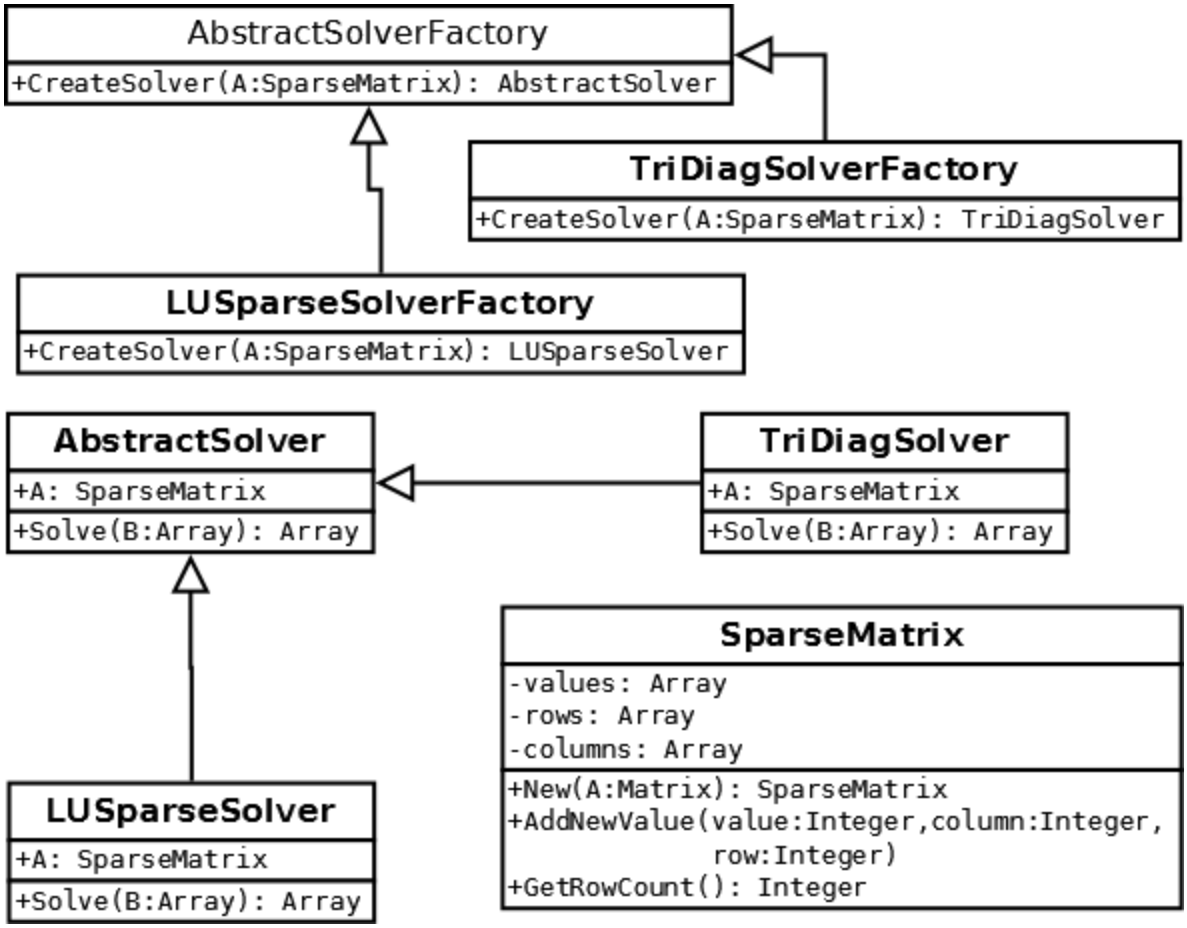
Sparse Matrix Design Report

The SparseMatrix package is designed to assist in solving linear equation problems involving sparse matrices. Our package supports expansions of different solving algorithms.

The package's class structure is shown in the UML diagram below:



Class Reference

SparseMatrix

The SparseMatrix Class is a storage class to efficiently represent the sparse matrix.
It contains three arrays, representing the value, row and column of each non-zero entry.
These three arrays are sorted sequentially from left to right, then top-down.
It also keeps track of the dimensions of the matrix in rowCount and columnCount.
It is used in other classes to solve linear equations.
It is constructed from a Ruby Matrix object.

AbstractSolverFactory

The abstract base class for all of our SolverFactories to implement.
All subclasses of this class are factories that create their respective solver objects.
It enforces the constructor to build their solvers around a SparseMatrix input.

AbstractSolver

The abstract base class for all of our Solver objects to implement.
All subclasses of this class are solvers that solve a system of linear equations problem.
It enforces the style in which all solvers implement their solving algorithm.

TriDiagSolverFactory

This factory class creates a TriDiagSolver object for a tridiagonal matrix.

TriDiagSolver

This solver class implements the Solve() method using a specialized algorithm for tridiagonal matrices. The algorithm solves the system of linear equations with a time complexity of O(n).

LUSparseSolverFactory

This factory class creates a LUSolver object for a SparseMatrix object.

LUSparseSolver

This solver class implements the Solve() method using the LU factorization method to solve the system of linear equations problem.

Additional Notes

We decided not to implement an inversion function in our library because:

1. Ruby's Matrix class already has that functionality
2. There are no specialized algorithms to improve inversion for sparse matrices as opposed to regular matrices.

We chose the 3 array system of representing an array because it supports efficient iteration, in which all the linear matrix system algorithms rely on. This means that we will not be able to support an O(1) time complexity for insertion. We decided that the benefit of faster solving time is worth the loss of insertion time.

We are not supporting matrices higher than 2 dimensions.