Assignment 1 part 3 report

1) Changes that were made to the proposed design was:
    a) The introduction of the classes named NDimensionalMatrix. The NDimensionalMatrix is a general matrix class that can store the data of a n-dimensional matrix in the data structure format of an array of arrays. This allows us to return results from sparse matrix operations as a general matrix if it doesn't follow the invariants of a sparse matrix.
    b) The additional method checkSum to the matrix classes. This method is mainly used for post conditions to validate the resulting matrix. Its precondition is that the input responds to getValues and the postcondition is that the result responds to round
    c) The design pattern delegation is now implemented into our design. For matrix functions such as determinate, multiplication of matrices, division of matrices, inverse of a matrix, and the power of matrices are delegated from the sparse matrix to our NDimensional matrix to ruby's Matrix to be done.
    d) The addition of a matrix factory class was introduced. It creates a matrix based on the input array representation of the matrix. Depending on the properties of the matrix array, the factory class will create a matrix of that type. Will be mostly used when returning a resulting matrix.
2) Deviations from the proposed contract are making them adopt duck typing ideas and making them check properties of results and inputs rather than the value. Such changes were:
    a) Pre_sparse_matrix_addition: check if input matrix responds to checkSum
    b) Post_sparse_matrix_addition: check if result is a matrix by checking if it responds to checkSum. Check if results sum is equal to the two matrices
    c) Pre_sparse_matrix_subtraction: check if input responds to checkSum.
    d) Post_sparse_matrix_subtraction: check if result responds to checkSum. Check if result sum equals the difference of the two matrices.
    e) Pre_scalar_subtraction: check if input responds to round.
    f) Post_scalar_subtraction: check if result if a matrix. Check if result sum is that of original minus value*matrix size
    g) Pre_scalar_addition: check if input responds to round
    h) Post_scalar_addition: check if result is a matrix. Check if sum of result is that of the original sum plus the value*matrix size.
    i) Pre_sparse_matrix_multiplication: check if input responds to getDimension. Check if matrix are (mxn)*(n*p)
    j) Post_sparse_matrix_multiplication: check if result responds to getDimension. Check if x dimension of result is same as first matrix and if y dimension of result is same as second matrix y dimension.
    k) Pre_scalar_multiplication: check if input responds to round
    l) Post_scalar_multiplication: check if result responds to checkSum. Check if sum of result is the same as the original * value

m) Pre_sparse_matrix_division: check if input responds to getDimension. Check if matrix are (mxn)*(n*p)
n) Post_sparse_matrix_division: Check if x dimension of result is same as first matrix and if y dimension of result is same as second matrix y dimension.
o) Pre_scalar_division: check if input responds to round
p) Post_scalar_division: check if result responds to getDimension.
q) PostDeterminant: check if result responds to round
r) PreTranspose: check if matrix responds to getDimension
s) postTranspose: check if result responds to getDimension. Check if sum or result and original matrix are the same. Removed check if result can transpose back
t) preInverse: check if result responds to getDimension
u) Pre_power: check if input responds to round
v) Post_power: check if result responds to getDimension. Check if dimension of result is the same as the original.

3) See attached files for a copy of the codes
4) Testing that was done involved testing the methods, preconditions, invariants, and postconditions on their own individually by inputting simple inputs that have an expected result.
5) Some missing functionality of the package is that the classes can store n-dimensional matrices, but none of the functions will work for any matrix other than of 2-D. But this matrix class can be extended to be used by N-dimensional matrices. Some of the more difficult methods such as determinate, inverse, matrix multiplication and division delegates the responsibility to Ruby's Matrix class to perform. Causing these methods to heavily rely on the Matrix class. The NDimensional matrices don't have much of the functionality. It is mainly there to perform operations with a sparse matrix, with the sparse matrix in front of the operation. (ie SparseMatrix + NDimensionalMatrix)