Augmented Design Report

Giving the user more control

We decided to give the user more control over the functionality of the library. Initially, we planned to detect when a tri-diagonal matrix is being operated on and use a specialized algorithm in such situations. But to aid testing and get rid of the checks we have to do before every operation, we decided to allow the user to decide whether to use Tri-diagonal matrices and their specialized implementations.

If the user is aware that the matrix they operate on is tri-diagonal, they can use the TridiagonalSparseMatrix class for their operations. We made sure to provide a uniform interface for both YaleSparseMatrix and TridiagonalSparseMatrix so that the learning curve isn't increased.

This decision was predicated on the principle that this project is a library and not an application and control, efficiency and flexibility should be prioritized over usability in this context.

Returning a 2D-array for tri-diagonal matrix inversion

Since the inverse of a tri-diagonal matrix isn't necessarily a tri-diagonal matrix, we decided to return a 2D-array as a result instead.

Shapes

We later realized that the Shape class was unnecessary and passing in 1-2 numbers as arguments simplified the design.

No sparse implementation of inverse in YaleSparseMatrix

Since NMatrix, the library used by YaleSparseMatrix to delegate functionality to, does not have a sparse implementation for inverse, we were forced to cast the matrix to dense first and then operate.

This is all done under the hood and does not affect the interface of the library.

Adding a to2DArray() method

A method to convert the sparse matrix classes to a 2D array was added for convenience in the client code.

Modified Factory Methods

Methods for creating specialized matrices in both YaleSparseMatrix and TridiagonalSparseMatrix were added for convenience and provide a more unified interface for creating matrices in the library.