AVALIAÇÃO DE DESEMPENHO
ETAPA 3: FINAL

Enzo Lisbôa Peixoto, Nathan Mattes e Pedro Scholz Soares

Outubro de 2025

- Problema: Resolução da Equação de Calor (Difusão) usando o método das Diferenças Finitas em 1D, 2D e 3D;
- Comparação: Linguagem Compilada JIT (Julia) vs. Linguagem Interpretada (Python);
- Fatores: Linguagem (2 níveis), Dimensão (3 níveis), Tamanho do Problema (3 níveis: Low, Mid, High);
- Métricas: Tempo de Execução e Pico de Memória;
- Repetições: 50 execuções por configuração para garantir significância estatística.

CONTROLE DO EXPERIMENTO

- CPU Pinning: Uso de –cpuset-cpus="0"para isolar o processo em um único núcleo e evitar trocas de contexto;
- Frequency Governor: Fixado em performance para evitar oscilação de clock;
- Turbo Boost: Desativado.

- 12th Gen Intel(R) Core(TM) i3-1215U (6/8);
- 8GB de Memória
- Uso de Docker para garantir reprodutibilidade do ambiente de software;
- Script automacao.py que orquestra todo o fluxo;

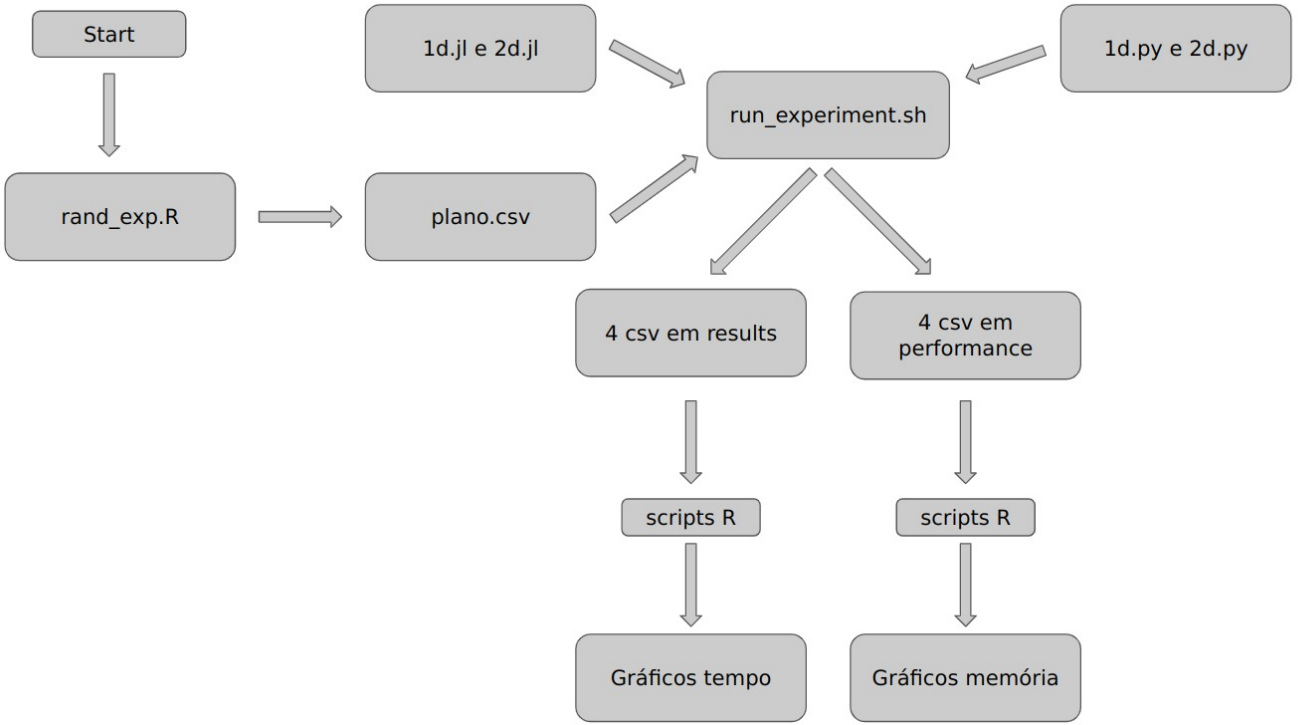
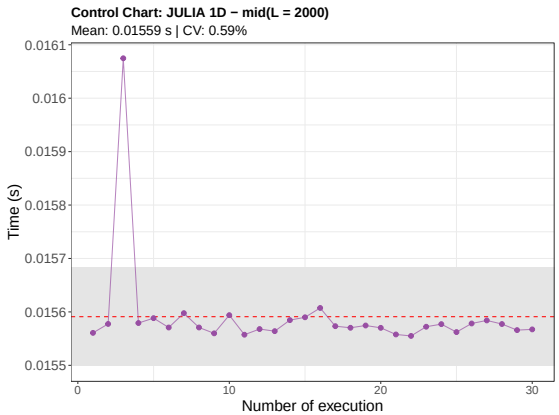
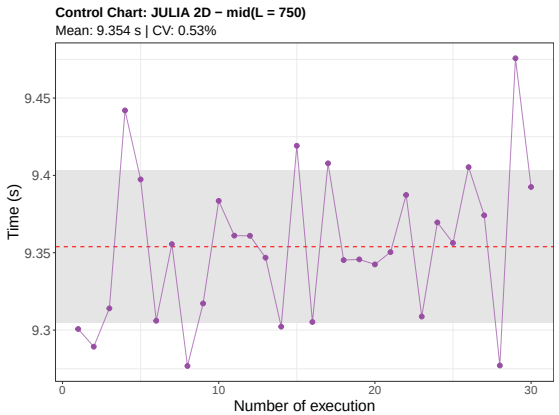


Figura: Organograma dos Experimentos

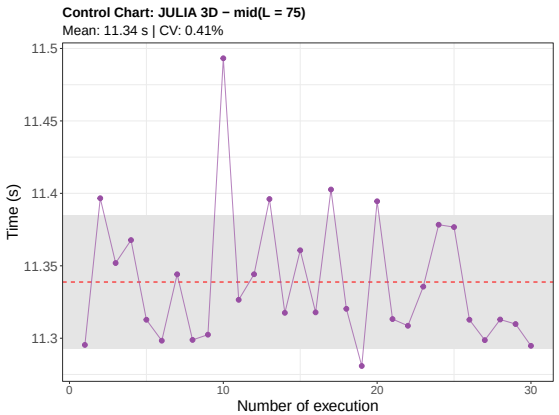
Gráficos de Tempo de Execução, carga intermediária - Julia



1D

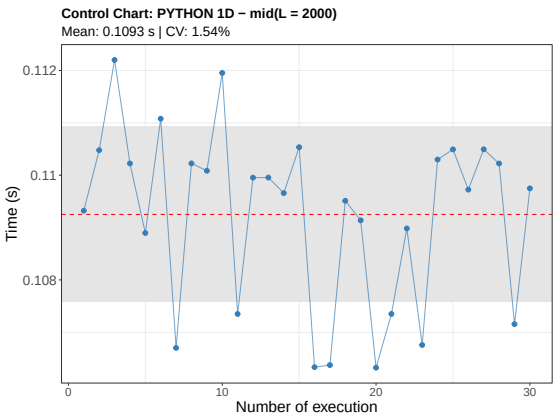


2D

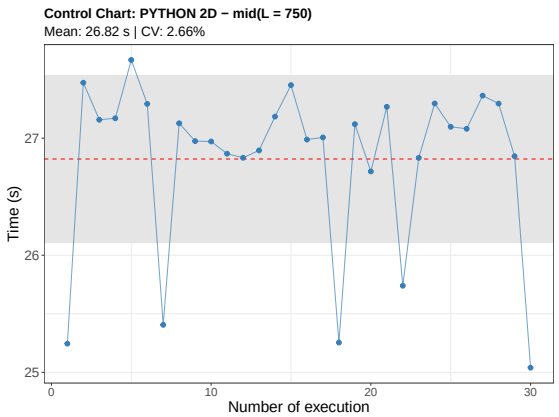


3D

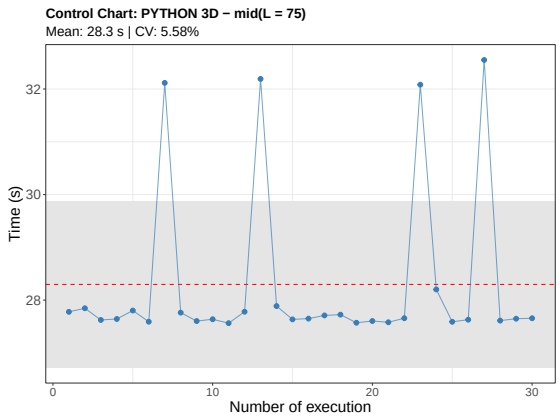
Gráficos de Tempo de Execução, carga intermediária - Python



1D

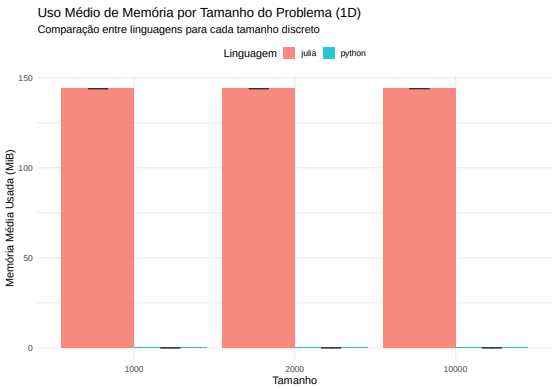


2D

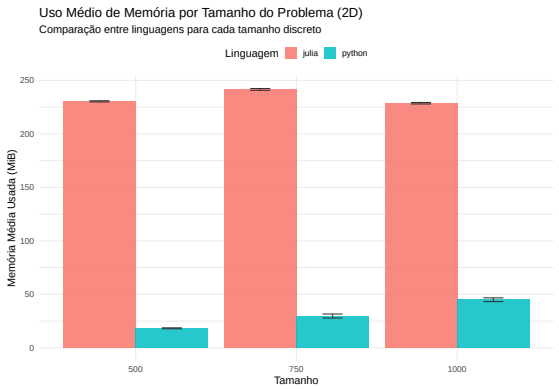


3D

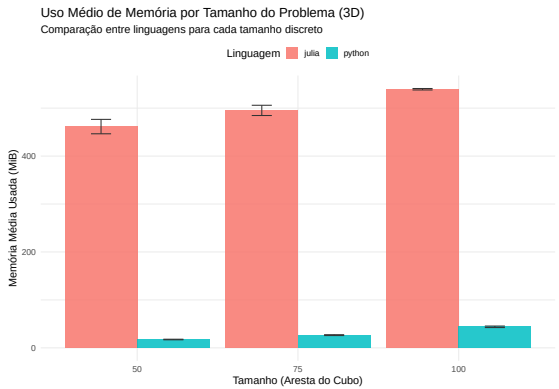
Gráficos Memória



1D



2D



3D

Tabela: Regressão Linear: Tempo de Execução

Linguagem	**Dimensão**	R^2	R^2_{adj}
Python	1D	**0.9631**	0.9627
Python	2D	**0.9720**	0.9717
Python	3D	**0.9773**	0.9771
Julia	1D	**0.9924**	0.9923
Julia	2D	**0.9760**	0.9758
Julia	3D	**0.9675**	0.9671

Tabela: Regressão Linear: Pico de Memória

Linguagem	**Dimensão**	R^2	R^2_{adj}
Python	1D	**1.0000**	1.0000
Python	2D	**0.9908**	0.9907
Python	3D	**0.9661**	0.9657
Julia	1D	**1.0000**	1.0000
Julia	2D	**0.9908**	0.9907
Julia	3D	**0.9667**	0.9663

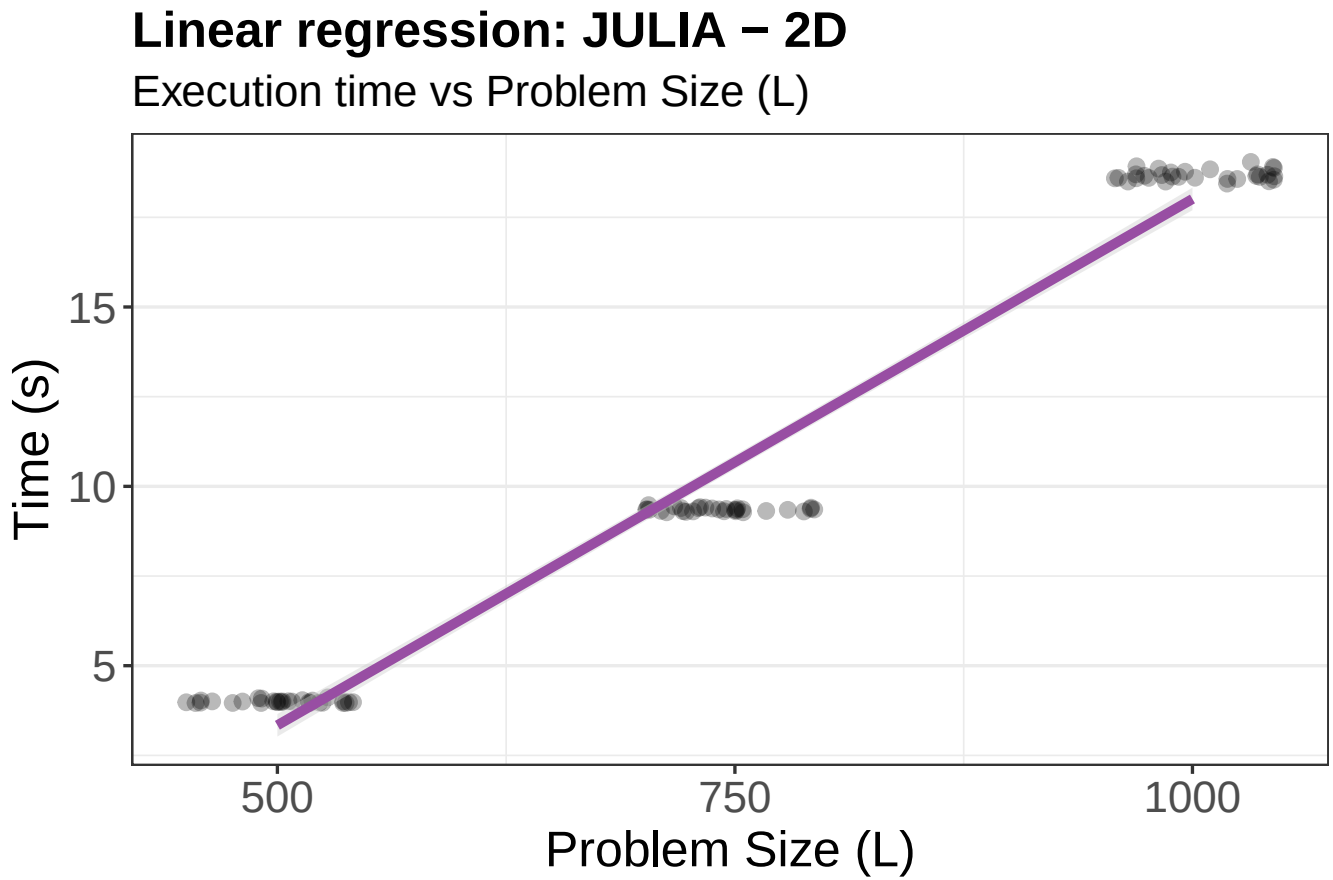
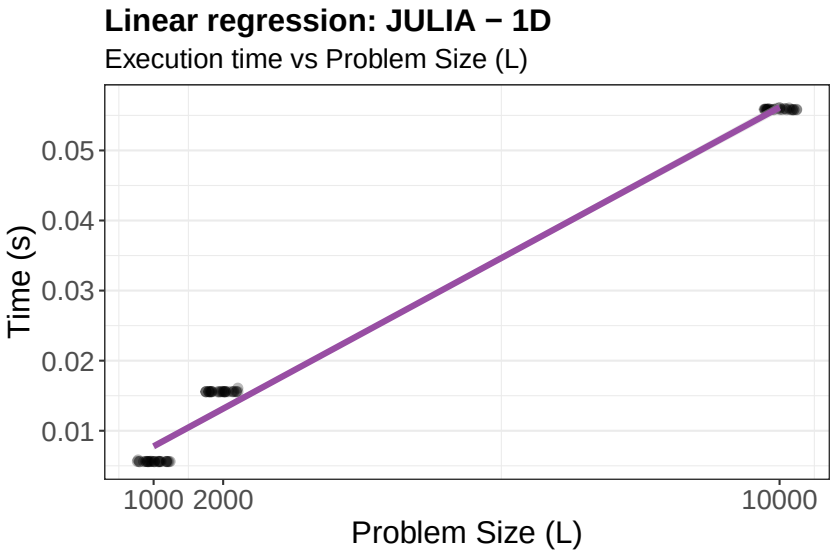
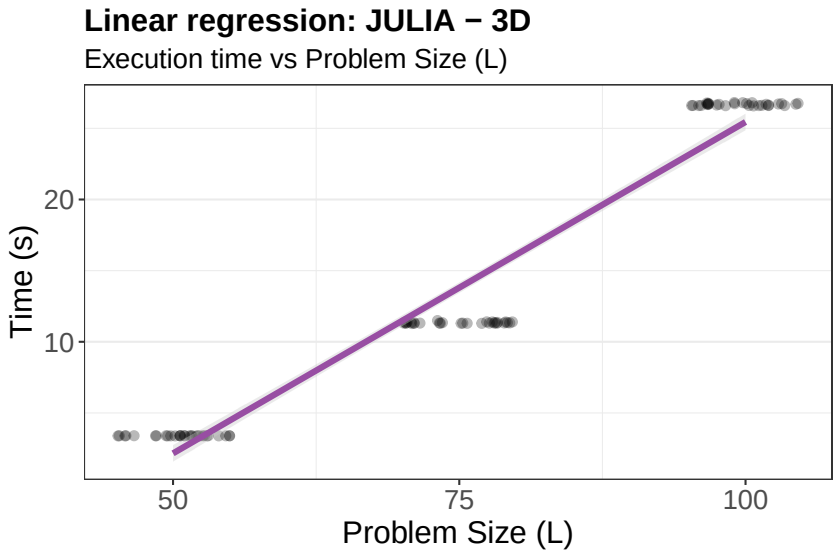


Figura: Regressão linear julia 2d

Regressão Tempo de Execução



1D



3D

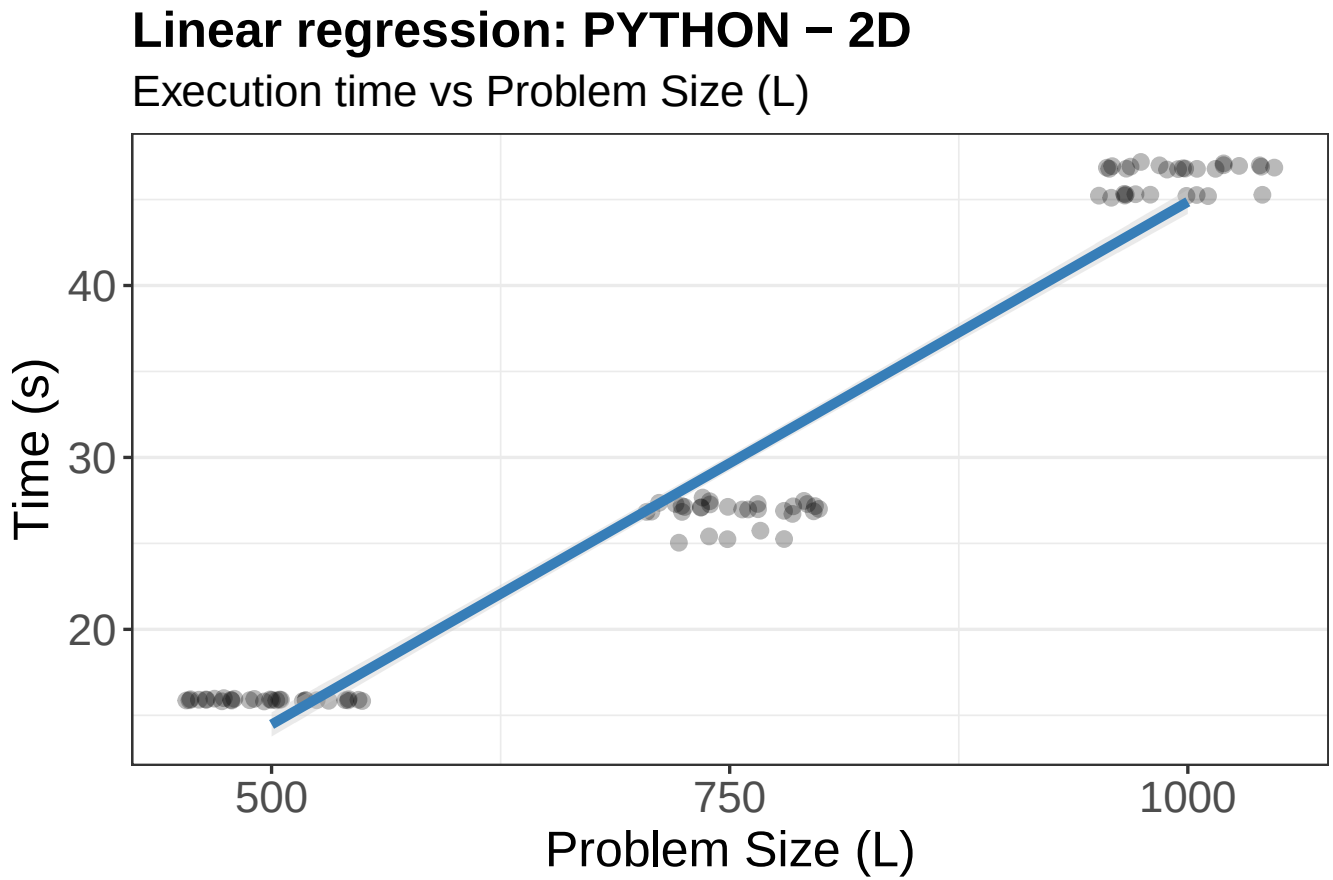
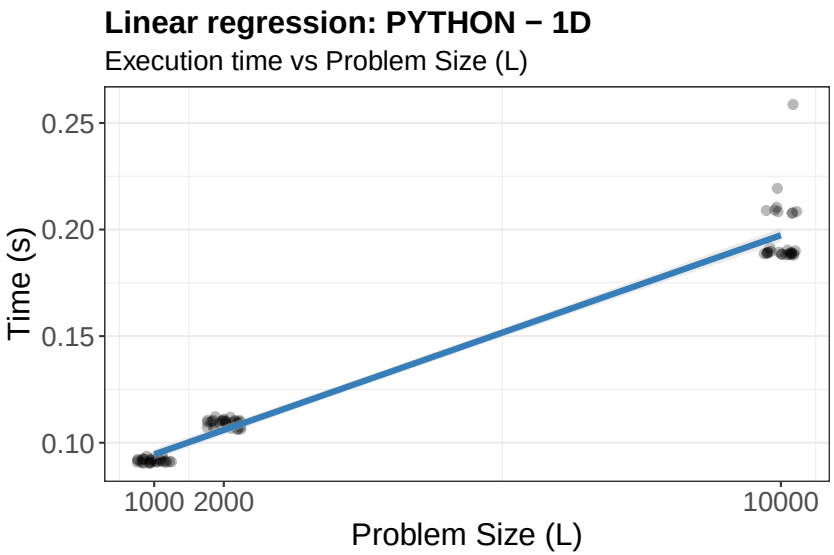
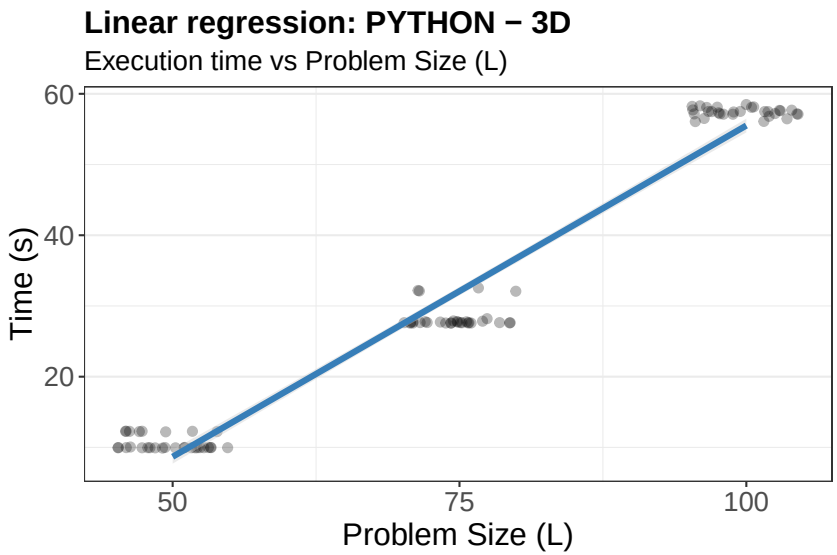


Figura: Regressão linear python 2d

Regressão Tempo de Execução



1D



3D

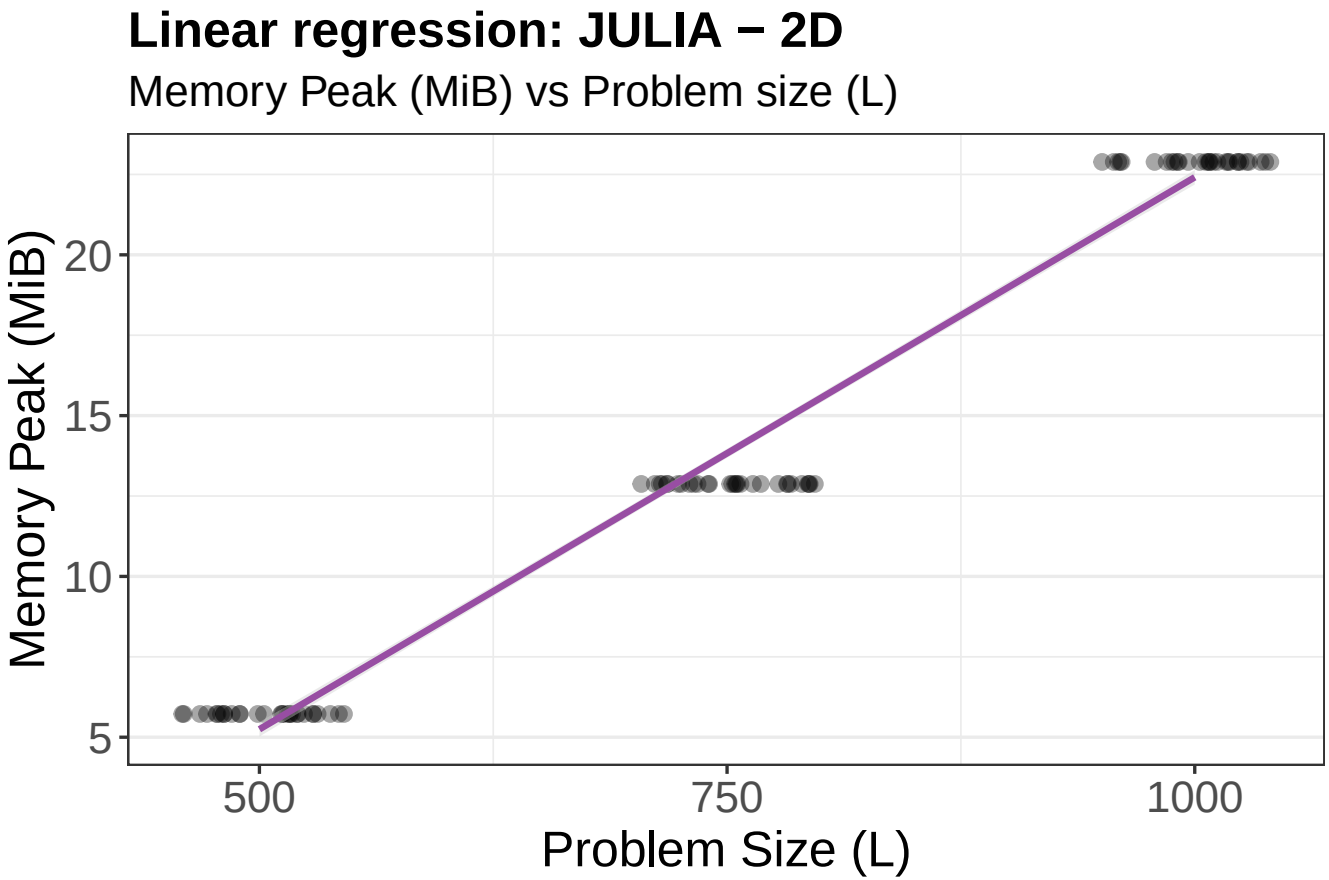
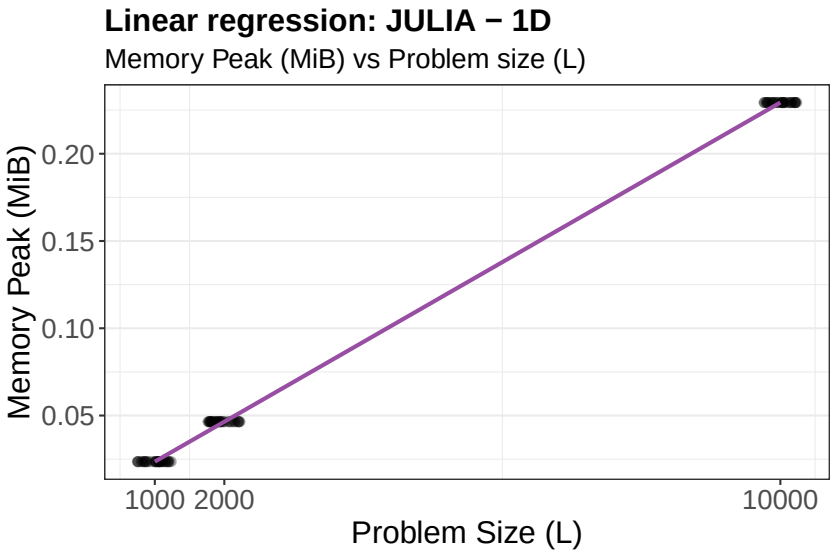
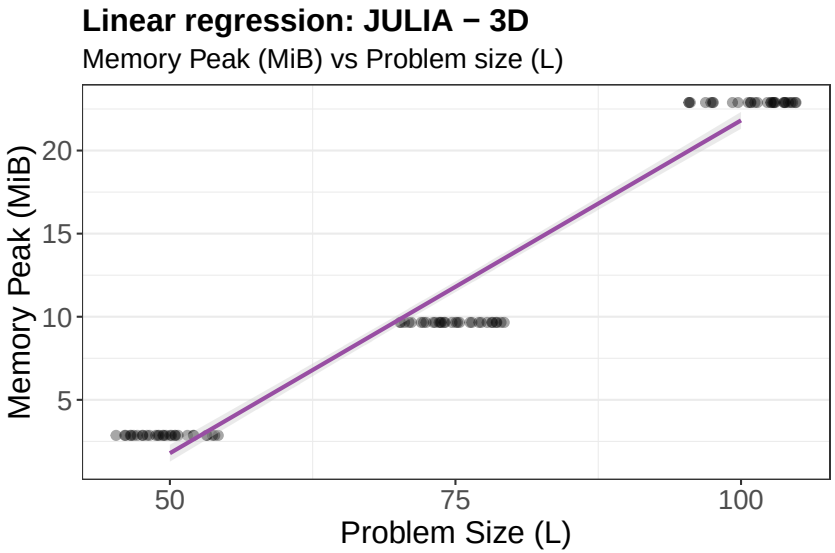


Figura: Regressão linear julia 2d

Regressão Pico de Memória



1D



3D

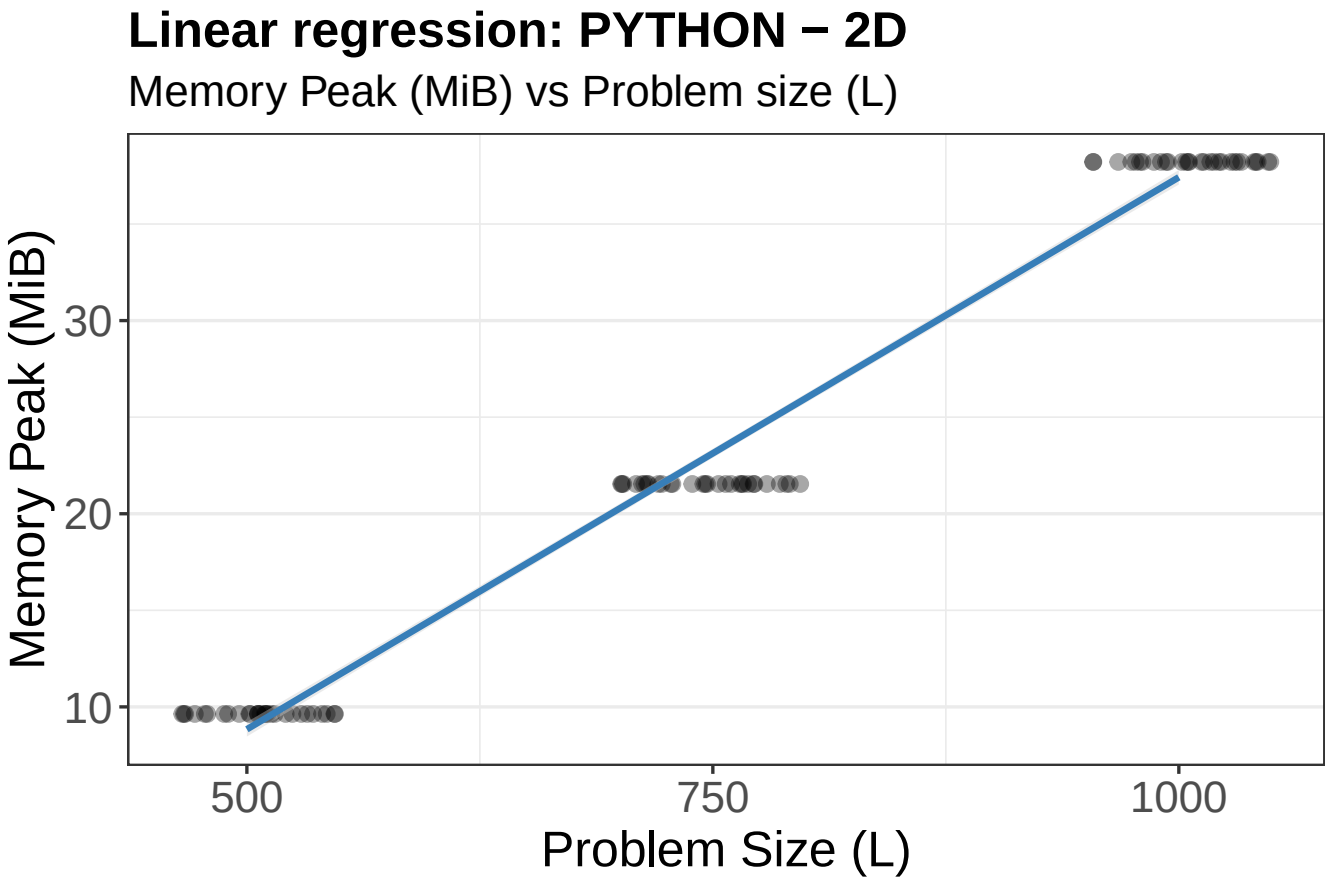
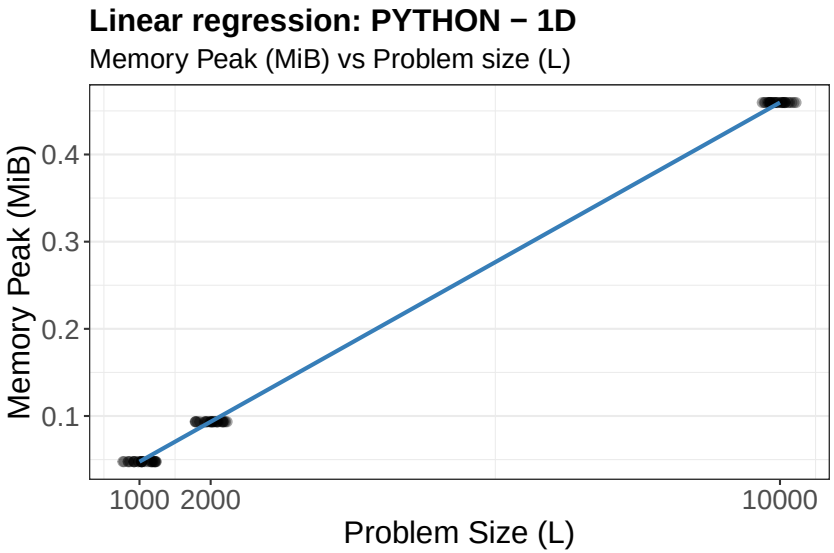
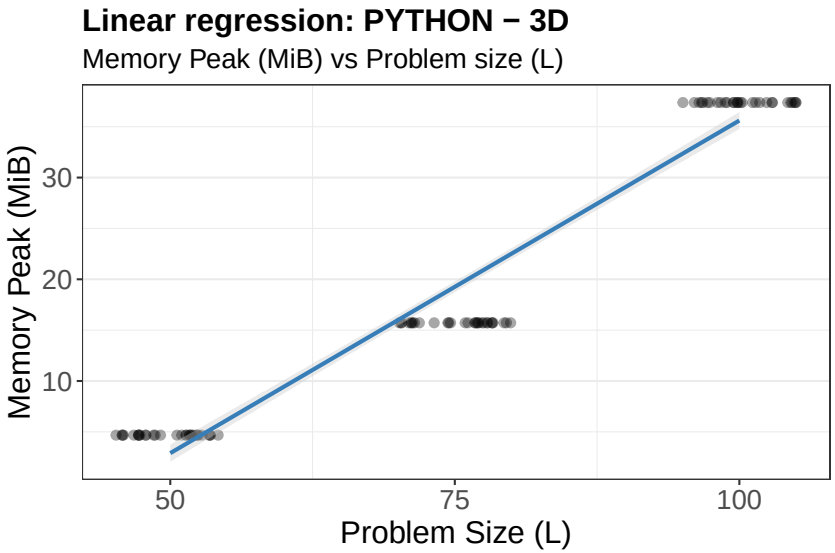


Figura: Regressão linear python 2d

Regressão Pico de Memória



1D



3D

DISCUSSÃO: O TRADE-OFF VELOCIDADE VS. MEMÓRIA

- **Desempenho Bruto:** Como esperado, Julia demonstrou ser ordens de magnitude mais rápida que Python em todas as dimensões testadas.
- **Contrapartida de Memória:**
  - Os dados mostram que **Julia consome significativamente mais memória RAM** que Python para realizar a mesma tarefa.
  - **Interpretação:** Este é o custo da velocidade. A infraestrutura de compilação *Just-In-Time* (JIT) e o gerenciamento de tipos do Julia exigem uma alocação de recursos muito mais agressiva do que o interpretador leve do Python.

DISCUSSÃO: O GARGALO DOS LOOPS EXPLÍCITOS

- **Por que Python foi lento?**
  - A implementação utilizou loops explícitos (‘for‘) iterando sobre a matriz ponto a ponto.
  - Em Python, cada iteração do loop carrega o *overhead* da tipagem dinâmica e verificação de objetos, o que torna o processamento escalar extremamente ineficiente.

- **A Vantagem de Julia:**
  - Julia foi projetada para otimizar loops. O compilador infere os tipos e gera código de máquina eficiente para as iterações, eliminando o gargalo que existe no Python puro.

DISCUSSÃO: PRODUTIVIDADE E ABSTRAÇÃO

- **O "Problema das Duas Linguagens":**
  - Python resolve a facilidade de escrita, mas exige C/C++ para performance crítica.
  - Julia resolve ambos: permite escrever código matemático de alto nível (idêntico ao Python) com a performance de linguagens compiladas.

- **Conclusão da Análise:**
  - Para algoritmos científicos que dependem de iterações complexas, Julia oferece a melhor relação entre **tempo de desenvolvimento** e **tempo de execução**, contanto que o maior uso de memória seja aceitável.

Conclusão Técnica

Para computação científica baseada em iterações explícitas (simulações físicas), Julia é a escolha superior, oferecendo desempenho próximo de C/Fortran com a facilidade de escrita de Python.

Síntese dos Resultados

O estudo evidenciou que não existe opção perfeita: o ganho de desempenho do Julia (JIT) exige uma alocação de memória agressiva. A escolha da linguagem deve ser baseada no recurso mais escasso do sistema (Tempo ou RAM).

RECOMENDAÇÕES E TRABALHOS FUTUROS

- **Melhoria de Desempenho:**
  - Implementar paralelismo (`Threads.@threads` em Julia).
- **Boas Práticas de Benchmarking:**
  - Sempre travar a frequência da CPU (*governor performance*).
  - Realizar rodadas de aquecimento (*warm-up*) antes da medição.
  - Usar Docker com *CPU Pinning* para reprodutibilidade.

OBRIGADO!

Enzo Lisbôa Peixoto, Nathan Mattes e Pedro Scholz Soares
Instituto de Informática — UFRGS
elpeixoto@inf.ufrgs.br
nmattes@inf.ufrgs.br
pedro.soares@inf.ufrgs.br