Relatório de Desenvolvimento — Calculadora Matricial

Nome: Bruno Videira Pinho DRE: 119.161.539

Estruturas de Dados Utilizadas

O projeto utiliza principalmente a estrutura listas de listas de floats (`list[list[float]]`) e lista de float (`list[float]`) para representar matrizes num geral. Para matrizes triangulares e diagonais, são usadas listas de listas de tamanhos variáveis e listas simples, respectivamente, otimizando o uso de memória ao armazenar apenas os elementos necessários (como, apenas a diagonal principal para matrizes diagonais).

Divisão de Módulos

- **src/matrices**: Armazenamento do tipo de matizes a serem representadas.
- **src/matrices/std_matrix.py**: Implementa a matriz padrão (m x n) e operações básicas.
- **src/matrices/sq_matrix.py**: Especializa para matrizes quadradas, incluindo operações como traço.
- **src/matrices/trg_matrix.py**: Implementa o esqueleto de matrizes triangulares, com métodos otimizados.
- **src/matrices/upt_matrix.py**: Implementa matrizes triangulares superiores, com métodos otimizados.
- **src/matrices/dwt_matrix.py**: Implementa matrizes triangulares inferiores, com métodos otimizados.
- **src/matrices/dig_matrix.py**: Implementa matrizes diagonais, usando lista simples para a diagonal principal.
- **src/matrices/custom_types.py**: Define tipos literais e aliases para operações.
- **src/interface/main_frame.py**: Interface de menu para manipulação da lista de matrizes e operações.
- **tests.py**: Contém testes unitários para todas as classes e operações.
- **main.py**: Arquivo principal para execução da interface.

Descrição das Rotinas e Métodos

- **Construtores**: Validam e armazenam os dados das matrizes conforme o tipo.
- **Operadores Sobrecarregados**: `__add__`, `__sub__`, `__mul__`, etc., permitem operações naturais entre matrizes e escalares.
- **Métodos Especiais**: Métodos como `trace()` e `determinant()` são implementados para matrizes quadradas e triangulares, respectivamente.
- **Gerenciamento de Lista**: Funções para inserir, remover, listar, salvar e carregar matrizes.
- **Tratamento de Exceções**: Verificações de compatibilidade de dimensões e tipos, com mensagens de erro claras.

Complexidade de Tempo e Espaço

- **Matrizes Gerais**: Operações como soma e multiplicação têm complexidade $O(n^2)$ e $O(n^3)$, respectivamente.
- **Matrizes Especiais**: Operações otimizadas, por exemplo, soma de diagonais é $O(n)$, multiplicação de triangulares é $O(n^2)$.
- **Espaço**: Matrizes especiais usam menos memória, armazenando apenas os elementos necessários.

Problemas e Observações

- **Padronização de Entrada**: Foi necessário definir um formato consistente para entrada de matrizes especiais, documentando claramente para evitar confusão.
- **Sobrecarga de Operadores**: Exigiu atenção para garantir que o tipo de retorno fosse o mais especializado possível.
    - OBS: Acabei me esfoçando muito para validar os tipos de objetos que resultam de cada operação. Por conta disso a visualização não ficou no formato que esperava.
- **Testes**: A cobertura de testes foi fundamental para garantir a robustez, especialmente em casos de erro e operações entre tipos diferentes.
- **Tratamento de Exceções**: Implementado para evitar operações inválidas e facilitar o uso da calculadora.

Conclusão

O projeto atendeu aos requisitos propostos, permitindo operações eficientes e seguras entre diferentes tipos de matrizes, com uso otimizado de memória e código modularizado. A abordagem orientada a objetos facilitou a extensão para novos tipos de matrizes e operações. O uso de testes automatizados garantiu a confiabilidade das implementações. Como melhoria futura, pode-se implementar uma interface gráfica ou expandir o suporte a outros tipos de matrizes especiais.