Programming languages

Worthy of further emphasis:

- **C**: serious computation
- **Perl**: text manipulation
- **R**: graphics and interactive analysis

A serious statistician should be fluent in *all* of these languages.

Steven Salzberg (Director of Bioinformatics, The Institute for Genomic Research) [personal communication]:

- Programs essentially exclusively in C and perl.
- Can read C++, and his programmers use it, but he doesn't need to use it himself.
- Wrote a Fortran program in the 70s, but (strongly) recommends against using it now.

Thus: I'm right. **Learn C and perl.** It will make it easier to (a) get things done, and (b) get a regular job (should you want one).

Programming

The central concepts:

- Readable
- Reusable

The methods:

- Find a clear style and stick to it.
- Write clearly.
- Comment the tricky parts.
- Make things general.
- Make things easy for the user (generally yourself).
- Incorporate error checking: data and output integrity; reasons/locations of crashes

Method of delivery: *R packages*

- R takes care of data I/O.
- Make use of R's functions and math library
- The pieces are unified.
- Quick, useful documentation
- Include sample data
- Seamless integration into R

R packages

Refs: "Writing R extensions," `http://cran.r-project.org`
MASS: "R complements"
S programming (Venables and Ripley)

Installing/Using R packages

Locations of libraries: `/usr/local/lib/R/library`
Alternatives: include `R_LIBS=/loc/of/mylibs` within `~/.Renviron`

Installing:

```
R INSTALL -l /loc/of/mylibs name.tar.gz
```

Using:

```
library(help=qtl)
library(qtl)
detach(package:qtl)
```

To use in all sessions:

Create a file `~/.Rprofile` containing commands to be invoked in every session.

For example:

```
library(ctest)
library(qtl)
attach("/home1/biostats/kbroman/R/.RData")
options(show.signif.stars = FALSE)
ps.options(horizontal = FALSE)
```

Contents of R packages

Files: DESCRIPTION, INDEX
Directories: R, data, man, src, tests, inst, exec

DESCRIPTION file:

```
Package: qtl
Version: 0.65
Date: 09/20/2000
Title: Tools for analyzing QTL experiments
Depends: R (>= 0.99)
Author: Karl W Broman (kbroman@jhsph.edu)
Description: Analysis of experimental crosses...
License: GPL (version 2 or later)
URL: http://biosun01.biostat.jhsph.edu/...
```

INDEX file: one line description for each object

```
R CMD Rdindex man/*.Rd > INDEX
```

R subdirectory:

Contains files containing the R code (preferably with .R extension); it should be possible to read in the files using source(), so R objects must be created via assignment. (File names don't matter.)

If you use compiled code, one file (historically named zzz.R) is used to load the code:

```
.First.lib <- function(lib,pkg)
    library.dynam("qtl", pkg, lib)
```

Contents of R packages (continued)

man subdirectory:

Contains documentation files for the package in R doc format (`.Rd`). There should be one file for each object. Here's an example:

```
\name{npem.em}
\alias{npem.em}
\title{Fit the normal-Poisson model...}
\description{
  Uses a version of the EM algorithm...
}
\usage{
npem.em(y, ests, cells=10^6, n=c(24, 24, 24, 22),
        n.plates=1,use.order.constraint=TRUE,
        maxit=2000, tol=1e-06, maxk=20, prnt=0)
}
\arguments{
  \item{y}{Vector of transformed...}
  \item{ests}{Initial parameter estimates...}
  \item{cells}{Number of cells per well....}
  \item{n}{Vector giving the number of wells...}
  ...
}
\details{
  Calculations are performed in a C routine.
}
\value{
  \item{ests}{The estimated parameters...}
  \item{k}{The estimated number of...}
  \item{ksq}{E(k^2|y)}
  \item{loglik}{The value of the log...}
  \item{n.iter}{Number of iterations performed.}
}
```

R documents (continued)

```
\references{Broman et al. (1996)...}
\author{Karl W Broman, \email{kbroman@jhsph.edu} \cr
\url{http://biosun01.biostat.jhsph.edu/...} }

\seealso{\code{\link{npem.sem}},
  \code{\link{npem.start}},\code{\link{npsim}},
  \code{\link{npem.ll}}}

\examples{
  # get access to the library and a dataset
  library(npem)
  data(p713)

  # analysis of plate3
    # get starting values
  start.pl3 <- npem.start(p713$counts[[3]],n=p713$n)
    # get estimates
  out.pl3 <- npem.em(p713$counts[[3]],
                     start.pl3,n=p713$n)
    # look at log likelihood
  npem.ll(p713$counts[[3]],start.pl3,n=p713$n)
  npem.ll(p713$counts[[3]],out.pl3$ests,n=p713$n)
}

\keyword{}
```

Note: The `R` and `man` subdirectories may contain OS-specific subdirectories named `unix`, `windows` and/or `mac`.

Contents of R packages (continued)

`src` subdirectory:

Contains C/FORTRAN source files. Also may contain the files `Makevars` and `Makefile`, if necessary. When the package is installed, the `make` program is used to compile and build these source files.

`data` subdirectory:

Contains additional (example) data files to be loaded with `data()`. These may be plain R code (`.R`), tables (`.tab`, `.txt` or `.csv`) or `save()` images (`.RData` or `.rda`).

The subdirectory should contain a `00Index` file with one-line descriptions of the datasets; full documentation should be placed in the `man` subdirectory.

`inst` subdirectory:

The contents of this directory are copied recursively to the installation directory (e.g., a `STATUS` file).

Contents of R packages (continued)

`tests` subdirectory:

Contains package-specific test code (`.R` files with `.Rout.save` files.) The `.Rout.save` files are compared to test results which go to `.Rout` files.)

`exec` subdirectory:

Contains any other executables needed by the package (typically shell of perl scripts). Apparently, this is still experimental.

Checking, building and installing

`R CMD build name`

Builds R package from source. Creates a gzipped tar file (e.g., `npem_0.50.tar.gz`)

`R INSTALL -l /loc/of/mylibs name.tar.gz`

Installs the `name` package to `/loc/of/mylibs`.

`R CMD check name`

Checks the examples in the package documentation; runs the tests in the `tests` subdirectory. You may need `--nsize=__ --vsize=__`

Calling C from R: outside a package

Arguments

I stick with *integers* and *doubles*. It is possible to work with strings and lists and such, but it seems complicated.

Within R, you need to use the functions `as.double()` and `as.integer()`.

Within C, everything is a pointer: `double *` or `int *`.

Return values must be given as arguments to the C function.

`.C()` is the function for calling your C programs.

Shared libraries

Use, at the unix prompt, `R SHLIB myprog.c` to create a "shared library" for loading into R.

Loading/unloading

Within R, use the functions `dyn.load()` and `dyn.unload()` to load and unload shared libraries.

Calling C from R: example

C code: test_r2c.c

```
#include <math.h>
void mean_and_sd(double *x, int *n,
                 double *mean, double *sd)
{
  int i; double a;

  *mean = *sd = 0.0;
  for(i=1; i< *n; i++) {
    a = x[i] - x[0];
    *mean += a;
    *sd += a*a;
  }
  *sd = sqrt((*sd - (*mean)*(*mean) / (double)*n)/
             ((double)(*n-1)));
  *mean = (*mean / (double)*n) + x[0];
}
```

R code: test_r2c.R

```
mean.and.sd <-
function(x)
{
  if(!is.loaded("mean_and_sd")) {
    dyn.load("test_r2c.so")
    cat(" -Loaded test_r2c.so\n")
  }
  z <- .C("mean_and_sd",
          as.double(x),
          as.integer(length(x)),
          mean = as.double(0),
          sd = as.double(0))
  c(mean=z$mean,sd=z$sd)
}
```

Calling C from R: example (continued)

From the unix prompt:

```
> R SHLIB test_r2c.c
```

[Creates the files `test_r2c.o` and `test_r2c.so`]

Within R:

```
> source("test_r2c.R")
> x <- rnorm(100,10,3)
> mean.and.sd(x)
 -Loaded test_r2c.so
    mean       sd
10.15874  2.94779
> dyn.unload("test_r2c.so")
```

Calling C from R within a package

Compilation

You don't need to worry about compiling the code or creating a shared library; that is done when the package is built (using `R CMD build`).

Loading

You don't need to worry about loading the shared library using `dyn.load`; this will be done when the library is loaded using `library()`.

Entry points for the R API

Header files: `/usr/local/lib/R/include/`

```
#include <R.h>
#include <R_ext/*****.h>
```

Memory allocation

- Send workspace as an argument.
- Transient method (R does the cleanup; taken from heap [`VSIZE`]): `R_alloc` and `S_realloc`.
- User-controlled method (in addition to R's memory; user must free): `Calloc`, `Realloc`, `Free`.

Error handling

`void error()` and `void warning()` take arguments like `printf`; act like `stop()` and `warning()` within R.

Random number generation

```
GetRNGstate();
PutRNGstate();
double unif_rand();
double norm_rand();
double exp_rand();
```

`rxxxx` functions — include `R_ext/Mathlib.h`.

Entry points for the R API (continued)

Printing: include `R_ext/PrtUntil.h`

`Rprintf` — guaranteed to write to R's output (be sure to write a `\n` before returning to R).

`REprintf` — writes to `stderr`

Numerical analysis

`R_ext/Linpack.h`
`R_ext/Mathlib.h`

Utilities

`R_ext/Mathlib.h`
`R_isort()`

Distribution functions

`R_ext/Mathlib.h`
`dnorm(), pnorm(), qnorm(), rnorm(), ...`

Other utilities

`log1p()` calculates `log(1+x)` for small `x`.

Constants (to 30 digits):

`M_E`, `M_PI`, etc. [within `R_ext/Mathlib.h`]

R package example: R/npem

http://biosun01.biostat.jhsph.edu/~kbroman/software.html

I develop this in the directory ~/Code/Rnpem, which contains the subdirectory npem containing the code within subdirectories R, data, inst, man and src.

R subdirectory:

npem_em.R, npem_sem.R, npem_start.R, zzz.R

The file npem_em.R contains definitions of the functions npem.em, npem.ll and npsim. The first two call C functions.

```
npem.em <-
function(y, ests, cells=10^6, n=c(24, 24, 24, 22),
         n.plates=1, use.order.constraint=TRUE,
         maxit=2000, tol=1e-6, maxk=20, prnt=0)
{
  ...

  z<-  .C("npem_em",
          as.double(y), as.integer(n.plates), ...)
  list(ests=z$ests,k=z$k,ksq=z$ksq,loglik=z$loglik,
       n.iter=z$n.iter)
}
```

The file npem_sem.R contains the definition of npem.sem, which calls a C function, but does some matrix multiplication within R to refine the output. The file npem_start.R defines a function which calls another C function to get starting points.

R package example: R/npem (continued)

data subdirectory:
Contains files `p711.RData`, `p713.2.RData` and `p713.RData` with *real* example data, as `save()` images.

inst subdirectory:
Contains a file `STATUS` describing the history of program updates.

man subdirectory:
Contains documentation files (`.Rd`) for the five functions and three datasets.

src subdirectory:
Contains four `.c` files and one header file (`.h`) with the code that does all of the work.

To build the package, I type `R CMD build npem` from within the directory `~/Code/Rnpem`; this creates the file `npem_0.50.tar.gz`.

I can then install this package using `R INSTALL npem_0.50.tar.gz`

I'll pass to the next example. Feel free to take a look at this code. The package was put together quickly, and so isn't even close to perfect.

R package example: R/qtl

I am in the process of developing an R package for mapping QTLs ("quantitative trait loci" — genes contributing to variation in quantitative traits) in experimental crosses.

The idea is to have a *QTL mapping environment* for the sophisticated user, composed of general, flexible tools for dealing with a lot of the grunt worth associated with QTL mapping, so that the user may focus more on modelling than on computing.

Data format:
I want to be able to work with different types of experimental crosses (F_2 intercrosses, backcrosses, RI lines, advanced intercross lines, etc.). The data from such a QTL mapping experiment is composed of the following:

- *Genetic map*: For each of a number of chromosomes, specify the order, locations and names of the typed genetic markers.
- *Genotype data*: A matrix giving the genotype for each individual at each marker.
- *Phenotype data*: A matrix giving the phenotype for each individual at each of possibly multiple traits. This may also contain information on *covariate* such as sex, age and treatment.

R package example: R/qtl (continued)

Data format in R: A list with *class* `"qtlcross"` and components:

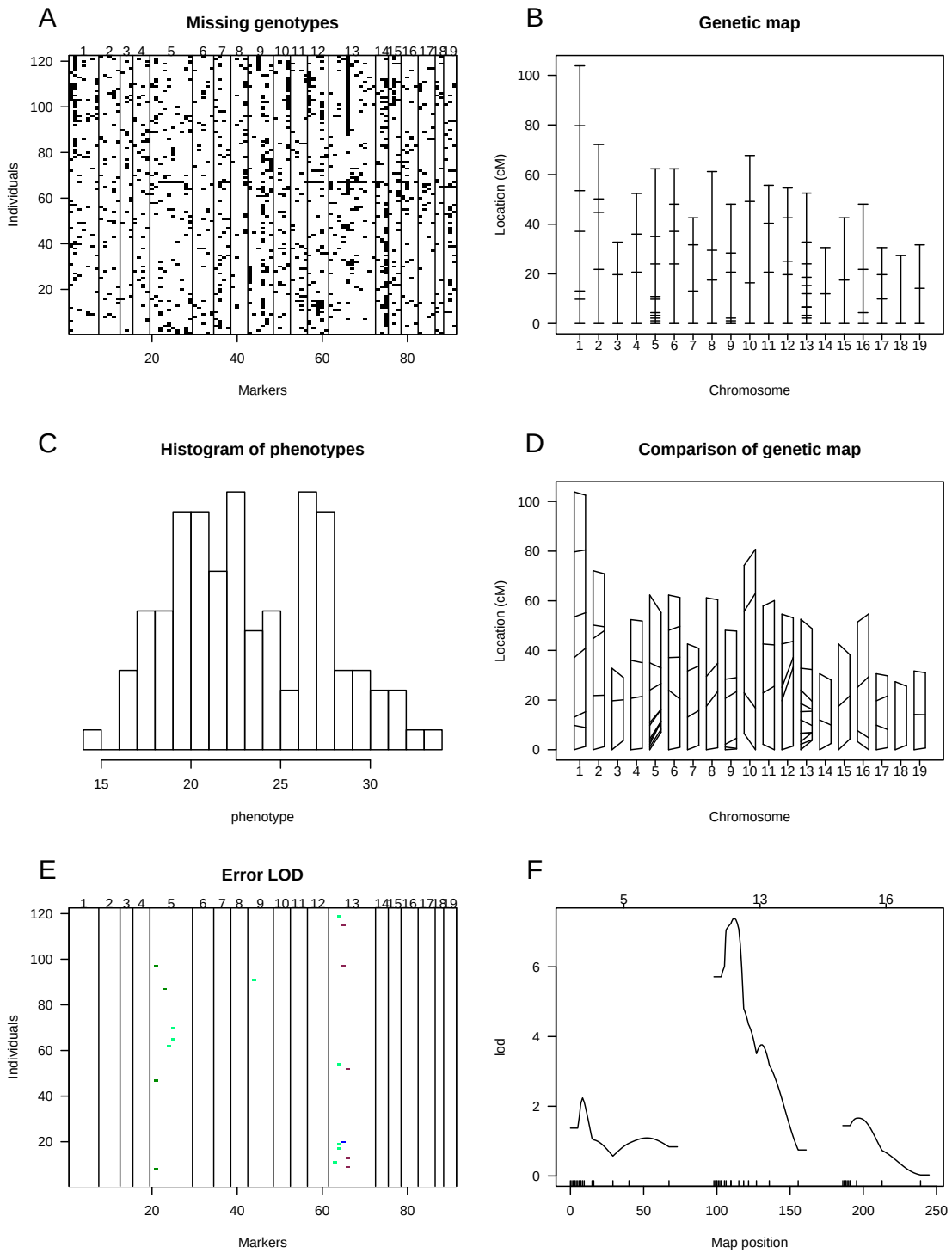
- `geno`: a matrix of integers of size `nind` by `totmar`; markers are in order along chromosomes. Column names are the marker names.
- `pheno`: a matrix of size `nind` by `nphe` containing the phenotypes.
- `map`: a list of length `nchr`; each component is a vector giving the locations of the markers on a chromosome. This component is given class `"map"`.
- `type`: takes value either `"f2"` or `"bc"`.

I've included a couple of example datasets (both are fake data); `data(fake.bc)` and `data(fake.f2)` allow access.

R package example: R/qtl (continued)

Utility functions:

- `read.cross` reads in data for a cross, from multiple files. The arguments are `format`, `dir`, `genfile`, `phefile`, `mapfile` and others. The allows the possibility of different data formats (possibly containing different numbers and types of files).
- `summary.qtlcross` allows one to get some simple information on a cross, using just `summary(cross)`, since the cross will have *class* `"qtlcross"`. This function returns a list with class `"qtlcross.summary"`, which is printed using the function `print.qtlcross.summary`.
- The functions `nind`, `nmar`, `totmar`, `nchr`, and `nphe` make it easy to get bits of information about a cross.
- Since the genotype data is a matrix with all chromosomes pasted together, `which.marpos` spits out which columns correspond to a particular chromosome. `pull.chr` creates a new `qtlcross` object with just the selected chromosomes.
- `sim.cross` allows for the simulation of data from QTL experiments. Like the rest of this, it's rather crude at the moment.



Example output from the prototype R/qtl program. A: Grid showing missing genotype data. B: Genetic map of typed markers. C: Histogram of observed phenotypes. D: Comparison of two genetic maps (e.g., the standard map against that estimated with the observed data). E: Grid indicating genotypes likely to be in error. F: LOD curves for three chromosomes, indicating inferred locations of QTLs.

R package example: R/qtl (continued)

Plotting routines:

- `plot.geno` uses the function `image` to plot a grid showing missing genotypes. Because this uses information on the genetic map (to put lines at the boundaries between chromosomes), it requires the full `qtlcross` object.
- `plot.map` plots a genetic map. It figures out whether it has been called with a `qtlcross` object or a `map` object. You can just use `plot(map)` if you want, since genetic maps are given class `map`. This function will also plot two different maps (with the same numbers of markers per chromosome) against each other.

  The function `est.map` re-estimates the genetic map (using the HMM technology and the EM algorithm) using the observed genotype data. It's nice to plot the original map against the new map.
- `plot.qtlcross`, which can be called using `plot(cross)`, calls `plot.geno`, `plot.map` and also plots histograms of each phenotype.

R package example: R/qtl (continued)

Intermediate calculations

There are a number of intermediate calculations that I want to do on a `qtlcross` object, and which I want to store with the original data.

For example, in the analysis of a cross to identify underlying QTLs, I need Pr(QTL genotype | marker data) at 1 cM steps along each chromosome.

The solution I came up with was to have many functions take a `qtlcross` object (a list) as input, and spit out the same object with another component added to the list. The name of the new component specifies what it contains.

For example, the function `calc.genoprob` calculates conditional genotype probabilities given observed genotype data. It takes a `qtlcross` object as input, and spits out the same object with an added component `genoprob`, which is itself a list composed of an array of genotype probabilities and a genetic map specifying the locations at which these probabilities were calculated.

The functions `calc.error.lod` and `est.rf` act the same way.

R/qtl: Summary

- Thinking carefully about how to store the data
- Try to write functions to make it easy to read the data from a file (or files).
- Make use of some of the object-oriented features of the S language.
- Provide functions for dealing with routine manipulation of data (summaries and specialized plotting functions).
- Start with prototypes of functions in R; move heavy computation to C.
- Write useful documentation.
- Learn from the R packages that others have written.