

Recursos para programação eficiente em R

Prof. Dr. Wagner Hugo Bonat

Estrutura e objetivos do módulo

Estrutura do módulo

- ▶ Estratégias para encontrar erros (*bugs*).
- ▶ Estratégias para medir a *performance*.
- ▶ Estratégias para melhorar a *performance*.
- ▶ Boas práticas para organização de código.
- ▶ Integrando o **R** com o **C++**.
- ▶ Paralelização.
- ▶ Exemplos práticos.



<https://www.pexels.com/photo/top-view-of-people-at-the-meeting-3184287/>

Objetivos do módulo

- ▶ Visitar algumas técnicas de *debug* em **R**.
- ▶ Visitar algumas ferramentas de *profilling* em **R** e **RStudio**.
- ▶ Estratégias para medir a *performance* computacional do **R**.
 - ▶ Pacotes {profvis} e {bench}.
- ▶ Estratégias para melhorar a *performance* computacional do **R**.
 - ▶ Integrando o **R** e o **C++**.
 - ▶ Ferramentas para paralelização.
- ▶ Prática com scripts.



Motivação

Motivação

- ▶ **R** não é uma linguagem rápida em termos de tempo computacional.
- ▶ **R** foi explicitamente desenhado para análise iterativa de dados.
- ▶ Fácil para humanos, não para o computador.
- ▶ Para a maioria das tarefas do dia-a-dia o **R** é rápido o suficiente.
- ▶ Comunidade ampla (em geral programadores não profissionais).
- ▶ Diversidade de pacotes extras.
- ▶ Tem de tudo em todos os sentidos!

- ▶ **C++** não é uma linguagem rápida em termos de tempo para programação e curva de aprendizado.
- ▶ **C++** foi desenhada para ser rápida computacionalmente.
- ▶ **C++** comunidade ampla e ativa.
- ▶ Fácil de obter suporte.
- ▶ Linguagem de propósito geral.
- ▶ Linguagem compilada → maior tempo de programação.
- ▶ Mais difícil de fazer pequenos protótipos de códigos.

Motivação

- ▶ **Computação científica** - Estamos interessados em desenvolver algoritmos computacionais com algum tipo de aplicação científica.
- ▶ Métodos numéricos
 - ▶ Sistemas lineares e não-lineares.
 - ▶ Derivação e integração numérica.
 - ▶ Otimização (linear, quadrática e não-linear).
 - ▶ Equações diferenciais.
- ▶ Performance computacional pode se tornar crítica.
- ▶ Métodos estatísticos
 - ▶ Manipulação de bases de dados.
 - ▶ Visualização de dados.
 - ▶ Distribuições de probabilidade.
 - ▶ Modelagem estatística de forma geral.

Objetivos

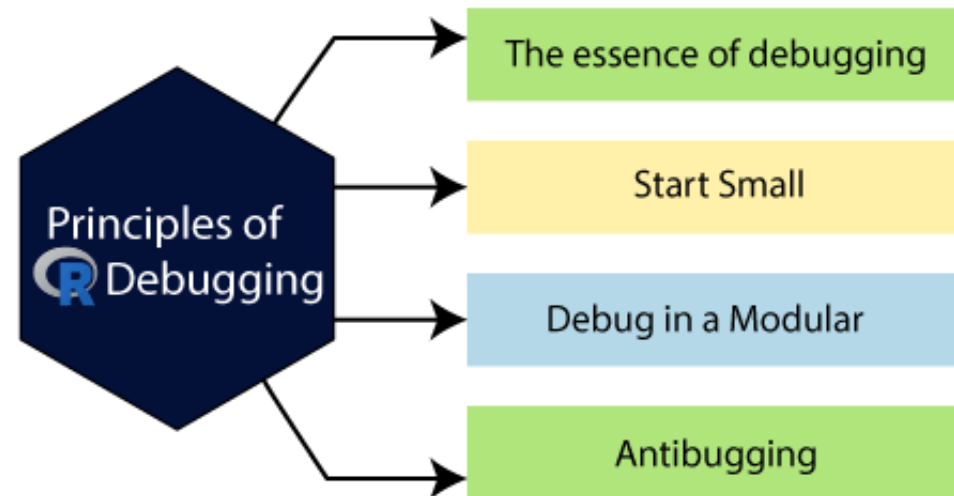
- ▶ Na programação de métodos científicos temos que lidar com duas técnicas de programação:
 1. Encontrar e arrumar erros de codificação (*bugs*).
 2. Encontrar e arrumar gargalos da performance computacional.
- ▶ Ferramentas para encontrar erros e medir a performance computacional são essenciais.
- ▶ **Objetivos:**
 1. Visitar algumas técnicas de *debug* em **R**.
 2. Visitar algumas ferramentas de *profiling* em **R** e **RStudio**.
 3. Discutir algumas estratégias para melhorar a performance computacional do **R**.



Encontrando erros

Ferramentas para *debug*

- ▶ O que fazer quando o **R** retorna uma mensagem de erro inesperada?
- ▶ Quais são as ferramentas para encontrar e arrumar o erro?
- ▶ Principal documentação [RStudio debugging documentation](#).
 - ▶ `traceback()` função que ajuda a encontrar onde um erro ocorreu.
 - ▶ `debug()` e `browser()`.
 - ▶ Breakpoints `Shift + F9`.
 - ▶ `options(error = recover)`.
- ▶ Código **R** (Modulo4_Script1.R).



Estratégia geral

Finding your bug is a process of confirming the many things that you believe are true — until you find one which is not true. - Norm Matloff

1. Google! Dê uma olhada nos pacotes `errorist` e `searcher`.
2. Faça o erro reproduzível.
 - ▶ Faça o exemplo ser o menor possível.
 - ▶ Desenvolva testes automatizados.
3. Descubra onde o erro está.
 - ▶ Adote o método científico.
 - ▶ Crie hipóteses, desenhe experimentos e teste.
4. Arrume e teste.
 - ▶ Cuidado para não incluir novos erros.
 - ▶ Importância de automatizar os testes.



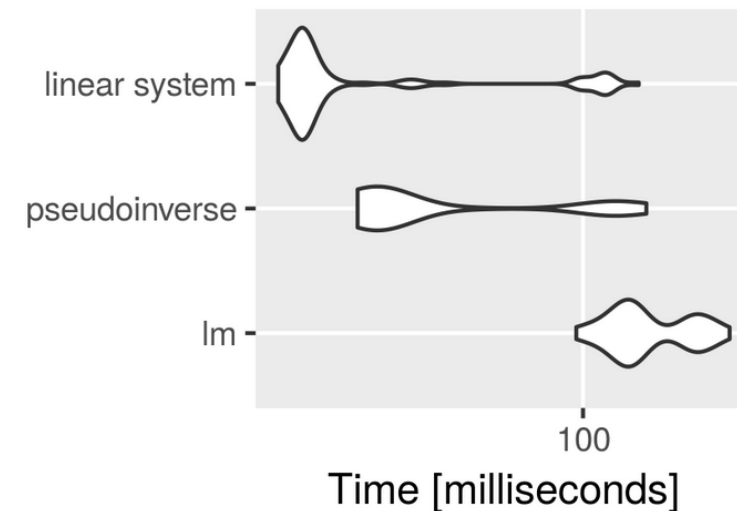
<https://pixabay.com/pt/users/graphicmama-team-2641041>

Medindo a *performance*

Medindo a *performance*

Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. - Donald Knuth.

- ▶ Para ter um código rápido, primeiro precisamos saber onde ele é lento.
- ▶ Identificar os chamados *bottlenecks*.
- ▶ *Profilling* - Medir o tempo computacional de cada linha de código.
- ▶ Encontrado o ponto crítico, testamos diferentes estratégias.
- ▶ Dois pacotes são populares [profvis](#) e [bench](#).
- ▶ Código [R](#) (Modulo4_Script1.R).



Estratégias para melhorar a *performance*

Estratégias para melhorar a *performance*



We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified. — Donald Knuth

Be pragmatic: don't spend hours of your time to save seconds of computer time. - Hadley Wickham.

- ▶ Organize o código para otimizar a performance e evitar *bugs*.
- ▶ Estratégia do preguiçoso: A função mais rápida é aquela que trabalha menos.
- ▶ Vetorize e evite cópias.
- ▶ Troque a linguagem.

Organização do código

- ▶ Armadilha da tentativa de melhorar o código:
 - ▶ Código rápido porém incorreto.
 - ▶ Código que você acha que é rápido, mas na verdade tem a mesma performance.
- ▶ Como evitar essas armadilhas?
 - ▶ Identifique o ponto crítico (*bottleneck*).
 - ▶ Esboce um conjunto de possibilidades para melhorar a performance.
 - ▶ Escreva cada uma em uma função separada.
 - ▶ Gere um exemplo representativo da situação.
 - ▶ Use *benchmark* para comparar as estratégias.



```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```


Como buscar estratégias de melhorias?

- ▶ Verifique opções existentes.
 - ▶ Consulte o *CRAN task view*.
 - ▶ Consulte as dependências reversas do pacote `Rcpp`.
- ▶ Procure na literatura termos para descrever o *bottleneck*.
- ▶ Pergunte para colegas.
- ▶ Use os termos encontrados para procurar no Google e StackOverflow.
- ▶ Restrinja sua busca a página relacionadas com o `R` (<https://rseek.org/>).
- ▶ Selecione as opções que parecem promissoras.
- ▶ Benchmark cada uma e tente combiná-las para criar uma melhor.
- ▶ Pare assim que o código for rápido o suficiente.
- ▶ ChatGPT.

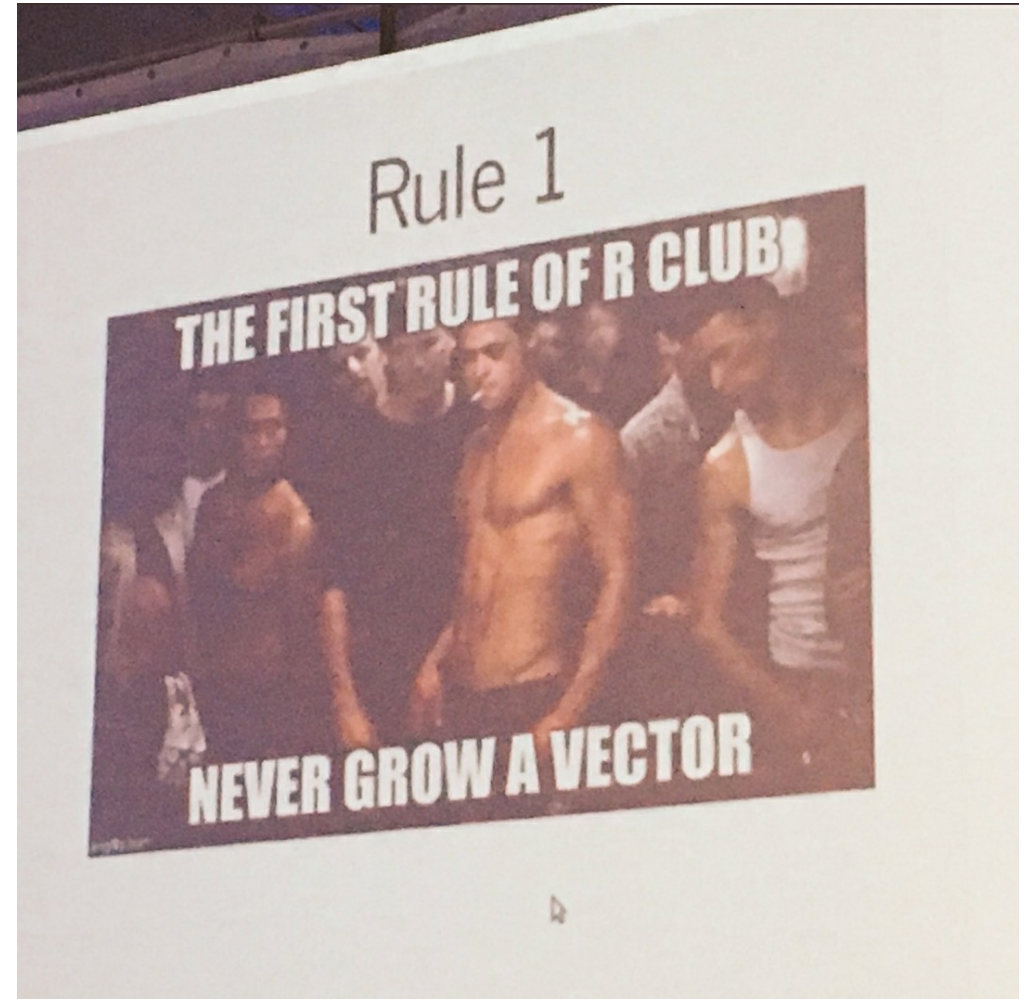
Vetorização

- ▶ Vetorizar não é apenas evitar `loops`.
- ▶ Pensar no código como um todo.
- ▶ Vetores ao invés de escalares.
- ▶ Em `R` vetorizar é simplesmente encontrar uma função em `C` que faz o que você quer :)
- ▶ Alguns exemplos: `rowSums()`, `colSums()`, `rowMeans()`, e `colMeans()`.
- ▶ Funções para tarefas específicas tendem a ser mais rápidas do que funções genéricas.
- ▶ Álgebra matricial é uma forma de `vetorização`.
- ▶ Bibliotecas de álgebra linear como a BLAS são altamente eficientes.



Evite cópias

- ▶ Regra número 1: Nunca cresça um objeto!
- ▶ Uso de funções como `c()`, `append()`, `cbind()`, `rbind()`, ou `paste()` deve ser feito com cuidado.
- ▶ Estudo de caso: Teste-t.
- ▶ Objetivo: Executar 1000 experimentos, cada um coletando amostras de 50 indivíduos. Os primeiros 25 indivíduos são designados ao grupo 1 e o resto ao grupo 2. Efetuar um teste t para comparar as médias dos grupos 1 e 2.
- ▶ Código `R`(Modulo4_Script1.R).



Dicas do Hadley para melhorar a sua programação

- ▶ Read R blogs to see what performance problems other people have struggled with, and how they have made their code faster.
- ▶ Read other R programming books, like The Art of R Programming or Patrick Burns' R Inferno to learn about common traps.
- ▶ Take an algorithms and data structure course to learn some well known ways of tackling certain classes of problems. I have heard good things about Princeton's Algorithms course offered on Coursera.
- ▶ Learn how to parallelise your code. Two places to start are Parallel R and Parallel Computing for Data Science.
- ▶ Read general books about optimisation like Mature optimisation or the Pragmatic Programmer.
- ▶ Reescreva suas funções em `C++`.



Hadley Wickham

Melhorando a *performance* usando C++

Típica situação

- ▶ Código está funcionando perfeitamente.
- ▶ Já fez o *profiling* e melhorou os principais *bottleneck*.
- ▶ Mas seu código ainda não está rápido o suficiente.
- ▶ Reescrever partes importantes do seu código em `C++` pode ajudar.
- ▶ Típicos *bottlenecks* que valem a pena escrever código em `C++`:
 - ▶ *Loop's* que não podem ser vetorizados, porque são usados em sequência.
 - ▶ Funções recursivas, ou problemas que envolvem chamar funções milhões de vezes.
 - ▶ Problemas avançados que requerem estruturas ou algoritmos que o `R` não tem.
- ▶ Página do `Rcpp`.
- ▶ Tutorial de `Rcpp` (Modulo4_Script2.R).



Logo C++.

Estudo de caso

Amostrador de Gibbs (Gibbs sampler) extraído do blog do Dirk.

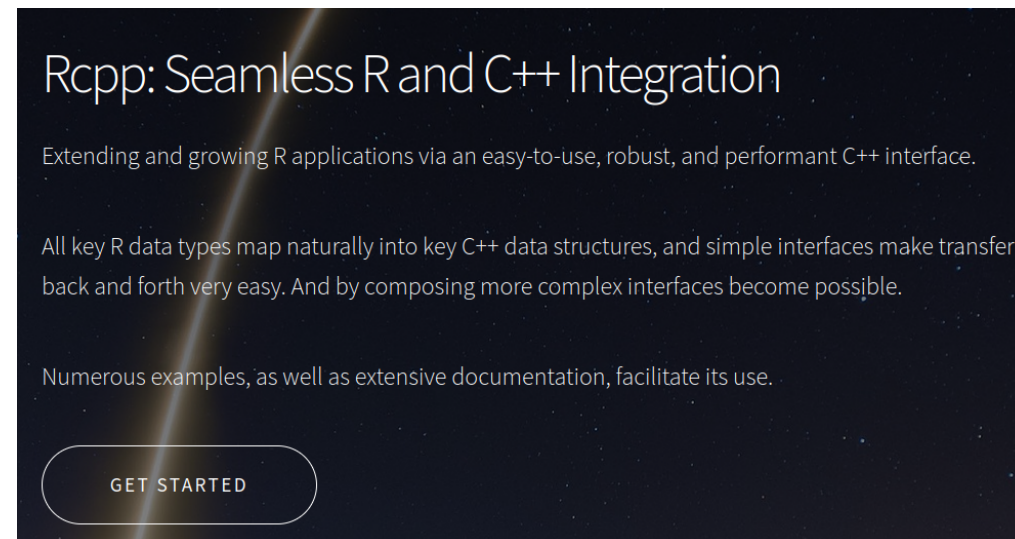
```
gibbs_r <- function(N, thin) {  
  mat <- matrix(nrow = N, ncol = 2)  
  x <- y <- 0  
  for (i in 1:N) {  
    for (j in 1:thin) {  
      x <- rgamma(1, 3, y * y + 4)  
      y <- rnorm(1, 1 / (x + 1),  
                1 / sqrt(2 * (x + 1)))  
    }  
    mat[i, ] <- c(x, y)  
  }  
  mat  
}
```

```
#include <Rcpp.h>  
using namespace Rcpp;  
// [[Rcpp::export]]  
NumericMatrix gibbs_cpp(int N, int thin) {  
  NumericMatrix mat(N, 2);  
  double x = 0, y = 0;  
  for(int i = 0; i < N; i++) {  
    for(int j = 0; j < thin; j++) {  
      x = rgamma(1, 3, 1 / (y * y + 4))[0];  
      y = rnorm(1, 1 / (x + 1),  
                1 / sqrt(2 * (x + 1)))[0];  
    }  
    mat(i, 0) = x;  
    mat(i, 1) = y;  
  }  
  return(mat);  
}
```

► Tutorial iniciando com o Rcpp.

Mais recursos

- ▶ O tutorial apresentou apenas os aspectos básicos do `Rcpp`.
- ▶ `vignette("Rcpp-quickref")` é uma excelente referência.
- ▶ Página do `Rcpp`.
- ▶ Effective C++ e Effective STL.
- ▶ C++ Annotations.
- ▶ Algorithm Libraries.
- ▶ Pacotes adicionais do ecossistema `Rcpp`: `RcppArmadillo`, `RcppEigen` e `RcppGSL`.

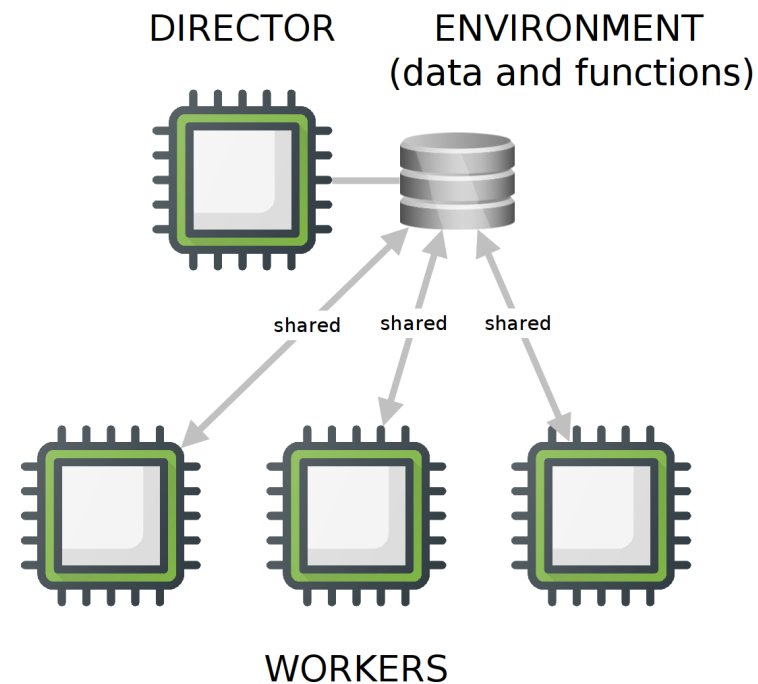


Página do Rcpp.

Paralelização

Paralelização em R

- ▶ Existem diversos esquemas de paralelização.
- ▶ Não vamos entrar em detalhes em nenhum :(
- ▶ Loops em R tendem a ser lentos, alternativas
 - ▶ Escreva código em C++ através do Rcpp.
 - ▶ Paralelize o seu for (se ele não for sequencial).
- ▶ Pacotes úteis: foreach, parallel e doParallel.
- ▶ Exemplos: Código R (Modulo4_Script3.R).



Estrutura de processamento.