Relatório do Lab1 de CCI-22

Trabalho 01 – Ambientação com o MATLAB

Aluno:

Bruno Costa Alves Freire

Turma:

T 22.4

Professor:

Marcos Ricardo Omena de Albuquerque Máximo

Data de realização do experimento: 11/03/2019

Instituto Tecnológico de Aeronáutica – ITA
Departamento de Computação

1. Produto Interno

Foram testados 2 métodos de realizar um produto interno entre dois vetores:

1) Utilizando o operador de multiplicação matricial: `pi = x' * y`;
2) Fazendo um laço `for` explícito:

$$\langle x, y \rangle = \sum_{i=1}^{N} x(i) \cdot y(i)$$

Para cada método, foram realizadas 1000 medições utilizando vetores de 10^7 elementos, gerados aleatoriamente pelo MATLAB. Medindo os tempos de cálculo dos três métodos, foram gerados os seguintes gráficos através do script `produtointerno.m`:

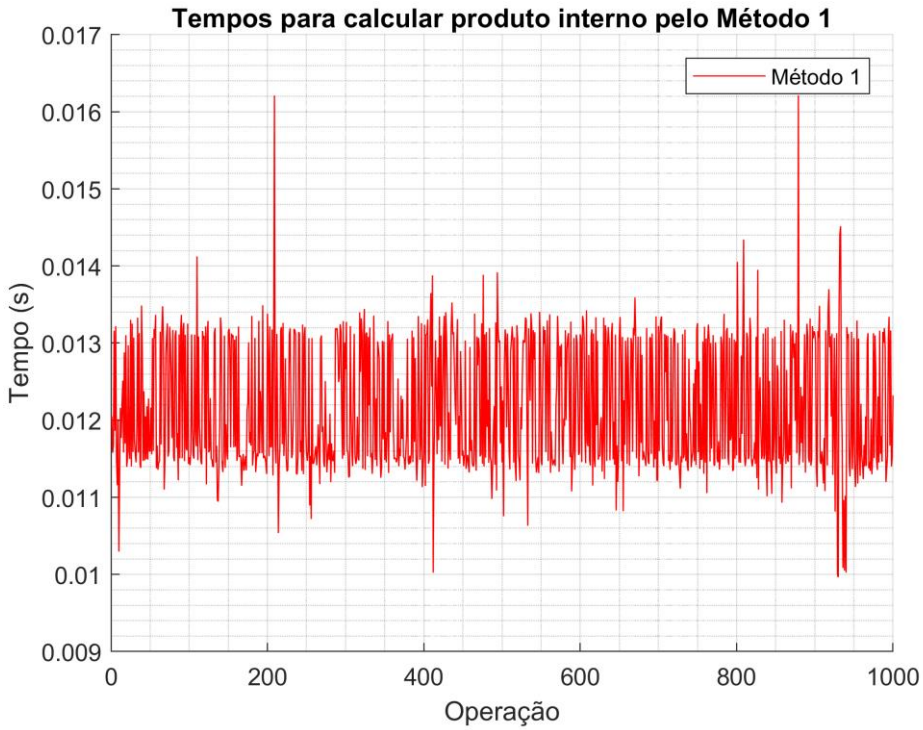

Figura 1: Gráfico do tempo de execução para o método 1

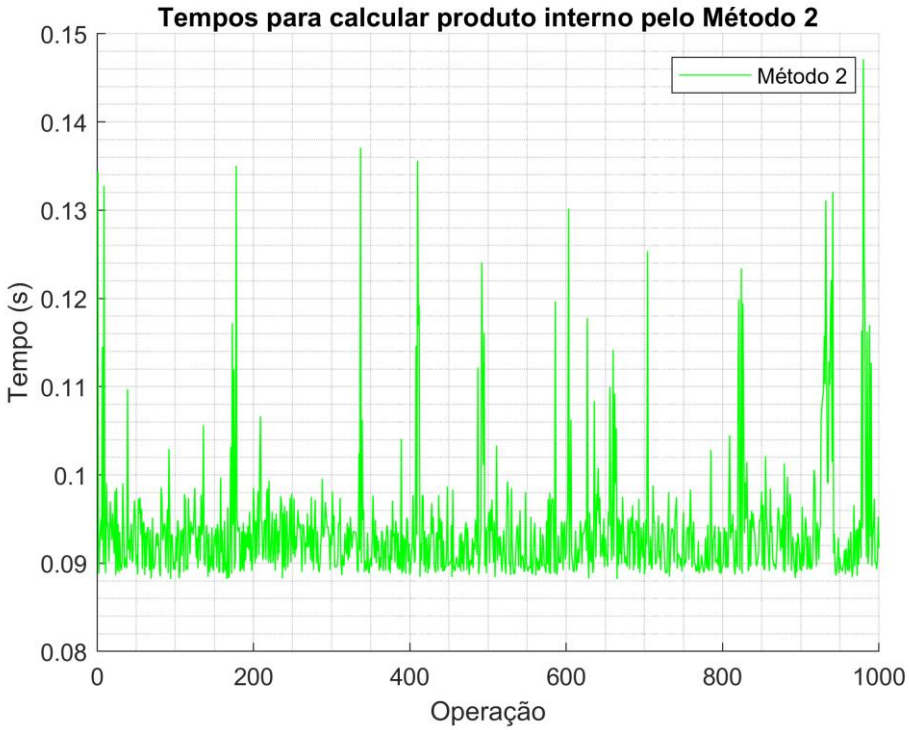

Figura 2: Gráfico do tempo de execução para o método 2

Para amenizar o aspecto ruidoso das medidas, plotou-se as médias dos tempos de execução para os dois métodos:

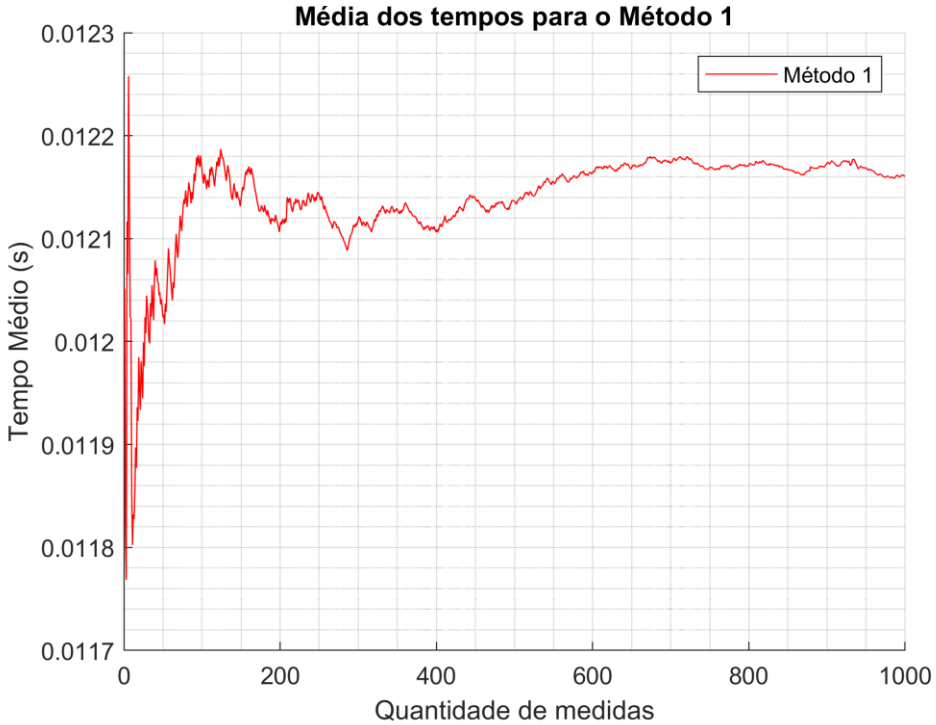

Figura 3: Médias dos tempos x quantidade de medidas para o método 1

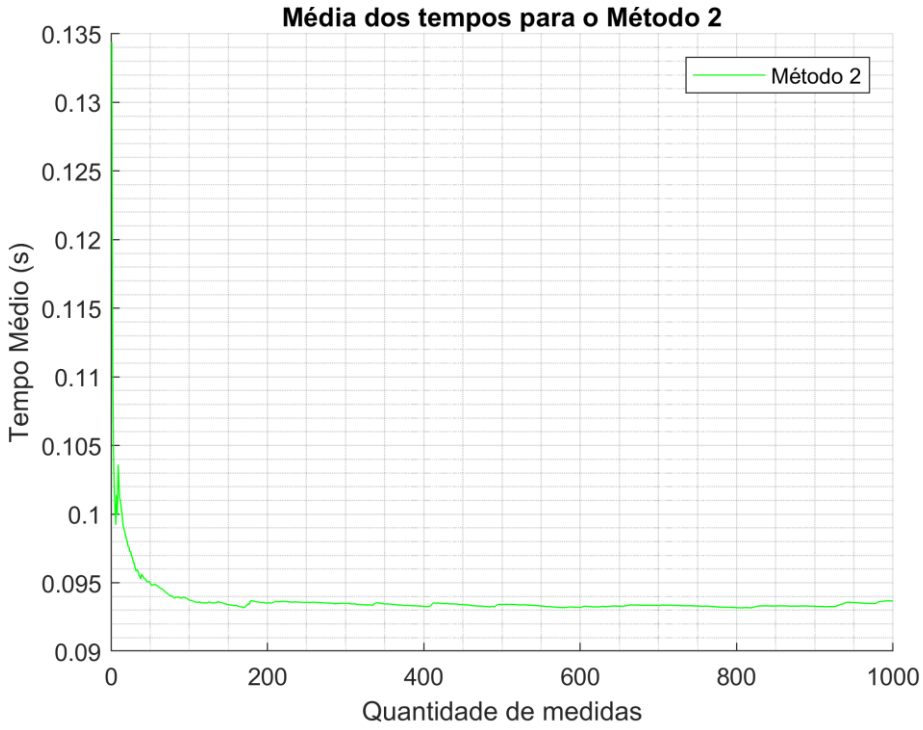

Figura 4: Médias dos tempos x quantidade de medidas para o método 2

Podemos observar através da escala de tempo de cada gráfico que o método 1 leva tempos consideravelmente menores do que o método 2 para realizar a tarefa de computar um produto interno. Portanto, podemos concluir que o uso da sintaxe nativa do MATLAB é a melhor maneira de realizar esse tipo de operação.

Para obter uma comparação justa, os dois métodos utilizaram os mesmos vetores aleatórios em uma dada medição. Os valores aproximados para das médias (para 4 dígitos significativos no primeiro método e para 2 no segundo) estão na tabela 1:

Tabela 1: Média final dos tempos

Método	Tempo médio (s)
`pi = x' * y`	0,01216
$pi = \sum_{i=1}^{n} x(i) * y(i)$	0,094

Como se pode ver pela tabela, a sintaxe para a operação matricial é muito mais eficiente do que laços de repetição.

2. Produto de Matrizes

Foi feita uma comparação entre as seguintes formas de calcular o produto de duas matrizes quadradas de ordem *N*:

1) Implementando laços `for` aninhados em para calcular cada elemento da matriz em C/C++:

$$Z(i,j) = \sum_{k=1}^{N} X(i,k) \cdot y(k,j)$$

2) Implementando o mesmo algoritmo descrito acima em MATLAB;
3) Utilizando o operador de multiplicação matricial: `Z = X * Y`;

Mediu-se o tempo de execução de cada implementação para *N* = 10, 100 e 1000 através dos *scripts* `matrixmult.m` e `matrixmult.c`. Os valores de tempo se encontram na tabela 2.

Tabela 2: Tempos de execução (em s) de duas matrizes quadradas de ordem N

	N = 10	N = 100	N = 1000
Iteração em C/C++	$8.4521 \cdot 10^{-6}$	$8.2486 \cdot 10^{-3}$	$1.1423 \cdot 10^{1}$
Iteração em MATLAB	$2.6564 \cdot 10^{-5}$	$1.4579 \cdot 10^{-2}$	$1.5902 \cdot 10^{1}$
Sintaxe MATLAB	$1.9319 \cdot 10^{-5}$	$1.8836 \cdot 10^{-4}$	$8.3237 \cdot 10^{-2}$

Comparando os dois métodos em MATLAB, vemos que a abordagem iterativa é muito mais lenta que a abordagem direta por meio da sintaxe nativa do MATLAB, diferindo por algumas ordens de grandeza conforme *N* aumenta. A iteração em C++, por outro lado, inicialmente se apresenta mais eficiente que os dois métodos em MATLAB, mas quando *N* cresce, tem seu desempenho reduzido e comparável ao mesmo método implementado em MATLAB.

Isso nos permite observar duas coisas: a linguagem C/C++, por ser mais baixo nível, é executada mais rapidamente que os *scripts* MATLAB. No entanto, para operações matriciais, a sintaxe nativa do MATLAB é extremamente otimizada, necessariamente a nível de hardware, de modo que supera facilmente a implementação iterativa mesmo em C/C++. Concluímos, finalmente, que é melhor evitar laços iterativos explícitos em MATLAB sempre que possível, visando aproveitar a otimização proporcionada pela sintaxe nativa do MATLAB.

3. Ordenação

Foram implementados como funções do MATLAB os algoritmos de ordenação MergeSort e BubbleSort, nos *scripts* `mergesort.m` e `bubblesort.m`, respectivamente.

Para mostrar o funcionamento correto das implementações, foram produzidas as figuras 5 e 6, que mostram num mesmo gráfico a disposição dos elementos de um vetor aleatório antes e depois de ser ordenado.

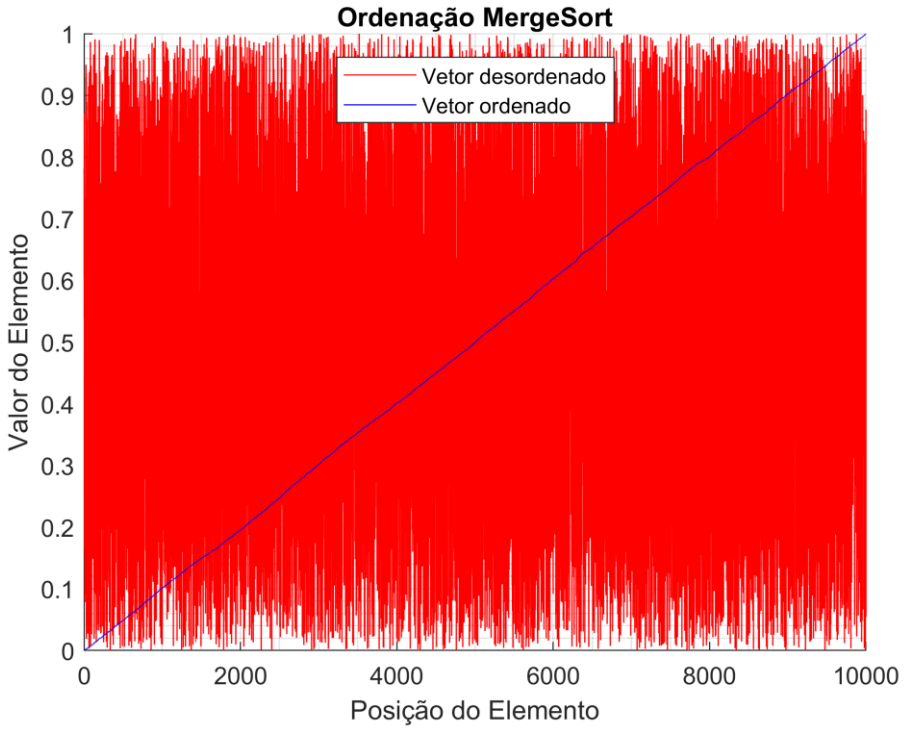

Figura 5: Vetor ordenado pelo MergeSort

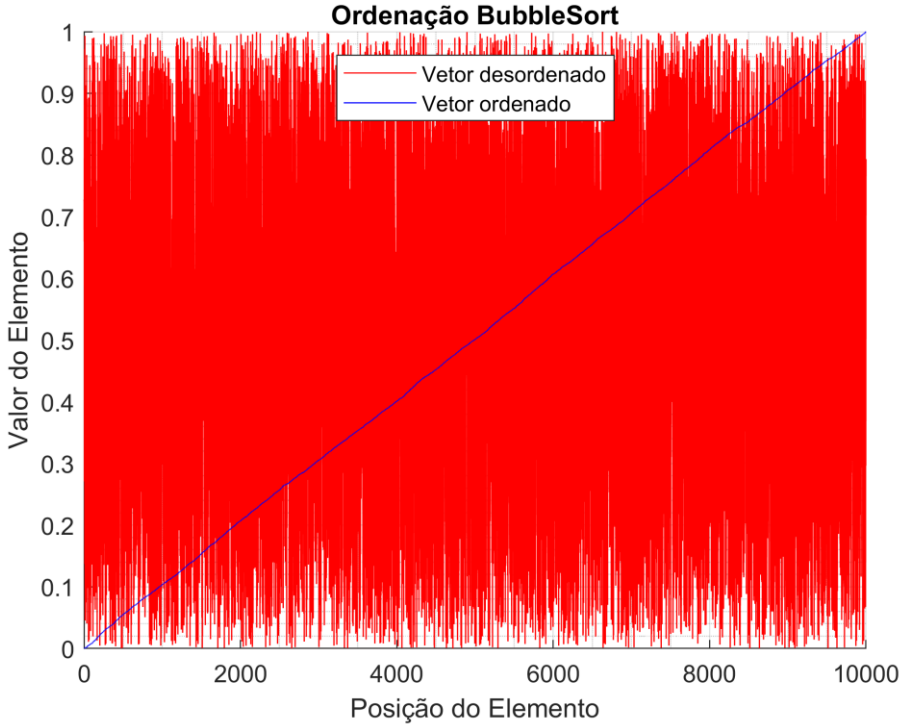

Figura 6: Vetor ordenado pelo BubbleSort

A ordenação dos vetores pelos dois métodos e a produção dos gráficos está implementada no *script* `comparaordenacao.m`.

Em seguida, o objetivo é mostrar como o tempo de execução varia com o tamanho da entrada para cada algoritmo, e depois comparar o desempenho de ambos os métodos com a função `sort` do MATLAB. No mesmo *script* `comparaordenacao.m`, foram medidos os tempos de execução dos três algoritmos, para vetores de tamanhos variando de 50 em 50, até 10000. Para cada vetor, foram realizadas 10 medições e em seguida foram tomadas as médias dessas medidas, numa tentativa de amenizar o ruído das medições. Com essas medições, foi produzido o gráfico da figura 7:

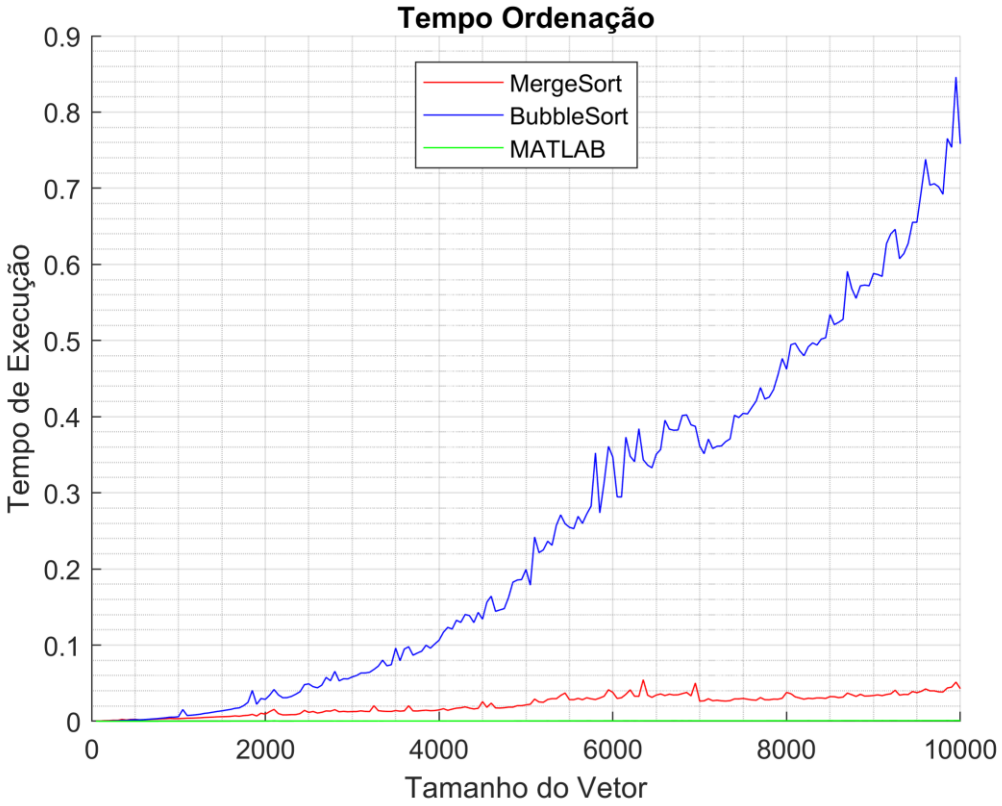

Figura 7: Tempos dos algoritmos de ordenação

Através da figura 7 é possível constatar o crescimento de natureza quadrática do tempo de execução do algoritmo BubbleSort. Um crescimento de aspecto aproximadamente linear, característico de uma curva $n \cdot log(n)$, pode ser observado para a curva do MergeSort. No entanto, as curvas verde e vermelha perdem foco nesse gráfico devido à ordem de grandeza muito superior da curva azul.

Visando uma análise comparativa mais minuciosa entre os métodos, foi realizada uma nova bateria de medições de tempo, agora com o tamanho dos vetores variando de 5 em 5, até 1000. Dessa vez, foram realizadas 100 medições para cada vetor, para cada método, para a extração de uma média, visando reduzir o ruído. A partir disso, produziu-se o gráfico da figura 8:

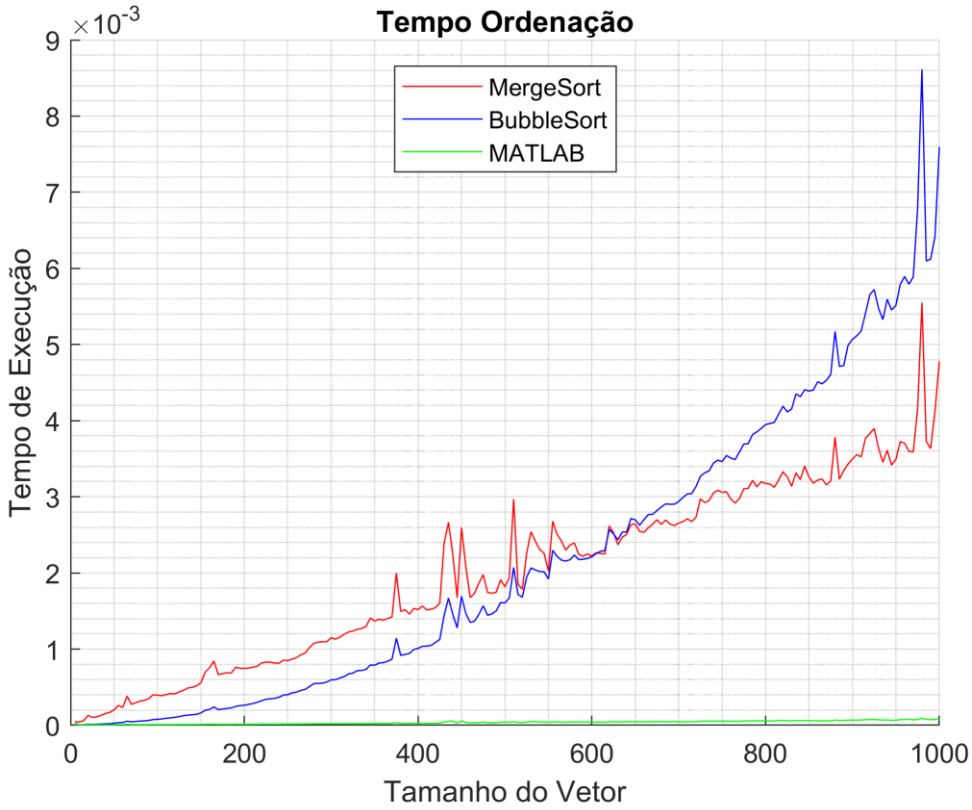

Figura 8: Tempos de ordenação com vetores menores

Apesar das médias, ainda se nota bastante ruído no gráfico, mas aqui podemos observar o instante em que o MergeSort passa a ser mais rápido que o BubbleSort, que é para um tamanho de vetor próximo de 600. O que podemos notar novamente é que a curva verde permanece muito mais baixa que as demais, evidenciando que a função `sort` do MATLAB é incomparavelmente mais eficiente para a ordenação do que ambas as implementações feitas no Lab, qualquer que seja o tamanho dos vetores sendo ordenados.