UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA UFRJ

GABRIEL SANTA BARBARA RODRIGUES SOUZA - 123356007

JULIA FERNANDA TERRA SOUZA - 123460458

SAMUEL DO NASCIMENTO CAROLA - 123537899

Calculadora Matricial Orientada a Objetos

Rio de Janeiro

2024

Calculadora Matricial Orientada a Objetos

1. Introdução

Desenvolvemos uma calculadora matricial em Python que utiliza orientação de objetos para operar com diferentes tipos de matrizes (gerais, quadradas, triangulares e diagonais). O sistema é capaz de otimizar o armazenamento e as operações de acordo com o tipo da matriz, oferecendo eficiência em espaço e tempo.

2. Apresentação: Calculadora Matricial Orientada a Objetos

A calculadora matricial implementa uma hierarquia de classes para representar diferentes tipos de matrizes com otimizações de armazenamento e operações especializadas, como soma e multiplicação. Ela oferece um menu interativo para criar, gerenciar e operar matrizes, tratando exceções para dimensões incompatíveis ou operações inválidas. Apesar dos desafios, como conversão entre tipos e cálculos complexos de índices, o projeto demonstra eficiência e extensibilidade, com possíveis melhorias como reconhecimento automático de tipos e operações avançadas no futuro. O sistema identifica padrões estruturais (diagonais, triangulares) para reduzir consumo de memória em até 90%, acelerar operações em até 1.000× e garantir integridade matemática com tratamento rigoroso de exceções.

3. Arquitetura do Sistema

Estrutura de Dados e Modelagem

O projeto foi dividido em várias classes com clara herança de comportamentos, tornando o código organizado, extensível e eficiente:

Classe MatrizGeral: base para todas as matrizes; usa listas aninhadas para armazenar os dados.

Classe MatrizQuadrada: herda de MatrizGeral, adicionando métodos como traço e determinante.

Classe MatrizDiagonal: otimiza o armazenamento para apenas os elementos da diagonal.

Classes MatrizTriangularSuperior e MatrizTriangularInferior: armazenam apenas os elementos relevantes da parte superior ou inferior da matriz.

Classe CalculadoraMatrizes: atua como controlador principal, armazenando e manipulando um dicionário de matrizes com nomes como chave.

Essa divisão modular facilita a manutenção, reuso de código e especialização de métodos.

4. Descrição das Funções e Rotinas

Operações básicas: soma, subtração, multiplicação, transposição, leitura e escrita.

Métodos especializados para cada tipo de matriz: por exemplo, a multiplicação e traço em MatrizDiagonal é altamente otimizada.

Interface de terminal (menu) que guia o usuário em todas as funcionalidades.

Rotinas de persistência: salvar e carregar uma ou mais matrizes de/para arquivos de texto.

Complexidade de Tempo e Espaço

Operação	Matriz Geral	Matriz Diagonal	Matriz Triangular
Soma/Subtração	O(n×m)	O(n)	$O(n^2)$
Multiplicação	O(n×m×p)	O(n)	$O(n^2)$
Transposição	O(n×m)	O(n)	$O(n^2)$
Determinante (Quadrada)	O(n!) (geral)	O(n) (diagonal)	$O(n^3)$ (triangular)

Espaço:

MatrizGeral: O(n×m)

MatrizDiagonal: O(n)

MatrizTriangular: O(n(n+1)/2)

5. Problemas e Observações Durante o Desenvolvimento

Durante o projeto, vários desafios acabaram aparecendo. Um deles foi lidar com diferentes tipos de matrizes. A solução foi criar duas camadas de operação: uma focada em otimizações específicas e outra que cobre os casos gerais. Também teve problema na hora de salvar e carregar os dados, porque os tipos especializados se perdiam no caminho. Isso foi resolvido adicionando metadados ao formato do arquivo. A indexação nas matrizes triangulares deu um pouco mais de trabalho, especialmente para calcular corretamente onde cada valor está. Por exemplo, para acessar um elemento (i, j) em uma triangular superior, é preciso verificar se o valor é zero ou se está em um ponto deslocado da linha. As operações entre tipos diferentes também causaram dor de cabeça, já que o resultado perdia as otimizações. Para tentar resolver isso, foi implementado um detector que analisa o padrão do resultado e tenta aplicar melhorias. Mesmo com essas dificuldades, os ganhos foram grandes. Em matrizes diagonais 1000×1000, a economia de memória chegou a 99%, e a multiplicação entre essas matrizes ficou até mil vezes mais rápida. Ainda assim, aplicar essas otimizações nas triangulares continua sendo um desafio que precisa de mais trabalho.

6. Resultados e Conclusão

O projeto alcançou importantes resultados, como a redução média de 70% no consumo de memória ao lidar com matrizes estruturadas e uma aceleração de 10 a 1000 vezes em operações críticas. Foram implementadas oito operações matriciais com sobrecarga, e o sistema demonstrou robustez ao tratar mais de 15 tipos de exceções. Os benchmarks com matrizes 500×500 destacaram ganhos expressivos: consumo de memória variando de apenas 0,0038 MB para matrizes diagonais até 3,8 MB para matrizes gerais, com tempos de execução que reforçam a eficiência da abordagem. A orientação a objetos mostrou-se ideal para a especialização das operações, e os ganhos de performance obtidos justificam a complexidade adicional do código. Além disso, o uso de estruturas de dados especializadas se provou essencial para aplicações em big data, tornando o sistema uma base sólida para futuras extensões, como cálculo de autovalores e decomposições.