library(AdvancedR2011)

**all the slides are here: browseVignettes("AdvancedR2011")**

**or here: http://www.bioconductor.org/help/course-materials/2011/AdvancedRFeb2011Seattle/**

Session 1: Efficient R programming

Easy stuff

• Input only the required data! (use colClasses feature of read.table)

Saves time and memory

• Preallocate and fill, do NOT append.

Something to do with R copying the whole data each time there’s an append.

Data.frames work as columns – sometimes things are slow by columns but not by rows. Might be better to build up the columns and THEN make the dataframe.

The apply functions are very efficient. One reason is because they pre-allocate space for the results. Although they still do an iteration, so if something can be vectorized instead that might be better.

• Vectorize, not iterate (often because someone’s written C code that gets operated under the hood that is very efficient).

e.g. rowSums is more efficient than iterating (including the apply way to iterate, which is still better than plain iteration)

R is an interpreted language. Does a lot of things each time it goes through a loop (e.g. how to subset, how to square-root, whatever), so if you can use a function that gets to systems level (compiled) language that’s better.

• avoid unnecessary character creation operations (e.g. use.names = F in unlist, USE.NAMES=F in sapply).

(list manipulation: e.g. in a list of multiple elements, each of which has a named variable “Het”, to get all those datapoints do this: lapply(lst,”[[“,”Het”)

Moderate stuff

• Use pre-existing packages. E.g. for linear model fit on microarray data, lmFit is an efficient way to implement lm using a constant model/designmatrix for lots of genes. Does mean you’re restricted to using same assumptions as package author.

• %in% is an O(N) operation, but a naïve implemetation (compare each versus each) would take O(N2).

Philosophical point: in order to speed things up need to (a) understand why its slow (realize that naïve implementation is inefficient) (b) find the equivalent efficient R function.

• alternative: use C or Fortran, but that’s quite a big step. Sometimes it’s worth it. E.g. sliding window approaches, maybe.

(multicore: lapply type operations across multiple cores, memory efficient. mclapply )

Measuring performance

system.time (gcFirst=T for garbage collection)

m <- matrix(runif(200000),20000)

replicate(5,system.time(apply(m,1,sum))[[1]])

replicate(5,system.time(rowSums(m))[[1]])

much quicker

tinyurl.com/29bd6xv – this says that read.table (and scan) can handle gz compressed files, and that it actually reads data FASTER from those than from the original plain text file.

Garbage collector: e.g. scans through the allocated memory and if it’s not stil being used, reclaims it. e.g. if x is a big thing, and you delete it, that memory might not be freed up until garbage is collected. Garbage collection gets triggered e.g. when you try to allocate new big memory object. Might not happen if we’re not close to running out of memory.

When replacing old code with more efficient code, use “identical” to check whether output is same (in all respects: class, name, etc). “all.equal” is a bit more sloppy (e.g. can ignore names if use check.attributes=F).

Floating point numbers. Double-precision. Rounding errors. Functions that truncate numerical representations: zapsmall signific

(typeof versus class: example y <- c(1,2) typeof and class give different results)

Measuring execution time

Rprof

tmpf <- tempfile()

Rprof(tmpf) ### opens the output file

res1 <- apply (m,1,sum)

Rprof(NULL) ; summaryRprof(tmpf)

can put any code between the two Rprof commands. Sampling approach: interrupts R at intervals and asks what its doing.

(the file itself may contain enough information to distinguish similar code lines. but a typical approach is to do it on big chunk of code, isolate the slow bit, break it down to smaller code chunks, etc, etc)

pct = percent

Measuring memory use

object.size()

R-devel : when you build, can configure to enable memory profiling ( --enable-memory-profiling) – then a function called tracemem is available.

x <- 1:10

tracemem(x)

### will spit out a message any time x changes its memory location

y <- x

### initially, x and y point to the same place in memory (for efficiency), but when we change x, now we need a new copy of y

x[1] <- 2L ### the L means integer, default is numeric

x[1] <- 2 ### because I’m changing integer to numeric it makes a whole new copy of x with the class change.

x[1] ## this call actually creates a new object in memory for the subsetted item that didn’t exist in memory before.

Functions: if we only access the object being passed in in a “read” manner, it doesn’t get copied. If we modify it, memory gets allocated/a copy gets made.

R also has some kind of global cache: if we a fasta file of lots of identical equal DNA sequences, R has an efficient way to store it – it knows they’re identical strings.

Session 2: R packages

require() is like library() but designed for use inside other functions: gives warning if package not available, rather than error). Both require and library attach the code to the current R session.

Why make a package?

organizes code, data, analyses

lets you share code

reliable access

there are tools allowing quality control

Package source has mandatory structure:

top level dir should be same name as package name

below that:

files:

DESCRIPTION file (has mandatory fields)

NAMESPACE file (optional, but encouraged)

other optional: configure, LICENSE, COPYING, NEWS

subdirs:

R (contains .R files with code)

data (will be loaded by data() )

inst (user can see all the stuff in the inst dir)

can include

doc (Sweave documentation: .Rnw format – used to document general description of what package does, demonstrate functions, informal)

extdata (misc data objects for the user to see e.g. ASCII files)

unitTest

man R documentation (.Rd format, strict formal structure = reference page for functions, objects, classes)

src foreign language code

tests test code (not as visible to the user) (R CMD CHECK tests this code. can also call stuff in unitTest from here)

Collate line in DESCRIPTION file determines order in which the files in R subdir will be loaded.

NAMESPACE file: we might want to import variables/functions from other packages. Also defines which functions in this package are visible to the user.

package.skeleton() is helpful – it would create the basic files. Then we would go to the shell and use e.g.

R CMD build package

R CMD check package

R CMD INSTALL package

R CMD check –help

R CMD INSTALL –build (this would generate binary archive, useful for Windows)

Package developer would mess with source code of the package, and then INSTALL then puts it in the R library

check: checks a lot of things

?str – see data structure

Session 3: SQL and databases

good intro book: SQL in 10 minutes

why do it? avoids storing data in RAM. fast access. data exchange with other people/tools.

relational dbs are used in Bioconductor e.g. annotation packages (org.Hs.eg.db), some software packages that relate to annotation (GenomicFeatures, Genominator, AnnotationDbi)

SELECT \* FROM gene (\* here = all fields. use WHERE to search the data)

gene.\_tx\_id (means the \_tx\_id field from gene table, although in the example it’s unnecessary)

AS: use little aliases to save typing

INSERT: OK to leave empty fields

INDEX: do this on things you look up a lot to speed up. Creates a file on hard drive, so only do it on thigs we use a lot. Don’t need to do anything differently after indexing.

Can be safer to ORDER BY because things sometimes come back in random orders.

In the subquery example, the subquery is actually redundant but in more complicated examples it can be useful.

Can be easier when figuring out SQL commands to do it directly in SQL not in R. So perhaps install SQLite, go to director that represents the annotation package e.g. R-2-12/library/org.Hs.eg.db/extdata, run sqlite3 org.Hs.eg.sqlite

.h on (turn headers on)

.m column ( go into column mode)

.ta (list the tables)

LIMIT 5 (would show the first 5 results)

(sqlite is a little different from mysql. has some nice functions, so Marc likes to use that.)

Inner Joins only take the ids that are in common between the two tables. Outer join might allow you to see all items in one of the tables whether or not they appear in the other table.

Session 4: Handling large data

NetCDF format is a special format, transferable, has special tools to access specific chunks quickly, has lots of libraries available for all kinds of programming languages. Originally developed for earth sciences, atmospheric sciences

version 3? version 4?

3 – easy to install, works on all platfoms

4; not on windows, harder to install, but offers additional features e.g. expanding data in multiple dimensions

in var.def.ncdf in the example we specify sampDim first as we want faster access to that

in slice2 command, specifying a -1 in the count will get the entire column or row or whatever)

Session 5: Documentation

Rnw = Sweave format

In help page: R CMD check will try to run the code in examples. It is also strict about checking all the help doc(s).

R CMD Rd2dvi –pdf is another way to check the documentation (just visually?)

Sweave file format: usepackage refers to Sweave package not R package.

code chunks enclosed by

<< chunkname, maybe some options>>=

@

Stangle extracts just the R code from an Sweave doc

Session 6: S4 class system

(old original system = S3 – S4 much more recommended)

part of methods package which is always loaded as base

when we call a method we don’t include class name. R uses method dispatch to figure out which to use.

core components: classes, generic functions and methods

method = a type of function that works out the appropriate actual function depending on object class.

is.generic (functionname)

ls(‘package:methods’)

methods package is big and scary. very different from other OO languages.

mostly transparent to end use but sometimes errors hard to understand and can be hard to find documentation.

S4 objects can come from all kinds of places – constructors, downloads, data packages, coercion

S4 way to change (coerce) class:

library (Matrix)  
m <- matrix (……)

as(m, “Matrix”)

S4 objects can be nested inside other S4 objects. “Slots” = portion of an S4 object. There are special accessor functions for things in these slots. e.g. getters, setters

str(S4object) = a way to list the slots and their contents of an object (i.e. an instance)

showClass("GWASdata") = lists the slots of the class definition

finding man pages:

class?graphNEL

?`graphNEL-class`

?function (for a generic function)

but the man page might not include all methods that use the function. Maybe the SeeAlso page will help

showMethods(generic-function)

showMethods(“reduce”)

?`qa,ShortReadQ-method` (normal single quotes)

??qa

class(thing) (shows class of an instance/object)

showClass(“classname”) (lists slots)

selectMethod("reduce","IRanges")

e.g. – shows the code of the reduce function for IRanges objects

# some signatures have two elements

selectMethod("writeFastq",c("ShortReadQ","character"))

Example of how we’d implement a class:

1. define the class

setClass – sets up the class with name and slots (but can’t get too specific at this point – e.g. can specify character but can’t specify that it’s a single string not a vector, or that it shouldn’t be NA)

2. set up a constructor function (recommended to name this same as class name)

new (can be a very simple wrapper, or can call stuff before we call “new”)

3. call the constructor

4. set up some methods e.g. the length method

setMethod

(and then the method dispatch mechanism will know to use length for this class, not the base length function). BUT: be careful of over-writing function names with generic functions.

If (and only if) setMethod calls a generic function that doesn’t yet exist, we have to make a generic function before we use setMethod

setGeneric

setGeneric is a little dangerous – only do it if there is no symbol existing already for that function.

slot accessor (for the developer) @

e.g. x@genome

(@ is S4 style. Older: $ is S3 style, because objects are lists in S3)

but the accessor functions are meant to shield the end user from using @

5. always useful to have a “show” method (to avoid getting long ugly output on the screen). show is a generic defined in methods. We need to know the correct signature for generic functions like this and follow the existing signature. Some generics include a “…” argument.

getGeneric(nrow) will show me the signature of a method. I should use the same variable names as the generic (perhaps same number inputs, outputs?)

6. setValidity : a method that will check the new objects look OK. Returns TRUE if object passes, or an informative error string if not. Don’t use stop inside validity method

validObject(object) – can be used to set off the checking.

it’s also automatically called when object is created (new function) but not when we modify it.

subclasses: don’t need to check validity of parent class again, just the extra stuff.

7. define some coercion methods

setAs (has special syntax using “from”)

child classes: setClass (…….. contains=”parentclass”)

inherits the methods (accessors etc) of the parent class. (but sometimes you need a slightly different method, so you’d overwrite the parent method)

new: if we don’t name an argument, it’s assumed to be an object of the parent class

(virtual class = abstract class in Java)

Class unions ?setClassUnion

Multiple inheritance: >1 parent class. use with caution. can be hard/impossible to maintain. Aim to keep it easy to understand and maintain.

?is ?as ?selectMethod ?getMethod Writing R Extensions web page

showMethods(class=…)

re-use chunks of code as much as possible (put all in functions)

Session 7: Foreign language calls

.C

.Call (more flexible but there’s a lot more to learn. It’s a whole API. Lots of complicated things about protecting memory)

the composite linkage diseq void routine preallocates “result” object

.onLoad

.onUnload

in one of the .R files

will make sure stuff happens when an R package is loaded

(there’s an error in the slides that’s fixed in the lab docs)

C indexing is 0-based

**(should I look at C or C++)**

debugger

R –d gdb mycodetotest.R

starts gdb

run

(gets me into an R session)

source (“mycode.R”)

ctrl-C to get back into gdb

break cld\_two\_snps (will break the C code at a specific poitn – can tab-complete)

c (continue)

at the break points we can check what the values of the variables are.

Final session –finishing up our R package

<http://www.bioconductor.org/help/course-materials/2010/AdvancedR/using-gdb-with-R.mov>

Rforge – somewhere we can store our own packages to be publically available – don’t have to be polished or finished. As a developer, it’s useful, but as a user be warned – they can be incomplete. CRAN and Bioconductor archives are more polished – should be mostly bug-free. Bioconductor review: does not look at the science – just the technical side – often a lot of changes during the review process in collaboration with authors).

Using BiocLite should avoid version incompatibility issues.

Depends vs Imports:

Depends – loads these packages, attaches to R session

Imports – not loaded/attached, and user won’t know/see they’re being used.

Suggests – other packages used in examples or man pages. Or if user probably won’t use the function so you can risk not requiring it.

prompt tool for making the man pages doesn’t do quite what BioC people want – they have their own way.

documentation for classes can be really long and complicated and tedious to make

“See Also” section of man pages can be very useful

no need to write man pages and vignettes for internal functions/classes etc just the ones the user will see

help.start () for html help on EVERYTHING and can search help from here.

General notes

(raw is a very compact type similar to integer, double, logical, character i.e. one of the five atomic vector types. raw uses 1 bytes per element, so can store values from 0-255 in each element. raw doesn’t support NAs and can’t do arithmetic – would need to coerce to integers first. Can do stuff like tabulate. Can do stuff at R level. Can use bitops package e.g. bit-ands can be useful in some cases)

John Chambers book: Programming With Data

Robert Gentleman: R for Bioinformatics

Learn about variable scope and whether variables used inside functions stick around