Seamless R and C++ Integration: Rcpp and RInside

Dirk Eddelbuettel
Debian Project

Joint work with Romain François

Invited seminar presentation
Institute for Statistics and Mathematics
Wirtschaftsuniversität Wien
20 May 2010
Preliminaries

We assume a recent version of R so that `install.packages(c("Rcpp","RInside","inline"))` gets us current versions of the packages. All examples shown should work 'as is' on Linux, OS X and Windows, provided a complete R development environment. The R Installation and Administration manual is an excellent start if you need to address the preceding point. In particular, one must use the same compilers used to build R in order to extend or embed the R engine. However, there is a known issue with the current RInside / Rcpp on Windows; but releases 0.2.1 and 0.7.1 do work.

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
We assume a recent version of R so that
\[
\text{install.packages(c("Rcpp","RInside","inline"))}
\]
gets us current versions of the packages
We assume a recent version of R so that
\[
\text{install.packages(c("Rcpp","RInside","inline"))}
\]
gets us current versions of the packages

All examples shown should work 'as is' on Linux, OS X and Windows *provided a complete R development environment*
Preliminaries

- We assume a recent version of R so that `install.packages(c("Rcpp","RInside","inline"))` gets us current versions of the packages.
- All examples shown should work 'as is' on Linux, OS X and Windows *provided a complete R development environment*.
- The *R Installation and Administration* manual is an excellent start if you need to address the preceding point.
We assume a recent version of R so that
\texttt{install.packages(c("Rcpp","RInside","inline"))}
gets us current versions of the packages.

All examples shown should work ‘as is’ on Linux, OS X and Windows\textit{ provided a complete R development environment}.

The \textit{R Installation and Administration} manual is an excellent start if you need to address the preceding point.

In particular, one must use the same compilers used to build R in order to extend or embed the R engine.
Preliminaries

- We assume a recent version of R so that
  `install.packages(c("Rcpp","RInside","inline"))`
  gets us current versions of the packages
- All examples shown should work ‘as is’ on Linux, OS X and Windows provided a complete R development environment
- The *R Installation and Administration* manual is an excellent start if you need to address the preceding point
- In particular, one must use the same compilers used to build R in order to extend or embed the R engine
- However, there is a known issue with the current RInside / Rcpp on Windows; but releases 0.2.1 and 0.7.1 do work
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
Chambers (2008) opens chapter 11 (Interfaces I: Using C and Fortran) with these words:

Chambers (2008) opens chapter 11 (Interfaces I: Using C and Fortran) with these words:

Since the core of R is in fact a program written in the C language, it’s not surprising that the most direct interface to non-R software is for code written in C, or directly callable from C. All the same, including additional C code is a serious step, with some added dangers and often a substantial amount of programming and debugging required. You should have a good reason.
Chambers (2008) then proceeds with this rough map of the road ahead:
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:

- It’s more work
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:
- It’s more work
- Bugs will bite
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:

- It’s more work
- Bugs will bite
- Potential platform dependency
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:
- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software

So is the deck stacked against us?

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:

- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software
Chambers (2008) then proceeds with this rough map of the road ahead:

**Against:**
- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software

**In Favor:**
- New and trusted computations
Chambers (2008) then proceeds with this rough map of the road ahead:

**Against:**
- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software

**In Favor:**
- New and trusted computations
- **Speed**
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:
- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software

In Favor:
- New and trusted computations
- Speed
- Object references
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:
- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software

In Favor:
- New and trusted computations
- Speed
- Object references
Chambers (2008) then proceeds with this rough map of the road ahead:

Against:
- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software

In Favor:
- New and trusted computations
- Speed
- Object references

So is the deck stacked against us?
Chambers (2008) then proceeds with this rough map of the road ahead:

**Against:**
- It’s more work
- Bugs will bite
- Potential platform dependency
- Less readable software

**In Favor:**
- New and trusted computations
- Speed
- Object references

So is the deck stacked against us?
1. Extending R
- Why?
- The standard API
- Inline

2. Rcpp
- Overview
- New API
- Examples
R offers several functions to access compiled code: we focus on `.C` and `.Call` here. (R Extensions, sections 5.2 and 5.9; Software for Data Analysis).
R offers several functions to access compiled code: we focus on `.C` and `.Call` here. (*R Extensions*, sections 5.2 and 5.9; *Software for Data Analysis*).

The canonical example is the convolution function:

```c
void convolve(double *a, int *na, double *b, int *nb, double *ab)
{
    int i, j, nab = *na + *nb - 1;
    for(i = 0; i < nab; i++)
        ab[i] = 0.0;
    for(i = 0; i < *na; i++)
        for(j = 0; j < *nb; j++)
            ab[i + j] += a[i] * b[j];
}
```

Dirk Eddelbuettel  
Seamless R and C++ Integration @ WU Wien, May 2010
The convolution function is called from R by

\begin{verbatim}
conv <function (a, b)
  C("convolve",
  as.double(a),
  as.integer(length(a)),
  as.double(b),
  as.integer(length(b)),
  ab = double(length(a) + length(b) - 1))
\end{verbatim}

As stated in the manual, one must take care to coerce all the arguments to the correct R storage mode before calling C as mistakes in matching the types can lead to wrong results or hard-to-catch errors.
The convolution function is called from R by

```r
conv <- function(a, b)
  .C("convolve",
      as.double(a),
      as.integer(length(a)),
      as.double(b),
      as.integer(length(b)),
      ab = double(length(a) + length(b) - 1))$ab
```

As stated in the manual, one must take care to coerce all the arguments to the correct R storage mode before calling `.C` as mistakes in matching the types can lead to wrong results or hard-to-catch errors.
Example: Running the convolution code via .C

All these files are at http://dirk.eddelbuettel.com/code/rcppTut

Turn the C source file into a dynamic library using

```
R CMD SHLIB convolve.C.c
```
Why? The standard API Inline

Example: Running the convolution code via .C
All these files are at http://dirk.eddelbuettel.com/code/rcppTut

- Turn the C source file into a dynamic library using
  
  R CMD SHLIB convolve.C.c

- Load it inside an R script or session using
  
  dyn.load("convolve.C.so")
Example: Running the convolution code via .C

All these files are at http://dirk.eddelbuettel.com/code/rcppTut

- Turn the C source file into a dynamic library using
  \[ \text{R CMD SHLIB convolve.C.c} \]

- Load it inside an R script or session using
  \[ \text{dyn.load("convolve.C.so")} \]

- Use it via the \text{.C()} interface as shown on previous slide
Example: Running the convolution code via `.C`

All these files are at [http://dirk.eddelbuettel.com/code/rcppTut](http://dirk.eddelbuettel.com/code/rcppTut)

- Turn the C source file into a dynamic library using
  
  ````
  R CMD SHLIB convolve.C.c
  ````

- Load it inside an R script or session using
  
  ````
  dyn.load("convolve.C.so")
  ````

- Use it via the `.C()` interface as shown on previous slide

- **All together in a helper file** `convolve.C.sh`
  
  ````
  #!/bin/sh
  
  R CMD SHLIB convolve.C.c
  
  cat convolve.C.call.R | R --no-save
  ````
Using `.Call`, the example becomes
Using `.Call`, the example becomes

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

extern "C" SEXP convolve2(SEXP a, SEXP b)
{
    int i, j, na, nb, nab;
    double *xa, *xb, *xab;
    SEXP ab;

    PROTECT(a = AS_NUMERIC(a));
    PROTECT(b = AS_NUMERIC(b));
    na = LENGTH(a); nb = LENGTH(b); nab = na + nb - 1;
    PROTECT(ab = NEW_NUMERIC(nab));
    xa = NUMERIC_POINTER(a); xb = NUMERIC_POINTER(b);
    xab = NUMERIC_POINTER(ab);
    for(i = 0; i < nab; i++) xab[i] = 0.0;
    for(i = 0; i < na; i++)
        for(j = 0; j < nb; j++) xab[i + j] += xa[i] * xb[j];
    UNPROTECT(3);
    return(ab);
}
```

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
Now the call simplifies to just the function name and the vector arguments—all other handling is done at the C/C++ level:

```r
conv <- function(a, b) .Call("convolve2", a, b)
```
Now the call simplifies to just the function name and the vector arguments—all other handling is done at the C/C++ level:

```r
conv <- function(a, b) .Call("convolve2", a, b)
```

In summary, we see that

- there are different entry points
Now the call simplifies to just the function name and the vector arguments—all other handling is done at the C/C++ level:

```
conv <- function(a, b) .Call("convolve2", a, b)
```

In summary, we see that

- there are different entry points
- using different calling conventions
Now the call simplifies to just the function name and the vector arguments—all other handling is done at the C/C++ level:

```r
conv <- function(a, b) .Call("convolve2", a, b)
```

In summary, we see that

- there are different entry points
- using different calling conventions
- leading to code that may need to do more work at the lower level.
Example: Running the convolution code via `.Call`

- Turn the C source file into a dynamic library using
  
  ```
  R CMD SHLIB convolve.Call.c
  ```
Example: Running the convolution code via `.Call`

- Turn the C source file into a dynamic library using
  
  ```
  R CMD SHLIB convolve.Call.c
  ```

- Load it inside an R script or session using
  
  ```
  dyn.load("convolve.Call.so")
  ```
Example: Running the convolution code via `.Call`

- Turn the C source file into a dynamic library using
  
  ```
  R CMD SHLIB convolve.Call.c
  ```

- Load it inside an R script or session using
  
  ```
  dyn.load("convolve.Call.so")
  ```

- Use it via the `.Call()` interface as shown previously
Example: Running the convolution code via `.Call`

- Turn the C source file into a dynamic library using
  ```
  R CMD SHLIB convolve.Call.c
  ```
- Load it inside an R script or session using
  ```
  dyn.load("convolve.Call.so")
  ```
- Use it via the `.Call()` interface as shown previously
- **All together in a helper file** `convolve.Call.sh`
  ```
  #!/bin/sh
  
  R CMD SHLIB convolve.Call.c
  
  cat convolve.Call.call.R | R --no-save
  ```
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
inline is a package by Oleg Sklyar et al that provides the function \texttt{cfunction} which can wrap Fortran, C or C++ code.
inline is a package by Oleg Sklyar et al that provides the function cfunction which can wrap Fortran, C or C++ code.

```r
## A simple Fortran example
code <- "
  integer i
  do 1 i = 1, n(1)
  1 x(i) = x(i)**3
"
cubefn <- cfunction(signature(n="integer", x="numeric"),
                   code, convention=".Fortran")
x <- as.numeric(1:10)
n <- as.integer(10)
cubefn(n, x)$x
```

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
inline is a package by Oleg Sklyar et al that provides the function cfunction which can wrap Fortran, C or C++ code.

```r
## A simple Fortran example

```c
code <- "
   integer i
   do 1 i=1, n(1)
      1 x(i) = x(i)**3

"
cubefn <- cfunction(signature(n="integer", x="numeric"),
   code, convention=".Fortran")

x <- as.numeric(1:10)
n <- as.integer(10)
cubefn(n, x)$x
```

cfunction takes care of compiling, linking, loading, ... by placing the resulting dynamically-loadable object code in the per-session temporary directory used by R.
Example: Convolution via .C with inline
Using the file `convolve.C.inline.R`

```r
require(inline)

code <- "int i, j, nab = *na + *nb - 1;

    for(i = 0; i < nab; i++)
        ab[i] = 0.0;

    for(i = 0; i < *na; i++) {
        for(j = 0; j < *nb; j++)
            ab[i + j] += a[i] * b[j];
    }
"

fun <- cfunction(signature(a="numeric", na="numeric",
                          b="numeric", nb="numeric",
                          ab="numeric"),
                 code, language="C", convention=".C")

fun(1:10, 10, 10:1, 10, numeric(19))$ab
```
Example: Convolution via `.Call` with inline
Using the file `convolve.Call.inline.R`
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
In a nutshell:
In a nutshell:

- **Rcpp** makes it easier to interface C++ and R code.
In a nutshell:

- **Rcpp** makes it easier to interface C++ and R code.
- Using the `.Call` interface, we can use features of the C++ language to automate the tedious bits of the macro-based C-level interface to R.
In a nutshell:

- **Rcpp** makes it easier to interface C++ and R code.
- Using the `.Call` interface, we can use features of the C++ language to automate the tedious bits of the macro-based C-level interface to R.
- One major advantage of using `.Call` is that richer R objects (vectors, matrices, lists, ... in fact most SEXP types incl functions, environments etc) can be passed directly between R and C++ without the need for explicit passing of dimension arguments.
Compiled Code: Rcpp

In a nutshell:

- **Rcpp** makes it easier to interface C++ and R code.
- Using the `.Call` interface, we can use features of the C++ language to automate the tedious bits of the macro-based C-level interface to R.
- One major advantage of using `.Call` is that richer R objects (vectors, matrices, lists, ... in fact most SEXP types incl functions, environments etc) can be passed directly between R and C++ without the need for explicit passing of dimension arguments.
- By using the C++ class layers, we do not need to manipulate the SEXP objects using any of the old-school C macros.
Compiled Code: Rcpp

In a nutshell:

- **Rcpp** makes it easier to interface C++ and R code.

- Using the `.Call` interface, we can use features of the C++ language to automate the tedious bits of the macro-based C-level interface to R.

- One major advantage of using `.Call` is that richer R objects (vectors, matrices, lists, ... in fact most SEXP types incl functions, environments etc) can be passed directly between R and C++ without the need for explicit passing of dimension arguments.

- By using the C++ class layers, we do not need to manipulate the SEXP objects using any of the old-school C macros.

- **inline** eases usage, development and testing.
Example: Convolution using classic Rcpp

Using the file `convolve.Call.Rcpp.classic.R`

```r
require(inline)
code <- 'require(inline)

code <- 
RcppVector<double> xa(a);
RcppVector<double> xb(b);

int nab = xa.size() + xb.size() - 1;
RcppVector<double> xab(nab);
for (int i = 0; i < nab; i++) xab(i) = 0.0;

for (int i = 0; i < xa.size(); i++)
    for (int j = 0; j < xb.size(); j++)
        xab(i + j) += xa(i) * xb(j);

RcppResultSet rs;
rs.add("ab", xab);
return rs.getReturnList();
',

fun <- cppfunction(signature(a="numeric", b="numeric"), code)
fun(1:10, 10:1)
'

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
Rcpp was significantly extended over the last few months to permit more natural expressions. Consider this comparison between the R API and the new Rcpp API:

Rcpp: The 'New API'

```r
# R API
1. SEXP ab ;
2. PROTECT(ab = R_Register Foreign Function (STRSXP, 2)) ;
3. SET_STRING_ELT(ab, 0, mkChar("foo")) ;
4. SET_STRING_ELT(ab, 1, mkChar("bar")) ;
5. UNPROTECT(1)
```

```r
# Rcpp API
1. CharacterVector ab (2) ;
2. ab [0] = "foo" ;
3. ab [1] = "bar" ;
```

Data types, including STL containers and iterators, can be nested and other niceties. Implicit converters allow us to combine types:

```r
1. std::vector<double> vec ;
2. [ . . . ]
3. List x (3) ;
4. x [0] = vec ;
5. x [1] = "some text" ;
6. x [2] = 42 ;
```

With Rcpp 0.7.11 or later we can do:

```r
1. std::vector<double> vec ;
2. [ . . . ]
3. List x = List::create (vec , "some text" , 42) ;
```
Rcpp was significantly extended over the last few months to permit more natural expressions. Consider this comparison between the R API and the new Rcpp API:

```r
SEXP ab;
PROTECT(ab = allocVector(STRSXP, 2));
SET_STRING_ELT(ab, 0, mkChar("foo"));
SET_STRING_ELT(ab, 1, mkChar("bar"));
UNPROTECT(1);
```

```r
CharacterVector ab(2);
ab[0] = "foo";
ab[1] = "bar";
```
Rcpp was significantly extended over the last few months to permit more natural expressions. Consider this comparison between the R API and the new Rcpp API:

```r
SEXP ab;
PROTECT(ab = allocVector(STRSXP, 2));
SET_STRING_ELT(ab, 0, mkChar("foo"));
SET_STRING_ELT(ab, 1, mkChar("bar"));
UNPROTECT(1);
```

1. SEXP ab;
2. PROTECT(ab = allocVector(STRSXP, 2));
3. SET_STRING_ELT(ab, 0, mkChar("foo"));
4. SET_STRING_ELT(ab, 1, mkChar("bar"));
5. UNPROTECT(1);

```r
CharacterVector ab(2);
ab[0] = "foo";
ab[1] = "bar";
```
Rcpp was significantly extended over the last few months to permit more natural expressions. Consider this comparison between the R API and the new Rcpp API:

```
1 SEXP ab;
2 PROTECT(ab = allocVector(STRSXP, 2));
3 SET_STRING_ELT(ab, 0, mkChar("foo"));
4 SET_STRING_ELT(ab, 1, mkChar("bar"));
5 UNPROTECT(1);
```

```
1 CharacterVector ab(2);
2 ab[0] = "foo";
3 ab[1] = "bar";
```

Data types, including STL containers and iterators, can be nested and other niceties. Implicit converters allow us to combine types:
Rcpp was significantly extended over the last few months to permit more natural expressions. Consider this comparison between the R API and the new Rcpp API:

```r
SEXP ab;
PROTECT(ab = allocVector(STRSXP, 2));
SET_STRING_ELT(ab, 0, mkChar("foo"));
SET_STRING_ELT(ab, 1, mkChar("bar"));
UNPROTECT(1);
```

```r
CharacterVector ab(2);
ab[0] = "foo";
ab[1] = "bar";
```

Data types, including STL containers and iterators, can be nested and other niceties. Implicit converters allow us to combine types:

```r
std::vector<double> vec;
[[...]]
List x(3);
x[0] = vec;
x[1] = "some text";
x[2] = 42;
```
Rcpp was significantly extended over the last few months to permit more natural expressions. Consider this comparison between the R API and the new Rcpp API:

```
1 SEXP ab;
2 PROTECT(ab = allocVector(STRSXP, 2));
3 SET_STRING_ELT(ab, 0, mkChar("foo"));
4 SET_STRING_ELT(ab, 1, mkChar("bar"));
5 UNPROTECT(1);
```

```
1 CharacterVector ab(2);
2 ab[0] = "foo";
3 ab[1] = "bar";
```

Data types, including STL containers and iterators, can be nested and other niceties. Implicit converters allow us to combine types:

```
1 std::vector<double> vec;
2 [...]  // With Rcpp 0.7.11 or later we can do:
3 List x(3);
4 x[0] = vec;
5 x[1] = "some text";
6 x[2] = 42;
```

```
1 std::vector<double> vec;
2 [...]  // With Rcpp 0.7.11 or later we can do:
3 List x = List::create(vec,
4                     "some text",
5                     42);
```
In R, functional programming is easy:

```r
data(faithful)
lapply(faithful, summary)
```

```cpp
Rcpp::List input(data)
Rcpp::Function f(fun)
Rcpp::List output(input.size())
std::transform(input.begin(), input.end(), output.begin(), f)
output.names() = input.names()
return output
```

### Functional programming in both languages

In **R**, functional programming is easy:

```r
R> data(faithful); lapply(faithful, summary)
```

<table>
<thead>
<tr>
<th></th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>eruptions</td>
<td>1.60</td>
<td>2.16</td>
<td>4.00</td>
<td>3.49</td>
<td>4.45</td>
<td>5.10</td>
</tr>
<tr>
<td>waiting</td>
<td>43.0</td>
<td>58.0</td>
<td>76.0</td>
<td>70.9</td>
<td>82.0</td>
<td>96.0</td>
</tr>
</tbody>
</table>
In R, functional programming is easy:

```r
R> data(faithful); lapply(faithful, summary)
$eruptions
   Min. 1st Qu. Median Mean 3rd Qu. Max. 
   1.60  2.16  4.00  3.49  4.45  5.10

$waiting
   Min. 1st Qu. Median Mean 3rd Qu. Max. 
   43.0  58.0  76.0  70.9  82.0  96.0
```

We can do that in C++ as well and pass the R function down to the data that we let the STL `transform` function iterate over:

```cpp
src <- 'Rcpp::List input(data);
    Rcpp::Function f(fun);
    Rcpp::List output(input.size());
    std::transform(input.begin(), input.end(), output.begin(), f);
    output.names() = input.names();
    return output;
' cpp_lapply <- cppfunction(signature(data="list", fun = "function"), src)
```
Exception handling

Automatic catching and conversion of C++ exceptions:

R> library(Rcpp); library(inline)
R> cpp <- '
+ Rcpp::NumericVector x(xs); // automatic conversion from SEXP
+ for (int i=0; i<x.size(); i++) {
+ if (x[i] < 0)
+ throw std::range_error("Non-negative values required");
+ x[i] = log(x[i]);
+ }
+ return x; // automatic conversion to SEXP
+ '
R> fun <- cppfunction(signature(xs="numeric"), cpp)
R> fun(seq(2, 5))
[1] 0.6931 1.0986 1.3863 1.6094
R> fun(seq(5, -2))
Error in fun(seq(5, -2)) : Non-negative values required
R> fun(LETTERS[1:5])
Error in fun(LETTERS[1:5]) : not compatible with INTSXP
R>
Exception handling

Automatic catching and conversion of C++ exceptions:

```r
R> library(Rcpp); library(inline)
R> cpp <- ' + Rcpp::NumericVector x(xs); // automatic conversion from SEXP
+ for (int i=0; i<x.size(); i++) {
+   if (x[i] < 0)
+     throw std::range_error("Non-negative values required");
+   x[i] = log(x[i]);
+ }
+ return x; // automatic conversion to SEXP
+
R> fun <- cppfunction(signature(xs="numeric"), cpp)
R> fun(seq(2, 5))
[1] 0.6931 1.0986 1.3863 1.6094
R> fun(seq(5, -2))
Error in fun(seq(5, -2)) : Non-negative values required
R> fun(LETTERS[1:5])
Error in fun(LETTERS[1:5]) : not compatible with INTSXP
R>
```

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
We attempted to automate forwarding of exceptions from the C++ layer to the R layer.
We attempted to automate forwarding of exceptions from the C++ layer to the R layer.

This works (thanks to some gcc magic) on operating system with an X in their name, but not on Windows.
Exception handling: Usage

- We attempted to automate forwarding of exceptions from the C++ layer to the R layer.
- This works (thanks to some gcc magic) on operating system with an X in their name, but not on Windows.
- We therefore once again recommend to wrap code with
  ```
  try {
    forward_exception_to_r(ex);
  } catch(...) {
    ::Rf_error("c++ exception (unknown reason)");
  }
  ```
Exception handling: Usage

- We attempted to automate forwarding of exceptions from the C++ layer to the R layer.
- This works (thanks to some gcc magic) on operating system with an X in their name, but not on Windows.
- We therefore once again recommend to wrap code with

  ```
  try {
  }
  ```

  and

  ```
  } catch( std::exception &ex) {
      forward_exception_to_r(ex);
  } catch(...) {
      ::Rf_error("c++ exception (unknown reason)");
  }
  ```

- Because this is invariant, we provide macros `BEGIN_RCCP` and `END_RCCP`.
Exception handling: Usage

- We attempted to automate forwarding of exceptions from the C++ layer to the R layer.
- This works (thanks to some gcc magic) on operating system with an X in their name, but not on Windows.
- We therefore once again recommend to wrap code with

```cpp
try {
    // Your code here
}
and

} catch( std::exception &ex) {
    forward_exception_to_r(ex);
} catch(...) {
    ::Rf_error("c++ exception (unknown reason)");
}
```

- Because this is invariant, we provide macros `BEGIN_RCPP` and `END_RCPP`.
- We provide a variant `cppfunction` of `inline::cfunction` which automatically inserts these at the beginning and end of the code snippets.
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
Example: Convolution using new Rcpp

Using the file `convolve.Call.Rcpp.new.R`

```r
require(inline)

code <- '
Rcpp::NumericVector xa(a); // automatic conversion from SEXP
Rcpp::NumericVector xb(b);

int n_xa = xa.size();
int n_xb = xb.size();
int nab = n_xa + n_xb - 1;

Rcpp::NumericVector xab(nab);

for (int i = 0; i < n_xa; i++)
  for (int j = 0; j < n_xb; j++)
    xab[i + j] += xa[i] * xb[j];

return xab; // automatic conversion to SEXP
',

fun <- cppfunction(signature(a="numeric", b="numeric"), code)

fun(1:10, 10:1)
```
In a recently-submitted paper, the following table summarises the performance of convolution examples:

Table 1: Performance for convolution example

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time in Relative millsec to R API</th>
</tr>
</thead>
<tbody>
<tr>
<td>R API (as benchmark)</td>
<td>32</td>
</tr>
<tr>
<td>RcppVector&lt;double&gt;</td>
<td>354</td>
</tr>
<tr>
<td>NumericVector::operator[]</td>
<td>52</td>
</tr>
<tr>
<td>NumericVector::begin</td>
<td>33</td>
</tr>
</tbody>
</table>

We averaged 1000 replications with two 100-element vectors – see examples/ConvolveBenchmarks in Rcpp for details.
In a recently-submitted paper, the following table summarises the performance of convolution examples:

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time in millisecc</th>
<th>Relative to R API</th>
</tr>
</thead>
<tbody>
<tr>
<td>R API (as benchmark)</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td><code>RcppVector&lt;double&gt;</code></td>
<td>354</td>
<td>11.1</td>
</tr>
<tr>
<td><code>NumericVector::operator[]</code></td>
<td>52</td>
<td>1.6</td>
</tr>
<tr>
<td><code>NumericVector::begin</code></td>
<td>33</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 1: Performance for convolution example
In a recently-submitted paper, the following table summarises the performance of convolution examples:

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Time in millisec</th>
<th>Relative to R API</th>
</tr>
</thead>
<tbody>
<tr>
<td>R API (as benchmark)</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>RcppVector&lt;double&gt;</td>
<td>354</td>
<td>11.1</td>
</tr>
<tr>
<td>NumericVector::operator[]</td>
<td>52</td>
<td>1.6</td>
</tr>
<tr>
<td>NumericVector::begin</td>
<td>33</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 1: Performance for convolution example

We averaged 1000 replications with two 100-element vectors – see examples/ConvolveBenchmarks/ in Rcpp for details.
Regression is a key component of many studies. In simulations, we often want to run a very large number of regressions.
Another Speed Comparison Example

Regression is a key component of many studies. In simulations, we often want to run a very large number of regressions.

- R has `lm()` as the general purposes function. It is very powerful and returns a rich object—but it is not lightweight.
Another Speed Comparison Example

Regression is a key component of many studies. In simulations, we often want to run a very large number of regressions.

R has \texttt{lm()} as the general purposes function. It is very powerful and returns a rich object—but it is not lightweight.

For this purpose, R has \texttt{lm.fit()}. But, this does not provide all relevant auxiliary data as e.g. the standard error of the estimate.
Regression is a key component of many studies. In simulations, we often want to run a very large number of regressions.

R has `lm()` as the general purposes function. It is very powerful and returns a rich object—but it is not lightweight.

For this purpose, R has `lm.fit()`. But, this does not provide all relevant auxiliary data as e.g. the standard error of the estimate.

For the most recent *Introduction to High-Performance Computing with R* tutorial, I had written a hybrid R/C/C++ solution using the GNU GSL.
Regression is a key component of many studies. In simulations, we often want to run a very large number of regressions.

R has `lm()` as the general purposes function. It is very powerful and returns a rich object—but it is not lightweight.

For this purpose, R has `lm.fit()`. But, this does not provide all relevant auxiliary data as e.g. the standard error of the estimate.

For the most recent *Introduction to High-Performance Computing with R* tutorial, I had written a hybrid R/C/C++ solution using the GNU GSL.

We complement this with a new C++ implementation around the Armadillo linear algebra classes.
Linear regression via GSL: lmGSL()
See the directory Rcpp/examples/FastLM

```
ImGSL <- function() {
  src <- '  
  RcppVectorView<double> Yr(Ysexp);
  RcppMatrixView<double> Xr(Xsexp);
  int i, j, n = Xr.dim1(), k = Xr.dim2();
  double chi2;
  gsl_matrix *X = gsl_matrix_alloc(n,k);
  gsl_vector *y = gsl_vector_alloc(n);
  gsl_vector *c = gsl_vector_alloc(k);
  gsl_matrix *cov = gsl_matrix_alloc(k,k);

  for (i = 0; i < n; i++) {
    for (j = 0; j < k; j++) {
      gsl_matrix_set (X, i, j, Xr(i,j));
    }
    gsl_vector_set (y, i, Yr(i));
  }

  gsl_multifit_linear_workspace *wk =
    gsl_multifit_linear_alloc(n,k);
  gsl_multifit_linear(X,y,c,cov,&chi2,wk);
  gsl_multifit_linear_free (wk);
  RcppVector<double> StdErr(k);  
  RcppVector<double> Coef(k);

  for (i = 0; i < k; i++) {
    Coef(i) = gsl_vector_get (c, i);
    StdErr (i) =
      sqrt(gsl_matrix_get (cov, i, i));
  }

  gsl_matrix_free(X);
  gsl_vector_free (y);
  gsl_vector_free (c);
  gsl_matrix_free (cov);

  RcppResultSet rs;
  rs.add( "coef", Coef);
  rs.add( "stderr", StdErr);

  return = rs.getList();
}
```

```cpp
fun <-
  cppfunction (signature(Ysexp="numeric ",
    Xsexp="numeric "), src,
    includes=  
      "#include <gsl/gsl_multifit.h>",
    cppargs="-l/usr/include ",
    libargs="-l gsl -l gslcblas ")
```
Linear regression via Armadillo: lmArmadillo example

Also see the directory Rcpp/examples/FastLM

```
# lmArmadillo <- function() {
  src <- "
  Rcpp::NumericVector yr(Ysexp);
  Rcpp::NumericVector Xr(Xsexp);
  // actually an n x k matrix
  std::vector<int> dims = Xr.attr("dim");
  int n = dims[0], k = dims[1];
  arma::mat X(Xr.begin(), n, k, false); // use advanced armadillo constructors
  arma::colvec y(yr.begin(), yr.size());
  arma::colvec coef = solve(X, y); // model fit
  arma::colvec resid = y - X*coef; // to comp. std.err of the coefficients
  arma::mat covmat = trans(resid)*resid/(n-k) * arma::inv(arma::trans(X)*X);

  Rcpp::NumericVector coefr(k), stderrrestr(k);
  for (int i=0; i<k; i++) {
    // with RcppArmadillo template converters
    coefr[i] = coef[i]; // this would not be needed but we only
    stderrrestr[i] = sqrt(covmat(i,i)); // have Rcpp.h here
  }

  return Rcpp::List::create( Rcpp::Named("coefficients", coefr), // Rcpp 0.7.11
    Rcpp::Named("stderr", stderrrestr));

  }

  ## turn into a function that R can call
  fun <- cppfunction(signature(Ysexp="numeric", Xsexp="numeric"),
    src, includes="#include <armadillo>",
    cppargs="-I/usr/include", libargs="-llarmadillo")
```
Linear regression via Armadillo: RcppArmadillo

See fastLm in the RcppArmadillo package

**fastLm in the new RcppArmadillo release does even better:**

```cpp
#include <RcppArmadillo.h>

extern "C" SEXP fastLm(SEXP ys, SEXP Xs) {

  try {
    Rcpp::NumericVector yr(ys);                  // creates Rcpp vector from SEXP
    Rcpp::NumericMatrix Xr(Xs);                 // creates Rcpp matrix from SEXP
    int n = Xr.nrow(), k = Xr.ncol();

    arma::mat X(Xr.begin(), n, k, false);       // reuses memory and avoids extra copy
    arma::colvec y(yr.begin(), yr.size(), false);
    arma::colvec coef = arma::solve(X, y);     // fit model y ~ X
    arma::colvec res = y - X*coef;             // residuals

    double s2 = std::inner_product(res.begin(), res.end(), res.begin(), double()) / (n-k);   // std.errors of coefficients
    arma::colvec stderr = arma::sqrt(s2*arma::diagvec(arma::inv(arma::trans(X)*X)));

    return Rcpp::List::create(Rcpp::Named("coefficients") = coef,
                               Rcpp::Named("stderr") = stderr,
                               Rcpp::Named("df") = n - k);
  } catch (std::exception &ex) {
    forward_exception_to_r(ex);
  } catch (...) {
    ::Rf_error("c++ exception (unknown reason)" );
  }

  return R_NilValue; // -Wall
}
```
We also wrote `fastLm` in a new package `RcppGSL`:

```c
extern "C" SEXP fastLm(SEXP ys, SEXP Xs) {
BEGIN_RCPP
RcppGSL::vector<double> y = ys; // create gsl data structures from SEXP
RcppGSL::matrix<double> X = Xs;
int n = X.nrow(), k = X.ncol();
double chisq;
RcppGSL::vector<double> coef(k); // to hold the coefficient vector
RcppGSL::matrix<double> cov(k, k); // and the covariance matrix
// the actual fit requires working memory we allocate and free
gsl_multifit_linear_workspace *work = gsl_multifit_linear_alloc(n, k);
gsl_multifit_linear(X, y, coef, cov, &chisq, work);
gsl_multifit_linear_free(work);
// extract the diagonal as a vector view
gsl_vector_view diag = gsl_matrix_diagonal(cov);
// currently there is not a more direct interface in Rcpp::NumericVector
// that takes advantage of wrap, so we have to do it in two steps
Rcpp::NumericVector stderr; stderr = diag;
std::transform(stderr.begin(), stderr.end(), stderr.begin(), sqrt);
Rcpp::List res = Rcpp::List::create(Rcpp::Named("coefficients") = coef,
                                      Rcpp::Named("stderr") = stderr,
                                      Rcpp::Named("df") = n - k);
// free all the GSL vectors and matrices — as these are really C data structures
// we cannot take advantage of automatic memory management
coef.free(); cov.free(); y.free(); X.free();
return res; // return the result list to R
END_RCPP
```

Dirk Eddelbuettel  
Seamless R and C++ Integration @ WU Wien, May 2010
Rcpp Example: Regression timings

Comparison of R and linear model fit routines

The small \texttt{longley} example exhibits less variability between methods, but the larger data set shows the gains more clearly.

For the small data set, all three appear to improve similarly on \texttt{lm}.

Source: Our calculations, see examples/FastLM/ in \texttt{Rcpp}. 

Dirk Eddelbuettel  
Seamless R and C++ Integration @ WU Wien, May 2010
Comparison of R and linear model fit routines

By dividing the `lm` time by the respective times, we obtain the 'possible gains' from switching.

One caveat, measurements depends critically on the size of the data as well as the cpu and libraries that are used.

Source: Our calculations, see examples/FastLM/ in Rcpp.
Armadillo uses delayed evaluation (via recursive template and template meta-programming) to combine several operations into one expression reducing / eliminating temporary objects.
Possible gains from template meta-programming

Armadillo uses delayed evaluation (via recursive template and template meta-programming) to combine several operations into one expression reducing / eliminating temporary objects.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Relative performance improvement for small matrices</th>
<th>Relative performance improvement for medium to large matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IT++  Newmat</td>
<td>IT++  Newmat</td>
</tr>
<tr>
<td>A + B</td>
<td>15.0      10.0</td>
<td>3.5      1.0</td>
</tr>
<tr>
<td>A + B + C + D</td>
<td>15.0      10.0</td>
<td>6.0      1.5</td>
</tr>
<tr>
<td>A * B * C * D</td>
<td>2.5       10.0</td>
<td>2.5      20.0</td>
</tr>
<tr>
<td>B.row(size-1) = A.row(0)</td>
<td>16.0      44.0</td>
<td>2.0      4.5</td>
</tr>
<tr>
<td>trans(p)*inv(diagmat(q))*r</td>
<td>77.0      23.0</td>
<td>1086.0  5.0</td>
</tr>
</tbody>
</table>

Table 2: Gains from C++ template programming

Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
Jeff Horner’s work on RA{}pache lead to joint work in littler, a scripting / cmdline front-end. As it embeds R and simply ’feeds’ the REPL loop, the next step was to embed R in proper C++ classes: RInside.
Jeff Horner’s work on **RApache** lead to joint work in **littler**, a scripting / cmdline front-end. As it embeds **R** and simply ‘feeds’ the REPL loop, the next step was to embed **R** in proper **C++** classes: **RInside**.

```cpp
#include <RInside.h> // for the embedded R via RInside

int main(int argc, char *argv[]) {
  RInside R(argc, argv); // create an embedded R instance
  R["txt"] = "Hello, world!\n"; // assign a char* (string) to 'txt'
  R.parseEvalQ("cat(txt)"); // eval the init string, ignoring any returns
  exit(0);
}
```

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
This example shows some of the new assignment and converter code:

```cpp
#include <RInside.h> // for the embedded R via RInside

int main(int argc, char *argv[]) {
    RInside R(argc, argv); // create an embedded R instance
    R["x"] = 10;
    R["y"] = 20;
    R.parseEvalQ("z <- x + y");
    int sum = R["z"];
    std::cout << "10 + 20 = " << sum << std::endl;
    exit(0);
}
```
#include <RInside.h>  // for the embedded R via RInside
#include <iomanip>

int main(int argc, char *argv[]) {
    RInside R(argc, argv);  // create an embedded R instance
    SEXP ans;
    R.parseEvalQ("suppressMessages(library(fPortfolio))");
    txt = "lppData <- 100 * LPP2005.RET[, 1:6];"
    "ewSpec <- portfolioSpec(); nAssets <- ncol(lppData);");
    R.parseEval(txt, ans);  // prepare problem
    const double dvec[6] = {0.1, 0.1, 0.1, 0.1, 0.3, 0.3};  // weights
    const std::vector<double> w(dvec, &dvec[6]);
    R.assign(w, "weightsvec");  // assign STL vec to R's 'weightsvec'
    R.parseEvalQ("setWeights(ewSpec) <- weightsvec");
    txt = "ewPortfolio <- feasiblePortfolio(data = lppData, spec = ewSpec, "
    "constraints = "LongOnly");"
    "print(ewPortfolio);"
    "vec <- getCovRiskBudgets(ewPortfolio@portfolio)";
    ans = R.parseEval(txt);  // assign covRiskBudget weights to ans
    Rcpp::NumericVector V(ans);  // convert SEXP variable to an RcppVector
    ans = R.parseEval("names(vec)");  // assign columns names to ans
    Rcpp::CharacterVector n(ans);
    for (int i=0; i<names.size(); i++) {
        std::cout << std::setw(16) << n[i] << "\t" << std::setw(11) << V[i] << "\n";
    }
    exit(0);
And another *parallel* example

See the file RInside/mpi/rinside_mpi_sample2.cpp

```c++
#include <mpi.h>     // mpi header
#include <RInside.h> // for the embedded R via RInside

int main(int argc, char *argv[]) {

    MPI::Init(argc, argv);       // mpi initialization
    int myrank = MPI::COMM_WORLD.Get_rank(); // obtain current node rank
    int nodesize = MPI::COMM_WORLD.Get_size(); // obtain total nodes running.

    RInside R(argc, argv);       // create an embedded R instance

    std::stringstream txt;
    txt << "Hello from node " << myrank
        << " of " << nodesize << " nodes!" << std::endl;
    R.assign( txt.str(), "txt" ); // assign string to R variable 'txt'

    std::string evalstr = "cat(txt)"; // show node information
    R.parseEvalQ(evalstr);           // eval the string, ign. any returns

    MPI::Finalize();               // mpi finalization

    exit(0);
}
```

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
C++ programs compute, gather or aggregate raw data.
RInside workflow

- C++ programs compute, gather or aggregate raw data.
- Data is saved and analysed before a new 'run' is launched.
RInside workflow

- C++ programs compute, gather or aggregate raw data.
- Data is saved and analysed before a new ’run’ is launched.
- **With RInside we now skip a step:**
C++ programs compute, gather or aggregate raw data. Data is saved and analysed before a new ’run’ is launched. With **RInside** we now skip a step:
- collect data in a vector or matrix
C++ programs compute, gather or aggregate raw data.
Data is saved and analysed before a new ‘run’ is launched.
With RInside we now skip a step:
- collect data in a vector or matrix
- pass data to R — easy thanks to Rcpp wrappers

RInside workflow
RInside workflow

- C++ programs compute, gather or aggregate raw data.
- Data is saved and analysed before a new ’run’ is launched.
- With RInside we now skip a step:
  - collect data in a vector or matrix
  - pass data to R — easy thanks to Rcpp wrappers
  - pass one or more short ’scripts’ as strings to R to evaluate
C++ programs compute, gather or aggregate raw data.
Data is saved and analysed before a new ‘run’ is launched.
With RInside we now skip a step:
- collect data in a vector or matrix
- pass data to R — easy thanks to Rcpp wrappers
- pass one or more short ’scripts’ as strings to R to evaluate
- pass data back to C++ programm — easy thanks to Rcpp converters
RInside workflow

- C++ programs compute, gather or aggregate raw data.
- Data is saved and analysed before a new ’run’ is launched.
- With RInside we now skip a step:
  - collect data in a vector or matrix
  - pass data to R — easy thanks to Rcpp wrappers
  - pass one or more short ’scripts’ as strings to R to evaluate
  - pass data back to C++ program — easy thanks to Rcpp converters
  - resume main execution based on new results
RInside workflow

- C++ programs compute, gather or aggregate raw data.
- Data is saved and analysed before a new 'run' is launched.
- With RInside we now skip a step:
  - collect data in a vector or matrix
  - pass data to R — easy thanks to Rcpp wrappers
  - pass one or more short 'scripts' as strings to R to evaluate
  - pass data back to C++ programm — easy thanks to Rcpp converters
  - resume main execution based on new results

- A number of simple examples ship with RInside
RInside workflow

- C++ programs compute, gather or aggregate raw data.
- Data is saved and analysed before a new ’run’ is launched.
- With RInside we now skip a step:
  - collect data in a vector or matrix
  - pass data to R — easy thanks to Rcpp wrappers
  - pass one or more short ’scripts’ as strings to R to evaluate
  - pass data back to C++ programm — easy thanks to Rcpp converters
  - resume main execution based on new results

- A number of simple examples ship with RInside
  - *nine different examples in* examples/standard
RInside workflow

- C++ programs compute, gather or aggregate raw data.
- Data is saved and analysed before a new 'run' is launched.
- With **RInside** we now skip a step:
  - collect data in a vector or matrix
  - pass data to R — easy thanks to **Rcpp** wrappers
  - pass one or more short 'scripts' as strings to R to evaluate
  - pass data back to C++ program — easy thanks to **Rcpp** converters
  - resume main execution based on new results

- A number of simple examples ship with **RInside**
  - *nine* different examples in `examples/standard`
  - *four* different examples in `examples/mpi`

*Dirk Eddelbuettel*  
Seamless R and C++ Integration @ WU Wien, May 2010
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
Quoting from the page at Google Code:

*Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data—think XML, but smaller, faster, and simpler.*
Quoting from the page at Google Code:

*Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data—think XML, but smaller, faster, and simpler.*

You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.
Quoting from the page at Google Code:

*Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data—think XML, but smaller, faster, and simpler.*

You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages.

You can even update your data structure without breaking deployed programs that are compiled against the "old" format.

Google provides native bindings for C++, Java and Python.
Extending R Rcpp

Google ProtoBuf

```r
# load the package
library(RProtoBuf)
# acquire protobuf information
readProtoFiles("addressbook.proto")
# create new object
bob <- new(tutorial.Person,
  email = "bob@example.com",
  name = "Bob",
  id = 123)
# serialize to stdout
writeLines(bob$toString())
# access and/or override
bob$email
bob$id <- 5
# serialize to compact binary format
serialize(bob, "person.pb")
```

Under the hood, Rcpp is used and works very well in conjunction with the rich C++ API provided by Google.
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
Users of Rcpp

- RInside uses Rcpp for object transfer and more
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
- RcppExamples is a 'this is how you can do it’ stanza
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
- RcppExamples is a 'this is how you can do it' stanza
- RProtoBuf is what got Romain and me here, it may get rewritten to take more advantage of Rcpp
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
- RcppExamples is a 'this is how you can do it' stanza
- RProtoBuf is what got Romain and me here, it may get rewritten to take more advantage of Rcpp
- RQuantLib is where Rcpp originally started
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
- RcppExamples is a ’this is how you can do it’ stanza
- RProtoBuf is what got Romain and me here, it may get rewritten to take more advantage of Rcpp
- RQuantLib is where Rcpp originally started
- highlight is Romain’s first re-use of Rcpp
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
- RcppExamples is a 'this is how you can do it' stanza
- RProtoBuf is what got Romain and me here, it may get rewritten to take more advantage of Rcpp
- RQuantLib is where Rcpp originally started
- highlight is Romain’s first re-use of Rcpp
- mvabund, sdcTable, bifactorial, minqa, pcaMethods (BioC), phylobase are truly external users which are all on CRAN
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
- RcppExamples is a 'this is how you can do it' stanza
- RProtoBuf is what got Romain and me here, it may get rewritten to take more advantage of Rcpp
- RQuantLib is where Rcpp originally started
- highlight is Romain’s first re-use of Rcpp
- mvabund, sdcTable, bifactorial, minqa, pcaMethods (BioC), phylobase are truly external users which are all on CRAN
- upcoming: possibly lme4a
Users of Rcpp

- RInside uses Rcpp for object transfer and more
- RcppArmadillo and RcppGSL (which contain fastLm())
- RcppExamples is a 'this is how you can do it' stanza
- RProtoBuf is what got Romain and me here, it may get rewritten to take more advantage of Rcpp
- RQuantLib is where Rcpp originally started
- highlight is Romain's first re-use of Rcpp
- mvabund, sdcTable, bifactorial, minqa, pcaMethods (BioC), phylobase are truly external users which are all on CRAN
- upcoming: possibly lme4a
- Your package here next?
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
This tutorial has tried to show you that
Wrapping up

This tutorial has tried to show you that

- While the deck way be stacked against you (when adding C/C++ to R), you can still pick where to play
Wrapping up

This tutorial has tried to show you that

- While the deck way be stacked against you (when adding C/C++ to R), you can still pick where to play
- R can be extended in many ways; we focus on something that allows us write extensions
This tutorial has tried to show you that

- While the deck way be stacked against you (when adding C/C++ to R), you can still pick where to play
- R can be extended in many ways; we focus on something that allows us write extensions
  - that are efficient: we want speed and features
Wrapping up

This tutorial has tried to show you that

- While the deck way be stacked against you (when adding C/C++ to R), you can still pick where to play
- R can be extended in many ways; we focus on something that allows us write extensions
  - that are efficient: we want speed and features
  - that correspond to the R object model
This tutorial has tried to show you that

- While the deck way be stacked against you (when adding C/C++ to R), you can still pick where to play
- R can be extended in many ways; we focus on something that allows us write extensions
  - that are efficient: we want speed and features
  - that correspond to the R object model
  - that also allow us to embed R inside C++
Wrapping up

This tutorial has tried to show you that

- While the deck way be stacked against you (when adding C/C++ to R), you can still pick where to play
- R can be extended in many ways; we focus on something that allows us write extensions
  - that are efficient: we want speed and features
  - that correspond to the R object model
  - that also allow us to embed R inside C++
- And all this while retaining ’high-level’ STL-alike semantics, templates and other goodies in C++
Wrapping up

This tutorial has tried to show you that

- While the deck way be stacked against you (when adding C/C++ to R), you can still pick where to play
- R can be extended in many ways; we focus on something that allows us write extensions
  - that are efficient: we want speed and features
  - that correspond to the R object model
  - that also allow us to embed R inside C++
- And all this while retaining ’high-level’ STL-alike semantics, templates and other goodies in C++
- Using C++ abstractions wisely can keep the code both clean and readable – yet very efficient
Outline

1. Extending R
   - Why?
   - The standard API
   - Inline

2. Rcpp
   - Overview
   - New API
   - Examples
Some pointers

http://dirk.eddelbuettel.com/code/rcpp.html
Some pointers

Some pointers

Some pointers

- http://cran.r-project.org/package=Rcpp
Some pointers

- [Link 1](http://dirk.eddelbuettel.com/code/rcpp.html)
- [Link 2](http://dirk.eddelbuettel.com/code/rcppTut/)
- [Link 3](http://romainfrancois.blog.free.fr/index.php?category/R-package/Rcpp)
- [Link 4](http://cran.r-project.org/package=Rcpp)
- [Link 5](http://r-forge.r-project.org/projects/rcpp/)

Dirk Eddelbuettel
Seamless R and C++ Integration @ WU Wien, May 2010
Some pointers

- [http://cran.r-project.org/package=Rcpp](http://cran.r-project.org/package=Rcpp)
- [http://r-forge.r-project.org/projects/rcpp/](http://r-forge.r-project.org/projects/rcpp/)
- and likewise for RInside, RProtoBuf and more.
The end

Thank you!