Group 4 members: Isaac Supeene, Braeden Soetaert

One main change that we made to our design in part 3 as opposed to part 1 and 2 is that we changed our contracts to be applied through the decorator pattern. The reason we did this is that our old way of applying contracts caused the stack to clog up too much and in some cases even resulted in buffer overflow. Using the decorator pattern resolves these issues but we had to turn our classes into a decorator class and their implementation underneath. (I.e sparse_matrix and sparse_matrix_impl) Using this example the sparse_matrix class has the constructors and includes the ContractDecorator class. Then when a non-constructor is called, the ContractDecorator class' method_missing function is called and this will call the contracts that should surround the function that is located in sparse_matrix_impl and the function itself if contracts are not already being evaluated. The reason for not evaluating contracts if one is already being evaluated is to prevent infinite recursion. Another note about the contracts is that we do have VectorBuilders but there are no contracts for these as these are intended for internal use and not for widespread use.

Another important change that we made was the decision for our matrix implementations to now inherit from the Matrix class. This decision was made so that we could leverage the working code in the matrix class for functions that were difficult to speed up. This decision also helped alleviate some of the coding burden as well as we did not have to reimplement functions that were already implemented in the Matrix class. Ideally, with more development time, we would change this so that we no longer inherited from the matrix class.

Other notes about the design that the contracts didn't really specifiy:
- We have two concrete matrix builders. One is called complete and one is called sparse. The complete one just creates Ruby's Matrix class while the sparse one will create a sparse matrix or a tridiagonal matrix if the conditions for being tridiagonal are met.
- There are also two concrete vector builders, complete and sparse, that work similarly to the matrix builders except not including the tridiagonal part.