



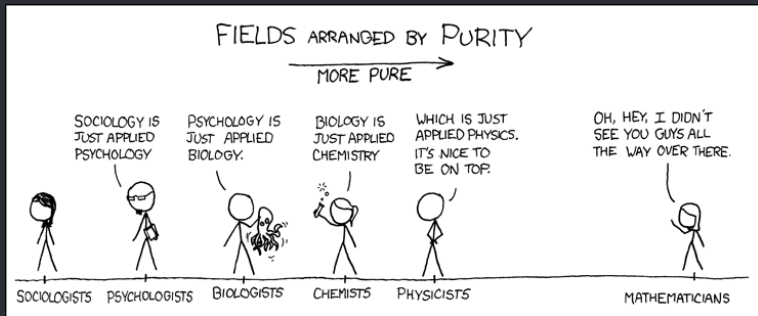
# De Fortran a Julia: Computação Científica no Mundo Real

.....

QConSP 2019

Melissa Weber Mendonça

## Quem sou eu?



# O que é Computação Científica?

Computação Científica não é só Data Science!!!!1!1ONE

```
\includegraphics{angry.gif}
```

# O que é Computação Científica?

“Computação Científica é a coleção de ferramentas, técnicas e teorias necessárias para a resolução computacional de modelos matemáticos de problemas nas Ciências e na Engenharia.” [1]

# O que é Computação Científica?

“Computação Científica é a coleção de ferramentas, técnicas e teorias necessárias para a resolução computacional de modelos matemáticos de problemas nas Ciências e na Engenharia.” [1]

- Previsão/modelagem de sistemas complexos em grande escala;

# O que é Computação Científica?

“Computação Científica é a coleção de ferramentas, técnicas e teorias necessárias para a resolução computacional de modelos matemáticos de problemas nas Ciências e na Engenharia.” [1]

- Previsão/modelagem de sistemas complexos em grande escala;
- Simulação de experimentos caros/perigosos;

# O que é Computação Científica?

“Computação Científica é a coleção de ferramentas, técnicas e teorias necessárias para a resolução computacional de modelos matemáticos de problemas nas Ciências e na Engenharia.” [1]

- Previsão/modelagem de sistemas complexos em grande escala;
- Simulação de experimentos caros/perigosos;
- Soluções interdisciplinares de problemas das Ciências e da Engenharia.

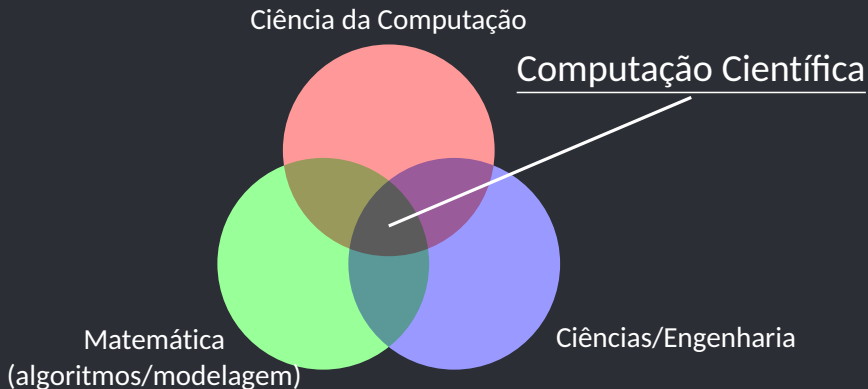
# O que é Computação Científica?

“Computação Científica é a coleção de ferramentas, técnicas e teorias necessárias para a resolução computacional de modelos matemáticos de problemas nas Ciências e na Engenharia.” [1]

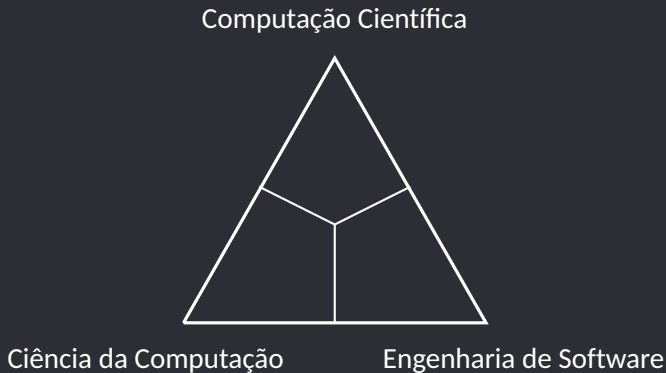
- Previsão/modelagem de sistemas complexos em grande escala;
- Simulação de experimentos caros/perigosos;
- Soluções interdisciplinares de problemas das Ciências e da Engenharia.
- “A Computação Científica é hoje o “terceiro pilar da ciência”, ao lado da análise teórica e dos experimentos para a descoberta científica.” [2]



# O que é Computação Científica?



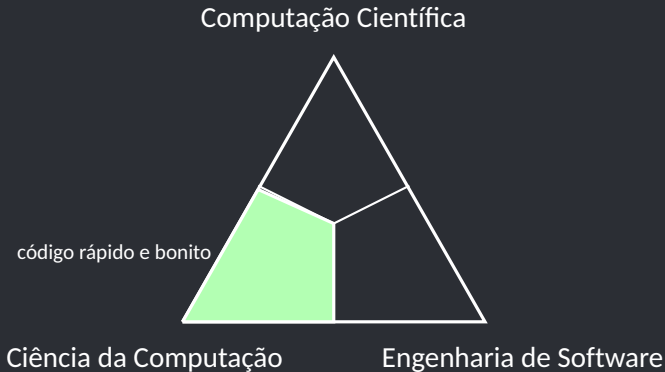
Mas, na prática...



---

Baseado na imagem [daqui](#)

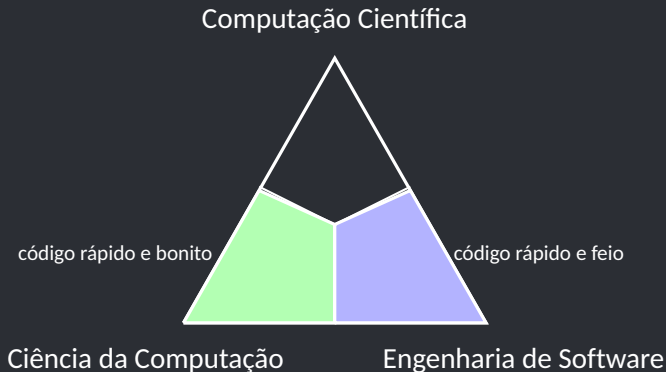
## Mas, na prática...



---

Baseado na imagem [daqui](#)

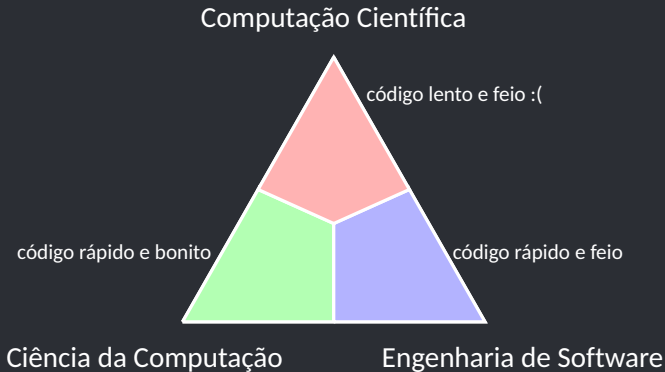
## Mas, na prática...



---

Baseado na imagem [daqui](#)

## Mas, na prática...



---

Baseado na imagem [daqui](#)

# Tipos de Problemas

- Solução Numérica de equações diferenciais: finanças, bioinformática, meteorologia, economia, física
- Estimação de parâmetros/Problemas inversos: estatística, Data Science, processamento de imagens, biomedicina, ajuste de curvas
- Simulações: física, engenharia, geociências, ciência de materiais, dinâmica de fluidos
- Otimização e Álgebra Linear Numérica

# A importância da Álgebra Linear

## Dimensão

Número de variáveis de um problema.

Exemplo:

$$\begin{cases} 2x + 3y = 1 \\ x - 2y = 1 \end{cases}$$

# A importância da Álgebra Linear

## Dimensão

Número de variáveis de um problema.

Exemplo:

$$\begin{cases} 2x + 3y = 1 \\ x - 2y = 1 \end{cases} \Leftrightarrow \begin{pmatrix} 2 & 3 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$



# A importância da Álgebra Linear

## Dimensão

Número de variáveis de um problema.

Exemplo:

$$\begin{cases} 2x + 3y = 1 \\ x - 2y = 1 \end{cases} \Leftrightarrow \begin{pmatrix} 2 & 3 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Leftrightarrow Ax = b.$$

# A importância da Álgebra Linear

A Álgebra Linear estuda espaços vetoriais e matrizes para resolver problemas do tipo

$$Ax = b$$

$$(A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^m)$$

# A importância da Álgebra Linear

A Álgebra Linear estuda espaços vetoriais e matrizes para resolver problemas do tipo

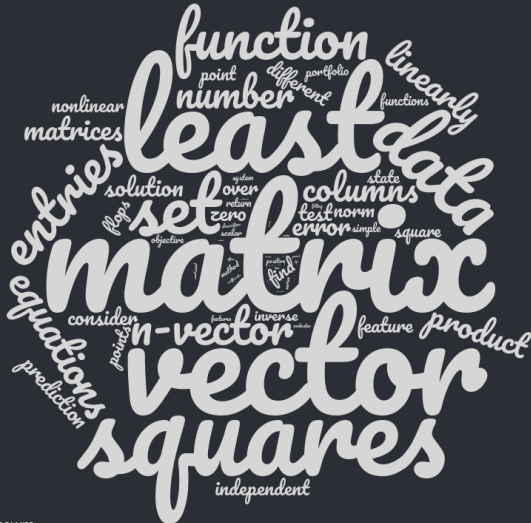
$$Ax = b$$

$$(A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^m)$$

## FUN FACT:

A solução de todos os problemas mencionados previamente dependem da solução de sistemas lineares!

# O que é Álgebra Linear Numérica?



# Álgebra Linear Numérica

Estudo de algoritmos para realizar cálculos computacionais de álgebra linear, principalmente operações em matrizes e vetores.

Algoritmos principais:

- Decomposição LU
- Decomposição QR
- Decomposição em Valores Singulares (SVD - usado em PCA)
- Cálculo de Autovalores/Autovetores
- Métodos iterativos especializados para a resolução de sistemas lineares

## BLAS (1979)

*Basic Linear Algebra Subprograms* (BLAS): especificação de rotinas de baixo nível para operações comuns de álgebra linear.

- ATLAS
- MKL
- OpenBLAS (GotoBLAS)

LAPACK, LINPACK, GNU Octave, Mathematica, MATLAB, NumPy/SciPy, R, e Julia usam a BLAS. (Gráficos, Compressão de áudio/vídeo, Compressão de arquivos, Jogos...)

# Operações Básicas da BLAS

- Level 1 (1979): tipicamente  $O(n)$ . Operações vetoriais em arrays: produto interno, norma de matriz (*axpy*):

$$y \leftarrow \alpha x + y$$

- Level 2 (1984-1988): tempo quadrático. Operações matriz-vetor (*gemv*):

$$y \leftarrow \alpha Ax + \beta y$$

- Level 3 (1990): tempo cúbico. Operações matriz-matriz (*gemm*):

$$C \leftarrow \alpha AB + \beta C$$

## LAPACK (antiga LINPACK)

Benchmarks LINPACK: TOP500 (que elenca e detalha os melhores sistemas computacionais não-distribuídos do mundo) é construída medindo a velocidade de resolução de um sistema linear  $n$  por  $n$  denso  $Ax = b$ .



Prof. Jack Dongarra

---

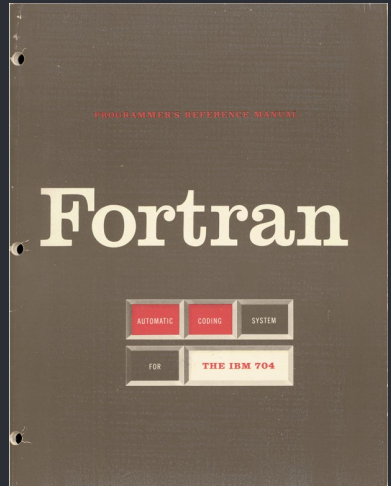
Foto da **Universidade do Tennessee, Knoxville**



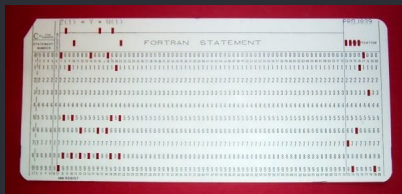
# FORTRAN/Fortran

"I don't know what the programming language of the year 2000 will look like, but I know it will be called FORTRAN."

—Tony Hoare, ganhador do Turing Award de 1980, in 1982



# Como assim "Fortran Moderno?!"



- FORTRAN 66/77, Fortran 90/95/2003/2008/2018 - *backwards compatible*
- Código maduro, bibliotecas, interoperabilidade
- Arquiteturas variadas

# Características

- A estrutura de dados primária é o vetor/matriz
- Sintaxe de slicing nativa, masked arrays, vetorização \*
- É difícil escrever código lento! (*non-aliasing*)
- Fortran é **concurrent** desde 2008: **coarrays** (com MPI)
- Legibilidade: feito para computação científica
- Compiladores muito eficientes (alguns "roubando") ▶ lista ;

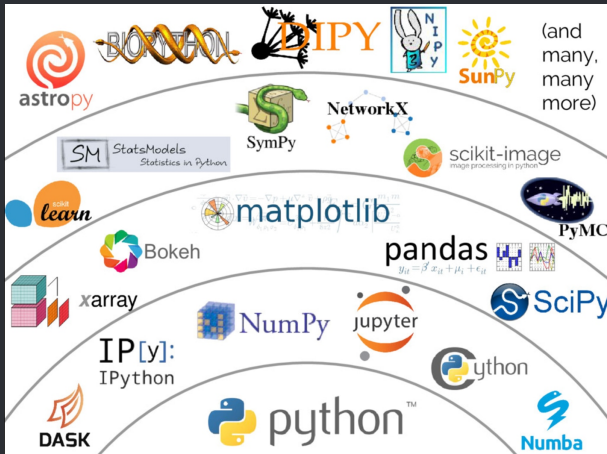
# Futuro do Fortran

Ainda muito saudável em computação científica e HPC!

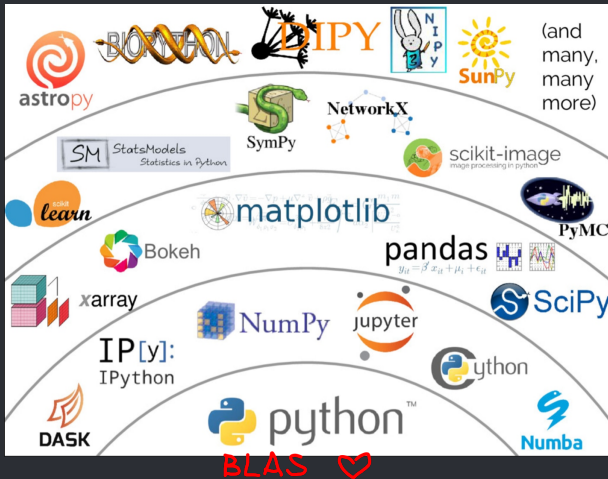
Alguns projetos:

- CHARMM (Dinâmica molecular)
- Code Saturne (Dinâmica de Fluidos Computacional)
- NEMO (Oceanografia)
- QUANTUMESPRESSO (Modelagem de Materiais)
- SPECFEM3D (Propagação de ondas sísmicas)
- WRF (Previsão do Tempo)

# E o Python? (Imagem de [3])



## E o Python? (Imagem de [3])



# Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python?

# Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python? Incompatíveis!



## Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python? Incompatíveis!
- Perda de modularidade/interoperabilidade (código monolítico)

## Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python? Incompatíveis!
- Perda de modularidade/interoperabilidade (código monolítico)
- Reescrever algoritmos especializados do zero?

# Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python? Incompatíveis!
- Perda de modularidade/interoperabilidade (código monolítico)
- Reescrever algoritmos especializados do zero?
- Numba
- Cython
- Pypy

# Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python? Incompatíveis!
- Perda de modularidade/interoperabilidade (código monolítico)
- Reescrever algoritmos especializados do zero?
- Numba
  - Fortran (SciPy) + Numba + Python ainda tem context switching
  - Não faz otimização interprocedural
  - Determinar quando o código deve usar um JIT nem sempre é óbvio
- Cython
- Pypy

# Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python? Incompatíveis!
- Perda de modularidade/interoperabilidade (código monolítico)
- Reescrever algoritmos especializados do zero?
- Numba
- Cython
  - Linguagem compilada: compila Python em C.
  - Grande parte da SciPy, pandas e scikit-learn estão escritas em Cython.
  - Pode ser estaticamente tipada, e é aí que o desempenho melhora (loops em C)
  - Pode desligar o GIL
- Pypy

# Cython/Numba/Pypy

- É suficiente usar um JIT, ou acelerar o Python? Incompatíveis!
- Perda de modularidade/interoperabilidade (código monolítico)
- Reescrever algoritmos especializados do zero?
- Numba
- Cython
- Pypy
  - JIT (RPython)
  - GIL: existem workarounds, mas não tão eficientes
  - Grande parte da SciPy, pandas e scikit-learn estão escritas em C(ython).

## E o C?

“C is pretty much a model of the computer, where you can glue on as much linear algebra as you want, provided you are prepared to pretend that it resembles the computer.

Fortran is pretty much a model of linear algebra, where you can glue on as many computer details as you want, provided you are prepared to pretend that they resemble linear algebra.

(...)

**Whether you prefer to pretend that the computer is made of linear algebra or that the linear algebra is made of computer, depends mostly on where you are coming from to begin with.”**

— Resposta em [7]

## E o C++?

- Blitz++: expression templates, template metaprogramming
- Já é possível escolher non-aliasing
- Sem garbage collection (assim como o Fortran, mas naturalmente usa-se mais ponteiros do que no Fortran)
- Reprodutibilidade?



# HPC

Em uma **pesquisa\*** de usuários de Fortran na 2014 Supercomputing Convention, 100% dos entrevistados disse que acreditavam que ainda estariam usando Fortran em 5 anos.

- C++ e Fortran moderno, com OpenMPI
- CUDA
- Clay Christiansen, em “The Innovators Dillema” diz que tecnologias disruptivas aparecem quando surge um novo mercado/grupo de usuários
- Loop fission/fusion, multithreading, coarrays não são *nice to have*; são **essenciais!**
- Exemplos: <http://www.archer.ac.uk/status/codes/>

## Julia: a solução para o problema das duas linguagens?

```
(juliaenv) [melissa@oneesk Downloads]$ julia
```

```
julia>
```


Documentation: <https://docs.julialang.org>

```
Type "?" for help, "]?" for Pkg help.
```

```
Version 1.0.0
conda-forge-julia release
```

## Julia: a solução para o problema das duas linguagens?

```
(juliaenv) [melissa@oneesk Downloads]$ julia
```

```
| Documentation: https://docs.julialang.org  
|  
| Type "?" for help, "]"?" for Pkg help.  
|  
| Version 1.0.0  
| conda-forge-julia release  
|  
  
julia>
```


- Não é um Python mais rápido, e sim um C++ mais produtivo

## Julia: a solução para o problema das duas linguagens?

[illegible]

- Não é um Python mais rápido, e sim um C++ mais produtivo
- Mais próximo de um AOT do que de um JIT

# Julia: a solução para o problema das duas linguagens?

```
(juliaenv) [melissa@oneesk Downloads]$ julia~  
 Documentation: https://docs.julialang.org  
Type "?" for help, "]?" for Pkg help.  
Version 1.0.0  
conda-forge-julia release  
  
julia> █
```

- Não é um Python mais rápido, e sim um C++ mais produtivo
- Mais próximo de um AOT do que de um JIT
- “We believe that the traditional HPC and “Big Data” worlds are converging, and we’re aiming Julia right at that convergence point.” (Stefan Karpinski, Viral Shah, Jeff Bezanson and Alan Edelman)

## Julia II: Revenge of the Loops

- Ideia principal: Multiple Dispatch + Estabilidade de Tipos => Velocidade + Legibilidade

## Julia II: Revenge of the Loops

- Ideia principal: Multiple Dispatch + Estabilidade de Tipos => Velocidade + Legibilidade
- LLVM permite competitividade em máquinas de todos os tipos

## Julia II: Revenge of the Loops

- Ideia principal: Multiple Dispatch + Estabilidade de Tipos => Velocidade + Legibilidade
- LLVM permite competitividade em máquinas de todos os tipos
- Interoperabilidade com Fortran, C, Python



## Julia II: Revenge of the Loops

- Ideia principal: Multiple Dispatch + Estabilidade de Tipos => Velocidade + Legibilidade
- LLVM permite competitividade em máquinas de todos os tipos
- Interoperabilidade com Fortran, C, Python
- Metaprogramming, code introspection

## Julia II: Revenge of the Loops

- Ideia principal: Multiple Dispatch + Estabilidade de Tipos => Velocidade + Legibilidade
- LLVM permite competitividade em máquinas de todos os tipos
- Interoperabilidade com Fortran, C, Python
- Metaprogramming, code introspection
- Desempenho previsível: IR legível @code\_llvm, @code\_native

## Julia II: Revenge of the Loops

- Ideia principal: Multiple Dispatch + Estabilidade de Tipos => Velocidade + Legibilidade
- LLVM permite competitividade em máquinas de todos os tipos
- Interoperabilidade com Fortran, C, Python
- Metaprogramming, code introspection
- Desempenho previsível: IR legível `@code_llvm`, `@code_native`
- Multiparadigmas

## Julia III: Computação Científica Edition

- O esquema de compilação facilita o desenvolvimento de software científico
- *A maior parte das pessoas não consegue escrever “Fortran ótimo”*. Cientistas não são engenheiros de software.
- Para scripts pequenos o benefício pode não ser óbvio
- Para desenvolvedores de pacotes de computação científica, pode ser revolucionário!

# Julia Paralelo, HPC, e para Ciência de Dados: Pacotes

- Paralelização simples: `@threads` paraleliza loops; `@parallel` adiciona a possibilidade de utilizar multiprocessamento
- `CUDANative.jl`, `GPUArrays.jl`
- `Flux.jl`, `Knet.jl` para ML
- `DifferentialEquations.jl`
- `IterativeSolvers.jl` (Matrix-free; `A_mul_B`)
- `Convex.jl`, `JuMP`, `JuliaSmoothOptimizers`
- <https://juliaobserver.com/>

# Desafios Técnicos

# Desafios Técnicos

- Obter o desempenho dos benchmarks nem sempre é fácil!

# Desafios Técnicos

- Obter o desempenho dos benchmarks nem sempre é fácil!
- “Better software, better research”: será que a velocidade é sempre o mais importante?



# Desafios Técnicos

- Obter o desempenho dos benchmarks nem sempre é fácil!
- “Better software, better research”: será que a velocidade é sempre o mais importante?
- Poderio computacional não é tudo: algoritmos podem ser mais importantes que a implementação

## Desafios Técnicos

- Obter o desempenho dos benchmarks nem sempre é fácil!
- “Better software, better research”: será que a velocidade é sempre o mais importante?
- Poderio computacional não é tudo: algoritmos podem ser mais importantes que a implementação
- Clojure? Haskell? Linguagens Funcionais? Ainda não tiveram sucesso na Computação Científica (cópias; dados imutáveis; funções puras)

## Desafios menos técnicos

- Padrões e linguagens novas nem sempre são adotados rapidamente

## Desafios menos técnicos

- Padrões e linguagens novas nem sempre são adotados rapidamente
- *Research Software Engineers*

## Desafios menos técnicos

- Padrões e linguagens novas nem sempre são adotados rapidamente
- *Research Software Engineers*
- Desempenho numérico em grande escala é um problema difícil e particular

## Desafios menos técnicos

- Padrões e linguagens novas nem sempre são adotados rapidamente
- *Research Software Engineers*
- Desempenho numérico em grande escala é um problema difícil e particular
- É difícil convencer pessoas a trabalharem em projetos open source porque o desenvolvimento de software não é visto como um produto de pesquisa válido para acadêmicos. “Toda grande biblioteca matemática open source é construída sobre as cinzas da carreira acadêmica de alguém”.

# Bibliografia

- [1] Gene H. Golub and James Ortega. *Scientific Computing and Differential Equations — An Introduction to Numerical Methods*. Academic Press, 1992.
- [2] *What is Scientific Computing?* [Technische Universität Kaiserslautern](#)
- [3] *Python Scientific Computing* [DataCamp](#)
- [4] *Scientific Computing's Future: Can any coding language top a 1950's behemoth?* [Ars Technica](#)
- [5] *Why Numba and Cython are not substitutes for Julia* [Stochastic Lifestyle Blog](#)
- [6] *Formula Translation in Blitz++, NumPy and Modern Fortran: A Case Study of the Language Choice Tradeoffs*
- [7] *For scientific computing, what are the advantages and disadvantages of using C as compared to using FORTRAN 90?*

# Fortran Compilers

- GFortran
- Flang: LLVM
- Intel ifort: SIMD vetorização e threading, OpenMP
- Lahey/Fujitsu
- NAG
- IBM XL Fortran: SIMD, OpenMP, Arquitetura Power9
- Arm Fortran Compiler

► back



# Aliasing

“Fortran, by forbidding aliasing, holds a significant advantage over C++. Aliasing means that if a Fortran function gets two arrays, slices, or values, modifying one of them is assumed not to affect the other. If the programmer accidentally passed two parameters that alias, there is no guarantee that the result will be correct. Fortran basically passes the responsibility to the programmer, while C++ tries to be safer. The impact on performance can be devastating for C++. In Fortran, because parameters are not supposed to alias, the compiler can keep a value from parameter A in a register even when it writes data to parameter B. In C++ this does not work this way, since the compiler can't always be sure if writing to B is guaranteed to not touch A. So C++ has to reload the value in the register of A, from memory every time. Here is where C is better than C++. In C it is possible to mark a parameter with restrict to signify that it can't alias, and get performance similar to Fortran for a single function. This keyword is not available in C++.”

— from [Is there something fundamental to the Fortran language itself \(not massive legacy code bases or time-proven optimizing compilers\) that causes it to still be preferable to speed-critical numerical computations/HPC rather than, say, C/C++?](#)