

Topics

- we are going to see:
 - how to pass a R function to a C function and use it within the C function

Pass A R Function I

- suppose we have a R function `targetDensity` which takes and returns a numeric value e.g.

```
targetDensity <- function(x) { dnorm(x, mean = 100, sd = 10, log = T) }
```

- to be able to use this R function from within your C code you need to do the following in R
 - create a temporary environment
 - attach the function to the temporary environment
 - pass to `.Call` or `.External` both the function and the environment

Pass A R Function II

- here is how you do it:

```
doStuff <-  
  function (nIters, timeInSecs, propBurnIn, targetDensity)  
  {  
    # check the function  
    checkVal <- 1.0  
    val <- targetDensity(checkVal)  
    if(!is.numeric(val) || length(val) != 1)  
      stop("Need a numeric result")  
  
    tmp.env <- new.env( )  
    environment(targetDensity) <- tmp.env  
    args <- list("n_iters" = as.integer(nIters),  
                "time_in_secs" = as.numeric(timeInSecs),  
                "prop_burn_in" = as.numeric(propBurnIn),  
                "target_density" = targetDensity,  
                "env" = tmp.env)  
    .Call("do_stuff2",  
          args)  
  }
```

Use A R Function In C |

- extract the function and the environment from the argument list (called list below):

```
SEXP target_density, env;  
target_density = getListElement(list, "target_density");  
env = getListElement(list, "env");  
if (!isFunction(target_density))  
    error("'targetDensity' should be a function");  
if (!isEnvironment(env))  
    error("'env' should be an environment");
```

- declare and allocate a R function call variable and an function “argument” variable:

```
SEXP R_fcall, xx;  
PROTECT(R_fcall = lang2(target_density, R_NilValue));  
++nProtected;  
PROTECT(xx = allocVector(REALSXP, 1));  
++nProtected;
```

- note the lang2 function creates a “cons-cell” or a “dotted-pair” of two SEXPs: (functionName . argument), initialize argument to R_NilValue

Use A R Function In C II

- suppose our goal is:

```
/* Lets say in our Monte Carlo iterations we want to do the
 * one of simplest thing in the world: create an vector of
 * n_iters many Normal(0, 1) random numbers and return it. We
 * also return the corresponding target_density values
 * evaluated at those generated samples.
 */
```

- toward this goal, say we have allocated two vectors like so:

```
SEXP samplesVec, densityVec;
PROTECT(samplesVec = allocVector(REALSXP, n_iters));
++nProtected;
PROTECT(densityVec = allocVector(REALSXP, n_iters));
++nProtected;
```

Use A R Function In C III

- now we do our work as follows:

```
GetRNGstate();
for (ii = 0; ii < n_iters; ++ii) {
    tmp = rnorm(0, 1.0);
    REAL(samplesVec)[ii] = tmp;
    REAL(xx)[0] = tmp;
    SETCADR(R_fcall, xx);
    REAL(densityVec)[ii] = REAL(eval(R_fcall, env))[0];
}
PutRNGstate();
```

- note `R_fcall` is a “cons-cell” or a “dotted-pair” and we are setting the `cdr` of it as the function argument
- note the usage of the `eval` function, it takes a function call object and an environment as arguments

Use A R Function In C IV

- now we return the two vectors `samplesVec` and `densityVec` packed in a list:

```
SEXP names, samplesVec, densityVec, retList;
PROTECT(names = allocVector(STRSXP, 2));
++nProtected;
SET_STRING_ELT(names, 0, mkChar("samples"));
SET_STRING_ELT(names, 1, mkChar("log_target_density"));

PROTECT(retList = allocVector(VECSXP, 2));
++nProtected;
SET_VECTOR_ELT(retList, 0, samplesVec);
SET_VECTOR_ELT(retList, 1, densityVec);

setAttrib(retList, R_NamesSymbol, names);

UNPROTECT(nProtected);
return retList;
```

Pass A R Function I

- what do you do if your function takes more than one argument?
- for the time being the only solution I know of is to pack all the arguments in a list and pass the list

- suppose we have the following R function:

```
### listOfArgs should be "list(x, y)"  
proposal <-  
  function(listOfArgs)  
  {  
    dnorm(listOfArgs[[1]], mean = listOfArgs[[2]], sd = 2.4, log = T)  
  }
```

- note two of this function's "potential" arguments have been packed into a list

Pass A R Function II

- lets look at the R side first:

```
doStuff <-  
  function (nIters, timeInSecs, propBurnIn, proposalDensity)  
  {  
    # check the function  
    checkVal <- list(0.0, 0.0)  
    val <- proposalDensity(checkVal)  
    if(!is.numeric(val) || length(val) != 1)  
      stop("Need a numeric result")  
  
    tmp.env <- new.env( )  
    environment(proposalDensity) <- tmp.env  
    args <- list("n_iters" = as.integer(nIters),  
                "time_in_secs" = as.numeric(timeInSecs),  
                "prop_burn_in" = as.numeric(propBurnIn),  
                "proposal_density" = proposalDensity,  
                "env" = tmp.env)  
    .Call("do_stuff",  
          args)  
  }
```

Use A R Function In C I

- as before process the function passed and the environment:

```
SEXP proposal_density, env;  
proposal_density = getListElement(list, "proposal_density");  
env = getListElement(list, "env");  
if (!isFunction(proposal_density))  
    error("'proposalDensity' should be a function");  
if (!isEnvironment(env))  
    error("'env' should be an environment");
```

Use A R Function In C II

- now declare and allocate a R function call variable and an function “argument” list variable:

```
SEXP R_fcall, xx, yy, ll;  
PROTECT(R_fcall = lang2(proposal_density, R_NilValue));  
++nProtected;  
PROTECT(xx = allocVector(REALSXP, 1));  
++nProtected;  
PROTECT(yy = allocVector(REALSXP, 1));  
++nProtected;  
PROTECT(ll = allocVector(VECSXP, 2));  
++nProtected;  
SET_VECTOR_ELT(ll, 0, xx);  
SET_VECTOR_ELT(ll, 1, yy);
```

Use A R Function In C III

- suppose our goal is:

```
/* Lets say in our Monte Carlo iterations we want to do the
 * one of simplest thing in the world: create an vector of
 * n_iters many Normal(0, 1) random numbers and return it. We
 * also return the corresponding proposal_density values
 * evaluated at the consecutive pairs of those generated
 * samples.
 */
```

- for this, say we have allocated two vectors like so:

```
SEXP samplesVec, densityVec;
PROTECT(samplesVec = allocVector(REALSXP, n_iters));
++nProtected;
PROTECT(densityVec = allocVector(REALSXP, n_iters - 1));
++nProtected;
```

Use A R Function In C IV

- now we do our work as follows:

```
GetRNGstate();
for (ii = 0; ii < n_iters; ++ii) {
    REAL(samplesVec)[ii] = rnorm(0, 1.0);
}
PutRNGstate();
for (ii = 0; ii < n_iters - 1; ++ii) {
    REAL(xx)[0] = REAL(samplesVec)[ii];
    REAL(yy)[0] = REAL(samplesVec)[ii + 1];
    SETCADR(R_fcall, ll);
    REAL(densityVec)[ii] = REAL(eval(R_fcall, env))[0];
}
```

Use A R Function In C V

- as before, we return the two vectors `samplesVec` and `densityVec` packed in a list:

```
SEXP names, samplesVec, densityVec, retList;
PROTECT(names = allocVector(STRSXP, 2));
++nProtected;
SET_STRING_ELT(names, 0, mkChar("samples"));
SET_STRING_ELT(names, 1, mkChar("log_proposal_density"));

PROTECT(retList = allocVector(VECSXP, 2));
++nProtected;
SET_VECTOR_ELT(retList, 0, samplesVec);
SET_VECTOR_ELT(retList, 1, densityVec);

setAttrib(retList, R_NamesSymbol, names);

UNPROTECT(nProtected);
return retList;
}
```

Code Files

```
prog9.c  
prog9.R  
prog10.c  
prog10.R
```