Using Rcpp

Hao Ye
R Users Group
December 5, 2013

What is Rcpp?

- Rcpp is a package that:
  - integrates C++ code with R
    - can be done in real time, or by making custom packages
  - provides a consistent map between C++ classes and R objects
  - simplifies new package development
    - works seamlessly with RStudio

Getting started

- C++ compiler
  - Windows: Rtools (simplest option)
  - OS X: xcode command-line tools*
  - Linux: r-base-dev (Debian/Ubuntu)
- install Rcpp package (may need to install from source)

* if using xcode 5, R's configuration file needs to be updated (/Library/Frameworks/R.framework/Versions/3.0/Resource/etc/Makeconf):

```
CC = clang
CXX = clang++
```

Then, Rcpp should be installed from source:

```
> install.packages("Rcpp", type = "source")
```

Example 1

- write a function to compute Fibonacci numbers:
  - sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, …
  - $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$ (for $n > 1$)

```
R_fib <- function(n)
{
  if(n < 2) # stopping condition
    return(n)
  return(R_fib(n-1) + R_fib(n-2))
}
```

Example 1 (cont'd)

- cppFunction compiles a C++ function and makes it visible (aka usable) in R:

```
cppFunction("
int cpp_fib(int n)
{
  if(n < 2)
    return n;
  return cpp_fib(n-1) + cpp_fib(n-2);
}")
```

Demo

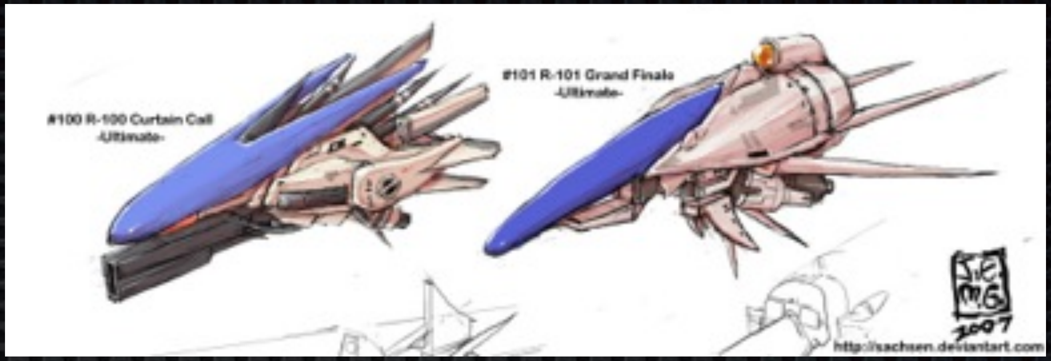
Notes

- C++ is a statically-typed language:
  - all variables have fixed types specified in advance
  - cannot reassign different types to the same variable name
- major syntax differences:
  - declarative lines end in a semicolon
  - only "=" used for assignment (no "<-" or "->")
  - function definition a little different

Transferring data to C++

- Rcpp: "provides a consistent map between C++ classes and R objects"
  - What does this mean?
  - C++ does not have the same variable types as R (e.g. data.frame, list, factor)
  - Rcpp add new types in C++ that are equivalent to some that exist in R

R types



R type	C++ type (from Rcpp)
integer	int
numeric	double
logical	bool
character	String
vector (of integer)	IntegerVector
vector (of numeric)	NumericVector
vector (of logical)	LogicalVector
vector (of character)	CharacterVector
matrix (of integer)	IntegerMatrix
…	…

Example 2

- write a function to multiply matrices
  - C = A B
    - $C_{i,j} = \sum_k (A_{i,k} * B_{k,j})$
    - built in operation in R:

```
a %*% b
```

  - sourceCpp compiles a C++ file and makes the "exported" functions usable in R:

```
sourceCpp("mmult.cpp")
```

Example 2 (cont’d)

- *mmult.cpp:*

```
#include <Rcpp.h>
using namespace Rcpp;

// [[Rcpp::export]]
NumericMatrix mmult(NumericMatrix a,
                    NumericMatrix b)
{
 /* function definition here */
}
```

Demo

Cautionary Notes

- in C++, vectors and matrices are 0-indexed
  - vectors can be indexed with [] or ()
  - matrices can only be indexed using ()
- Rcpp vectors and matrices cannot be resized at runtime
  - so use C++ native types (e.g. stl vector) and convert to R types on output
- lists and data frames are tricky to deal with
- NA values are not supported by most C++ types

Lists and Data Frames

- Hadley Wickham explains best:
- "Rcpp also provides classes List and DataFrame. These are more useful for output than input, because lists and data frames can contain arbitrary classes, and this does not fit well with C++'s desire to have the types of all inputs known in advance. If, however, the list is an S3 object with components of known types, you can extract the components and manually convert to their C++ equivalents with as."

NA values

- C++ types do not have universal support for NA, so Rcpp adds several constants (e.g. NA_INTEGER, NA_STRING, NA_REAL)
  - NA_INTEGER resolves to the -2147483648 (-2^31), and vice-versa:

```
> cppFunction("int n() {return -2147483648;}")
[1] NA
```

  - NA_LOGICAL resolves to true:

```
> cppFunction("bool n() {return NA_LOGICAL;}")
[1] TRUE
```

`

- Hadley Wickham's page on Rcpp:
  - http://adv-r.had.co.nz/Rcpp.html
- Rcpp vignettes (most useful are probably Rcpp-quickref, Rcpp-FAQ, Rcpp-introduction)
- Rcpp-devel mailing list:
  - http://lists.r-forge.r-project.org/mailman/listinfo/rcpp-devel
- Google

Advanced Stuff

Making Packages

- using RStudio:
  - new project -> New Directory -> R Package
    - select Package w/ Rcpp
  - "build and reload" will compile and install
- using R:
  - in R: `Rcpp.package.skeleton("new_package")`
  - in shell: `R CMD BUILD new_package`
  - in shell: `R CMD INSTALL new_package`

C++ classes

- classes: a major feature of C++ over C
- what is a class?
  - C++ object type
    - (usually) contains internal data
    - (usually) contains associated functions (called "methods")
    - can be derived from other classes or have derived classes
- very useful paradigm (object-oriented programming)

R classes

- R also has classes (S4 and S3 types)
  - example: model-fitting objects
    - `lm()` function help:

```
lm returns an object of class "lm" or for
multiple responses of class c("mlm", "lm").
```

  - lots of objects have similar methods (e.g. "summary", "print", "predict")
  - the R definitions of those functions dispatch to the class-specific methods

C++ classes in R

- We want to do the following (given C++ class):
  - make an object of that class
  - send it some data
  - do multiple operations, returning a result each time
- What we know how to do so far:
  - give C++ code some data and return an R object
  - to do the above, **every time** we want to do an operation, need to make a new C++ object
  - inefficient!

Rcpp modules

- can expose a class and its methods to R
  - make an R object which has a pointer to a C++ object
  - R functions which act as wrappers for class methods of that C++ object

```
> u <- new(Uniform, 0, 10)
> u$draw(10)
> u$range()
```

How to use Rcpp modules

- define the module (i.e. describe which objects, methods, fields, etc. to make visible to R)
- using a package:
  - make a custom Rcpp package
  - set package to load module
  - build, install, use custom package
- manually:
  - sourceCpp loads modules automatically

Defining an Rcpp module

- in C++ file:

```
RCPP_MODULE(module_name) {
  function("func_name", &func_name);

  class_<class_name>("class_name")
     .constructor<arguments>()
     .field( "variable", &class_name::variable )
     .method( "method", &class_name::method )
  ;
}
```

Adding Rcpp modules to a package

- in "DESCRIPTION": add RcppModules field
- in "zzz.R": add code to auto-load module:

```
loadModule("module_name", TRUE)
```

- (demo'ed by Rcpp.package.skeleton, with module argument set to TRUE):

```
Rcpp.package.skeleton("new_package",
                      module = TRUE)
```