Devin Delfino
PHYS 228: Scientific Computing
Final Project: Fun with Matrices

For my final project, I am implementing a custom Matrix class that includes many of the standard matrix operations, including some iterative techniques to calculate eigenvalues and solve linear equations. This project works in tandem with the self-initiated connection that I proposed between this course and Linear Algebra in order to see how some of these functions are executed on a computer. Because of the fact that in order to create a custom Matrix class with a relatively *complete* arsenal of functionality and operations lies beyond the scope of this assignment, I focused on implementing the basic, standard linear algebra functions. After completing most of the functionality that I had originally planned in a *fairly* accurate manner, I had to judge when I had reached a reasonable point to terminate this first sprint of development.

Shown below is a list of the functionality I have implemented within the custom Matrix class.

- Helper Functions
  - **Dimensions** - Returns the dimensions of a matrix
  - **Elements** - Returns a specific element(s) from the matrix in the form of a standard MATLAB matrix.
  - **Absolute Value** - Constructs a new matrix containing the absolute values of each element of a starting matrix
  - **Less Than** - Returns a boolean indicating whether or not each and every element within a matrix is less than a specified value.
  - **Equality** - Returns a boolean indicating whether or not the elements of two M *by* N matrices are equal within an error of 10^{-4}.
- Matrix Operations
  - **Addition, Subtraction, Multiplication**
  - **Transposition**
  - **Normalizing Vectors**
  - **LU Factorization** - Decomposes a matrix into a lower triangular matrix, upper triangular matrix, and permutation matrix.
  - **Calculating the Determinant** - Uses LU Factorization for simplicity because the determinant of a matrix is equal to the product of the determinants of the lower, upper, and permutation matrices of which it is composed.
  - **Inversion** - Also utilizes LU Factorization.

  - **Solving a Linear System** - Uses LU Factorization and forward and backwards substitutions to solve linear systems directly with Gaussian Elimination techniques.

- Iterative Methods
  - Calculating Eigenvectors and Eigenvalues
    * **Power Method** - Calculates the dominant eigenvalue and the corresponding eigenvector (dominant meaning the eigenvalue with the largest absolute value).
    * **Inverse Iteration** - Calculates the least dominant eigenvalue and the corresponding eigenvector (least dominant meaning the eigenvalue with the smallest absolute value).
  - Solving a Linear System
    * **Gauss-Seidel Method** - Approximates the solution to a linear system using a modification of the method we learned in class.

- Calculates eigenvectors and eigenvalues using the following iterative techniques:
  1. Power Method
  2. Inverse Iteration Method

- Solves linear equations using the following techniques:
  1. Gaussian Elimination
  2. Jacobi Method
  3. Gauss-Seidel Method

I believe that I satisfied my original goal to explore some of the algorithms used to execute linear algebra techniques through a programming language, as well as a few approximate iterative techniques. The *test.m* script can be run to test the main methods of the Matrix class, which uses assert statements to compare my implemented code to the standard MATLAB functions that accomplish the similar task. Each time the script is run, new matrices of designated sizes are randomly generated, so each run will test the methods with new matrices. There are a still few minor bugs that result in occasional inaccurate results for inverting a matrix, calculating the least dominant eigenvalue, and solving a linear system, but running the test script multiple times will demonstrate the general performance of the code.

I have abided by the Wheaton Honor Code in this work.