Trabalho Prático Determinante de Matrizes

Ruan Da Silva Sathler

[1]Instituto de Informática – Universidade Federal do Amazonas – (UFAM)
Manaus – AM – Brasil

ruan.sathler@icomp.ufam.edu.br

Abstract. *This work aims to document the results obtained in a research project in which Professor Moises Carvalho, from the Introduction to Programming course, provided a code for calculating matrix determinants. This code was to be optimized and compared with a newly developed code, and the results of this comparison are available in this document.*

Resumo. *Esse trabalho tem por objetivo documentar os resultados obtidos em uma pesquisa na qual o professor Moises Carvalho da disciplina de Introdução à programação disponibilizou com código que calcula determinante de matrizes, este deveria ser otimizado e comparado com o novo código, resultados estes que estão disponíveis neste documento.*

1. Introdução

Calcular o determinante de matrizes pode ser uma tarefa relativamente simples para nós humanos, mas será que é assim para maquinas? Na verdade, isso não é tão simples assim, à medida que a ordem de uma matriz aumenta a dificuldade e a exigência das capacidades da maquina que se está utilizando aumenta exponencialmente, para reduzir essa dificuldade a escolha de um bom algoritmo é indispensável, essa pesquisa foi desenvolvida com o objetivo de mostrar a diferença que uma simples melhoria no algoritmo pode causar no seu tempo de execução.

Obs.: todos os atributos de classe listados neste documento foram acessados seguindo as normas da programação orientada a objeto, sendo este sempre acessadas pelos seus respectivos sets e gets.

2. Ambiente de execução

Todos os testes desta pesquisa foram feitos em um notebook da marca Dell modelo G7 7588, com processador Intel core I7, 16g de memória RAM e memória SSD de 250G. O sistema operacional utilizado foi o Windows 11 Pro, versão 23H2.

3. Baseline e versão otimizada

3.1 Baseline:

O código Baseline disponibilizado pelo professor utiliza a regra de La Place, que percorre uma linha e coluna e para cada elemento segue a seguinte regra.

$$detA = \sum_{i=1}^{n} (-1)^{i+j} \cdot a_{ij} \cdot (detA_{ij})$$

Figura 1. determinante de La Place

Para calcular o determinante de uma matriz de ordem N o código Baseline funciona usando dois métodos principais que são “determinante” e “detOrdemN”, eles criam uma chamada recursiva onde o primeiro método chama o segundo até chegarem ao critério de parada que é uma matriz de ordem 2, para obter a matriz de ordem menor até que ela seja de ordem 1, o método “detOrdemN” chama ainda outro método chamado “copiaMatrizMaiorParaMenor”, esse método criar uma matriz de ordem N-1, onde ela exclui uma linha e uma coluna que são passados como parâmetros na hora que esse método é chamado.

Com esses métodos o cálculo é feito da seguinte maneira, uma matriz de ordem N é passada para o método determinante que verifica a sua ordem, se maior que 2, ela é passada para o “detOrdemN”, que chama o método “copiaMatrizMaiorParaMenor” onde ele excluirá a primeira linha e a primeira coluna matriz, em sequência ele chama o método “determinante” que repete o processor até a ordem ser2.

Utilizando da propriedade dos determinante onde é possível obtê-lo através da soma dos produtos de elementos e de determinantes de matrizes menores, ele pega o determinante da matriz de ordem 2 e usa-o para descobrir o determinante da matriz de ordem 3 até a ordem N da matriz, após isso esse valor é armazenado em um somatória e as etapas anteriores são repetidas para todos os elementos da primeira linha da matriz, então ao somar o valor obtido em cada um dos elementos ela retorna o determinante da matriz.

3.2 Versão otimizada:

A versão otimizada funciona de forma muito similar, porém seu objetivo e achar a coluna ou linha com mais zeros, já que se o determinante é obtido a partir da soma de produtos sabemos que o valor após todas as operações envolvendo 0 serão 0, assim o elemento 0 na linha ou coluna pode ser ignorado e o algoritmo pula imediatamente para o próximo elemento.

Para tal objetivo foram criados 4 atributos novos na classe “linhaComMaiszeros”, “colunaComMaiszeros”, “zerosColuna” e “zerosLinha”, bem como seus gets e sets, além disso foram criados os métodos “achaColunaComMaisZeros” e “achaLinhaComMaisZeros”, eles funcionam de forma parecida, o primeiro percorre a primeira coluna e conta quantos zeros tem, após isso joga o número da primeira coluna dentro do “colunaZomMaisZeros” e a quantidades de zeros contados dentro do “ZerosColuna”, depois ele percorre a segunda coluna e conta seus zeros, ao final se a quantidade de zeros for maior do que a quantidade de zeros no atributo “zerosColuna” ele substitui os valores da “ColunaComMaisZeros” e “ZerosColuna” pelos da segunda coluna, e repete os passos com todas as colunas da matriz, no fim teremos a coluna na qual foi encontrado mais zeros, o mesmo ocorre para achar a linha porém ele percorre as linhas da matriz invés das colunas.

Com os métodos acima criados foram criados mais dois métodos “determinanteOtimizado” e “detOrdemNOtimizado”, o “determinanteOtimizado” funciona exatamente como o segundo a única diferença é que ele chama o método “detOrdemNOtimizado”, o “detOrdemNOtimizado” funciona como o “detOrdemN” a diferença é que ele analisa duas situações antes de encaminhar a matriz para o “copiaMatrizMaiorParaMenor”, a primeira quem é maior entre “zerosColuna” e “zerosLinha”, depois ele verifica se o elemento em que ele está é, o método vai percorrer os elementos da linha ou coluna que tiver mais zeros, se o elemento em que ele estiver for zeros então ele pula automaticamente para o próximo elemento ao final ele retorna o somatório obtidos utilizando todos os números diferentes de 0.

4. Dados obtidos

A seguir encontra-se os dados obtidos em todos os testes e os gráficos comparativos, nota-se que o algoritmo otimizado é muito mais rápido que a versão baseline, chegando a ser mais de 100% mais rápido em alguns momentos, porém sua eficiência diminui um pouco à medida que a ordem cresce e é cada vez mais raro a ocorrência de zeros na matriz. Segue os dados:

Tabela 1. Comparativo detalhado entre os testes de algoritmo

base-line				algoritimo incrementado		
ordem da matriz:	numero do teste	tempo de execução:		ordem da matriz:	numero do teste	tempo de execução:
ordem 3	teste 1	23.500		ordem 3	teste 1	15600
	teste 2	2400			teste 2	6800
	teste 3	37800			teste 3	34100
ordem 5	teste 1	323.700		ordem 5	teste 1	128200
	teste 2	13700			teste 2	20400
	teste 3	103200			teste 3	59300
ordem 7	teste 1	1908400		ordem 7	teste 1	894100
	teste 2	2033300			teste 2	836100
	teste 3	1530000			teste 3	811900
ordem 9	teste 1	38778400		ordem 9	teste 1	30055500
	teste 2	38152200			teste 2	35631900
	teste 3	38329300			teste 3	34750500
ordem 11	teste 1	3705556900		ordem 11	teste 1	3053749800
	teste 2	3572009900			teste 2	3302770100
	teste 3	3549248600			teste 3	2440387500

Tabela 2. Comparativo geral entre os testes de algoritmo

tempo médio de execução					
base-line		versão otimizada		porcentagem de melhoria	
ordem:	média de tempo:	ordem :	média de tempo:	ordem:	melhoria:
3	21.233,33	3	18.833,33	3	12,74%
5	146.866,67	5	69.300,00	5	111,93%
7	1.823.900,00	7	847.366,67	7	115,24
9	38.419.966,67	9	33.479.300,00	9	14,76
11	3.608.938.466,67	11	2.932.302.466,67	11	23,08

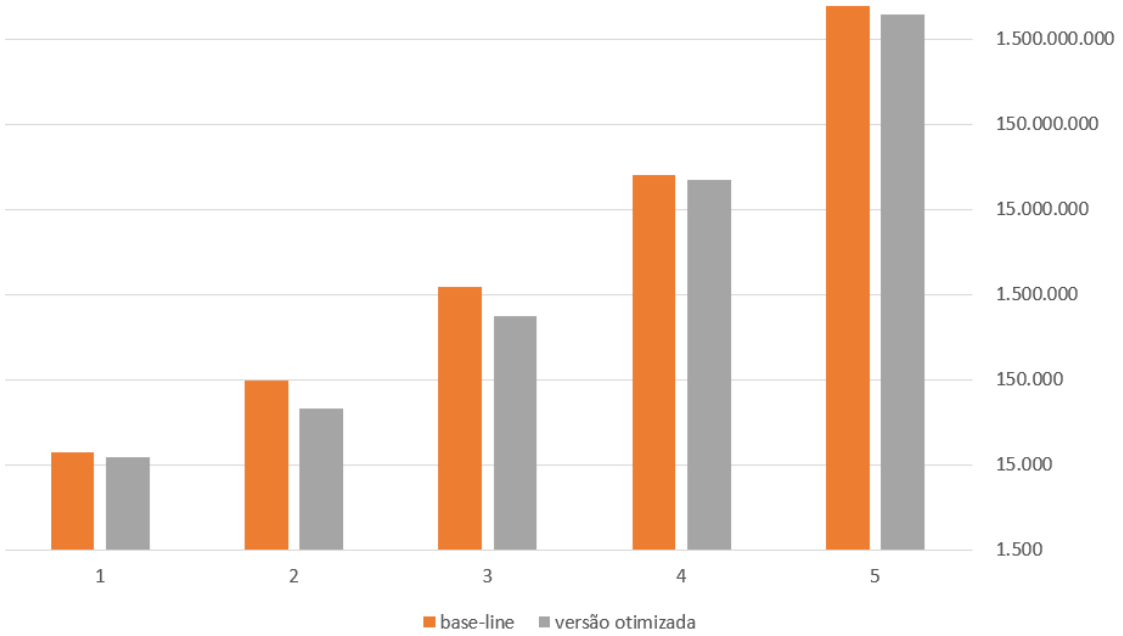

Figura 2. Gráfico comparativo de tempo entre os testes

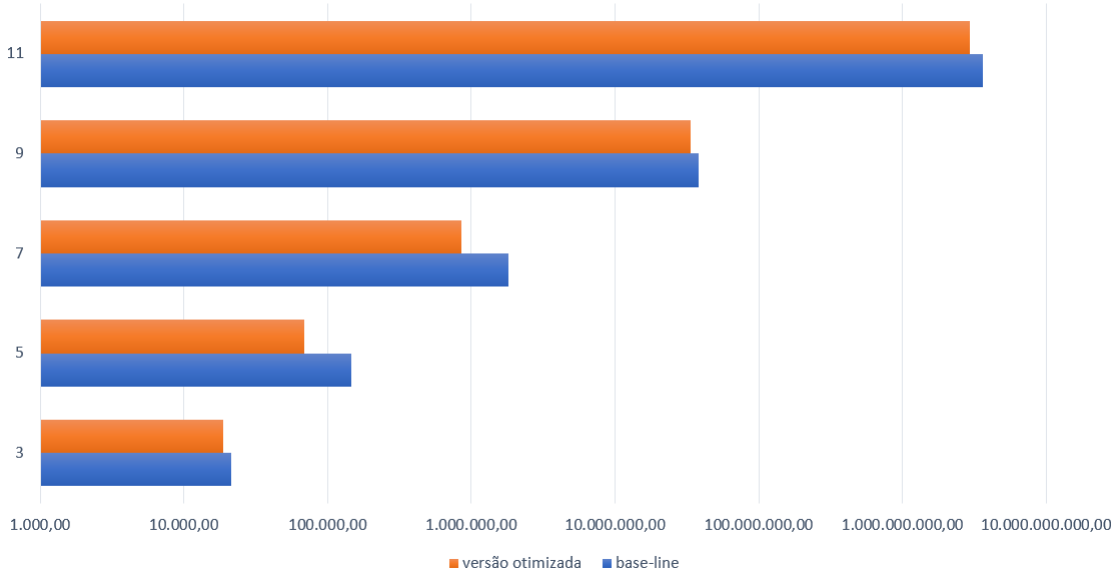

Figura 3. Gráfico comparativo de tempo entre os testes 2

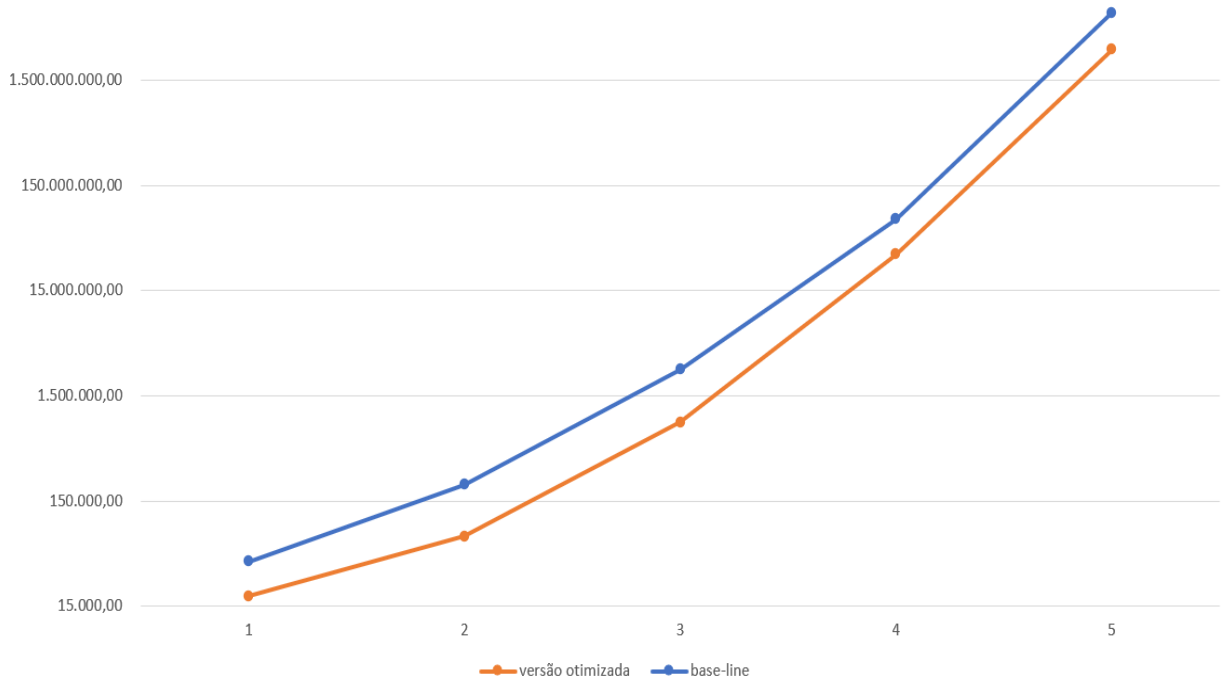

Figura 4. Curva de tempo comparativa

7. Referências

.

Gabriel Alessandro de Oliveira. (2011) Teorema de Laplace. Calculando determinantes através do teorema de Laplace. Disponível em: <https://brasilescola.uol.com.br/matematica/teorema-laplace.htm/>. Acesso em: 19 jul. 2024.

LESSA, J. R. Teorema de Laplace. Disponível em: <https://www.infoescola.com/matematica/teorema-de-laplace/>. Acesso em: 20 jul. 2024.