Introduction

This software package is an implementation of various sparse matrix operations written in the Ruby programming language. Arithmetic operations, inverse, transpose, determinant, trace, round and equivalency are provided.

We consulted classes implemented in other languages, as well as resources on the topic of linear algebra and engineering to determine what functionality was needed for this project.

Design

The class uses a dictionary of keys to store the sparse matrix. The keys are pairs of numbers [i,j] representing the value's location in the matrix. This storage technique allows for O(1) access to individual elements in the matrix, and approximately O(1) for inserting new elements.

The delegate design pattern is where a certain class or function calls upon an already implemented class or function to complete a task. The dictionary of keys is implemented using a Hash in Ruby. This allowed us to use the delegate pattern to use Hash's functionality when needed. For example, getting the number of non zero elements in the matrix uses Hash.size.

The abstract factory pattern has one class that makes factories for specific instances, and those factories then make concrete classes of those instances. Since our package only implements the sparse matrix as one class, it does not use this pattern.

We chose to implement the sparse matrix as a single class which allows for efficient access, updates, and storage. This would be useful for applications in things such as graph theory. Adjacency matrices could be represented for directed and undirected graphs in a spatially efficient manner.

A sparse matrix can be initialized by passing in just dimensions, dimensions and a set of elements, or another sparse matrix. The first will create an empty matrix of the given size. The second will create a sparse matrix filled with the data supplied. And the third will create a copy of the matrix passed to it. The class always requires that it know the size of the matrix in terms of number of rows and columns.

Methods

The methods possible with the SparseMatrix class are matrix addition, subtraction, multiplication, division, and exponentiation, scalar multiplication and exponentiation, matrix transpose, inverse, determinant, round and trace. Other methods include real?, regular?, singular?, square?, symmetric? and zero? and these check for various properties of the matrix. Contracts are provided for each method to ensure it adheres to the expected rules and gives an appropriate output.

The addition and subtraction functions work for two matrices or equal dimensions, and will give an error if they are not.

Multiplication can be between either two matrices, or a matrix and a scalar number. In the multiplication A*B, the columns in A must equal the rows in B. Division is implemented by multiplying by the inverse of a matrix: $A/B=A*B^{-1}$. This requires that A and B both be square matrices of the same size, in order to compute the inverse of B, and then perform the multiplication. Scalar division can be achieved by multiplying by a fraction.

Matrix exponentiation can be performed on a square matrix, and scalar exponentiation can be performed on any matrix.

The transpose of a matrix can be computed given any SparseMatrix. The inverse can be computed provided that the matrix is square, and the same goes for the determinant.

Methods are also provided which allow the user to get the number of rows or columns, and to get the dictionary representation of the matrix.

Updating the matrix can be achieved by using the put method, or the standard matrix notation A[i,j]=value. i and j must be within the bounds of the matrix, otherwise an error will occur. Getting an element from the matrix can be done with the get method or the standard notation A[i,j]. If the element is not in the matrix, this will return 0.

A matrix can be cleared, which sets all the entries to zero by emptying the dictionary.

A submatrix of a matrix can be obtained using the minor method given the ranges of the rows and columns that need to be extracted.

All the entries in the matrix can be rounded to a given precision using the method round.

The trace method computes the sum of the entries in the diagonal of the matrix.

Some methods implemented check for various properties of the matrix and return boolean values. The method real? checks if all entries are real. The methods regular?, singular?, square?, and symmetric? check if the matrix is regular, singular, square, and symmetric respectively. The method zero? checks if all the entries in a matrix are zeros.

Iteration is able to be used for manipulation, since the data is stored as a hash, which supports efficient iteration using enumerables.

Quality characteristics

The SparseMatrix class provides efficient memory usage for matrices where many elements are 0. Each elements requires three numbers to describe it: the row, column, and value. This could be extended, however, to accommodate higher dimensions by adding more positional information: (i,j,k,l…,value) for each element.