Relatório do Lab1 de CCI-22

Trabalho 01 - Números

Aluno:

Bruno Costa Alves Freire

Turma:

T 21.4

Professor:

Luiz Gustavo Bizarro Mirisola

Data de realização do experimento: 06/03/2018

Instituto Tecnológico de Aeronáutica – ITA
Departamento de Computação

1. Mais um ϵ!

A partir de um programa em linguagem C e scripts MATLAB, foram obtidos os valores correspondentes aos *épsilons* da máquina, compilados na tabela 1:

Tabela 1: Valores de épsilon decimais

	MATLAB eps	Programa MATLAB	Programa C
`single`	$1{,}1920928955078125 \cdot 10^{-7}$	$1{,}1920928955078125 \cdot 10^{-7}$	-
`float`	-	-	$1{,}1920928955078125 \cdot 10^{-7}$
`double`	$2{,}2204460492503131 \cdot 10^{-16}$	$2{,}2204460492503131 \cdot 10^{-16}$	$2{,}2204460492503131 \cdot 10^{-16}$

Como pode ser observado, os valores encontrados para o script MATLAB e a função pronta foram iguais, e ainda foram iguais aos valores correspondentes em C. Vale observar que no compilador utilizado, os tamanhos para os tipos em C são de 32 bits para `float` e 64 bits para `double`. O mesmo padrão é seguido pelo MATLAB para os tipos `single` e `double`, respectivamente. Ou seja, ambos seguem o padrão IEEE-754 de representação em ponto flutuante.

Escrevendo esses números em notação exponencial de base 2 e conhecendo a quantidade de bits reservada para cada campo do padrão IEEE-754, podemos inferir sobre a forma como esses épsilons interagem com o número 1.

Tabela 2: Valores de épsilon em ponto flutuante

Épsilon	Base 2	IEEE 754 (Hexadecimal)
`single/float`	$1{,}0 \cdot 2^{-23}$	0x3E000000
`double`	$1{,}0 \cdot 2^{-52}$	0x3CB0000000000000

Os números `float/single` possuem 23 bits na mantissa, os `double`, 52. Ao operar esses épsilons com o número 1, eles são convertidos de modo a igualar os expoentes entre os operandos, e então realizar a soma das mantissas. Contudo, quando isso é feito, a representação dos épsilons se reduz a um único bit 1 no final da mantissa. Ou seja, qualquer valor menor do que esse seria arredondado para zero ao operar com números da faixa do 1. Por isso, eles são os menores números capazes de operar somas com 1 alterando seu valor.

Os arquivos dos programas utilizados para calcular os valores da tabela 1 são: `epsilon.c` e `epsilon.m`.

2. Operador ou loop?

Foram testados 3 métodos de realizar um produto interno entre dois vetores:

1) Utilizando o operador de multiplicação matricial: `pi = x' * y`;
2) Utilizando a função dot: `pi = dot(x, y)`;
3) Fazendo um laço `for` explícito:

$$\sum_{i=1}^{n} x(i) * y(i)$$

Para cada método, foram realizadas 1000 medições utilizando vetores de 10^6 elementos, gerados aleatoriamente pelo MATLAB. Medindo os tempos de cálculo dos três métodos, foram gerados os seguintes gráficos através do script `myip.m`:

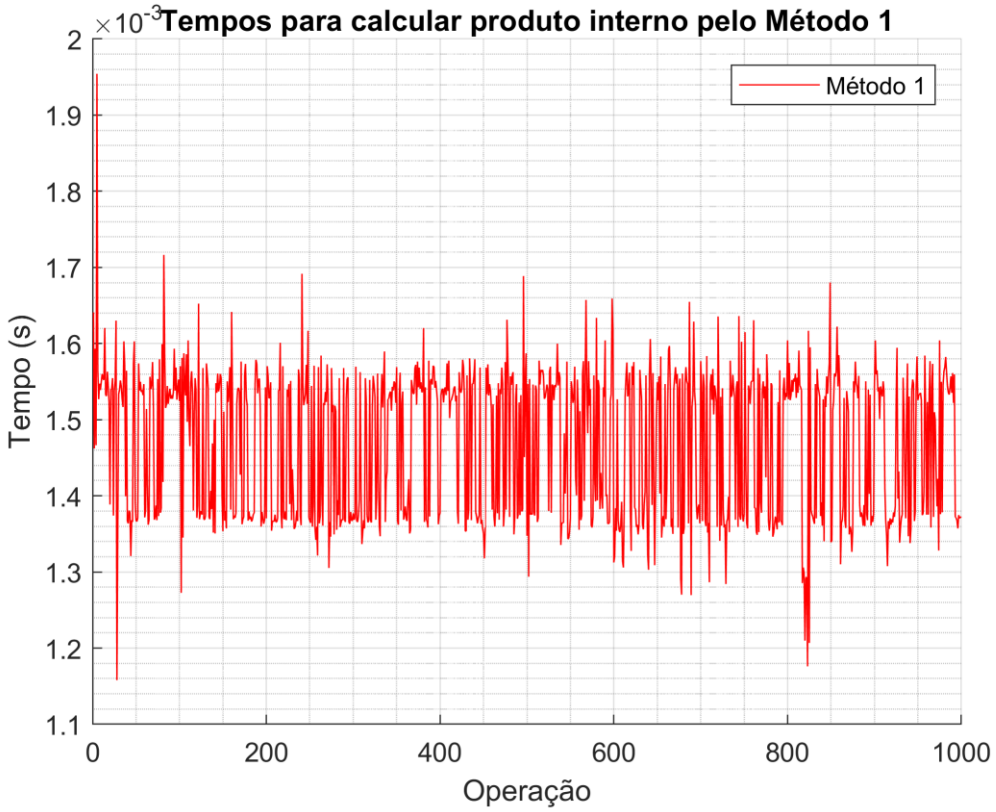

Figura 1: Tempos do Método 1

Cada curva teve de ser plotada em um gráfico diferente por questões visuais: a região ocupada pela curva vermelha parece ínfima perto das demais.

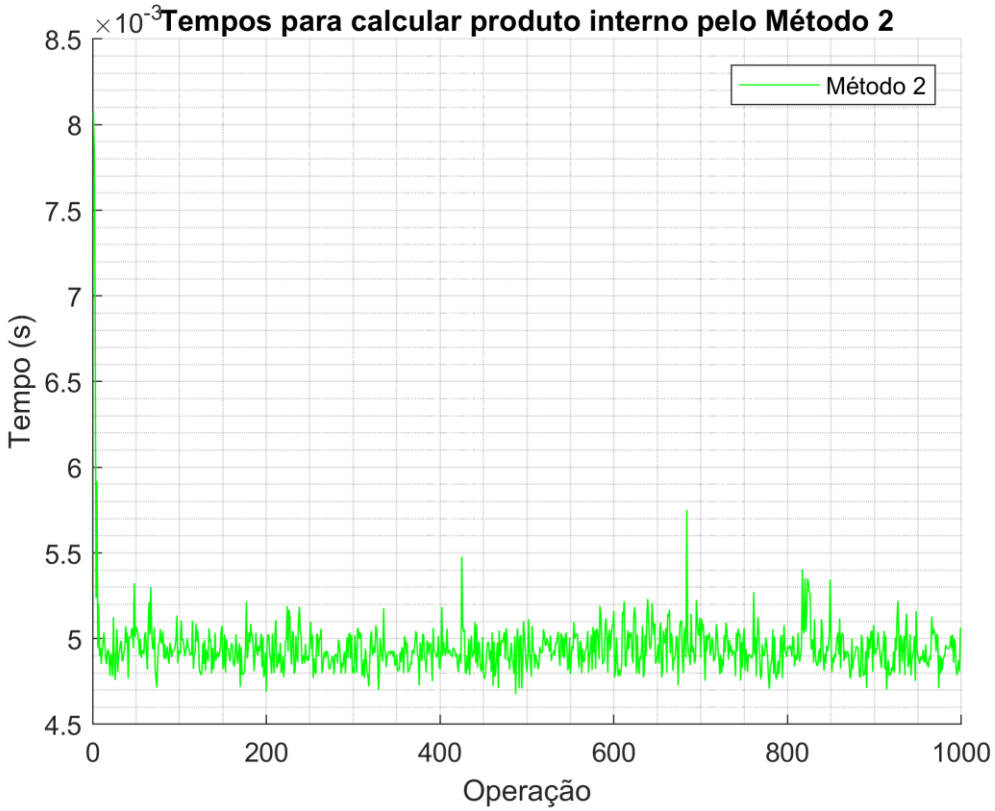

Figura 2: Tempos do Método 2

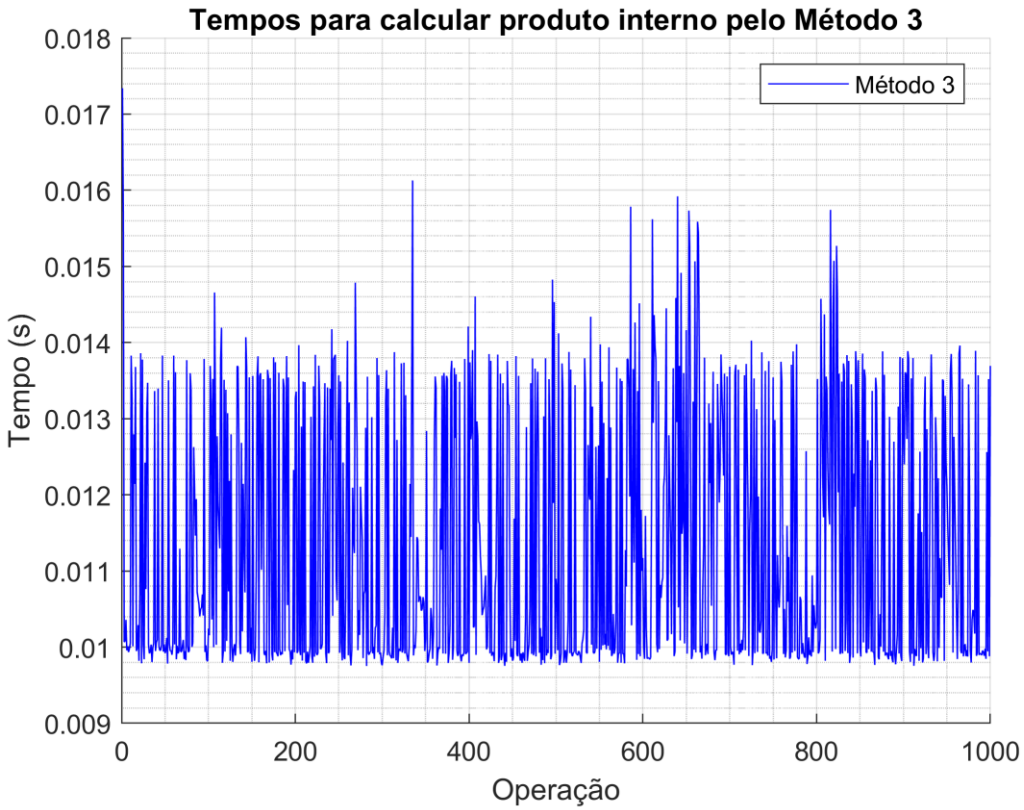

Figura 3: Tempos do Método 3

Devido ao aspecto notavelmente ruidoso das medidas, foram produzidos também gráficos das médias dos tempos em função do número de medições:

Figura 4: Média dos tempos do Método 1

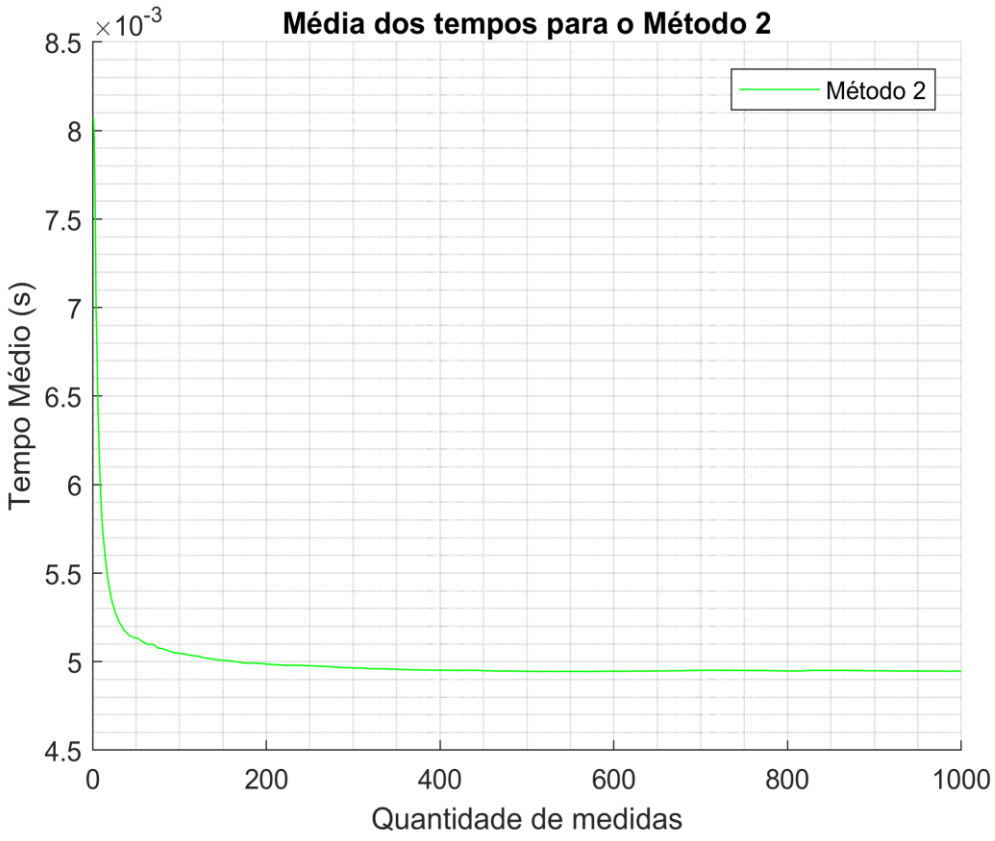

Figura 5: Média dos tempos do Método 2

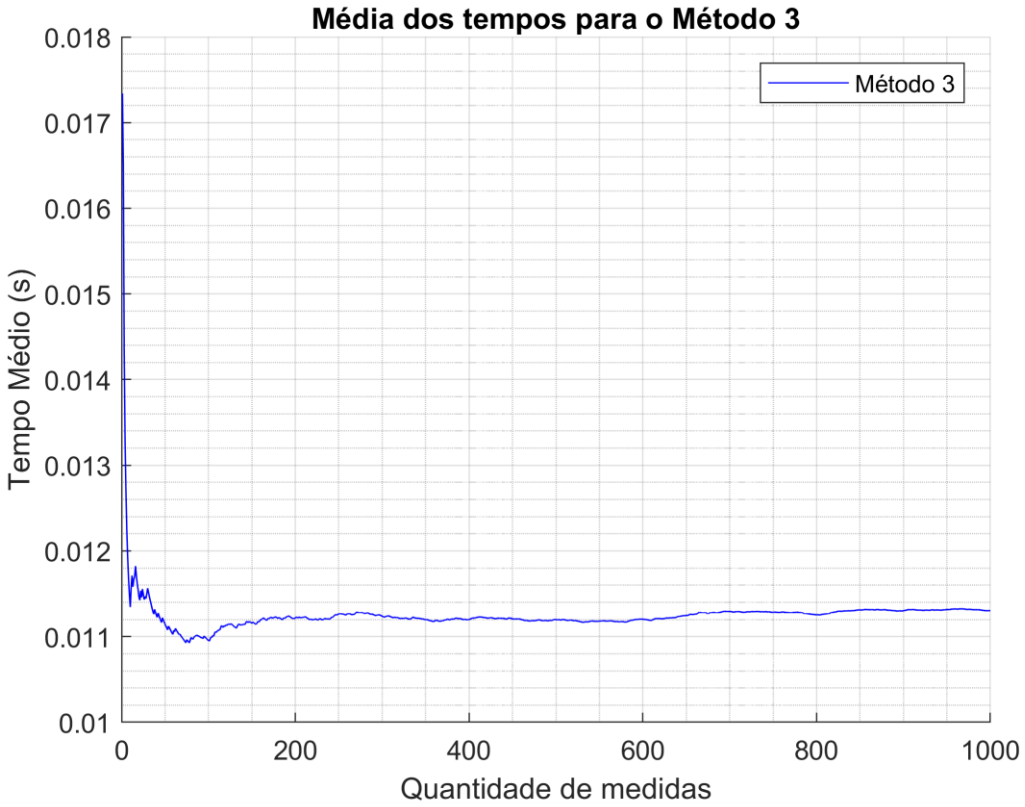

Figura 6: Média dos tempos do Método 3

Para obter uma comparação justa entre os métodos, os três métodos utilizaram os mesmos vetores aleatórios em uma dada medição. Os valores aproximados para 3 dígitos significativos das médias estão na tabela 3:

Tabela 3: Média final dos tempos de execução

Método	Média final dos tempos (ms)
`pi = x' * y`	1,47
`pi = dot(x, y)`	4,95
$pi = \sum_{i=1}^{n} x(i) * y(i)$	11,3

Como se pode ver pela tabela, os métodos estão ranqueados do mais eficiente pro menos eficiente, o que evidencia que a sintaxe para a operação matricial é mais otimizada no MATLAB do que a função nativa, e ambas são muito mais eficientes do que laços de repetição.

3. exp( ) com precisão (AUTOTEST)

4. Somas não causam erros, certo?

Através do console do MATLAB, calculou-se a expressão:

$$D = 10000 - \sum_{i=1}^{n} x$$

Com:

a) $n = 100000$ (10^5) e $x = 0{,}1$;
b) $n = 80000$ ($8 \cdot 10^4$) e $x = 0{,}125$;

E os valores obtidos foram, respectivamente:

a) $D = -1{,}884836819954216 \cdot 10^{-8}$
b) $D = 0$

Ao passo que matematicamente os dois resultados deveriam claramente ser iguais a zero, vale lembrar que os números nem sempre são armazenados de maneira exata nos computadores. Por fazerem uso de representação binária, os computadores não conseguem precisão absoluta para números que são *dízimas* em base 2.

Acontece que o número 0,1 é uma dízima em binário. Portanto, para ser armazenado, seu valor tem de ser truncado, e por conta disso algumas operações com o mesmo serão afetadas por esse truncamento, propagando um erro ao longo dos cálculos.

Por outro lado, o valor 0,125 tem representação exata em binário, e por isso os cálculos efetuados com ele não sofrem por erros de representação, levando a resultados matematicamente exatos.

- Aplicação: Comparação de Double

Tendo em vista o exposto acima, é corriqueiro que haja contas matematicamente corretas sendo invalidadas por erros de representação numérica. Em particular, um problema que surge é o da comparação entre valores do tipo `double`.

A depender da magnitude dos valores com que se está trabalhando, da precisão buscada nas operações e da propagação de erros de representação, dois valores que deveriam ser iguais podem não coincidir, e eventualmente valores que precisam ser diferenciados podem não ter uma diferença significativa.

Portanto, para comparar dois valores `double`, todos esses fatores devem ser levados em consideração. Para decidir sobre a igualdade de dois valores, deve-se analisar se o valor absoluto da diferença entre ambos é menor do que um dado valor que deve ser escolhido levando em consideração o ruído provocado por erros de representação e pela quantidade de operações.

5. Ordenação (AUTOTEST).