Uma Introdução ao Rcpp

5º Encontro da Pós-Graduação em Estatística do IME - USP

José Augusto Sartori Junior

Instituto de Matemática e Estatística
Universidade de São Paulo

19 de Novembro de 2021

Overview

1. Introdução e C++

2. Estruturas de Dados e Exemplo

Introdução

R é uma linguagem maravilhosa para lidar com dados:

```
mydata <- read.table()
fit <- lm(y ~ x, data = mydata)
plot(fit)
```

Introdução

R é uma linguagem maravilhosa para lidar com dados:

```
mydata <- read.table()
fit <- lm(y ~ x, data = mydata)
plot(fit)
```

Com seus pacotes então...

```
read_csv() %>%
    filter() %>%
    group_by() %>%
    summarise() %>%
    ggplot() +
    geom_col()
```

Introdução

Também é muito boa para escrever protótipos de código:

- alto nível
- interpretada
- tipagem dinâmica

Introdução

Também é muito boa para escrever protótipos de código:

- alto nível
- interpretada
- tipagem dinâmica

Benefícios adicionais: a comunidade e o RStudio!

Introdução

R é um programa escrito em C que incorpora bibliotecas super otimizadas do Fortran.

Introdução

R é um programa escrito em C que incorpora bibliotecas super otimizadas do Fortran.

Está fadado a ser mais lento que C ou Fortran (ou C++ ou Assembly ou...)

Introdução

R é um programa escrito em C que incorpora bibliotecas super otimizadas do Fortran.

Está fadado a ser mais lento que C ou Fortran (ou C++ ou Assembly ou...)

O que importa mais, velocidade de execução ou velocidade de implementação?

Introdução

R é um programa escrito em C que incorpora bibliotecas super otimizadas do Fortran.

Está fadado a ser mais lento que C ou Fortran (ou C++ ou Assembly ou...)

O que importa mais, velocidade de execução ou velocidade de implementação?

Depende! Você usaria R se o código abaixo:

- demorasse 3 horas para rodar...
- mas fosse seu e estivesse em fase de testes?

```
mydata <- read.table()
fit <- lm(y ~ x, data = mydata)
plot(fit)
```

Rcpp

Pacote do R que permite fácil integração de códigos escritos em C++.

Rcpp

Pacote do R que permite fácil integração de códigos escritos em C++.

Não seria mais fácil integrar com C ou Fortran mesmo?

Rcpp

Pacote do R que permite fácil integração de códigos escritos em C++.

Não seria mais fácil integrar com C ou Fortran mesmo?

```
.C()
.Fortran()
.Call()
```

Rcpp

Pacote do R que permite fácil integração de códigos escritos em C++.

Não seria mais fácil integrar com C ou Fortran mesmo?

```
.C()
.Fortran()
.Call()
```

Muito do C que importa para nós é entendido pelo compilador do C++.

Rcpp

Pacote do R que permite fácil integração de códigos escritos em C++.

Não seria mais fácil integrar com C ou Fortran mesmo?

```
.C()
.Fortran()
.Call()
```

Muito do C que importa para nós é entendido pelo compilador do C++.

C++ vivenciou uma renascença com o padrão C++11.

Instalação

- Linux

```
install.packages("Rcpp")
```

- Windows

```
# Baixe e instale Rtools
# https://cran.r-project.org/bin/windows/Rtools/index.html
install.packages("Rcpp")
```

- Mac

```
# Instale Xcode no terminal:
# xcode-select --instal
install.packages("Rcpp")
```

Instalação

- Linux

```
install.packages("Rcpp")
```

- Windows

```
# Baixe e instale Rtools
# https://cran.r-project.org/bin/windows/Rtools/index.html
install.packages("Rcpp")
```

- Mac

```
# Instale Xcode no terminal:
# xcode-select --instal
install.packages("Rcpp")
```

- Para testar a instalação:

```
library(Rcpp)
evalCpp("2+2")
```

Bacana, mas eu não sei C++!

C++ é totalmente o oposto do que falamos sobre o R:

- baixo nível
- compilada
- tipagem estática

Bacana, mas eu não sei C++!

C++ é totalmente o oposto do que falamos sobre o R:

- baixo nível
- compilada
- tipagem estática

Declarações mais comuns:

```
// Scalar
    int k;      double x;     const int c;     long int n;

// Array
    int a[n];  double v[n];    int A[n][m];  double Y[n][m]
```

Bacana, mas eu não sei C++!

C++ é totalmente o oposto do que falamos sobre o R:

- baixo nível
- compilada
- tipagem estática

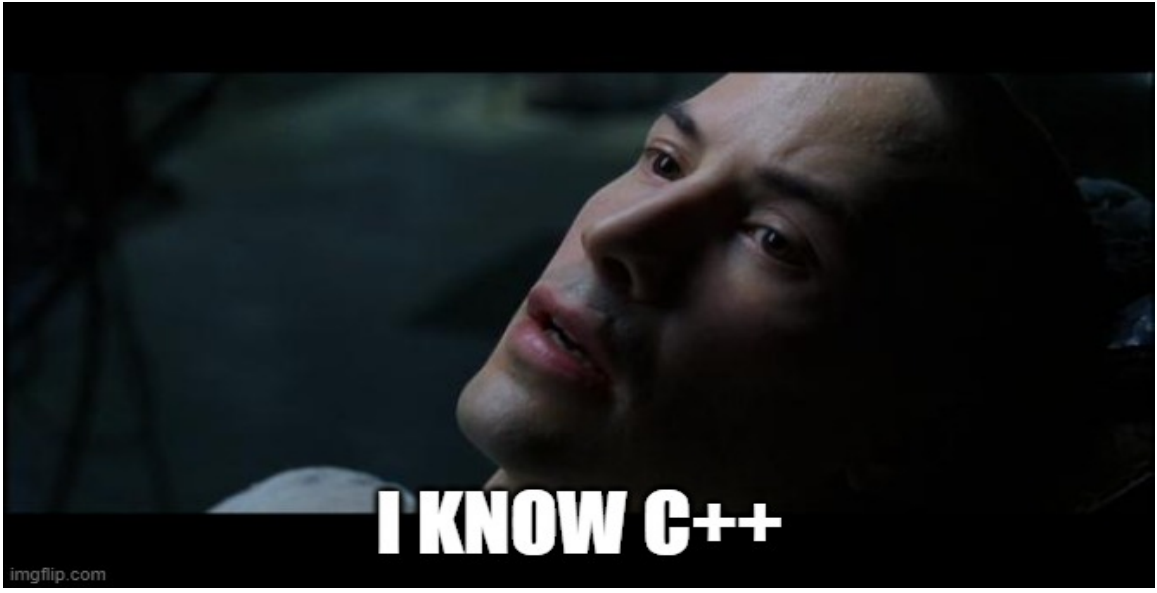
Declarações mais comuns:

```
// Scalar
    int k;      double x;     const int c;     long int n;

// Array
    int a[n];  double v[n];    int A[n][m];  double Y[n][m]
```

Os índices em C++ começam em zero e não em um!

Bacana, mas eu não sei C++!

```
#include <iostream>
using namespace std;

int main() {
    double x[4] = {1.0, -1.0, 2.0, -2.0};
    for (int i = 0; i < 4; i++) {
        if (x[i] < 0) {
            x[i] = - x[i];
        } else {
            x[i] = 2 * x[i];
        }
    }
    int j = 0;
    while (j < 4) {
        cout << x[j] << endl;
        j++;
    }
    return 0; // Entendeu tudo!? :)
}
```


I KNOW C++
imgflip.com

Show me!

Sequência de Fibonnaci: $f_n = f_{n-1} + f_{n-2}$

- $f_0 = 0$, $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, ...

Show me!

Sequência de Fibonnaci: $f_n = f_{n-1} + f_{n-2}$

- $f_0 = 0$, $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, ...

Implementação recursiva em R:

```
fibR <- function(n) {
    if (n == 0) return(0)
    if (n == 1) return(1)
    return( fibR(n-1) + fibR(n-2) )
}
```

Implementação recursiva em C++:

```
int fibC(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibC(n-1) + fibC(n-2);
}
```

Show me!

A função do R vai funcionar de imediato, mas a do C++ precisa ser compilada.

Já vimos a função **evalCpp()**. Agora vamos usar uma mais flexível:

```
Rcpp::cppFunction("
int fibC(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibC(n-1) + fibC(n-2);
}")
```

Vamos fazer alguns testes:

```
fibR(0); fibC(0)
fibR(1); fibC(1)
fibR(35); fibC(35)
```

Estruturas de Dados

O Rcpp e os pacotes que o usam têm muitas estruturas de dados.

Estruturas de Dados

O Rcpp e os pacotes que o usam têm muitas estruturas de dados.

As mais usuais são:

```
Rcpp::IntegerVector;     Rcpp::IntegerMatrix;
Rcpp::NumericVector;     Rcpp::NumericMatrix;
Rcpp::LogicalVector;     Rcpp::CharacterVector
Rcpp::List;              Rcpp::DataFrame;
```

Estruturas de Dados

O Rcpp e os pacotes que o usam têm muitas estruturas de dados.

As mais usuais são:

```
Rcpp::IntegerVector;     Rcpp::IntegerMatrix;
Rcpp::NumericVector;     Rcpp::NumericMatrix;
Rcpp::LogicalVector;     Rcpp::CharacterVector
Rcpp::List;              Rcpp::DataFrame;
```

Vamos focar aqui nos vetores, matrizes e listas, mas o resto quase que praticamente igual!

Estruturas de Dados

O Rcpp e os pacotes que o usam têm muitas estruturas de dados.

As mais usuais são:

```
Rcpp::IntegerVector;     Rcpp::IntegerMatrix;
Rcpp::NumericVector;     Rcpp::NumericMatrix;
Rcpp::LogicalVector;     Rcpp::CharacterVector
Rcpp::List;              Rcpp::DataFrame;
```

Vamos focar aqui nos vetores, matrizes e listas, mas o resto quase que praticamente igual!

Também vamos remover os **Rcpp::** começando o código com

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

Vetores

Inicialização:

```
// x <- c(0, 0)
IntegerVector x(2);

// y <- rep(1, 3)
NumericVector y(3, 1.0);

// z <- c(TRUE, FALSE, TRUE)
LogicalVector z = {true, false, true};
```

Vetores

Inicialização:

```
// x <- c(0, 0)
IntegerVector x(2);

// y <- rep(1, 3)
NumericVector y(3, 1.0);

// z <- c(TRUE, FALSE, TRUE)
LogicalVector z = {true, false, true};
```

Acessando os elementos:

```
// a <- 0 e b <- 1.0
int a = x[0]; double b = y(0);

// z[1] <- FALSE
z[0] = false;
```

Matrizes (com exercícios)

Inicialização:

```
// quais os equivalentes do R?
IntegerMatrix Z(100, 100);

NumericMatrix Y(10,2);
Y.fill(1.0);

NumericMatrix X(2,2);
X.fill_diag(1.0);
```

Matrizes (com exercícios)

Inicialização:

```
// quais os equivalentes do R?
IntegerMatrix Z(100, 100);

NumericMatrix Y(10,2);
Y.fill(1.0);

NumericMatrix X(2,2);
X.fill_diag(1.0);
```

Acessando os elementos:

```
int z = Z(0,0);

NumericVector x = X(1,_);

NumericVector y = Y(_,2); // Cuidado!
```

Listas

Inicialização:

```
List X = List::create(x1, x2); // x1 vetor e x2 escalar, por exemplo

List Y = List::create(Named("y1") = x1, Named("y2") = x2);

List Z;
Z["z1"] = x1;
Z["z2"] = x2;
```

Listas

Inicialização:

```
List X = List::create(x1, x2); // x1 vetor e x2 escalar, por exemplo

List Y = List::create(Named("y1") = x1, Named("y2") = x2);

List Z;
Z["z1"] = x1;
Z["z2"] = x2;
```

Acessando os elementos:

```
NumericVector x = X[0];

double y = Y["y2"];
```

Exemplo 1: Simulando um processo AR(1)

Processo AR(1) com erros normais: $Y_t = \phi Y_{t-1} + e_t$.

Exemplo 1: Simulando um processo AR(1)

Processo AR(1) com erros normais: $Y_t = \phi Y_{t-1} + e_t$.

Algoritmo iterativo no R:

```
ar1R <- function(n, phi, sigma) {

    y <- numeric(n)

    for (t in 2:n) {
        y[t] <- phi * y[t-1] + rnorm(1, 0, sigma)
    }

    return(y)

}
```

Exemplo: Simulando um processo AR(1)

Algoritmo iterativo no C++ (agora completinho!):

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

// [[Rcpp::export]]
NumericVector ar1C(int n, double phi, double sigma) {

    NumericVector y(n);

    for (int t = 1; t < n; t++) {
        y[t] = phi * y[t-1] + rnorm(1, 0, sigma)[0];
    }

    return y;
}
```

Vamos salvar como **.cpp** e compilar pelo R mesmo:

```
Rcpp::sourceCpp("ar1C.cpp")
```

Exemplo: Simulando um processo AR(1)

Quem será o vencedor? Vamos comparar usando o pacote **microbenchmark**:

```
microbenchmark(
    ar1R(n = 1000, phi = 0.8, sigma = 1),   # Mediana: 1340 microsegundos
    ar1C(n = 1000, phi = 0.8, sigma = 1),   # Mediana: 90 microsegundos
    times = 100
)
```

Com Rcpp ficou 15 vezes mais rápido e concluímos que não vale a pena nesse caso!

Moral da história: otimize apenas os gargalos das suas implementações.

Obrigado!