Starting points for future work

L Gatto and S J Eglen

October 30, 2012

Outline

The shell

The unix `shell` is an extremely powerful environment that features many extremely handy tools, that do simple things, and that can be piped (|) together.

- `wc`, `grep`, `cut`
- `tr`, `sed`, `awk`

`shell` can also be used for scripting.

grep, sed, awk

- All exploit regular expressions. See ItDT book (later).
- grep: find matching lines
- sed: stream-editor. Incredibly handy for one-liners:

http://sed.sourceforge.net/sed1line.txt

```
sed 's/foo/bar/g'                  # replaces ALL instances in line
# print section of file between two regular expressions
sed -n '/Iowa/,/Montana/p'                  # case sensitive
```

- awk: flexible pattern matching/ processing of text files.

http://www.pement.org/awk/awk1line.txt

```
# print the sums of the fields of every line
awk '{s=0; for (i=1; i<=NF; i++) s=s+$i; print s}'
```

diff: where do my files differ?

version1.dat

```
0.701 -0.764 -0.226 0.796 -0.337
0.249 -1.51 0.876 2.25 -0.879
-0.523 -1.29 0.354 -0.378 -1.39
0.565 1.31 -0.237 -0.844 0.28
2 -0.128 -0.841 1.31 -0.651
-0.565 0.81 -0.116 0.582 -0.0334
1.03 -0.75 1.7 -0.829 2.3
0.797 -0.988 0.667 -0.492 -0.78
0.94 -0.0931 -0.22 -1.29 -1.21
-0.456 -0.0231 0.603 1.43 0.734
0.598 -0.113 0.852 -1.58 -0.165
0.126 -0.0806 0.951 0.49 0.328
```

version2.dat

```
0.701 -0.764 -0.226 0.796 -0.337
0.249 -1.51 0.876 2.25 -0.879
-0.523 -1.29 0.354 -0.378 -1.39
0.565 1.31 -0.235 -0.844 0.28
2 -0.128 -0.841 1.31 -0.651
-0.565 0.81 -0.116 0.582 -0.0334
1.03 -0.75 1.7 -0.829 2.3
0.797 -0.988 0.667 -0.492 -0.78
0.94 -0.0932 -0.22 -1.29 -1.21
-0.456 -0.0231 0.603 1.43 0.734
0.598 -0.113 0.852 -1.58 -0.165
0.126 -0.0806 0.951 0.49 0.328
```

diff and patch

- *diff* shows the differences between version1 and version 2.

```
diff nextsteps/version1.dat  nextsteps/version2.dat
```

- *patch*: new file = old file + diff
- *patches* are efficient ways of sending updates. Useful for syncing and version control.

```
diff version1.dat version2.dat > p
patch version1.dat p
diff version1.dat version2.dat
```

Perl: Practical Extraction and Report Language

- Most unix tools (used to be) limited by length of lines. Perl removed those restrictions, combining features of awk, sh and C.
- 'duct tape' programming language.
- Useful in computational biology. See http://www.bioperl.org
- Excellent Ensembl API, http://www.ensembl.org/info/data/api.html
- G. Valiente. Combinatorial Pattern Matching Algorithms in Computational Biology using Perl and R . Taylor & Francis/CRC Press (2009).
- Verdict: yucky, but probably [essential | good to now].
- Bidirectional R /Perl interfaces http://www.omegahat.org/RSPerl/

R can also regexp

- `grep`, `sub`, `gsub`, `strsplit`, `nchar`, `substr`, ...
- also stringr package

and for sequence data storing and manipulation

- Biostrings package

Python

- Modern programming language; less compact than perl:

```
while (<>) {                    | import sys
    print if /perl/i;           | for line in sys.stdin.readlines():
}                               |     if line.lower().find("perl") > -1:
                                |         print line,
```

http://www.sabren.net/articles/againstperl.php3

- Clean syntax
- Properly object-oriented.
- Not as much support in computational biology (yet). See http://www.biopython.org
- Verdict: More general programming language than `R` ; lacking (perhaps?) in core numerics and graphics – see NumPy and RPy(2).
- Bidirectional `R` /Python interface http://www.omegahat.org/RSPython/

Outline

C

- Low-level programming language
- Very fast, but takes a long time to write code.
- You have to worry about memory allocation yourself.
- All variables have predefined type.
- Critical for numerical-intensive work. (FORTRAN less-popular.)

C from R

- R has build-in C interfaces
  - Better know how to program in C.
  - Documentation is not always easy to follow: R-Ext, R Internals as well as R and other package's code.
- `.C` Arguments and return values must be *primitive* (vectors of doubles or integers)
- `.Call` Accepts R data structures as arguments and return values (`SEXP` and friends) (no type checking is done though).
- Memory management: memory allocated for R objects is garbage collected. Thus R objects in C code, you must be explicitely `PROTECT`ed to avoid being `gc()`ed, and subsequently `UNPROTECT`ed.

Using .Call

```
#include <R.h>
#include <Rdefines.h>

SEXP gccount(SEXP inseq) {
  int i, l;
  SEXP ans, dnaseq;
  PROTECT(dnaseq = STRING_ELT(inseq, 0));
  l = LENGTH(dnaseq);
  printf("length %d\n",l);
  PROTECT(ans = NEW_NUMERIC(4));

  for (i = 0; i < 4; i++)
    REAL(ans)[i] = 0;

  for (i = 0; i < l; i++) {
    char p = CHAR(dnaseq)[i];
    if (p=='A')
      REAL(ans)[0]++;
    else if (p=='C')
      REAL(ans)[1]++;
    else if (p=='G')
      REAL(ans)[2]++;
    else if (p=='T')
      REAL(ans)[3]++;
    else
      error("Wrong alphabet");
  }
  UNPROTECT(2);
  return(ans);
}
```

Using our C code

- Create a shared library: `R CMD SHLIB gccount.c`
- Load the shared object: `dyn.load("gccount.so")`
- Create an R function that uses it: `gccount <- function(inseq) .Call("gccount",inseq)`
- Use the C code: `gccount("GACAGCATCA")`

```
s <- "GACTACGA"
gccount
gccount(s)
table(strsplit(s, ""))
system.time(replicate(10000, gccount(s)))
system.time(replicate(10000, table(strsplit(s, ""))))
```

Rcpp for C++

- Rcpp is a great package for writing both C and C++ code:
- It comes with loads of documentation and examples.
- No need to worry about garbage collection.
- All basic R types are implemented as C++ classes.
- Easy to interface C++ classes (via modules)
- With package inline code can be easily compiled in R .

```
library(Rcpp)
library(inline)
cppCode <- '
    Rcpp::NumericVector cx(x);
    Rcpp::NumericVector ret(1);
    ret[0] = cx[0] * cx[0];
    return(ret);
    '
squareOne <- cxxfunction(signature(x="numeric"),
                         plugin="Rcpp", body=cppCode)
squareOne(10)
```

Outline

Syncing your files

- How do you keep two directories in synchrony, e.g. your home directory on laptop and desktop?
- `sftp`, `ssh`, `rsync`
- `Unison` gets Stephen's vote since 2003 – http://www.damtp.cam.ac.uk/internal/computing/unison/
- Modern services like Dropbox are useful and build upon these unix tools.

Version control / revision control system (RCS)

- How to keep backup copies over time?
- Just copy files, e.g. *mycode.jan1.R*, *mycode.jan2.R*, ...
- Leads to many large copies, with no trace of what you did over time.
- more principled way is to use version control: every time you make significant changes, you *commit* a new version with a succint log file saying what you changed.
- RCS: going since 1982... old and simple but stable. Typically single-user.

`http://www.cl.cam.ac.uk/~mgk25/rcsintro.html`

- More modern approaches: *cvs*, *svn*, *git*, ...

For packages, analysis projects, papers and slides

- Github, google code, bitbucket, ...
- R-forge: svn and build system

Outline

Handling large data files.

- Computational Biology requires access to large data files.
- Reading them all into memory is difficult, when files are very large (> 1 Gb).
- Some approaches:
  1. Compress files.
  2. Selectively use scan or connections.
  3. Use a database.

1. Compress files.

- This produces typically x2 compression:

```
Rscript -e 'write(rnorm(99999), file="largefile.dat")'
ls -lh largefile.dat
gzip largefile.dat
ls -lh largefile.dat.gz
gunzip largefile.dat
```

- R can read in compressed files natively.

```
x <- scan('largefile.dat.gz')
```

- Other compression options also recognised: xz, bzip2

2. Scan and Connections.

- scan() is very flexible; e.g. read just 2nd column:

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
     quote = if(identical(sep, "\n")) "" else "'\"", dec = ".",
     skip = 0, nlines = 0, na.strings = "NA",
     flush = FALSE, fill = FALSE, strip.white = FALSE,
     quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE,
     comment.char = "", allowEscapes = FALSE,
     fileEncoding = "", encoding = "unknown")

x <- scan(file, what=list(NULL,"",NULL), skip=2, sep='\t')
```

- connections allow you to maintain state between accesses to a file.

```
con <- file("version1.dat", "r")
while (length(dat <- scan(con,n = 5,quiet = TRUE)) > 0) {
  print(mean(dat))
}
close(con)
```

3. Relational databases

- Relational database: data stored in tables, very similar in nature to `R` 's data.frames.
- Databases allow for multiple-accesses, locks for restricted changes, very scalable.
- Many databases available: Oracle, Postgres, Access, MySQL.
- SQL – Structured Query Language: language to interrogate databses.

What is SQLite?

- Most databases run on remote server; SQLite is embedded into your program.
- Embedding the database simplifies setup of server, but means your databases are not shared in the same way that others are. (You have to share the .sql files.)
- Incredibly small (1/4 Mb) and useful. Widely used (e.g. mac, iOS, Firefox, Android). Not as fast as e.g. Oracle.
- You compile your SQLite within your program.
- All handled with you by `R` , care of *RSQLite* package. (e.g. Bioconductor uses it for data files.)

Using databases in R , a simple session (Gentleman, p239)

- package *DBI* interfaces to all database platforms.

```
library(RSQLite)
m = dbDriver("SQLite")

## Create a new database from an R data frame.
con = dbConnect(m, dbname = "arrest.db")
data(USArrests)
dbWriteTable(con, "USArrests", USArrests, overwrite=TRUE)
dbListTables(con)

## Later, query the database.
rs = dbSendQuery(con, "select * from USArrests")
d1 = fetch(rs, n=5)      ## get first five
print(d1)
d1 = fetch(rs, n=-1)
dbDisconnect(con)
```

Other uses for sqlite

- `sqldf` Performs SQL selects on `R` data frames.
- supports SQLite backend database (by default), the H2 java db and PostgreSQL and MySQL.
- avoid read.csv entirely http://code.google.com/p/sqldf/

  "See ?read.csv.sql in sqldf. It uses RSQLite and SQLite to read the file into an sqlite database (which it sets up for you) completely bypassing R and from there grabs it into R removing the database it created at the end." (G. Grothendieck, r-help mailing list).

- Good book: `^((HT|X)M|SQ)L|R$`
  Introduction to Data Technologies.

`http://www.stat.auckland.ac.nz/~paul/ItDT/`

ff: back to the future?

- *ff* package stores objects on disk, but looks like they are in memory.
- "back to the future": S used to store objects in disk.
- Sorting a single column of 81e6 entries. Time-taken in seconds.

Oct 2010 results from.
http://tolstoy.newcastle.edu.au/R/packages/10/0697.html

	ruinteger	rinteger	rusingle	rsingle	rudouble	rdouble	rfactor
ram	5.58	3.23	NA	NA	NA	NA	0.49
ff	10.70	8.54	51.35	28.98	70.20	44.13	7.91
R	OOM	OOM	OOM	OOM	OOM	OOM	OOM
SAS	61.45	44.94	NA	NA	63.14	46.56	NA

(ram=in-memory, optimized for speed, not ram; ff=on disk).

Other

- The `bigmemory` package by Kane and Emerson permits storing large objects such as matrices in memory (as well as via files) and uses external pointer objects to refer to them.
- netCDF data files: `ncdf` and `RNetCDF` packages.
- hdf5 format: `rhdf5` package
- `XML` package to parse xml data
- ...

Outline

Introduction

- Applicable when repeating *independent* computations a certain number of times; results just need to be combined after parallel executions are done.
- A cluster of nodes: generate multiple workers listening to the master; these workers are new processes that can run on the current machine or a similar one with an identical R installation. Should work on all R plateforms (as in package snow).
- The R process is *forked* to create new R ~processes by taking a complete copy of the masters process, including workspace (pioneered by package multicore). Does not work on Windows.
- Grid computing.

Packages

- Package parallel, first included in R 2.14.0 builds on CRAN packages multicore and snow.

`mclapply(X, FUN, ...)` (adapted from multicore).
`parLapply(cl, X, FUN, ...)` (adapted from snow).

- Package foreach, introducing a new looping construct supporting parallel execution. Natural choice to parallelise a `for` loop.

```
library(doMC)
library(foreach)
registerDoMC(2)
foreach(i = ll) %dopar% f(i)
foreach(i = ll) %do% f(i) ## serial version
library(plyr)
llply(ll, f, .parallel=TRUE)
```

High performance computing

- Find information about managing and chunking big data:
  - High performance computing CRAN task view
  - http://cran.r-project.org/web/views/HighPerformanceComputing.html

Outline

Pass by . . .

- **value** is the default in `R`
- **reference** using S4 ReferenceClasses (OO)
- can emulate pass by ref using an `environment`

```
e <- new.env()
e$x <- 1
f <- function(myenv) myenv$x <- 2
f(e)
e$x
```

Profiling

```
m <- matrix(rnorm(1e6), ncol=100)
Rprof("rprof")
res <- apply(m,1,mean,trim=.3)
Rprof(NULL)
summaryRprof("rprof")
```

Benchmarking

```
m <- matrix(rnorm(1e6), ncol=100)
f1 <- function(x, t = 0.3) {
  xx <- 0
  for (i in 1:nrow(x)) {
    xx <- c(xx, sum(m[i, ]))
  }
  mean(xx, trim = t)
}
f2 <- function(x, t = 0.3) mean(rowSums(x), trim = t)

library(rbenchmark)
benchmark(f1(m), f2(m),
          columns=c("test", "replications",
            "elapsed", "relative"),
          order = "relative", replications = 10)
```

Outline

Lotka–Volterra equations

Describe simple models of populations dynamics of species competing for some common resource. When two species are not interacting, their population evolve according to the logistic equations and the rate of reproduction is proportional to both the existing population and the amount of available resources

$$\frac{\partial x}{\partial t} = r_1 x \left(1 - \frac{x}{k_1}\right)$$
$$\frac{\partial y}{\partial t} = r_2 y \left(1 - \frac{y}{k_2}\right)$$

where the constant r_i defines the growth rate and k_i is the carrying capacity of the environment.

Competitive Lotka–Volterra equations

When competing for the same resource, the animals have a negative influence on their competitors growth.

$$\frac{\partial x}{\partial t} = r_1 x\ (1 - \frac{x}{k_1}) - axy$$
$$\frac{\partial y}{\partial t} = r_2 y\ (1 - \frac{y}{k_2}) - bxy$$

Rabbits vs sheep (Strogatz, p155)

Here is an example with $r_1 = 3$, $k_1 = 3$, $a = 2$, $r_2 = 2$, $k_2 = 2$, $b = 1$, which simplifies to

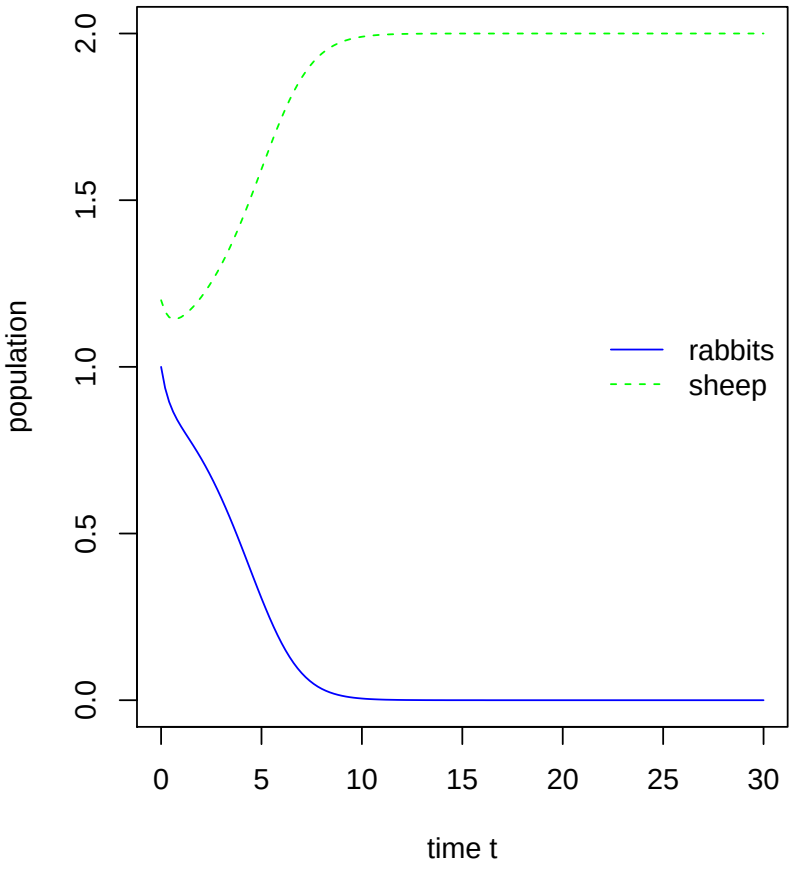
$$\frac{\partial r}{\partial t} = r(3 - r - 2s)$$
$$\frac{\partial s}{\partial t} = s(2 - r - s)$$

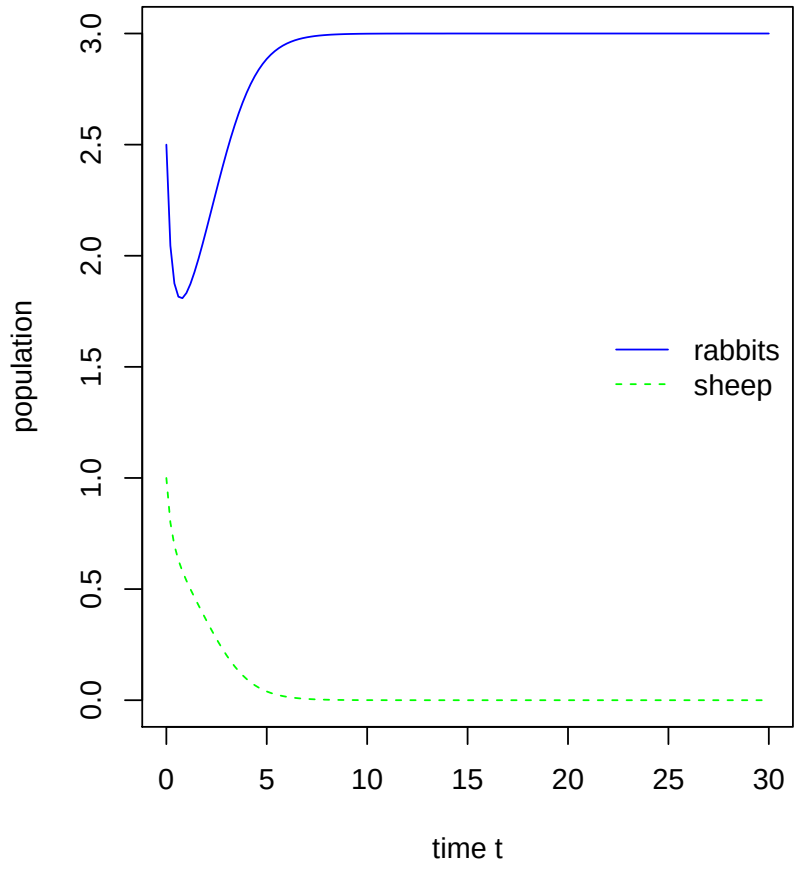
Computing a trajectory over time

i.e. use numerical integration, with $r_0 = 1$ and $s_0 = 1.2$

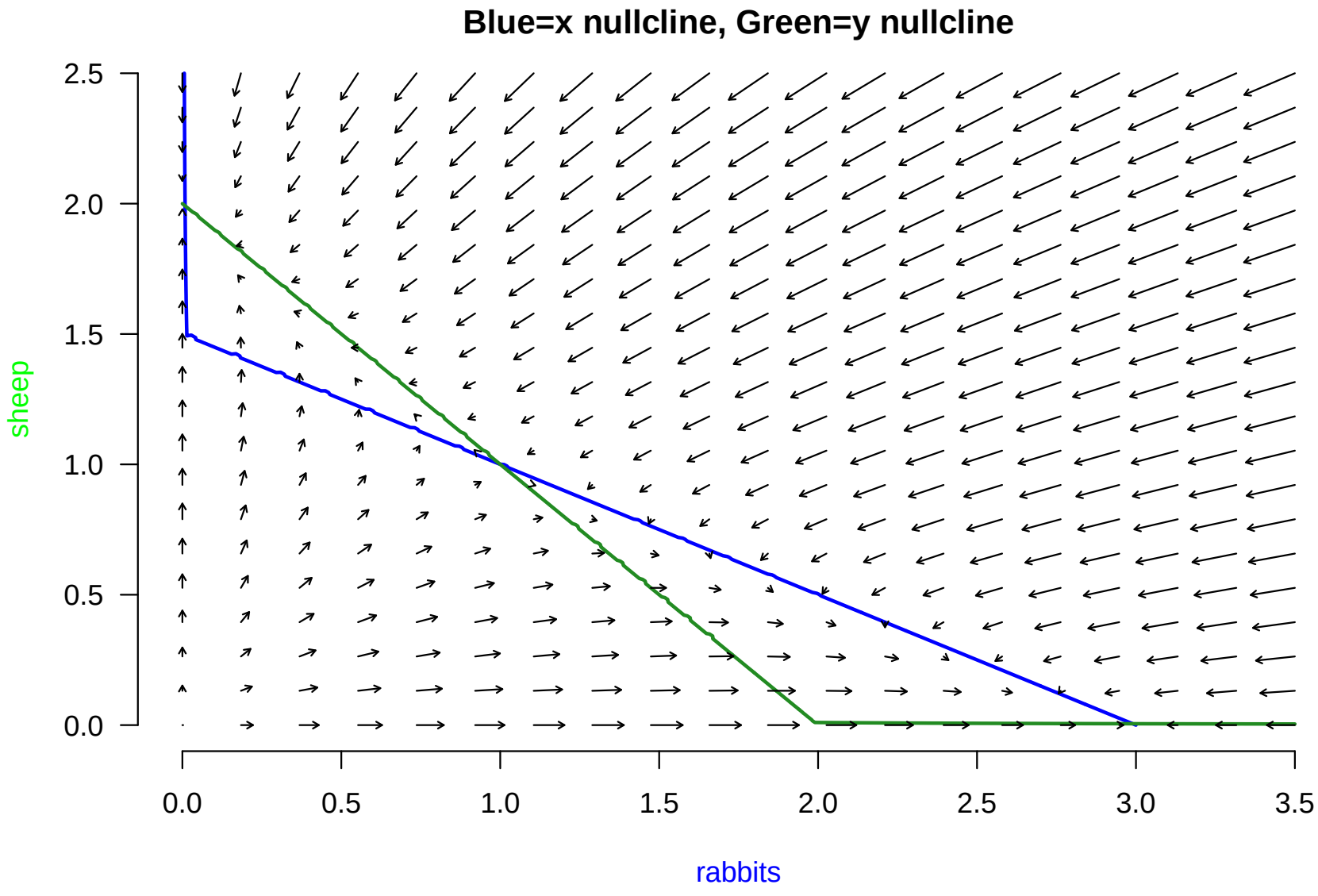
```
library(deSolve)
Sheep <- function(t, y, parms) {
  r=y[1]; s=y[2]
  drdt = r * (3 - r - (2*s))
  dsdt = s * (2 - r - s)
  list(c(drdt, dsdt))
}

x0 <- c(1, 1.2)
times <- seq(0, 30, by=0.2)
parms <- 0
out <- rk4(x0, times, Sheep, parms)
head(out)
```

Plotting population growth





Phase plane analysis



Starting points

- `deSolve` package
- phase planes and nullclines (`DMBpplane.r` from DMB site, modified from Daniel Kaplan)
- `integrate()` – quadrature
- `D()` – symbolic differentiation
- `optimize()` (1d) and `optim()` (n-d)
- Steven Strogatz. Nonlinear dynamics and chaos.
- NR: William Press et al. Numerical Recipes in C/C++
- More slides about DE and phase plane – `de.pdf`

Outline

Conclusions

- Looking for packages
  - CRAN Task Views http://cran.r-project.org/web/views/
  - Bioconductor biocViews http://bioconductor.org/packages/release/BiocViews.html
- Reproducibility is crucial
- Have several tools at hand
  - editor, programming languages, shell, ...
- Practice to keep learning
- Have fun! ☺