ECE 421 Assignment 1 Part 3

Winter2019_Group4:

Nathan Klapstein (1449872)

Tony Qian (1396109)

Thomas Lorincz (1461567)

Zach Drever (1446384)

Table of Contents

Summary of Deviations

Tridiagonal Matrix Class Deviations

Description

In our Part 2 report, we specified that we were not going to create a standalone class for tridiagonal matrices and would instead allow users to specify the bandwidth of the diagonal matrix that they are creating. As we developed the diagonal matrix functionality, we decided to follow many storage optimizations for matrices with a single diagonal. Because of the optimizations, it was no longer feasible to support a variable bandwidth. So, we have instead implemented a standalone tridiagonal matrix class. This new class is specified based on the contracts of its constructor and its determinant method.

Contract Updates

def determinant()

- Purpose: Computes the determinant of the matrix. A special property of tridiagonal matrices is that its determinant can be calculated as the continuant of its elements. This allows its determinant calculation to be faster than regular determinants for dense matrices (in sufficiently large cases).
- Pre:
    - @row_count > 3
    - @column_count > 3
- Post:
    - result = matrix[n-1, n-1] * determinant(matrix, n-1) - matrix[n-1, n-2] * matrix[n-2, n-1] * determinant(matrix, n-2)

Diagonal Matrix Class Deviations

Description

As mentioned above, the development of the diagonal matrix class included consideration for many optimizations that made supporting a variable bandwidth unfeasible to implement. For this reason, there is no bandwidth field in the DiagonalMatrix class.

Contract Updates

def rows(rows, column_count = rows[0].size)

- Purpose: Construct a new SparseMatrix from an array of rows (arrays) where there are only elements stored along the main diagonal (rows[i][i]).

- Pre:
    - rows.is_a?(Array)
    - rows.length > 0
    - column_count > 0 || rows[0].size > 0
- Post:
    - @matrix.length = @column_size && @matrix[0].length = @matrix[i] = rows[i, i]
    - assert_equal(@column_count, column_count)
    - assert_equal(@column_size, rows.length)

Sparse Matrix Fields/Data Deviations

Description

We initially planned to have a @nonzero_values field that all sparse matrices would store. This field was to be an array of the nonzero values contained in the sparse matrix. However, we determined that this would not be helpful because it would increase the storage of the sparse matrix classes. Instead, we implemented a method read_all that will return an array of all nonzero values when called. This way, we don't waste storage.

Functional Deviations

Description

We had specified that the return of all applicable methods would be a new SparseMatrix. Instead, the return of these methods is a new Matrix. Conversion back to a SparseMatrix can be easily performed. This decision was made because many operations on sparse matrices will create matrices that are better stored/represented as dense matrices.