Advanced R and C

Ed Davis

18/02/2021

Rcpp sugar

`Rcpp sugar` allows for simplification of `C++` code such that it is more similar to the syntax of `R`, while often maintaining a boost in computational speed.

Binary Arithmetic Operators

In the following, we see that writing a function to add vectors in standard `Rcpp` involves a lot more code in comparison to one written exploiting `Rcpp sugar`. We see that in the latter, the `+` operator has been overloaded such that variables of type `Rcpp::NumericVector` can be added.

```
# Example: Standard Rcpp

library(Rcpp)
sourceCpp(code = '
#include <Rcpp.h>

// [[Rcpp::export(name = "vsum")]]
Rcpp::NumericVector vsum_I(const Rcpp::NumericVector x1, const Rcpp::NumericVector x2)
{
  int ni = x1.size();
  Rcpp::NumericVector out(ni);

  for(int ii = 0; ii < ni; ii++){
    out[ii] = x1[ii] + x2[ii];
  }

  return out;
}')

vsum(c(1,2), c(3,4))
```

```
## [1] 4 6
```

```
# Example: Rcpp sugar

sourceCpp(code = '
#include <Rcpp.h>

// [[Rcpp::export(name = "vsumVett")]]
Rcpp::NumericVector vsum_I(const Rcpp::NumericVector x1, const Rcpp::NumericVector x2)
{
  return x1 + x2;
}')

vsumVett(c(1,2), c(3,4))
```

```
## [1] 4 6
```

Due to `Rcpp sugar`, the arithmetic operators `+`, `-`, `*`, `/` are each overloaded for `Rcpp::NumericVector` vector-vector or vector-scalar combinations. However, as the arithmetic operators in `R` are written with `C` or `C++`, in this simple example, `R` will be the quickest.

```
d <- 1e2
x1 <- rnorm(d)
x2 <- rnorm(d)

library(microbenchmark)
microbenchmark(R = x1 + x2, Rcpp = vsum(x1, x2), RcppSugar = vsumVett(x1, x2))
```

```
## Unit: nanoseconds
##       expr  min   lq  mean median   uq    max neval
##          R  200  300   403    300  400   6700   100
##       Rcpp 1300 1500 10426   1600 1800 876100   100
##  RcppSugar 1300 1500  7711   1600 1700 608000   100
```

Binary Logical Operators

Similar to the arithmetic operators, `Rcpp sugar` also has overloads of the logical operators, `<`, `>`, `=`, such that a `logical` sugar expression can be created from comparisons between two sugar expressions or a sugar expression with a primitive value of acceptable type.

```
# Example: < operator
sourceCpp(code = '
  #include <Rcpp.h>

  // [[Rcpp::export(name = "is_positive")]]
  Rcpp::LogicalVector is_positive_I(Rcpp::NumericVector num)
  {
    return num > 0;
  }
')

is_positive(5)
```

```
## [1] TRUE
```

```
is_positive(-5)
```

```
## [1] FALSE
```

```
# Example: = operator
sourceCpp(code = '
  #include <Rcpp.h>

  // [[Rcpp::export(name = "is_equal")]]
  Rcpp::LogicalVector is_equal_I(Rcpp::NumericVector num1, Rcpp::NumericVector num2)
  {
  return num1 == num2;
  }
')

is_equal(5, 5)
```

```
## [1] TRUE
```

```
is_equal(5,6)
```

```
## [1] FALSE
```

Rcpp versions of R functions

Operations, functions and distributions that are found in `R` can also be accessed through `Rcpp sugar`, with a full list of the functions here.

RcppArmadillo