Rcpp part II: Sugar and RcppArmadillo

Carlos Trucíos Maza

Universidade Estadual de Campinas
R Campinas User Group

30 de setembro, 2015

Rcpp: Sugar

Rcpp Sugar is bases on expression templates and provides some syntactic sugar facilities direclty in Rcpp

```
suppressMessages(library(Rcpp))
```

```
## Warning: package 'Rcpp' was built under R version 3.1.3
```

```
suppressMessages(library(RcppArmadillo))
```

```
## Warning: package 'RcppArmadillo' was built under R versi
```

```
suppressMessages(library(inline))
suppressMessages(library(rbenchmark))
suppressMessages(library(microbenchmark))
```

Sugar: Motivation

```
// [[Rcpp::export]]
SEXP foo (SEXP xs, SEXP ys) {
    Rcpp::NumericVector xv(xs), yv(ys);
    int n= xv.size();
    Rcpp::NumericVector res(n);
    for (int i=0; i<n; i++){
        double x=xv[i];
        double y=yv[i];
        if (x<y){
            res[i]=x*x;
        } else{
            res[i]=-(y*y);
        }
    }
    return res;
}
```

Sugar: Motivation

```
foo_R = function(x,y){
  ifelse(x<y, x*x, -(y*y))
}
```

```
// [[Rcpp::export]]
SEXP foo_Sugar (SEXP xs, SEXP ys) {
  Rcpp::NumericVector x(xs), y(ys);
  return Rcpp::wrap(ifelse (x<y, x*x, -(y*y)));
}
```

Sugar: Motivation

```
x = rnorm(10); y = rnorm(10)
microbenchmark(foo(x,y),foo_Sugar(x,y), foo_R(x,y))
```

```
## Unit: microseconds
##              expr    min      lq    mean median      uq     m
##        foo(x, y)  1.832  3.478  5.350  4.157  6.141  46.
##  foo_Sugar(x, y)  2.020  3.404  5.259  4.165  6.357  34.
##      foo_R(x, y) 12.008 19.643 23.533 21.389 24.758 104.
```

```
c(sum(foo(x,y)),sum(foo_Sugar(x,y)),sum(foo_R(x,y)))
```

```
## [1] 0.3899 0.3899 0.3899
```

Sugar: Operators

- Aritmetic operators: +, -. *, /
- Logical operatorns: <, > <=, >=, ==, !=
- Unary operatorns: "!" negates a logical sugar expression

Sugar: Operators

```
x = c(8,9,7,6)
y = c(1,7,2,3)
# Rcpp Code.
// [[Rcpp::export]]
SEXP Sugar_Ex (NumericVector x, NumericVector y){
  NumericVector soma = x + y, res = x - y ;
  NumericVector prod = x * y, div = x / y;
  LogicalVector menor = x<y,  maior = x>y;
  LogicalVector igual = x==y, dif = x!=y;

  return Rcpp::List::create(soma,div,menor,dif);
}
```

Sugar: Operators

```
Sugar_Ex(x,y)
```

```
## [[1]]
## [1]  9 16  9  9
##
## [[2]]
## [1] 8.000 1.286 3.500 2.000
##
## [[3]]
## [1] FALSE FALSE FALSE FALSE
##
## [[4]]
## [1] TRUE TRUE TRUE TRUE
```

Sugar: Functions

```
// [[Rcpp::export]]
SEXP Ex_1(IntegerVector y){
  int n = y.size();
  IntegerVector x = seq_len(n);
  return any(x*y<5);
}
```

```
Ex_1(c(3,4,5,1,2,6))
```

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

Sugar: Functions

```
// [[Rcpp::export]]
SEXP Ex_2(IntegerVector y){
  int n = y.size();
  IntegerVector x = seq_len(n);
  return all(x*y<5);
}
```

```
Ex_2(c(3,4,5,1,2,6))
```

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

Sugar: Functions

```
// [[Rcpp::export]]
SEXP Ex_3(IntegerVector y){
  int n = y.size();
  IntegerVector x = seq_len(n);
  bool B1 = is_false(all(x*y<5));
  bool B2 = is_true(any(x*y<5));
  bool B3 = is_na(any(x*y<2));
  return Rcpp::List::create(B1,B2,B3);
}
```

Sugar: Functions

```
Ex_3(c(3,4,5,1,2,6))
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] FALSE
```

Sugar: Functions

```
// [[Rcpp::export]]
SEXP Ex_4(IntegerVector y){
  int n = y.size();
  IntegerVector x = seq_along(y);
  IntegerVector z = seq_len(n);
  IntegerVector min = pmin(z,z*z);
  IntegerVector max = pmax(z,3);
  return Rcpp::List::create(x,z,min,max);
}
```

Sugar: Functions

```
Ex_4(c(3,4,5,1,2,6))
```

```
## [[1]]
## [1] 1 2 3 4 5 6
##
## [[2]]
## [1] 1 2 3 4 5 6
##
## [[3]]
## [1] 1 2 3 4 5 6
##
## [[4]]
## [1] 3 3 3 4 5 6
```

Sugar: Functions

```
// [[Rcpp::export]]
SEXP Ex_5(IntegerVector y){
  int n = y.size();
  IntegerVector x=y+2;
  IntegerVector SIG = sign(y);
  IntegerVector DIF = diff(y);
  IntegerVector DIF2 = setdiff(y,x);
  return Rcpp::List::create(SIG,DIF,DIF2,x);
}
```

Sugar: Functions

```
Ex_5(c(-3,4,0,1))
```

```
## [[1]]
## [1] -1  1  0  1
##
## [[2]]
## [1]  7 -4  1
##
## [[3]]
## [1] -3  0  1  4
##
## [[4]]
## [1] -1  6  2  3
```

Sugar: Functions

```
// [[Rcpp::export]]
SEXP Ex_6(IntegerVector y, IntegerVector x){
IntegerVector UNI = union_(x,y);
IntegerVector INTER = intersect(x,y);
return Rcpp::List::create(UNI,INTER);
}
```

Sugar: Functions

```
y = c(-3,4,0,1)
x = c(5,6,0,1)
Ex_6(y,x)
```

```
## [[1]]
## [1] -3  0  1  4  5  6
##
## [[2]]
## [1] 0 1
```

Sugar: Functions

```
atan(x);
gamma(x);
lgamma(x);  # log gamma
digamma(x);
trigamma(x);
tetragamma(x);
pentagamma(x);
expm1(x);
log1p(x);
factorial(x);
lfactorial(x);
choose(n, k);
lchoose(n, k);
beta(n, k);
lbeta(n, k);
psigamma(n, k);
trunc(x);
round(x, k);
signif(x, k);
mean(x);
var(x);
sd(x);
```

Sugar: Probability

beta, binom, cauchy, chisq, exp, F, gamma, hyper, lnorm, logis, norm, t, pois, unif and weibull.

- dnorm() density
- qnorm() quantile
- pnorm() probability
- rnorm() random

Sugar: Probability

```
// [[Rcpp::export]]
SEXP  Ex_gamma(int n){
  RNGScope scope;
  NumericVector x = rgamma(n,1,1);
  return x;
}
```

Sugar: Probability

```
Ex_gamma(5)
```

```
## [1] 0.3250 0.9629 0.3029 2.1092 0.4427
```

Sugar: Probability

```
// [[Rcpp::export]]
SEXP  Ex_norm(int n){
  RNGScope scope;
  NumericVector x = rnorm(n,0,1);
  return x;
}
```

Sugar: Probability

```
Ex_norm(5)
```

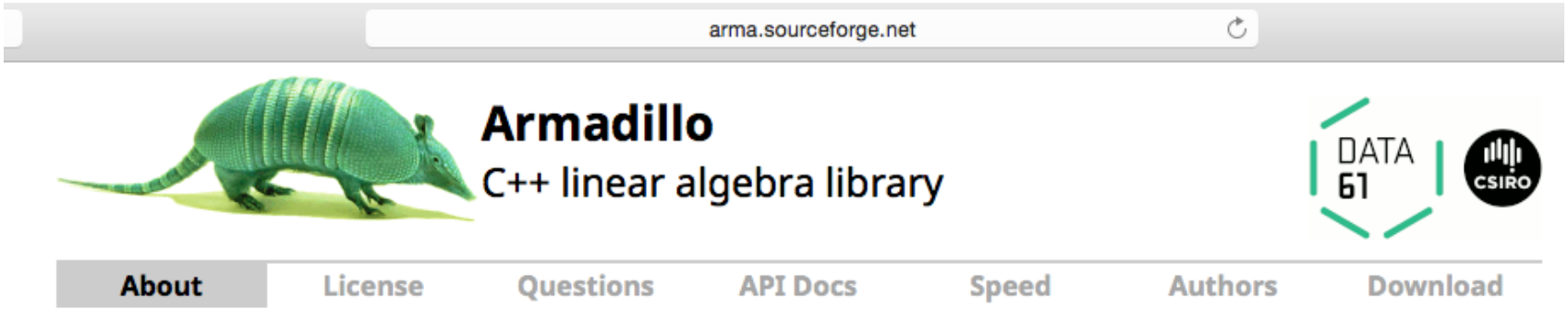
```
## [1] -0.34876  0.31335 -0.09215  0.02970  1.28520
```

```
benchmark(Ex_norm(1000),rnorm(1000),
          columns = c("test", "replications", "relative"))
```

```
##           test replications relative
## 1 Ex_norm(1000)          100    1.000
## 2   rnorm(1000)          100    1.125
```

Rcpp: RcppArmadillo

- RcppArmadillo provides an interface from R to and from Armadillo by utilising the Rcpp R/C++ interface library.
- Various matrix decompositions are provided through optional integration with LAPACK and ATLAS libraries.
- More information about Armadillo is available at `http://arma.sourceforge.net`



- Armadillo is a C++ library with a focus on linear algebra and related topics.

RcppArmadillo: Motivation

Why Linear Algebra?

Linear algebra is very useful in statistical computations, for instance:

- Multiplication
- Inversion
- Descomposition
- Transpose
- etc.

Statistical routines are commonly implemented using linerar algebra.

RcppArmadillo: Motivation

```
e = rnorm(10000)
x1 = rnorm(10000,14,2)
x2 = rt(10000,7)
y = 0.1+0.8*x1 + 0.65*x2+e
dados = data.frame(y,x1,x2,e)
microbenchmark("lm" = lm(y~x1+x2,data=dados),
               "fastLm" = fastLm(y~x1+x2,data=dados))
```

```
## Unit: milliseconds
##     expr   min    lq  mean median    uq   max neval
##       lm 7.050 7.542 10.31  8.610 8.886 47.16   100
##  fastLm 7.194 8.427 10.25  8.737 9.217 48.90   100
```

RcppArmadillo: Define an object

- mat (double)
- umat (unsigned integer)
- imat (signed integer)
- fmat (float)
- vec
- uvec
- ivec
- fvec

RcppArmadillo: Functions

Armadillo function	Description
`X(1,2) = 3`	Assign value 3 to element at location (1,2) of matrix X
`X = A + B`	Add matrices A and B
`X( span(1,2), span(3,4) )`	Provide read/write access to submatrix of X
`zeros(rows [, cols [, slices]])`	Generate vector (or matrix or cube) of zeros
`ones(rows [, cols [, slices]])`	Generate vector (or matrix or cube) of ones
`eye(rows, cols)`	Matrix diagonal set to 1, off-diagonal elements set to 0
`repmat(X, row_copies, col_copies)`	Replicate matrix X in block-like manner
`det(X)`	Returns the determinant of matrix X
`norm(X, p)`	Compute the p-norm of matrix or vector X
`rank(X)`	Compute the rank of matrix X
`min(X, dim=0); max(X, dim=0)`	Extremum value of each column of X (row if `dim=1`)
`trans(X)` or `X.t()`	Return transpose of X
`R = chol(X)`	Cholesky decomposition of X such that $R^T R = X$
`inv(X)` or `X.i()`	Returns the inverse of square matrix X
`pinv(X)`	Returns the pseudo-inverse of matrix X
`lu(L, U, P, X)`	LU decomp. with partial pivoting; also `lu(L, U, X)`
`qr(Q, R, X)`	QR decomp. into orthogonal Q and right-triangular R
`X = solve(A, B)`	Solve system $AX = B$ for X
`s = svd(X); svd(U, s, V, X)`	Singular-value decomposition of X

Table 1: Selected Armadillo functions with brief descriptions; see `http://arma.sf.net/docs.html` for more complete documentation. Several optional additional arguments have been omitted here for brevity.

RcppArmadillo: Functions

Some other examples of functions using RcppArmadillo.

- arma::colvec
- arma::mat
- arma::vec
- arma::diagvec
- arma::pinv
- arma::trans
- arma::solve
- arma::sqrt
- matrix.t()
- matrix.eye(m,n)
- matrix.zeros(m,n)

Rcpp: Example

```
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::export]]
arma::mat M1 (NumericMatrix x,NumericVector y) {
  arma::mat x_ = as<arma::mat>(x);
  arma::vec y_ = as<arma::vec>(y);
  arma::mat z = inv(x_) + y_*y_.t();
  return(arma::diagvec(z)) ;
}
```

Rcpp: Example

```
M1(matrix(c(1,3,2,4),2,2), c(2,3))
```

```
##      [,1]
## [1,]  2.0
## [2,]  8.5
```

Rcpp: Example

```
// [[Rcpp::depends(RcppArmadillo)]]
// [[Rcpp::export]]
SEXP M2 (arma::mat x) {
  arma::mat z = chol(x);
  int n = x.n_rows;
  int m = x.n_cols;
  arma::mat ONE = arma::ones(m,n);
  arma::mat y = z+ONE;
  return (wrap(arma::det(y)));
}
```

```
M2(matrix(c(3.7,3.4,2.1,4),2,2))
```

```
## [1] 5.731
```

Rcpp Part II