OPTIMIZING R CODE WITH RCPP

C++ functions belong to C++ files

Romain François
Consulting Datactive, ThinkR

Previously, in this course

evalCpp()

```
evalCpp( "40 + 2" )

42
```

cppFunction()

```
cppFunction( "int fun(){ return 42; }")

fun()

42
```

Using .cpp files

C++ code in `code.cpp`

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

// [[Rcpp::export]]
int timesTwo( int x ){
  return 2*x ;
}
```

The `sourceCpp()` function compiles and loads it

```
library(Rcpp)

sourceCpp( "code.cpp" )
timesTwo( 21 )

42
```

Include the Rcpp header file

```
#include <Rcpp.h>
_____ _________ ____ _

__ ________________
___ __________ ___ _ __
  _______ ___ _
_
```

- Include **only** `Rcpp.h`
    - It includes all other header files automatically

Using the Rcpp namespace

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


__ ________________
___ __________ ___ _ __
  _______ ___ _
_
```

- Use `Something` instead of `Rcpp::Something`, when `Something` is in `Rcpp`

Exporting the function to R

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

// [[Rcpp::export]]

___ __________ ___ _ __
  _______ ___ _
_
```

The function itself

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

// [[Rcpp::export]]
int timesTwo( int x ){
  return 2*x ;
}
```

source the C++ file

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

// [[Rcpp::export]]
int timesTwo( int x ){
  return 2*x ;
}
```

load the function into R

```
library(Rcpp)

sourceCpp( "code.cpp" )
```

Call it, just as any other R function.

```
timesTwo( 21 )

42
```

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

Writing functions in C++

Romain François
Consulting Datactive, ThinkR

Just don't export internal functions

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

int twice( int x ){
  return 2*x ;
}

// [[Rcpp::export]]
int universal(){
  return twice(21) ;
}
```

Calling from R:

```
# Not possible, twice is internal
twice(21)
Error in twice(21) : could not find function "twice"

# Fine
universal()
42
```

C++ comments

Comment until the end of the line:

```
// A comment
```

Comments spanning multiple lines

R code special comment

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

int twice( int x ){
  return 2*x ;
}

// [[Rcpp::export]]
int universal(){
  return twice(21) ;
}

/*** R
  # This is R code
  12 + 30

  # Calling the `universal` function
  universal()
*/
```

if and else

```
if( condition ){
  // code if true
} else {
  // code otherwise
}
```

if/else example

```
// [[Rcpp::export]]
void info( double x){
    if( x < 0 ){
        Rprintf( "x is negative" ) ;
    } else if( x == 0 ){
        Rprintf( "x is zero" ) ;
    } else if( x > 0 ){
        Rprintf( "x is positive" ) ;
    } else {
        Rprintf( "x is not a number" ) ;
    }
}
```

Calling the function with various arguments:

```
info(-2)                    info(0)

x is negative               x is zero

info(3)                     info(NaN)

x is positive               x is not a number
```

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

For loops

Romain François
Consulting Datactive, ThinkR

The 4 parts of C++ for loops

- Initialization
- Continue condition
- Increment
- Body

For loops - the initialization

What happens at the very beginning of the loop:

```
for( init ;   ;   ){

}
```

For loops - the continue condition

Logical condition to control if the loop continues

```
for(   ; condition ;   ){

}
```

For loops - the increment

Executed at the end of each iteration

```
for(   ;   ; increment ){

}
```

For loops - the body

Executed at each iteration. What the loop does.

```
for(   ;   ;   ){
    body
}
```

Typical for loop

```
for (int i=0; i<n; i++ ){
    // some code using i
}
```

Typical for loop

```
for (int i=0; ; ){

}
```

Typical for loop

```
for (int i=0; i<n; ){

}
```

Typical for loop

```
for (int i=0; i<n; i++){

}
```

Example: sum of n first integers

```
// [[Rcpp::export]]
int nfirst( int n ){
    if( n < 0 ) {
        stop( "n must be positive, I see n=%d", n ) ;
    }

    int result = 0 ;
    for( int i=0; i<n; i++){
        result = result + (i+1) ;
    }

    return result ;
}
```

Breaking out of a for loop

```
// [[Rcpp::export]]
int nfirst( int n ){
    if( n < 0 ) {
        stop( "n must be positive, I see n=%d", n ) ;
    }

    int result = 0 ;
    for( int i=0; i<n; i++){
        if( i == 13 ){
            Rprintf( "I cannot handle that, I am superstitious" ) ;
            break ;
        }
        result = result + (i+1) ;
    }

    return result ;
}
```

Newton iterative method to calculate square roots

Finding $\sqrt{S}$ is the same as finding the root of $f(x) = x^2 - S$

Leading to the iterative expression:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - S}{2x_n} = \frac{1}{2}(x_n + \frac{S}{x_n})$$

Algorithm:

- Take an initial value x_0
- Update x using the formula above a given number of times

Newton's method in C++

$$x_{n+1} = \frac{1}{2}(x_n + \frac{S}{x_n})$$

translates to the pseudo code

```
int n = ... // number of iterations

double res = ... // initialization

for( int i=0; i<n; i++){

    // update the value of res
    // i.e. calculate x_{n+1} given x_{n}
    res = ( res + S / res ) / 2.0 ;

}

return res ;
```

OPTIMIZING R CODE WITH RCPP

Let's practice!

OPTIMIZING R CODE WITH RCPP

While loops

Romain François
Consulting Datactive, ThinkR

While loops are simpler

```
while( condition ){
    body
}
```

- Continue condition
- Loop body

Example

```
// [[Rcpp::export]]
int power( int n ){

    if( n < 0 ){
        stop( "n must be positive" )  ;
    }

    int value = 1 ;
    while( value < n ){
        value = value * 2 ;
    }
    return value ;
}
```

Once the function is compiled with `sourceCpp`, you can call it:

```
power( 1000 )
1024

power( 17 )
32
```

For loops are just while loops

```
for( init ; condition; increment ){
    body
}
```

is equivalent to

```
init
while( condition ){
    body
    increment
}
```

do / while loops

```
do {
    body
} while( condition ) ;
```

Example of a do / while loop

```
// [[Rcpp::export]]
int power( int n ){

    if( n < 0 ){
        stop( "n must be positive" )  ;
    }

    int value = 1 ;
    do {
        value = value * 2 ;
    } while( value < n );
    return value ;
}
```

OPTIMIZING R CODE WITH RCPP

Let's practice!