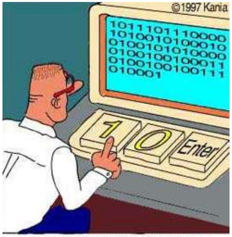
Break out group II: extending R with C

Caspar Hallmann   Marco D. Visser   Sean McMahon

October 7, 2015

Goals



Real programmers code in binary.

1. Extending R with C
2. Read and write (simple) C
3. Call C from R

II: Extending R with C

Resources
http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004140

1. .C interface (section 5)
2. .Call interface (section 6.2)
3. SEXP (section 6.2.2)
4. PROTECT, UNPROTECT and gc() (sect. 6.2.2)
5. random number generation and other functions (sect. 6.2.3)

Getting prepared

What we need:

- Windows users need Rtools
  http://cran.r-project.org/bin/windows/Rtools/
- Linux users:

  ```
   sudo apt-get update
  sudo apt-get install build-essential r-base-dev
  ```
- Mac users:... not sure, but see
  Xcode application and *Command Line Tools package*

Getting prepared

Windows:
PATH variables need to be set

- ...\Rtools\bin;
- ...\Rtools\gcc-4.6.3\bin;
- ...\R-3.1.0\bin;

where ... refers to the directories on your machine.
Install *Rtools* as close as possible to the root, e.g. C:\Rtools

R's interface to C

Function	Use today
.C()	Yes!
.Call()	Yep!
.Interal	Nope
.External	No
PACKAGES	
inline	superficially
Rcpp	Nope (but really cool)
RcppArmadillo	(Even more cool)

Calling C workflow

1. Write C code to a file (e.g myfunction.c)
2. Compile c file
3. load so/dll into R
4. construct a R-wrapper (optional)
5. run program

A simple R program

Suppose you have R's SimpleR function for $a_i = \frac{N}{N+1}$

```
SimpleR<-function(N){
answer<-numeric(N)
for (i in 1:N) answer[i]<-N/(1+N)
return(answer)
}
```

A simple C program

The equivalent in C is

```
void SimpleC(int *nc, double *dnc, double *answerc) {
int n = nc[0];
double dn = dnc[0];
int i;
for (i=0; i<n; i++){
answerc[i] = (dn/(1+dn));
}
}
```

C code explained

What was in here?

- headers:

  ```
  #include <stuff.h>
  ```
- commenting: "/* stuff for human eyes only */"
- declarations: **int*, **double* ,**char*, etc
- always end with semicolons ";"

C code explained

- Everything needs to be declared

Rmode	C equivalent	example
integer	int	0,1,2,...
numeric	double	1.23, -45.1,...
character	char	"a", "h"

- Use 'pointers' (e.g. int *nc, double *dnc) to declare where the object is located in the memory (by R)
- Access content from pointers

```
int n = nc[0];
double dn = dnc[0];
```

- Need to provide both "input" (nc,dnc) as well as "output" (answerc)

C code explained

`for` loops

```
int i;
for (i=0; i<n; i++){
answerc[i] = dn/(1+dn)
}
```

- Loops start with position 0, unlike in R.
- You can increment a variable by stating "++" behind it, e.g. i++.
- setting values in variables with "[...]", as in R.

compiling C code

Unlike R, you need to compile first!

```
R CMD SHLIB SimpleC.so SimpleC.c
```

OR in windows

```
R CMD SHLIB SimpleC.dll SimpleC.c
```

using

```
system("R CMD SHLIB SimpleC.dll SimpleC.c")
```

in R.

calling C programn in R using .C

We can then use:

```
.C("SimpleC",nc = as.integer(N),
dnc = as.double(N),answer = as.double(rep(0,N)))
```

Or in a function like

```
SimpleCWrapper <- function(N){
out <- .C("SimpleC",
nc = as.integer(N),
dnc = as.double(N),
answer = as.double(rep(0,N))
)
return(out$answer)
}
```

Run the whole lot as:

```
sink("Simple.c")
## Send C code to this file with "cat"

cat("
void SimpleC(int *nc, double *dnc, double *answerc){

int n = nc[0];
double dn = dnc[0];
int i;
for (i=0; i<n; i++){
answerc[i] = (dn/(1+dn));
}

}
")
sink(NULL)
```

Run with

```r
system("R CMD SHLIB -o Simple.dll Simple.c")
dyn.load("Simple.dll")
SimpleCWrapper <- function(N){
out <- .C("SimpleC",
nc = as.integer(N),
dnc = as.double(N),
answer = as.double(rep(0,N))

)
return(out$answer)
}
```

Programmers While Coding
It Doesn't Work....... Why?
It Work....... Why?
©IndianFunnyPicture.com

Benchmark

```
system.time(Ranswers <- SimpleR(1e+06))

##    user  system elapsed
##    5.61    0.00    5.62

system.time(Canswers <- SimpleCWrapper(1e+06))

##    user  system elapsed
##       0       0       0
```

- http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pc
- try for yourself: .C interface (section 5) (+- 10 mins)

the .Call interface

- The .C interface is too simplistic
- Only accepts vectors of integers or doubles
- Only produces vectors of integers or doubles
- We want to use R's objects such as `list` and `matrix`

SOLUTION:

- We can use them with .Call instead of .C function
- Drawback: C code more complicated

SEXP objects

R objects are called SEXP in C

SEXP type	R mode	accessor
REALSXP	numeric vector	REAL(x)
INTSXP	integer vector	INTEGER(x)
LGLSXP	logical vector	LOGICAL(x)
STRSXP	character vector	STRING_ELT(x,index)
VECSXP	list	VECTOR_ELT(x,index)

We can declare SEXP types eg as:

```
allocVector(x, index)
allocMatrix
```

Example Lotka model

$$\frac{dx_1}{dt} = r_1 x_1 \left(1 - \left(\frac{x_1 + a_{12} x_2}{K}\right)\right) \tag{1}$$

$$\frac{dx_2}{dt} = r_2 x_2 \left(1 - \left(\frac{x_1 + a_{21} x_1}{K}\right)\right) \tag{2}$$

Example Lotka model (in R)

```
MatrixLotka <- function(T = N, pop = population, am = alpha
as = alpha.sd, rm = r.means, rs = r.sd, K = CarryingCapacit
r1 <- rnorm(T, rm[1], rs[1])
a1 <- rnorm(T, am[1], as[1])
r2 <- rnorm(T, rm[2], rs[2])
a2 <- rnorm(T, am[2], as[2])
for (i in 2:T) {
pop[i, 1] <- pop[i - 1, 1] * r1[i] * (1 - (pop[i - 1, 1] +
(a2[i] * pop[i - 1, 2]))/K)
pop[i, 2] <- pop[i - 1, 2] * r2[i] * (1 - (pop[i - 1, 2] +
(a1[i] * pop[i - 1, 1]))/K)
}
return(pop)
}
```

Example Lotka model (in C)

```
/*Stochastic Lotka example in C*/
#include <R.h>
#include <Rmath.h>
#include <Rinternals.h>
SEXP lotkac(SEXP time, SEXP alphamean, SEXP alphasd,
SEXP rmean, SEXP rsd, SEXP K)
{
int n=length(time);
int i;
/* create new R objects in C. */
SEXP P1, P2, result;
/* protect and allocate R objects in C. */
PROTECT(P1=allocVector(REALSXP,n));
PROTECT(P2=allocVector(REALSXP,n));
/* set list that returns results to R*/
PROTECT(result = allocVector(VECSXP, 2));
/* assign pointers to R objects */
double *p1=REAL(P1);
double *p2=REAL(P2);
double *am=REAL(alphamean);
double *as=REAL(alphasd);
double *rm=REAL(rmean);
double *rs=REAL(rsd);
double *k=REAL(K);
double R, A;
/* set initial population sizes*/
p2[0]=1.0;
p1[0]=1.0;
for(i=1; i<n; i++) {
/* actual simulation, in which we update the random number generator*/
GetRNGstate();
```

PROTECT

- R's internal GARBAGE COLLECTION
- Need to PROTECT objects from being wiped away
- e.g

  ```
  PROTECT(P1=allocVector(REALSXP,n));
  ```
- Need to UNPROTECT them at the end
- e.g

  ```
  UNPROTECT(1);
  ```

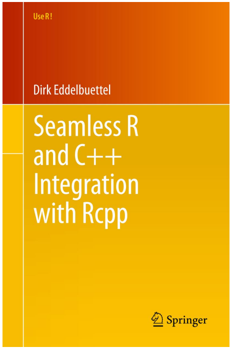
Example .Call

```
system("R CMD SHLIB -o lotka.dll lotka.c")
dyn.load("lotka.so")
LotkaCWrapper<-function(time=1:N,alphamean=alpha.means,
alphasd=alpha.sd,rmean=r.means,rsd=r.sd, K=CarryingCapacity)
{
out <- .Call("lotkac",
time=as.double(time),
alphamean=as.double(alphamean),
alphasd=as.double(alphasd),
rmean=as.double(rmean),
rsd=as.double(rsd),
K=as.double(K))
return(out)
}
## reset seeds for simulations
set.seed(1)
#Set N to 10000 to compare with previous results
N=10000
#time the function
LotkaCT<-system.time(
ResultsC <- LotkaCWrapper()
)[3]
print(LotkaCT)
```

Break out group II: extending R with C

- http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004140
- try for yourself: .C interface (section 6) (+- 10 mins)
- OR try own code

More advanced interface



- Rccp
- Rccp ... addons
- see https://cran.r-project.org/package=Rcpp

THE END