimentData(afxsubRMAES)

Experiment data
  Experimenter name: Shippy R
  Laboratory: GE Healthcare, 7700 S. River Pkwy., Suite #2603, Tempe, Arizona 85284, US
  Contact information:
  Title: Using RNA sample titrations to assess microarray platform performance and norm
  URL:
  PMIDs: 16964226

  Abstract: A 158 word abstract is available. Use 'abstract' method.

> #
> # look at some expression values
```

```
> #
> exprs(afxsubRMAES)[1:5,1:5]

           AFX_1_A1.CEL AFX_1_A2.CEL AFX_1_B1.CEL AFX_1_B2.CEL AFX_1_C1.CEL
1007_s_at      9.505351     9.539333    10.219630    10.247826     9.648155
1053_at        8.790974     8.921003     6.874935     6.844444     8.618656
117_at         5.927965     5.846260     6.246920     6.276300     5.763446
121_at         7.512177     7.282186     7.217067     7.249829     7.610581
1255_g_at      3.681744     3.453635     3.925925     3.662053     3.517195

> #
> # what 'phenoData' values are present
> #
> names(pData(afxsubRMAES))

[1] "site"     "samp"     "repl"     "pctBrain"
```

[This space reserved for jelly stains.]

Boxplots

```
> #
> # show distributions of expression of a gene
> #  stratified by percent ambion brain hybridized
> #
> boxplot(split(exprs(afxsubRMAES)[4000,], afxsubRMAES$pctBrain),
+   xlab="percent brain", ylab="expression of gene 4000")
```



Regression modeling

```
> mylm = lm(exprs(afxsubRMAES)[4000, ] ~ afxsubRMAES$pctBrain)
> summary(mylm)

Call:
lm(formula = exprs(afxsubRMAES)[4000, ] ~ afxsubRMAES$pctBrain)

Residuals:
      Min        1Q    Median        3Q       Max
-0.409667 -0.152021  0.007191  0.113659  0.443999

Coefficients:
```

```
                     Estimate Std. Error t value Pr(>|t|)
(Intercept)           5.96040    0.07667   77.74  < 2e-16 ***
afxsubRMAES$pctBrain  1.22192    0.12029   10.16 9.07e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2329 on 22 degrees of freedom
Multiple R-squared: 0.8243,        Adjusted R-squared: 0.8163
F-statistic: 103.2 on 1 and 22 DF,  p-value: 9.07e-10
```

```
> summary(mylm)$coef[2, 4]
```

```
[1] 9.070416e-10
```

Annotation

```
> #
> # attach a package for annotation management, get the
> #  annotation data for hgu133 plus 2.0 chip, GO
> #
> library(annotate)
> library(hgu133plus2.db)
> library(GO.db)
> #
> # get the affy identifier for gene 4000 (in the
> #  ordering of featureNames)
> #
> fn = featureNames(afxsubRMAES)[4000]
> #
> # now look it up in various dictionaries
> #
> lookUp(fn, "hgu133plus2", "ENTREZID")
```

```
$`1557970_s_at`
[1] "6196"
```

```
> lookUp(fn, "hgu133plus2", "GENENAME")
```

```
$`1557970_s_at`
[1] "ribosomal protein S6 kinase, 90kDa, polypeptide 2"
```

```
> lookUp(fn, "hgu133plus2", "SYMBOL")
```

```
$`1557970_s_at`
[1] "RPS6KA2"
```

```
> gotags = lookUp(fn, "hgu133plus2", "GO")
> length(gotags)

[1] 1
```

List manipulation

```
> gotags[[1]][1:2]

$`GO:0006468`
$`GO:0006468`$GOID
[1] "GO:0006468"

$`GO:0006468`$Evidence
[1] "IEA"

$`GO:0006468`$Ontology
[1] "BP"


$`GO:0007243`
$`GO:0007243`$GOID
[1] "GO:0007243"

$`GO:0007243`$Evidence
[1] "TAS"

$`GO:0007243`$Ontology
[1] "BP"

> lookUp("GO:0004674", "GO", "TERM")

$`GO:0004674`
GOID: GO:0004674
Term: protein serine/threonine kinase activity
Ontology: MF
Definition: Catalysis of the reaction: ATP + a protein serine/threonine
    = ADP + protein serine/threonine phosphate.
Synonym: SAP kinase 3 activity
Synonym: SAP kinase 4 activity
Synonym: SAP kinase 5 activity

> sapply(lookUp(names(gotags[[1]]), "GO", "TERM"), Term)
```

```
                                   GO:0006468
        "protein amino acid phosphorylation"
                                   GO:0007243
                    "protein kinase cascade"
                                   GO:0005634
                                   "nucleus"
                                   GO:0000166
                       "nucleotide binding"
                                   GO:0000287
                     "magnesium ion binding"
                                   GO:0004674
"protein serine/threonine kinase activity"
                                   GO:0005524
                              "ATP binding"
                                   GO:0016740
                     "transferase activity"
```

3 Review

3.1 Essential R repertory

Fundamental operations reviewed above include

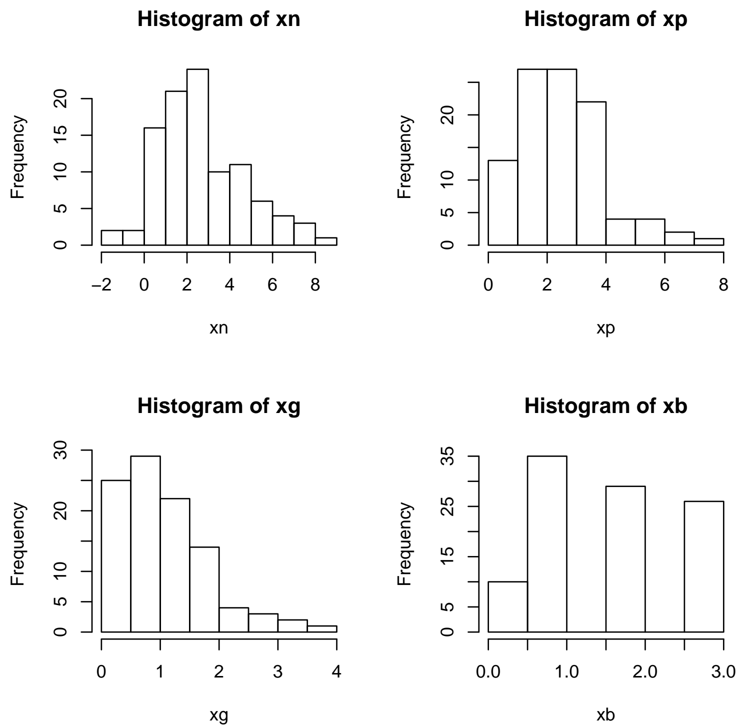
1. *Most generally:* **calling functions.** Every action in R results from the evaluation of a function. Typically the lexical form of a function call is `f(x,y,z)`, where "`f`" names a function that has been defined in some package and `x`, `y`, `z` are inputs suitable for use with `f`. This mention of 'suitability' is vague but inevitable at this point. We learn what is suitable for a given function by reading the manual pages and other documentation resources.

   Some operations in R do not look like function calls, but they are: `x[1,2]` can be written `"["(x,1,2)`.

   Distinguish carefully between vector inputs to functions and multiple parameters. `x[1,2]` and `x[c(1,2)]` are very different computations.

2. **create numerical data via simulation; create (pseudo) random samples from given data.** Random number generation is accomplished using the r[dist] family of functions:

```
> set.seed(1234)
> xn = rnorm(100, 3, 2)
> xp = rpois(100, 3)
> xg = rgamma(100, 2, 2)
> xb = rbinom(100, 4, .4)
> par(mfrow=c(2,2))
> hist(xn)
> hist(xp)
> hist(xg)
> hist(xb)
> par(mfrow=c(1,1))
```



To draw a random sample from a vector, use sample:

```
> mean(sample(xn, size = 20))
```

```
[1] 2.667159
```

```
> mean(sample(xn, size = 20))
```

```
[1] 2.742264
```

Note that set.seed can be used to control reproducibility of 'random' events.

3. **package attachment** – `library([packagename])`

   Note that you can get information on manual pages available for a package by doing `help(package=[packagename])`.

4. **run examples** – `example([topic])`

5. **load stored data** – `data([objectname])`

   You can get information on data sets provided by a package with the command `data(package=[packagename])`

6. **data capture from files**: functions `read.csv`, `read.delim`, `readLines`, `scan` are relevant in different contexts. Practice 'round trips': create some data in excel, export as CSV, import to R, analyze/extend, export results using write.table, and import to excel.

7. **create new (or rewrite old) objects in an R session** – `y = f(x)` or `x = f(x)`

8. **use indexing on vector-like or matrix-like data structures to refer to subsets** – the forms `x[i]` and `x[i,j]` need to be fully appreciated; `i` and `j` may themselves be vectors or matrices

   (a) named vector elements

```
> x = c(1, 2, 3, 4)
> names(x) = c("a", "b", "A", "B")
> x[3]

A
3

> x["A"]

A
3

> x[c("A", "a")]

A a
3 1
```

   (b) named matrix margins

```
> y = matrix(1:16, nr=4, nc=4)
> y # no names

     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16

> rownames(y) = colnames(y) = c("a", "b", "c", "d")
> y # names on both margins

  a b  c  d
a 1 5  9 13
b 2 6 10 14
c 3 7 11 15
d 4 8 12 16

> y["b",]  # extract a row
```

```
 a  b  c  d
 2  6 10 14
> y[c("b","c"), c("b", "d")]  # extract a submatrix
  b  d
b 6 14
c 7 15
```

9. **recycling rule for binary operations**: when a binary operation is applied to two vectors of different lengths, the shorter is replicated to the length of the longer (with truncation and warning if the length of the longer is not an exact multiple of that of the shorter)

   ```
   > sum(y)
   [1] 136
   > sum(y+1)
   [1] 152
   > sum(y+c(1,1,1))  # will warn
   [1] 152
   ```

   Note also that we can compose operations within indexing contexts (above, creating a vector within []).

10. **visualize data**: `plot(x, y)`, `boxplot(split(x,y))`, `pairs` are important for exploration; the lattice package is essential for visualizing multivariate data

11. **compute simple statistics**: function `mean`, `median`, `sd`, `cor` are self-explanatory, but behavior with incomplete data (i.e., some elements take value NA) must be understood with care

12. **execute statistical procedures and derive summaries:** `lm` and its formula interface illustrates a basic paradigm for working with linear and nonlinear models

13. **create simple functions.** Example: suppose we are given a set of numeric strings of different lengths and we wish to pad them to a common length with "0" on the left. We break this up into two phases. First we write a program that solves the problem of creating the padding prefix. Given a prefix length, the following function creates the padding string:

    ```
    > makePref = function(prefLen) paste(rep("0", prefLen), collapse="")
    > makePref(4)  # works
    ```

```
[1] "0000"
```

```
> try( makePref(c(3,4,5)))  # fails!
```

We see that our program fails on a vector input. An exercise is to create a variation that succeeds on a vector input. However, for our purposes, this function is sufficient.

We now build a function that solves the original problem, using makePref iteratively to get the padded strings. This function MUST SUCCEED on vector inputs.

```
> padTo = function(x, length=10) {
+  nx = length(x)   # number of elements passed
+  lens = nchar(x)   # character length of each element
+  add = length-lens  # deficits
+  pads = sapply(add, makePref)
+  paste(pads, x, sep="")
+ }
> strs = c("12334", "1322", "144445")
> padTo(strs,10)
```

```
[1] "0000012334" "0000001322" "0000144445"
```

14. **use applicative programming.** In the example function above, we used `sapply` to quickly iterate over the elements of a vector (the vector `add`). Instead of using a 'for' loop, we can use a single statement to have a more concise and efficient calculation. Other applicative programming tools are `lapply`, `sapply`.

3.2 Essential Bioconductor repertory

1. **obtain desired software and data packages**, and resolve relevant dependencies: `biocLite`

2. **understand the introspective capacity of Bioconductor's high-throughput containers**

   (a) view (and populate) MIAME schemas

   ```
   > experimentData(afxsubRMAES)
   ```

   ```
   Experiment data
     Experimenter name: Shippy R
     Laboratory: GE Healthcare, 7700 S. River Pkwy., Suite #2603, Tempe, Arizona
     Contact information:
     Title: Using RNA sample titrations to assess microarray platform performance
     URL:
   ```

```
PMIDs: 16964226

Abstract: A 158 word abstract is available. Use 'abstract' method.
```

(b) abstracts

```
> substr(abstract(afxsubRMAES), 1, 70)
[1] "We have assessed the utility of RNA titration samples for evaluating m"
```

(c) sample-level variables

```
> names(pData(afxsubRMAES))
[1] "site"      "samp"      "repl"      "pctBrain"
```

(d) for some 'custom' assays, `featureData` details can be important for understanding assay reporters; for manufactured assays these are factored out into annotation packages

3. **investigate values of expression assays** – `exprs([esetname])[r,c]` where `r` and `c` delimit 'genes' and samples of interest respectively.

   Note that Bioconductor structures support associative memory/polymorphic indexing in various ways:

```
> afxsubRMAES["1557970_s_at", ]  # explain

ExpressionSet (storageMode: lockedEnvironment)
assayData: 1 features, 24 samples
  element names: exprs
phenoData
  sampleNames: AFX_1_A1.CEL, AFX_1_A2.CEL, ..., AFX_3_D2.CEL  (24 total)
  varLabels and varMetadata description:
    site: from cel
    samp: rna src/mixture code
    repl: replicate
    pctBrain: pct of mixture from Ambion brain
featureData
  featureNames: 1557970_s_at
  fvarLabels and fvarMetadata description: none
experimentData: use 'experimentData(object)'
  pubMedIds: 16964226
Annotation: hgu133plus2

> exprs(afxsubRMAES)[4000,1:3]

AFX_1_A1.CEL AFX_1_A2.CEL AFX_1_B1.CEL
    5.572803     5.718472     7.285622
```

```
> exprs(afxsubRMAES)["1557970_s_at",1:3]

AFX_1_A1.CEL AFX_1_A2.CEL AFX_1_B1.CEL
    5.572803     5.718472     7.285622

> exprs(afxsubRMAES)["1557970_s_at",c("AFX_1_A1.CEL", "AFX_1_A2.CEL", "AFX_1_B1.C

AFX_1_A1.CEL AFX_1_A2.CEL AFX_1_B1.CEL
    5.572803     5.718472     7.285622
```

4. **understand resolution of identifiers using annotation maps**

```
> library(annotate)
> library(hgu133plus2.db)
> lookUp("1557970_s_at", "hgu133plus2", "GENENAME")

$`1557970_s_at`
[1] "ribosomal protein S6 kinase, 90kDa, polypeptide 2"
```

The annotate lookUp facility is very basic. It was designed when Bioconductor annotation maps were one-directional, with probe identifiers as keys and biological terms or tokens as values, as shown above.

Newer facilities (with which original facilities currently coexist) are based on the relational database system SQLite. A simple application, commonly required, finds the probe identifier(s) for a certain gene symbol:

```
> rmap = revmap(hgu133plus2SYMBOL)
> get("A2M", rmap)

[1] "217757_at"

> mget(c("A2M", "CRP"), rmap)

$A2M
[1] "217757_at"

$CRP
[1] "205753_at" "37020_at"
```

A deeper view of what is going on here can be obtained via DBI (database interface) facilities. This requires an acquaintance with SQL.

```
> co = hgu133plus2_dbconn()  # hook to the database
> co
```

```
<SQLiteConnection:(17287,2)>

> dbListTables(co)

 [1] "alias"                  "chrlengths"             "chromosome_locations"
 [4] "chromosomes"            "cytogenetic_locations"  "ec"
 [7] "ensembl"                "gene_info"              "genes"
[10] "go_bp"                  "go_bp_all"              "go_cc"
[13] "go_cc_all"              "go_mf"                  "go_mf_all"
[16] "kegg"                   "map_counts"             "map_metadata"
[19] "metadata"               "omim"                   "pfam"
[22] "probes"                 "prosite"                "pubmed"
[25] "refseq"                 "sqlite_stat1"           "unigene"

> lk133 = function(x) dbGetQuery(co, x)  # shorthand helper
> lk133("select * from probes limit 200,5")

     probe_id accession   _id
1   1552541_at NM_138810 18492
2 1552542_s_at NM_138810 18492
3 1552543_a_at NM_033104 17469
4   1552544_at  BC040857 19732
5 1552546_a_at NM_144652 19354

> lk133("select * from gene_info limit 200,5")

  _id
1 254
2 255
3 257
4 258
5 259

1
2                                                                           ankyrin 3,
3 alanyl (membrane) aminopeptidase (aminopeptidase N, aminopeptidase M, microsomal
4                  solute carrier family 25 (mitochondrial carrier; adenine nucleo
5                  solute carrier family 25 (mitochondrial carrier; adenine nucleo
   symbol
1    ANK2
2    ANK3
3   ANPEP
4 SLC25A4
5 SLC25A5
```

From the AnnotationDbi vignette, we adapt an example. We wish to find all the probes on illuminaHumanv2 that have a GO MF function ascribed, with evidence code TAS (traceable author statement).

```
> mft = lk133("SELECT symbol FROM go_mf INNER JOIN gene_info
+    USING(_id) WHERE go_mf.evidence = 'TAS'")
> dim(mft) # will be data.frame

[1] 5100    1

> mft[1:3,]

[1] "NAT1"     "NAT2"     "SERPINA3"
```

5. **Perform analyses of high-throughput experiments**. This 'essential' repertory component is not regarded as a preliminary for the course. This is the intended content of the course.

4 Exercises

1. **Vector element annotation.** Attach MAQCsubset package, and load the afxsubRMAES dataset:

   ```
   > library(MAQCsubset)
   > data(afxsubRMAES)
   ```

   Now grab some expression values:

   ```
   > myv = exprs(afxsubRMAES)[1000:1004, 1]
   > myv

     1553623_at   1553625_at 1553626_a_at 1553627_s_at 1553629_a_at
       3.772405     2.688321     4.047763     3.229765     4.777014
   ```

   Replace the probe names on this vector with the HUGO symbols. You should get:

   ```
   > myv
   MGC15705   FAM98B C17orf57 C17orf57   FAM71B
   3.772405 2.688321 4.047763 3.229765 4.777014
   ```

2. **Revising phenoData values**. Create a revised version of afxsubRMAES where the percentage of Ambion brain has the customary units of percentage measures ranging from 0-100 (the current representation has proportions, even though the name pctBrain suggests percentages).

3. **Improving the self-documentation of an ExpressionSet**. MAQCsubset package has illumina data in an object called `ilmMAQCsubR`. Access it

   ```
   > data(ilmMAQCsubR)
   ```

   Create a vector, `myb`, defining the percentage of Ambion brain present in each sample. Attach it to the phenoData of the ilmMAQCsubR through the operation

   ```
   > ilmMAQCsubR$pctBrain = myb
   ```

   Now mention the resulting ExpressionSet to R and note the difference in the reports. An NA is shown in the report; modify the varMetadata component to repair this.

   Insert correct information for experimentData and annotation slots as well. The experimentData can be borrowed from the afxsubRMAES, and the annotation is illuminaHumanv1.

4. **Writing a statistical function.** A two-sided t distribution-based $1-\alpha\%$ confidence interval for a mean takes the form $\bar{x} \pm k_{1-\alpha/2,n-1} s_x/\sqrt{n}$ where $\bar{x}$ is the sample mean, s_x is the sample standard deviation, n is the number of observations used for the mean, and $k_{1-\alpha/2,n-1}$ is the $1-\alpha/2$ quantile of the t distribution with $n-1$ degrees of freedom. Write a function that takes a vector as input along with a parameter clev $= 1-\alpha$ and returns a vector with the lower and upper bounds of the two-sided $1-\alpha$ level confidence interval for the mean.

   Solution:

```
> myCI_t = function(x, clev) {
+     mx = mean(x)
+     n = length(x)
+     sxc = sd(x)/sqrt(n)
+     k = abs(qt((1 - clev)/2, df = n - 1))
+     c(-k * sxc + mx, k * sxc + mx)
+ }
> set.seed(1234)
> tt = rnorm(100)
> myCI_t(tt, 0.95)
```

```
[1] -0.35605755  0.04253406
```

   Additional problems.

   (a) Verify that this function computes the correct CI by comparing with calls to t.test for one sample testing.

   (b) create a new function, `myCI_n`, in which you substitute `qnorm` for the `qt` call in `myCI_t`. Compare widths of CIs generated under normal- and t-based radius lengths for samples of size 10, 50 and 100.

5. **Damping outlier effects: winsorization.** The k-winsorization of an n-sample x with order statistics $(x_{(1)}, \ldots, x_{(n)})$ is the sample with $2k$ modifications leading to order statistics $(x^{*k}_{(k)}, x_{(k+1)}, \ldots, x_{(n-k)}, x^{*k}_{(n-k+1)})$, where the operator x^{*k} produces k copies of datum x. Concretely, suppose the sorted data are 1,2,3,4,5,6,7, then the 2-winsorization is 2,2,3,4,5,6,6 – we replace the k extreme values at each end with copies of the kth order statistic at the low end and the $n-k+1$st at the high end.

   Write a function `k_wins`, with parameters `x, k` that k-winsorizes a vector `x`.

6. **Simulating a test to evaluate its size.** Explain the following:

```
> ps = sapply(1:5000, function(x) t.test(rnorm(20))$p.value)
> mean(ps < 0.05)
```

```
[1] 0.0484
```

```
> psw = sapply(1:5000, function(x) t.test(k_wins(rnorm(20), 2))$p.value)
> mean(psw < 0.05)
```

```
[1] 0.073
```