Topic 11: Rcpp and Armadillo Library

Irina Gaynanova

Linear Algebra via RcppArmadillo

What is Armadillo?

In Spanish, “little armored one”

Linear Algebra via RcppArmadillo

What is Armadillo in C++ context?

A very powerful C++ library for linear algebra

Armadillo clickable reference

What this means in lay-man terms: Has its own vector/matrix classes (different from Rcpp NumericVector, NumericMatrix) that are highly optimized for various linear algebra operations (package functions that work on these classes)

Another powerful library (we will not use): Eigen with corresponding RcppEigen for R integration

HW 5 - Lasso algorithm in C++

- Use Armadillo matrix and vector classes
- Should see amazing speed improvements from moving the loops into C++
- **Purpose** - be able to search on your own examples in C++ and Armadillo library to figure out how to implement what you need; there is more than one way to do it

Linear Algebra via RcppArmadillo

To be able to use it from R, need to have RcppArmadillo package

```
install.packages("RcppArmadillo")
```

When writing C++ code, this will require the use of different header, i.e.

```
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
using namespace Rcpp; // can be omitted for most cases
```

The first include will automatically do <Rcpp.h> as well. The second line is needed for sourceCpp to work (a short answer, a longer answer here). The third line is needed for Rcpp specific types if used (e.g. List instead of Rcpp::List)

Matrix algebra with Rcpp Armadillo - matrix multiplication

Example matrix multiplication using Armadillo library

```
// [[Rcpp::export]]
arma::mat matrix_mult(const arma::mat& X,
                      const arma::mat& Y) {
  int m = X.n_rows;
  int n = Y.n_cols;
  arma::mat Z(m,n);
  Z = X * Y;
  return Z;
}
```

- **const** - prevents direct modifications (more on this later)
- **arma::mat** - matrix class within Armadillo library (class **mat** within namespace **arma**)
- **& X** - uses pointer rather than copying the whole matrix (more on this later)
- **.n_rows** and **.n_cols** allow to get dimensions
- Overloaded * instead of %*%

Matrix algebra with Rcpp Armadillo - matrix multiplication

```
// [[Rcpp::export]]
arma::mat matrix_mult(const arma::mat& X,
                      const arma::mat& Y) {
  int m = X.n_rows;
  int n = Y.n_cols;
  arma::mat Z(m,n);
  Z = X * Y;
  return Z;
}
```

Question: Writing arma::mat every time is annoying, can I just do

```
use namespace arma;
```

Answer: Yes, but it would not work (*easily*) in R package, and putting arma:: explicit will ensure you always know which functions/classes are from Armadillo and which ones are just plain C++

Matrix multiplication

```
library(Rcpp)
library(RcppArmadillo)
sourceCpp("ArmadilloExamples.cpp")

X = matrix(rnorm(300), 30, 10)
Y = matrix(rnorm(200), 10, 20)
prodCpp = matrix_mult(X, Y)
prodR = X%*%Y
all.equal(prodCpp, prodR)
```

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

Matrix multiplication

```
library(Rcpp)
library(RcppArmadillo)
sourceCpp("ArmadilloExamples.cpp")

X = matrix(rnorm(30000), 300, 100)
Y = matrix(rnorm(20000), 100, 200)
library(microbenchmark)
microbenchmark(
  matrix_mult(X, Y),
  X%*%Y
)
```

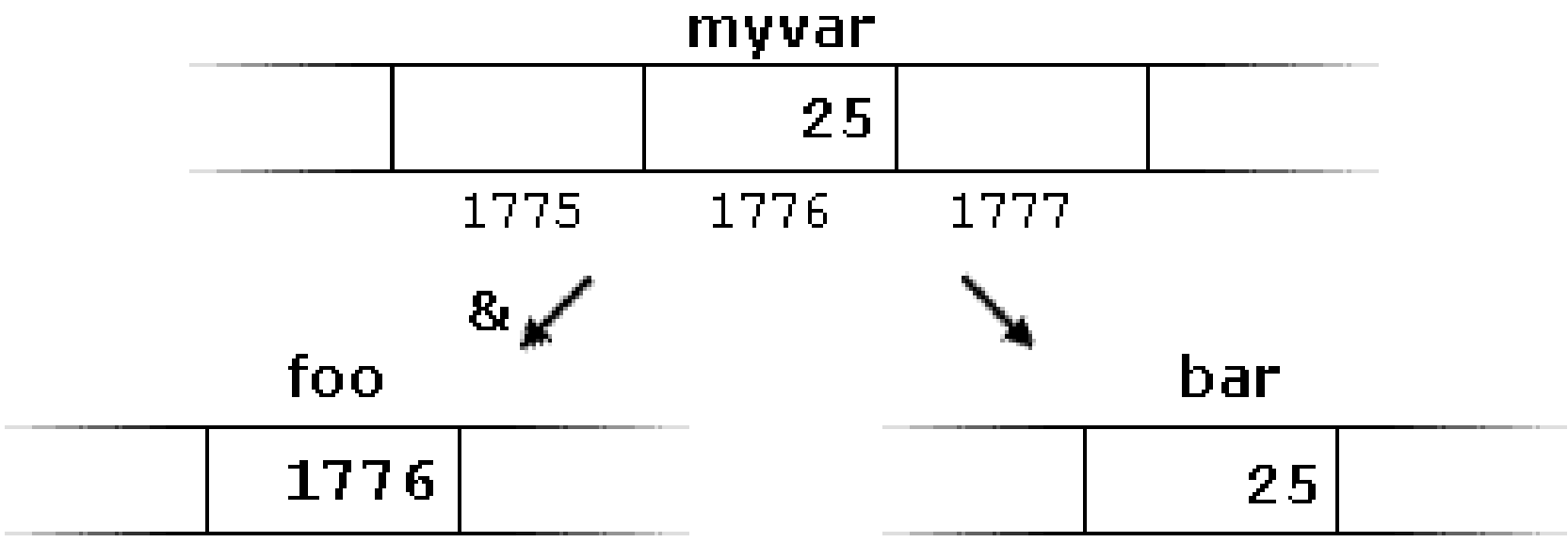
```
## Warning in microbenchmark(matrix_mult(X, Y), X %*% Y): I
## times to avoid potential integer overflows
```

```
## Unit: milliseconds
##                 expr      min       lq     mean   median
##  matrix_mult(X, Y) 2.039258 2.077675 2.133300 2.099610 2
##             X %*% Y 2.032985 2.046761 2.077366 2.070234 2
```

More on pointers, addresses and passes by value

- &X passes the address of X rather than X itself
- Simple example

```
myvar = 25;
foo = &myvar;
bar = mayvar;
```



const, addresses and pointers & versus direct passing of a vector

- Unlike **NumericVector** in Rcpp, the vector and matrix types in Armadillo are by default passed by value unless you use &
- **Pointers_Armadillo.R** illustrates the difference in behavior
- **const** will make the compiler complain if you try to directly modify the corresponding variable in the code

More on pointers, addresses and passes by value

- **For the purpose of this class,** we will use & to pass matrices and vectors to functions by address rather than by value. We will still be able to access all the elements in the usual way (more examples to come)
- For those familiar with C/C++: We will not use the **derefence operator** * in class or for assignments, but am happy to show some examples with plain C from R that uses it towards the end of the semester if there is an interest
- For more on pointers and dereference operator

Linear model fit in Armadillo

- **ArmadilloExamples.cpp** contains linear model example

```
Rcpp::List fastLm(const arma::mat& X,
                  const arma::colvec& y) {
  # Return
  return Rcpp::List::create(
    Rcpp::Named("coefficients") = coef,
    Rcpp::Named("stderr") = std_err,
    Rcpp::Named("df.residual") = n - p );
}
```

- Both X and y are passed by address with **const** (no changes)
- In Armadillo, can specify vectors directly as **colvec** or **rowvec** or just **vec** (useful to keep track of dimensions)
- To return a list, use Rcpp List class. The R equivalent is

```
return(list(coefficients = coef, stderr = std_err,
            df.residual = n - p))
```

Linear model fit in Armadillo

```
// Fit model y ~ X
  arma::colvec coef = arma::solve(X, y);
```

- **arma::solve(X, y)** - calculates $(X^\top X)^{-1} X^\top Y$

Linear model fit in Armadillo

```
// Compute the residuals
arma::colvec res  = y - X * coef;
// Estimated variance of the random error
double s2 = std::inner_product(res.begin(), res.end(),
                               res.begin(), 0.0) / (n - p);
```

- **std::innerproduct** - inner product function within std namespace (standard library in C++), 4 arguments: beginning and end of 1st vector, beginning of 2nd vector, initial value. [Not the only way to compute inner product]
- **.begin()** - iterator pointing to the 1st element of the vector
- **.end()** - iterator pointing to the last element of the vector

std:: - function from standard C++ library

- std:: ensures that you don't have conflict with some other function with the same name
- std is actually a collection of many libraries that are considered **standard**
- a full list of std libraries, useful for searching functions that you need
- Particularly useful ones
- cmath for absolute values, trigonometric functions, power functions, square root, etc. (all on scalars)

-vector to work with arrays (this is where **.begin()** and **.end()** live)

-random for various random number generators

-numeric numeric operations on values in ranges (this is where **inner_product** lives)

Words of caution

The function with the same name from a different library may have unexpected behavior

```
// designed for integers x only,
// will work incorrectly on 3.2
std::abs(x)
// designed for floating point types x,
// so will work correctly on any real number
std::fabs(x)
// designed for vectors/matrices from armadillo library,
// will work correctly on vectors/matrices x that
// are integers/floating point types
arma::abs(x)
```

Linear model fit in Armadillo

```
// Standard error matrix of coefficients
  arma::colvec std_err =
    arma::sqrt(s2 * arma::diagvec(arma::pinv(X.t()*X)));
```

- **X.t()** - transpose of matrix X
- **pinv** - pseudo-inverse (generalized inverse based on SVD, just inverse if full rank)
- **diagvec** - takes diagonal vector

Linear model fit in Armadillo

```
// Standard error matrix of coefficients
  arma::colvec std_err =
    arma::sqrt(s2 * arma::diagvec(arma::pinv(X.t()*X)));
```

This is calculating st.dev for each $\widehat{\beta}_j$ since

$$\begin{aligned}\widehat{\beta} &= (X^\top X)^{-1} X^\top y = (X^\top X)^{-1} X^\top X \beta^* + (X^\top X)^{-1} X^\top \varepsilon \\ &= \beta^* + (X^\top X)^{-1} X^\top \varepsilon\end{aligned}$$

$$\begin{aligned}Cov(\widehat{\beta}) &= Cov((X^\top X)^{-1} X^\top \varepsilon) = \sigma^2 (X^\top X)^{-1} X^\top X (X^\top X)^{-1} \\ &= \sigma^2 (X^\top X)^{-1}\end{aligned}$$

Linear model fit in Armadillo

```
sourceCpp("ArmadilloExamples.cpp")
set.seed(20386)
X = matrix(rnorm(100), 25, 4)
beta = rep(1, 4)
Y = X %*% beta + rnorm(25, sd = 0.5)
outC = fastLm(X, Y); names(outC)
```

```
## [1] "coefficients" "stderr"       "df.residual"
```

```
cbind(outC$coefficients,
      solve(crossprod(X), crossprod(X, Y)))
```

```
##           [,1]      [,2]
## [1,] 0.9827557 0.9827557
## [2,] 1.1114222 1.1114222
## [3,] 0.8683201 0.8683201
## [4,] 1.0017851 1.0017851
```

RcppArmadillo - conversion examples

R syntax	Armadillo syntax
X[i,j]	X(i-1,j-1) (indexing starts from 0)
X[, j]	X.col(j-1)
X[i,]	X.row(i-1)
t(X)	X.t()
nrow(X)	X.n_rows
ncol(X)	X.n_cols

RcppArmadillo - conversion examples

R syntax	Armadillo syntax
X %*% Y	X * Y (matrix multiplication)
X * Y	X % Y (element-wise multiplication)
solve(X)	inv(X) (for square X)
solve(X, Y)	solve(X, Y)
as.vector(X)	vectorize(X)

RcppArmadillo - vector norms

```
sourceCpp("ArmadilloExamples.cpp"); Y = rnorm(100)
normArmaV(Y, 2)
```

```
## [1] 9.544771
```

```
sqrt(sum(Y^2))
```

```
## [1] 9.544771
```

```
normArmaV(Y, 1)
```

```
## [1] 77.15552
```

```
sum(abs(Y))
```

```
## [1] 77.15552
```

RcppArmadillo - matrix norms

```
X = matrix(rnorm(300), 30, 10)
normArmaM(X, 2)
```

```
## [1] 8.302613
```

```
svd(X)$d[1]
```

```
## [1] 8.302613
```

```
normArmaM(X, 1)
```

```
## [1] 30.7933
```

```
max(colSums(abs(X)))
```

```
## [1] 30.7933
```

RcppArmadillo - contiguos subsetting

R syntax	Armadillo syntax
X[i,j]	X(i-1,j-1) (indexing starts from 0)
X[, j]	X.col(j-1)
X[i,]	X.row(i-1)
X[i:j,]	X.rows(i, j)
X[, i:j]	X.cols(i, j)
X[i:j, k:m]	X(span(i, j), span(k, m))

See **submatrix views** in Armadillo documentation

RcppArmadillo - noncontiguos subsetting

For vectors

```
X(vector_of_indices)
```

For matrices

```
X.cols(vector_of_column_indices)
X.rows(vector_of_row_indices)
X(vector_of_row_indices, vector_of_column_indices)
```

IMPORTANT: the indexing vectors have to be of type **uvec**

RcppArmadillo - noncontiguos subsetting

In R

```
X <- matrix(rnorm(15), 5, 3)
X[X>1.5]
```

```
## [1] 1.803663
```

In Cpp

```
arma::uvec indexX = arma::find(X > 1.5);
X(indexX);
```

See **find** in Armadillo documentation

RcppArmadillo - noncontiguos subsetting

IN R

```
X <- matrix(rnorm(15), 5, 3)
Y <- rnorm(5)
X[Y>0,]
```

```
## [1] -0.5281037  1.6403530 -0.5753602
```

IN Cpp

```
arma::uvec indexY = arma::find(Y > 0);
X.rows(indexY);
```

RcppArmadillo - procrustes problem

Consider the following minimization problem, where $X \in \mathbb{R}^{n \times p}$, $V \in \mathbb{R}^{p \times r}$ are given, and $U \in \mathbb{R}^{n \times r}$ is the argument

$$\text{minimize}_U \|X - UV^\top\|_F^2 \quad \text{subject to} \quad U^\top U = I.$$

- This is a **constrained** optimization problem with 1 equality constraint
- This is a **non-convex** problem because the constraint is non-convex (more on this later)
- Orthogonal Procrustes problem (commonly arises in matrix decomposition problems)

Procrustes problem

RcppArmadillo - procrustes problem

Orthogonal Procrustes Problem: $X \in \mathbb{R}^{n \times p}$, $V \in \mathbb{R}^{p \times r}$ are given, and $U \in \mathbb{R}^{n \times r}$ is the argument

$$\text{minimize}_U \|X - UV^\top\|_F^2 \quad \text{subject to} \quad U^\top U = I.$$

- Can rewrite objective function as

$$f(U) = -2\text{Trace}\{U^\top (XV)\} + C$$

- Despite non-convexity of the problem, the global solution is known

$$U^* = RQ^\top, \quad \text{where} \quad XV = RDQ^\top (\text{SVD})$$

Procrustes problem in base R

```
procrustesR <- function(X, V){
  svdXV <- svd(X %*% V)
  U <- tcrossprod(svdXV$u, svdXV$v)
  return(U)
}

set.seed(308723)
X <- matrix(rnorm(110), 11, 10)
V <- matrix(rnorm(30), 10, 3)
U <- procrustesR(X, V)
```

Procrustes problem in Rcpp Armadillo

```
library(Rcpp)
library(RcppArmadillo)
sourceCpp("ArmadilloExamples.cpp")

U_Cpp <- procrustes(X, V)
sum(abs(U_Cpp - U))
```

```
## [1] 0
```

Procrustes problem in Rcpp Armadillo

- Dimensions have been automatically determined by svd_econ function (economical SVD)
- Using just svd function here will result in incompatible dimensions
- Need to create s even though it is not used

Sparse PCA via Procrustes Problem

Consider

$$\text{minimize}_{U,V} \left\{ \frac{1}{2}\|X - UV^\top\|_F^2 + \lambda \sum_{j=1}^{p}\sum_{k=1}^{r} |v_{jk}| \right\}$$
$$\text{subject to} \quad U^\top U = I.$$

This problem is one of the variants of **sparse Principal Component Analysis (PCA)**

- When V is fixed, this is **Orthogonal Procrustes Problem** with respect to U
- When U is fixed, this is a **convex** unconstrained problem in V that is very similar to Lasso problem

Sparse PCA via Procrustes Problem

When U is fixed

$$\text{minimize}_V \left\{ \frac{1}{2}\|X - UV^\top\|_F^2 + \lambda \sum_{j=1}^{p}\sum_{k=1}^{r} |v_{jk}| \right\}$$

- Can rewrite objective function as

$$f(V) = -\text{Trace}\{V^\top(X^\top U)\} + \frac{1}{2}\text{Trace}\{V^\top V\} + \lambda \sum_{j=1}^{p}\sum_{k=1}^{r} |v_{jk}| + C.$$

- Optimality conditions with respect to V (S - subgradient)

$$-X^\top U + V + \lambda S = 0$$

- Solution V is element-wise soft-thresholding operator

Sparse PCA via Procrustes in R

- Write the following code to alternate optimization with respect to U and V

```
sparsePCAR <- function(X, Vstart, lambda, tol){
  # Evaluate U for given Vstart, and the value of objectiv

  # While not converged, repeat
    ## Update V via soft-thresholding of X'U

    ## Update U via Procrustes

  # Return a list of U, V and error on solution
  return(list(U = U, V = V, error = error))
}
```

Sparse PCA via Procrustes Problem in R

- How does this work?

```
set.seed(308723)
X <- matrix(rnorm(110), 11, 5)
V <- matrix(rnorm(30), 5, 3)
lambda = 1
eps = 1e-2
outR = sparsePCAR(X, V, lambda, eps)
outR$V
```

```
##             [,1]       [,2]        [,3]
## [1,]  0.000000  0.0000000 -2.28653318
## [2,] -1.512586  0.0000000  0.00000000
## [3,] -1.513491 -0.7032196  0.08888249
## [4,]  0.000000 -2.4774902  0.00000000
## [5,]  2.008046  0.0000000  0.00000000
```

Sparse PCA via Procrustes Problem in C++

```
library(Rcpp); library(RcppArmadillo)
sourceCpp("ArmadilloExamples.cpp")
set.seed(308723)
X <- matrix(rnorm(110), 11, 5)
V <- matrix(rnorm(30), 5, 3)
lambda = 1
eps = 1e-2
out = sparsePCA(X, V, lambda, eps)
out$V
```

```
##             [,1]       [,2]        [,3]
## [1,]  0.000000  0.0000000 -2.28653318
## [2,] -1.512586  0.0000000  0.00000000
## [3,] -1.513491 -0.7032196  0.08888249
## [4,]  0.000000 -2.4774902  0.00000000
## [5,]  2.008046  0.0000000  0.00000000
```

Sparse PCA via Procrustes Problem in C++

```
library(Rcpp)
library(RcppArmadillo)
sourceCpp("ArmadilloExamples.cpp")
set.seed(308723)
X <- matrix(rnorm(110), 11, 5)
V <- matrix(rnorm(30), 5, 3)
lambda = 1
eps = 1e-2
library(microbenchmark)
microbenchmark(
  sparsePCA(X, V, lambda, eps),
  sparsePCAR(X, V, lambda, eps)
)
```

```
## Unit: microseconds
##                                expr     min       lq      mean
##   sparsePCA(X, V, lambda, eps)  33.128  33.661  39.67734
##  sparsePCAR(X, V, lambda, eps) 233.618 237.390 243.03898
##       max neval cld
```

Rcpp and RcppArmadillo - summary

- Do not try to memorize all C/C++ classes/commands - rather learn how to search for what you need and how to **learn from examples you can find**
- Some good references are in the beginning of the slides
- Another good source is **Armadillo library** and **gallery of Rcpp examples**