IntPy: Casos de uso e análise de performance

Alysson Geraldo Gomes da Silva
Universidade Federal Fluminense
Niterói, Brazil
alyssongomes@id.uff.br

André Luiz
Universidade Federal Fluminense
Niterói, Brazil
andrelpoj@id.uff.br

João Pedro Sá Medeiros
Universidade Federal Fluminense
Niterói, Brazil
jpsmedeiros@id.uff.br

RESUMO

O uso de scripts para experimentos científicos abriu um novo mundo de oportunidades e descobertas, mas alguns desses experimentos levam muito tempo para ser executados, e são executados diversas vezes durante o estudo, o que toma muito tempo. Para ajudar a solucionar tal problema, ferramentas como o IntPy surgiram com soluções para reduzir o tempo de execução de scripts. O objetivo deste trabalho é analisar a viabilidade do uso da ferramenta em experimentos científicos, levantando benefícios do uso da ferramenta e também seus problemas e casos em que seu uso não trás vantagens. Para realizar tal análise foi utilizada uma metodologia de coleta de dados de execução dos experimentos, que coleta e compara os dados das execuções com e sem o IntPy de forma consistente. O estudo levantou casos onde o uso da ferramenta trouxe até 81.37% redução no tempo de execução em relação ao Python convencional e casos em que houve também melhoria no uso de memória RAM durante a execução, chegando a economizar até 9.06% em relação à execução utilizando o Python convencional para os experimentos estudados.

CCS Concepts

• **Theory of computation→** Caching and paging algorithms**;**
• **Software and its engineering→** Scripting languages;

Keywords

memoization; python; caching; experiments; intpy

1. INTRODUÇÃO

Cientistas têm utilizado cada vez mais scripts para a realização e auxílio de experimentos científicos[1]. Seja para a realizar um experimento completo ou somente para facilitar a análise de dados de outros experimentos, os scripts estão sendo cada vez mais utilizados pela sua praticidade e alto poder expressivo, onde o uso de bibliotecas e outras ferramentas auxiliam os cientistas em diversos aspectos[1]. Dentre o ferramental disponível para o auxílio dos experimentos, encontram-se as ferramentas que diminuem o tempo de execução dos scripts por meio de memorização das funções, dentre as quais se destacam IncPy[2] e IntPy[3].

Algumas ferramentas tal como o sistema de workflow Vistrails[4] e o interpretador modificado de Python IncPy fornecem a memorização de forma transparente, mas, enquanto este se limita à uma versão específica da linguagem Python, aquela adiciona a complexidade e limitação de um novo ambiente para desenvolvimento e execução de experimentos. Em alguns casos, cientistas recorrem ao uso de arquivos com resultados intermediários escritos manualmente, visando poupar processamento para a reexecução de experimentos[2]. Entretanto, tal estratégia adicionada a necessidade de controle manual sobre a escrita e leitura dos dados.

Para auxiliar na memorização de resultados intermediários e não sobrecarregar o cientista com a necessidade de se controlar o ciclo de vida dos arquivos intermediários, a ferramenta IntPy surge para fornecer uma estratégia simples e controlada de implementar a memorização, podendo ser executada script em diferentes versões de Python.

Este trabalho visa analisar o IntPy de maneira a destacar em que cenários o seu uso traria vantagens na execução do experimento, e em quais não traria. Para realizar tal estudo, foram executados diversos experimentos, de naturezas distintas com intuito de mapear de forma prática e aplicada alguns dos possíveis caso de uso da ferramenta. Durante as execuções, métricas foram coletadas e compiladas, permitindo a análise da viabilidade do uso da ferramenta para cada um dos casos.

Pode-se observar após a execução dos experimentos que existe perda de desempenho nos casos onde o processamento da função anotada é muito simples e veloz, por mais que sejam realizadas diversas chamadas. Mas em casos onde há muito processamento para cada chamada da função, tem-se um ganho de desempenho considerável. Pode-se verificar também que apesar da ferramenta de propor a otimizar apenas tempo de execução de scripts, em alguns casos também havia economia na utilização de memória RAM.

2. INTPY

2.1 Definição e exemplo

IntPy é uma ferramenta para viabilizar a memorização de resultados de funções de forma simples. Para usar realizar a memorização da função, é necessário somente anotar a função ou método desejado e, a partir dos valores de entrada, o retorno da execução será guardado para ser acessado em execuções futuras.
Tal comportamento é possível, pois o IntPy foi desenvolvido de forma a interceptar scripts[3], portanto tendo-se uma função determinística a única ação necessária por parte do cientista, para que o seu programa passe a ser executado utilizando a ferramenta é anotar essa função com a anotação *@deterministic*. A Figura 1 demonstra o uso da anotação do IntPy.

```
from src.intpy import deterministic

@deterministic
def fib(n):
    if n < 2:
        return n

    return fib(n-1) + fib(n-2)

```

Figura 1: Exemplo de utilização da anotação do IntPy (@deterministic) para a função fib.

Na execução do código acima, o resultado da função fib será guardado na primeira execução. Nas execuções subsequentes, o valor será recuperado do cache ao invés de ser reprocessado. No caso da primeira execução ser a chamada da função com n = 3, uma chamada subsequente fib(5) usará os dados gerados pela função fib(3) guardados no cache, não necessitando reprocessar os dados de fib(2) e fib(1) que seriam chamados a partir de fib(3).

2.2 Problemas encontrados

Um dos problemas encontrado na versão inicial da ferramenta foi a falta de compatibilidade com o Python 2 por causa do uso da função getfullargspec(), que só foi definida no python 3. Para contornar este problema, foi necessário alterar a chamada desta função para a função getargspec(), que é compatível com Python 2. Tal substituição foi possível pois o uso da função se limitava a descobrir se a chamada era executada a partir de um objeto ou não, ou seja, definir se era função ou método.

Outro problema encontrado foi o de colisões na criação do cache. Durante alguns experimentos, havia erro na checagem da existência de um dado no cache e isso gerava uma tentativa de inserção de um dado que já existia. Foi adicionada uma checagem de existência no método de criação de registro no cache para evitar este problema.

Também foi encontrado um problema na execução da função fib para um n inicial muito grande. A ferramenta gerava um erro de erro no empilhamento de chamadas para verificar o cache. Este problema não foi corrigido, pois não ocorreu durante as execuções dos experimentos.

Ao tentar executar o experimento *Menger Sponge Slice* a utilização do IntPy aumentou o tempo de execução tão consideravelmente que cancelamos a execução. Não sabemos com certeza se o script terminaria de executar, porém a execução sem o uso do IntPy levava apenas 5 segundos.

3. EXPERIMENTOS REALIZADOS

3.1 Metodologia

Para realizar o estudo proposto sobre casos de uso e análises de performance quanto a utilização do IntPy adotamos metodologias para a obtenção de dados consistentes entre os diversos experimentos. Todos experimentos foram executados seguindo os padrões descritos abaixo.

Utilizamos duas configurações de máquina diferentes para rodar os experimentos. A primeira utilizando máquinas virtuais, hospedadas na AWS(Amazon Web Services), com as seguintes características: Sistema operacional Ubuntu Server 18.04, 1Gb de memória RAM e um processador Intel Xeon. A segunda utilizando uma máquina local devido ao aumento de complexidade para executar experimentos com interface gráfica, com as seguintes características: Sistema operacional Ubuntu 16.04, 4GB de memória RAM, Intel ® Core(™) i5-3230M CPU @ 2.60GHz. A máquina da AWS foi usada nos experimentos 3.1, 3.2, 3.3 e 3.4 a máquina local foi usada no experimento 3.5, por necessitar de um display para a execução.

Para evitar que anomalias e variações decorrentes do sistema tivessem impacto sobre os resultados finais do experimento foram realizadas dez execuções de cada experimento, com e sem anotações do IntPy, gerando ao final a média das execuções.

Foram coletados para realizar as análises de performance, o tempo de execução e a média de memória RAM utilizada pelo processo do Python durante o tempo da execução. Apesar da proposta do IntPy ser otimizar o tempo de execução de scripts, também foi analisada a utilização de memória RAM, com o intuito de evidenciar novos benefícios ou malefícios da ferramenta. Para a coleta da memória, foi criada uma ferramenta que pode ser encontrada em *https://github.com/AlyssonG/mem_monitor*.

Antes da realização de cada experimento, foi verificada e anotada na tabela de captura as versões do Python ou do IntPy que foram utilizadas. As versões dos experimentos tanto modificadas com IntPy quanto sem as modificações podem ser encontradas em *https://github.com/jpsmedeiros/IntPy-experiments*.

3.2 Distribuição de calor

Este experimento, tem como objetivo avaliar a evolução da distribuição de calor em uma placa quadrada, através de uma equação de difusão pura. Tal problema exige que sistemas lineares sejam resolvidos para que sua solução seja encontrada, neste caso foi utilizado o método de fatoração LU para solucionar tais sistemas.

Apenas o método da fatoração LU foi anotado, e portanto os resultados observados para esse experimento se dão ao uso do IntPy para a memorização resultados de funções determinísticas que realizam processamento sobre matrizes.

Pode-se verificar que o ganho obtido quanto ao tempo de execução no uso do IntPy em relação ao Python convencional, para este caso, foi de 81.37%, e o ganho na média de memória RAM utilizada na execução, foi de 9.06%. Os resultados obtidos em cada execução e a média podem ser observados na Figura 2 para o tempo de execução, e na Figura 3 para a utilização de memória RAM.

Podemos concluir, portanto que o uso da ferramenta para o caso abordado neste experimento é extremamente vantajoso, obtendo uma diferença muito grande no tempo de execução e também uma economia no uso de memória RAM. Em especial, verificando os resultados encontrados na Figura 2, podemos observar que a primeira execução do IntPy possui uma diferença bem pouco notável no tempo de execução em relação às demais (Aproximadamente 1 segundo). Isso acontece devido a estruturação do experimento, que utiliza a cada passo os resultados obtidos no laço anterior, portanto desde a primeira execução, os resultados memorizados pela ferramenta já são utilizados.

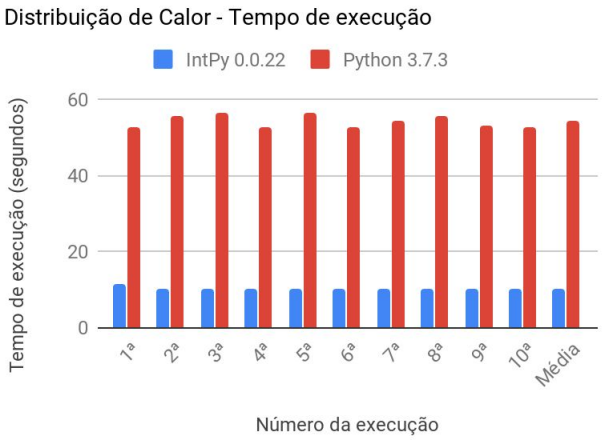

Figura 2: Comparação do tempo de execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de Distribuição de calor.

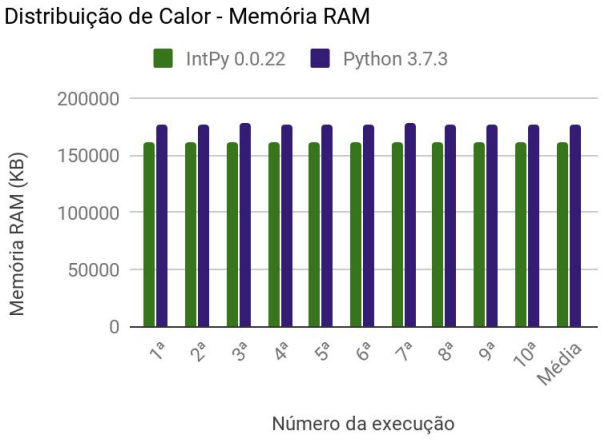

Figura 3: Comparação do uso de memória RAM durante a execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de Distribuição de calor.

3.3 Permutação utilizando Algoritmo de Heap

O objetivo desse experimento é encontrar através do Algoritmo de Heap todas as possíveis combinações de resultados (permutação) para a organização de um array de números inteiros. O Algoritmo de Heap já foi considerado como o mais eficaz para gerar permutações[5].

As operações realizadas a cada laço são executadas com muita rapidez, devido ao algoritmo minimizar as operações realizadas e pela própria natureza do problema. Mas apesar da execução de cada laço ser muito rápida, o algoritmo possui complexidade fatorial, portanto, utilizando um array com 6 números, teremos 720 chamadas a função.A função anotada para este experimento realiza operações de troca de posição de elementos de um array diversas vezes.

Este é um exemplo claro que nem sempre a utilização do IntPy irá trazer benefícios. Neste caso tivemos uma piora de 922780% do tempo do Python em relação ao tempo obtido com o IntPy, e uma piora de 176.3% no uso da memória RAM. Os resultados obtidos em cada execução e a média podem ser observados na Figura 4 para o tempo de execução, e na Figura 5 para a utilização de memória RAM.

O eixo vertical do gráfico da Figura 4 utiliza a escala logarítmica para permitir a visualização dos dados, devido a grande diferença entre os resultados obtidos com e sem o uso da ferramenta.

Podemos concluir, portanto que a utilização do IntPy para casos onde a função realiza operações de maneira muito veloz não é vantajosa. O tempo gasto com o acesso aos dados armazenados em cache acaba por ser maior do que o tempo de executar a operação de fato, valendo mais a pena realizar as operações novamente, do que ler os valores armazenados. A memória gasta com as operações necessárias para a execução das funções de leitura e escrita de cache acabam também por aumentar o consumo médio de memória RAM durante a execução.

Neste experimento podemos observar que o tempo da primeira execução utilizando o IntPy é muito mais alto do que o das execuções seguintes. Isso se dá ao fato do IntPy nesta primeira execução, precisar criar o banco de dados para armazenamento do cache, executar a função e então realizar a escrita dos resultados dela. Nas execuções seguintes, o banco já existe e os valores já estão escritos, o tempo gasto é apenas de consulta dos dados.

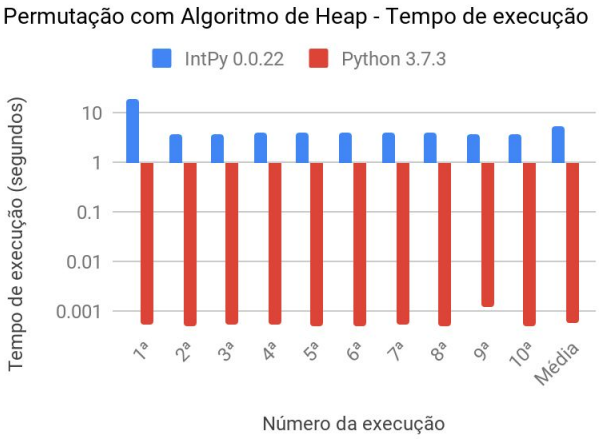

Figura 4: Comparação do tempo de execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de Permutação com Algoritmo de Heap.

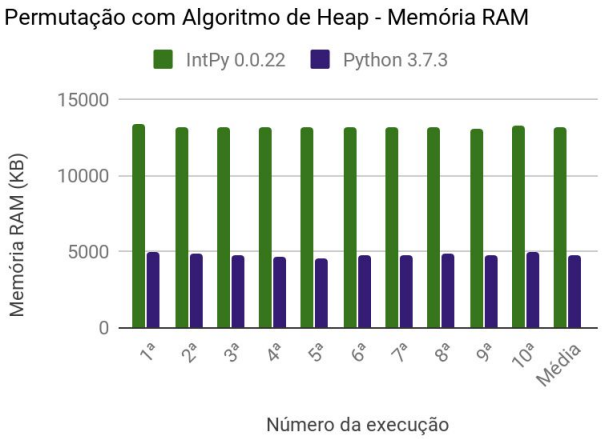

Figura 5: Comparação do uso de memória RAM durante a execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de Permutação com Algoritmo de Heap.

3.3 RAD-Seq

Este experimento foi modificado para se testar o uso do IntPy. Nele, um arquivo inicial de entrada é dado, junto com a sequência de base nitrogenadas para as buscas e o script busca a sequência dentre os dados do arquivo de entrada. Na primeira execução do experimento, o script gerava arquivos intermediários para acelerar a execução das próximas execuções. O uso de arquivos intermediários foi removido e o programa passou a ler o arquivo de entrada todas as vezes para buscar o dado.

A leitura inicial do arquivo e a leitura dos dados foram divididas em funções que aparentemente favorecem o uso do IntPy, mas encontramos uma piora tanto no uso de tempo de execução quanto no uso de memória.

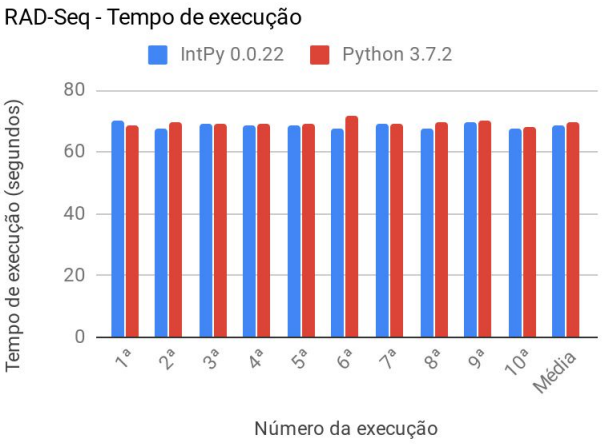

Figura 6: Comparação do tempo de execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de RAD-Seq.

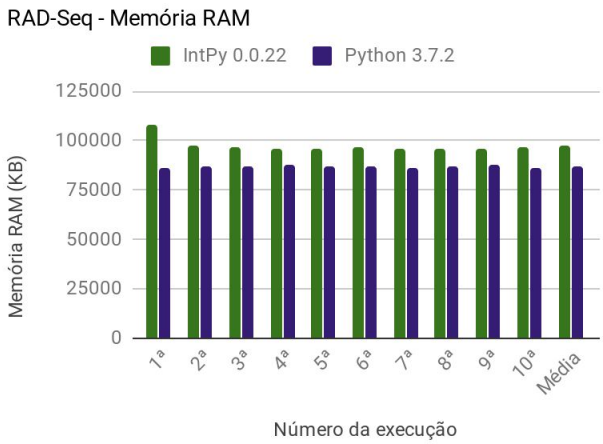

Figura 7: Comparação do uso de memória RAM durante a execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de RAD-Seq.

3.4 Simulador de Diversidade

Este experimento tem como objetivo encontrar a quantidade de interações gerada por conjuntos a partir de uma lista aleatória.

Pelo caráter totalmente determinístico, pelo uso de uma mesma semente para geração de números aleatórios, do experimento, o uso do IntPy em execuções sucessivas se mostrou extremamente vantajoso. Embora o uso de memória tenha se mantido constante, o tempo de execução do experimentos foi reduzido em 240x, pois se limitou à uma chamada no cache. A quantidade de memória utilizada se manteve bem próxima da versão do script sem a ferramenta, pois a geração da lista ainda ocorria e ela ocupa a maior parte do uso da memória.

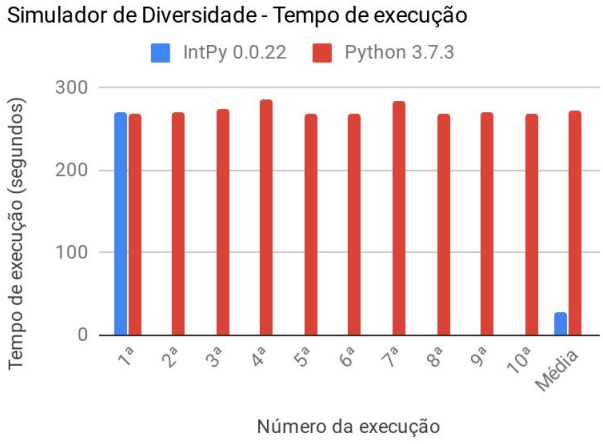

Figura 6: Comparação do tempo de execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de Simulador de diversidade.

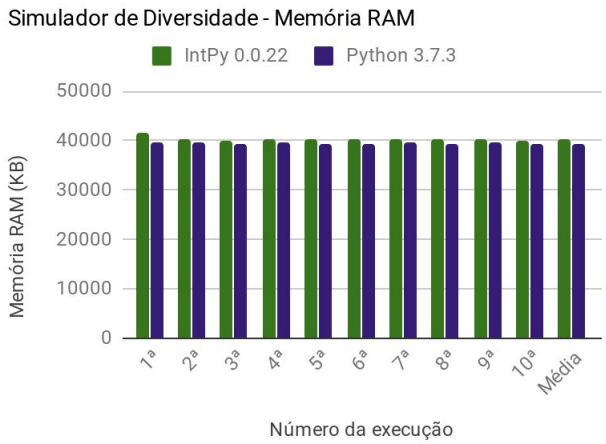

Figura 7: Comparação do uso de memória RAM durante a execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de Simulador de diversidade.

3.5 Quantum Harmonic Simulator

Este experimento, tem como objetivo simular um oscilador harmônico quântico com a interação entre dois elétrons.Os resultados observados para esse experimento se dão ao uso do IntPy para a memorização dos resultados de funções determinísticas que realizam processamento sobre pontos no espaço.

Pode-se verificar que o IntPy obteve um tempo 241% mais lento do que a versão sem a ferramenta e um ganho de apenas 0.36% no uso de memória o que poderia ser facilmente descartado. Os resultados obtidos em cada execução e a média podem ser observados na Figura 8 para o tempo de execução, e na Figura 9 para a utilização de memória RAM.

Nesse experimento não obtivemos ganho, pois o número de vezes em que o armazenamento do resultado de funções foi utilizado não compensou o custo de seu armazenamento.

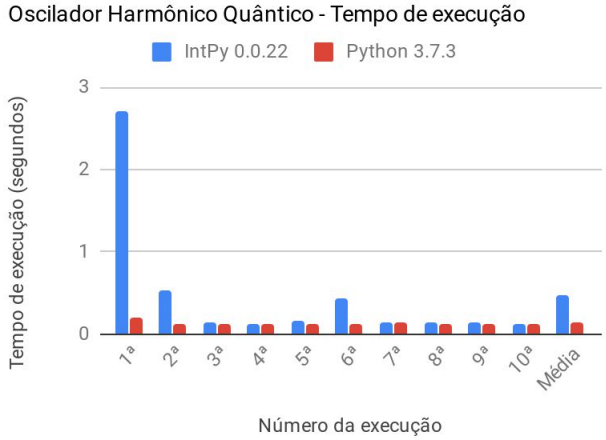

Figura 8: Comparação do tempo de execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de oscilador harmônico quântico.

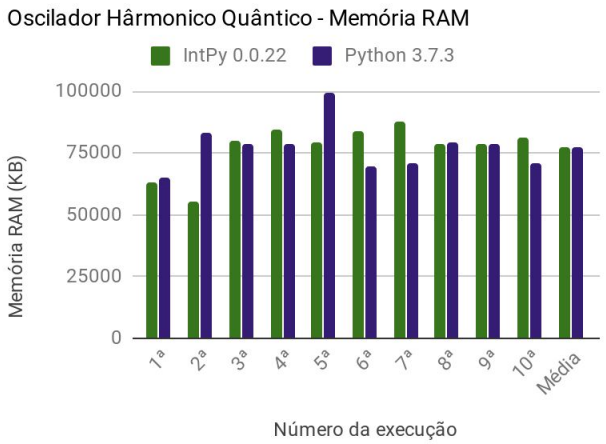

Figura 9: Comparação da média de uso de memória RAM durante a execução do algoritmo utilizando o IntPy 0.0.22 e utilizando o Python 3.7.3 para o experimento de oscilador harmônico quântico.

4. CONCLUSÕES

Conseguimos com os experimentos explorados durante o desenvolvimento deste trabalho entender melhor o funcionamento prático do IntPy e verificar que outras métricas além do tempo de execução, como o uso de memória RAM, podem também ser otimizadas com o uso da ferramenta.

Funções com execução muito veloz podem acabar demorando mais para executar utilizando o IntPy, devido ao tempo de leitura dos dados memorizados ser maior do que o tempo de executar novamente. Porém, em funções onde há muito processamento sobre uma estrutura, como operações com matrizes, pode-se melhorar muito o tempo de execução gasto, atingindo assim o propósito da ferramenta. Portanto nem todas as funções serão beneficiadas pela memorização de seus resultados, pelo contrário, muitas podem ter um tempo de execução maior do que quando não utilizada a ferramenta, cabendo ao cientista avaliar e anotar apenas aquelas que irão trazer benefícios para seu experimento.

O uso de orientação a objetos também não favorece o uso de memorização a partir do IntPy, pois muitas vezes o único argumento da função é o próprio objeto e os valores são obtidos a partir dele. A ferramenta não consegue lidar com igualdade entre objetos e, por isso, o uso de memorização em experimentos organizados sob este paradigma não foram possíveis.

Outras métricas poderiam ser avaliadas, com intuito de explorar mais os benefícios (ou malefícios) que o uso da memorização de resultados pode trazer para a execução de scripts, tais como taxa de escrita e leitura de disco, espaço de armazenamento utilizado, uso de CPU, entre outras.

5. REFERÊNCIAS

[1] Marta Mattoso, Cláudia Werner, G Travassos, Vanessa Braganholo, Leonardo Murta, Eduardo Ogasawara, F Oliveira, and Wallace Martinho. 2009. Desafios no apoio à composição de experimentos científicos em larga escala. Seminário Integrado de Software e Hardware, SEMISH 9 (2009), 36.

[2] Philip J Guo and Dawson Engler. 2011. Using automatic persistent memoization to facilitate data analysis scripting. In

Proceedings of the 2011 International Symposium on Software Testing and Analysis. ACM, 287–297.

[3] Leonardo Fiório, Mariana Teixeira, and Max Fratane. 2018. IntPy: Um Interceptador para Python. In Proceedings of eScience. ACM, New York, NY, USA, Article 4, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[4] Claudio T. Silva, Juliana Freire, and Steven P. Callahan. 2007. Provenance for Visualizations: Reproducibility and Beyond. *Computing in Science & Engineering* 9, 5 (2007), 82–89. DOI:http://dx.doi.org/10.1109/mcse.2007.106

[5] Robert Sedgewick. 1977. Corrigenda: Permutation Generation Methods. *ACM Computing Surveys* 9, 4 (1977), 314. DOI:http://dx.doi.org/10.1145/356707.356712