Hans Guthrie
4/1/2014
ECE 540
Dr. Day

Project 2: Matrices, and Statistics

This project reads in matrices from two different file formats and then performs advanced statistical analysis on the data.

Part 1: Reading Data

The first part consisted of building two function to read different input files. We had to read the same data formatted as a CSV file as well as binary file. We were also supposed to use the built in Timer library to track how long it took to import each file.

I found that it was slightly easier to code the `ReadCsvMatrix` function than the `ReadBinaryMatrix` function. It was also convenient that we were given a significant portion of the code needed to import the csv files. The big advantage of the binary file, however, is that it executes much faster than the csv file since the computer doesn't have to encode the text into binary as it would for the CSV file. In the end, I think the Binary format is superior since at the end of the day all we care about is performance.

Here is the output for the first part.

```
Text01.mtx was read in with 4 Rows and 200000 Columns
First entry in Text01.mtx:            -3.3620059834680900
Second entry in Text01.mtx:           -1.4198563122219019
Pinultimate entry in Text01.mtx:       0.4925230592762726
Last entry in Text01.mtx:             -0.6611294508844503
------------------
Binary01.mtx was read in with 4 Rows and 200000 Columns
First entry in Binary01.mtx:          -3.3620059834680904
Second entry in Binary01.mtx:         -1.4198563122219019
Pinultimate entry in Binary01.mtx:     0.4925230592762726
Last entry in Binary01.mtx:           -0.6611294508844503
-----------------------------------
Elapsed time for reading in CSV File (Text01.mtx) = 11.032000 seconds
Elapsed time for reading in Binary File (Binary01.mtx) = 0.272000 seconds
-----------------------------------
```

Part 2: Eigenvalues and EigenVectors

This part of the project involved writing three functions: `LargestEigenValue`, `DeflateMatrix`, and `EigenV_V`.

a) `LargestEigenValue` returns the largest eigenvector and eigenvalue for the given matrix.
b) `DeflateMatrix` removes the eigenvalue vector set from the matrix. This removal (or deflation) is shown by the following equation.
$M = [m_{i,j} - \lambda * v_i * v_j]$
where $m_{i,j}$ is the i,j element in the matrix, λ is the eigenvalue, and v_i is the i'th entry in the eigenvector
c) `EigenV_V` emulates the Eigen function in matlab. This function calculates all the eigenvalues and eigenvectors for the given matrix. The eigenvectors are returned as a matrix. The eigenvalues are returned in the diagonal values of a matrix passed by reference.

Part 3: Eigenvalues and Condition

The third and final part of the project was basically the execution of all the code we wrote.

a) We were given additional binary files to import and test with our functions. The different files are "`Binary02.mtx`", "`Binary03.mtx`", and "`Binary04.mtx`". Here the output for these can be found in Appendix A.
b) Next we were to calculate the condition of the matrix using the matrix class' built in `Condition()` function, and then compare it to our custom function. These results can also be found in Appendix A.
c) We were also asked to validate the condition of the matrices by multiplying them by a scalar value of 10,000 and the computing the condition of these scaled matrices. The initial Eigen value is identical to the scaled version, which proves the condition is not scalable.
d) An interesting fact is that if the input matrix in `EigenV_V` is scaled by a particular factor, the resulting Eigen value is also scaled by the same factor. The Eigen vector however is unaffected.

Appendix A – Output

Input Matrix 'Binary02.mtx'

```
 Columns 1 to 5
|    0.960757,   -0.268234,   -0.157583,   -0.271823,    0.063957,  |
|   -0.268234,    0.331513,    0.310962,   -0.359465,    0.117738,  |
|   -0.157583,    0.310962,    1.221752,   -0.058142,    0.104391,  |
|   -0.271823,   -0.359465,   -0.058142,    0.949024,   -0.258110,  |
|    0.063957,    0.117738,    0.104391,   -0.258110,    0.076648,  |
```

condition from object = 2441797301.790704
condition from eigens = 2441797228.996604

EigenValues Matrix

```
 Columns 1 to 5
|    1.466538,    0.000000,    0.000000,    0.000000,    0.000000,  |
|    0.000000,    1.284872,    0.000000,    0.000000,    0.000000,  |
|    0.000000,    0.000000,    0.788283,    0.000000,    0.000000,  |
|    0.000000,    0.000000,    0.000000,    0.000000,    0.000000,  |
|    0.000000,    0.000000,    0.000000,    0.000000,    0.000000,  |
```

EigenVector Matrix

```
 Columns 1 to 5
|   -0.298831,    0.628003,    0.640175,   -0.186935,    0.267476,  |
|    0.394114,    0.095334,   -0.341710,   -0.479206,    0.699415,  |
|    0.814247,   -0.096348,    0.548911,    0.024595,   -0.160658,  |
|   -0.273936,   -0.741704,    0.409417,   -0.000408,    0.455206,  |
|    0.131662,    0.192657,   -0.066975,    0.857211,    0.454149,  |
```

Test of Eigen Decomposition (V*D*V') = A

```
 Columns 1 to 5
|    0.960757,   -0.268234,   -0.157583,   -0.271823,    0.063957,  |
|   -0.268234,    0.331513,    0.310962,   -0.359465,    0.117738,  |
|   -0.157583,    0.310962,    1.221752,   -0.058142,    0.104391,  |
|   -0.271823,   -0.359465,   -0.058142,    0.949024,   -0.258110,  |
|    0.063957,    0.117738,    0.104391,   -0.258110,    0.076648,  |
```

Input Matrix 'Binary03.mtx'

Columns 1 to 7

| 1.137282, 0.065824, -0.300513, 0.325165, -0.023954, 0.095367, -0.078446, |

| 0.065824, 0.938825, 0.026421, -0.236018, 0.296991, 0.351226, 0.037831, |

| -0.300513, 0.026421, 0.771951, 0.261080, 0.022720, 0.037560, -0.043594, |

| 0.325165, -0.236018, 0.261080, 1.027120, -0.128629, -0.098836, 0.058680, |

| -0.023954, 0.296991, 0.022720, -0.128629, 1.063868, 0.258894, 0.057784, |

| 0.095367, 0.351226, 0.037560, -0.098836, 0.258894, 0.529034, -0.139937, |

| -0.078446, 0.037831, -0.043594, 0.058680, 0.057784, -0.139937, 1.097963, |

condition from object = 5.778800

condition from eigens = 5.778800

EigenValues Matrix

Columns 1 to 7

| 1.638072, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 1.389187, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 1.144893, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 1.087339, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.720785, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.302305, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.283462, |

EigenVector Matrix

Columns 1 to 7

| -0.191293, 0.838036, 0.036514, -0.167793, 0.028098, 0.435256, -0.203417, |

| 0.548073, 0.227655, 0.105757, 0.082992, 0.629854, -0.363446, -0.317660, |

| -0.038358, -0.248841, 0.135094, 0.680517, 0.231836, 0.588363, -0.235232, |

| -0.500215, 0.277740, 0.352566, 0.507681, -0.014866, -0.517852, 0.149033, |

| 0.549687, 0.163444, 0.290935, 0.246040, -0.709224, -0.026587, -0.149149, |

| 0.330212, 0.221040, -0.057761, 0.189826, 0.160908, 0.189463, 0.860781, |

| -0.011824, -0.176736, 0.870028, -0.384116, 0.140130, 0.165874, 0.130305, |

Test of Eigen Decomposition (V*D*V') = A

Columns 1 to 7

| 1.137282, 0.065824, -0.300513, 0.325165, -0.023954, 0.095367, -0.078446, |

| 0.065824, 0.938825, 0.026421, -0.236018, 0.296991, 0.351226, 0.037831, |

| -0.300513, 0.026421, 0.771951, 0.261080, 0.022720, 0.037560, -0.043594, |

| 0.325165, -0.236018, 0.261080, 1.027120, -0.128629, -0.098836, 0.058680, |

| -0.023954, 0.296991, 0.022720, -0.128629, 1.063868, 0.258894, 0.057784, |

| 0.095367, 0.351226, 0.037560, -0.098836, 0.258894, 0.529034, -0.139937, |

| -0.078446, 0.037831, -0.043594, 0.058680, 0.057784, -0.139937, 1.097963, |

Input Matrix 'Binary04.mtx'

Columns 1 to 9

| 0.702164, 0.428501, 0.078759, 0.003158, -0.270337, -0.015028, 0.149959, 0.202722, -0.120105, |

| 0.428501, 0.791707, 0.330810, 0.049941, -0.387769, -0.276266, 0.055959, 0.027694, 0.011544, |

| 0.078759, 0.330810, 0.650300, -0.210712, -0.136929, 0.064310, 0.031763, 0.001883, -0.076062, |

| 0.003158, 0.049941, -0.210712, 0.285511, 0.243198, 0.031650, -0.028542, -0.129867, 0.079296, |

| -0.270337, -0.387769, -0.136929, 0.243198, 0.767772, 0.276166, -0.234244, -0.135363, -0.017313, |

| -0.015028, -0.276266, 0.064310, 0.031650, 0.276166, 0.499974, 0.136369, 0.001676, 0.106260, |

| 0.149959, 0.055959, 0.031763, -0.028542, -0.234244, 0.136369, 0.280657, -0.078865, -0.006834, |

| 0.202722, 0.027694, 0.001883, -0.129867, -0.135363, 0.001676, -0.078865, 0.388956, 0.146244, |

| -0.120105, 0.011544, -0.076062, 0.079296, -0.017313, 0.106260, -0.006834, 0.146244, 0.398127, |

condition from object = 50424841953427.633000

condition from eigens = 50267199622998.219000

EigenValues Matrix

Columns 1 to 9

| 1.758367, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.718116, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.671593, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.649456, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.517736, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.449899, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

| 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, 0.000000, |

EigenVector Matrix

Columns 1 to 9

| 0.443949, -0.063960, 0.611434, 0.226111, -0.222440, -0.308330, 0.057668, -0.173728, -0.440136, |

| 0.577060, -0.007710, -0.059243, 0.456503, 0.300742, 0.219126, -0.130497, -0.224227, 0.499953, |

| 0.288734, 0.783207, -0.238497, 0.020123, 0.210840, -0.061998, 0.171485, 0.284915, -0.297004, |

| -0.130100, -0.218897, 0.185121, 0.485146, 0.093275, 0.301729, -0.063992, 0.734134, -0.150356, |

| -0.530038, 0.216878, 0.150177, 0.520911, 0.184244, -0.262698, 0.428736, -0.257157, 0.158601, |

| -0.227058, 0.498491, 0.547456, -0.113855, -0.052799, 0.209414, -0.538443, -0.034917, 0.223008, |

| 0.132082, 0.109963, 0.223540, -0.163765, -0.423311, 0.465636, 0.641535, 0.083406, 0.281411, |

| 0.128669, -0.114407, 0.330658, -0.371443, 0.419788, -0.462354, 0.171997, 0.384003, 0.397922, |

| -0.055852, -0.117519, 0.223434, -0.235813, 0.644029, 0.467525, 0.176798, -0.278653, -0.364813, |

Test of Eigen Decomposition (V*D*V') = A

Columns 1 to 9

| 0.702164, 0.428501, 0.078759, 0.003158, -0.270337, -0.015028, 0.149959, 0.202722, -0.120105, |

| 0.428501, 0.791707, 0.330810, 0.049941, -0.387769, -0.276266, 0.055959, 0.027694, 0.011544, |

| 0.078759, 0.330810, 0.650300, -0.210712, -0.136929, 0.064310, 0.031763, 0.001883, -0.076062, |

| 0.003158, 0.049941, -0.210712, 0.285511, 0.243198, 0.031650, -0.028542, -0.129867, 0.079296, |

| -0.270337, -0.387769, -0.136929, 0.243198, 0.767772, 0.276166, -0.234244, -0.135363, -0.017313, |

| -0.015028, -0.276266, 0.064310, 0.031650, 0.276166, 0.499974, 0.136369, 0.001676, 0.106260, |

| 0.149959, 0.055959, 0.031763, -0.028542, -0.234244, 0.136369, 0.280657, -0.078865, -0.006834, |

| 0.202722, 0.027694, 0.001883, -0.129867, -0.135363, 0.001676, -0.078865, 0.388956, 0.146244, |

| -0.120105, 0.011544, -0.076062, 0.079296, -0.017313, 0.106260, -0.006834, 0.146244, 0.398127, |

Appendix B - General Code

Project2.cpp

```
#include "MatrixRead.h"
#include "EigenValues.h"
#include <time.h>
#include "MatrixOutputs.hpp"
#define eps 2.22e-16;


#define _CRT_SECURE_NO_WARNINGS

///Part 1 of the project
void part1( )
{
	int Rows, Columns;
	clock_t Time0, Time1, Time2; //Start time, Time after CSVRead, Time after
BinaryRead
	matrix CsvMatrix, BinaryMatrix;
	/*char* csvfilename = "Text01.mtx";
	char* binaryfilename = "Binary01.mtx";*/
	char* filenames[] = { "Text01.mtx", "Binary01.mtx" }; //array holding the file
names
	Time0 = clock( ); //take initial time

	//Read CSV Matrix
	CsvMatrix = ReadCsvMatrix( filenames[ 0 ] );
	Rows = CsvMatrix.high( ); //sets the bounds fo the matrix.
	Columns = CsvMatrix.wide( ); //sets the bounds fo the matrix.
	printf( "%s was read in with %d Rows and %d Columns\n\n", filenames[ 0 ], Rows,
Columns );
	printf( "First entry in %s: \t\t%18.16lf\n", filenames[ 0 ], CsvMatrix( 0, 0 ) );
	printf( "Second entry in %s: \t\t%18.16lf\n", filenames[ 0 ], CsvMatrix( 0, 1 ) );
	printf( "Pinultimate entry in %s: \t%18.16lf\n", filenames[ 0 ], CsvMatrix( Rows -
1, Columns - 1 ) );
	printf( "Last entry in %s: \t\t%18.16lf\n", filenames[ 0 ], CsvMatrix( Rows - 1,
Columns - 2 ) );
	Time1 = clock( ); //time after CSVRead
	printf( "\n-------------------\n" );

	//Read Binary Matrix
	BinaryMatrix = ReadBinaryMatrix( filenames[ 1 ] );
	Rows = BinaryMatrix.high( ); //sets the bounds fo the matrix.
	Columns = BinaryMatrix.wide( ); //sets the bounds fo the matrix.
	printf( "%s was read in with %d Rows and %d Columns\n\n", filenames[ 1 ], Rows,
Columns );
	printf( "First entry in %s: \t\t%18.16lf\n", filenames[ 1 ], BinaryMatrix( 0, 0 )
);
	printf( "Second entry in %s: \t\t%18.16lf\n", filenames[ 1 ], BinaryMatrix( 0, 1 )
);
	printf( "Pinultimate entry in %s: \t%18.16lf\n", filenames[ 1 ], BinaryMatrix(
Rows - 1, Columns - 1 ) );
	printf( "Last entry in %s: \t\t%18.16lf\n", filenames[ 1 ], BinaryMatrix( Rows -
1, Columns - 2 ) );
	Time2 = clock( ); //time after BinaryRead
	printf( "\n-----------------------------------\n" );
```

```
	printf( "Elapsed time for reading in CSV File (%s) = %lf seconds\n",
		filenames[ 0 ],
		(double)( Time1 - Time0 ) / (double)CLOCKS_PER_SEC
		);

	printf( "Elapsed time for reading in Binary File (%s) = %lf seconds\n",
		filenames[ 1 ],
		(double)( Time2 - Time1 ) / (double)CLOCKS_PER_SEC
		);


	//Was going to try to use a large loop to avoid dupliate print statements above,
but can't do this because each matrix is processed using a different function.
	//for each( char* c in filenames )
}


///Part 3 of the project
void part3( )
{
	//array containing the filenames of different files used in the project.
	//this makes is easy to quickly switch files
	char* filenames[] = { "Binary02.mtx", "Binary03.mtx", "Binary04.mtx" };

	matrix A, EigenVec, EigenVal,EigDecomposition;
	int Rows, Columns;
	double EigCondition, MatCondition;
	char* filename = filenames[ 2 ]; //set the filename from the array of filenames
	//reads the BinaryMatrix for the particular file chosen from the 'filename' array
	A = ReadBinaryMatrix( filename );

	Rows = A.wide( );
	Columns = A.high( );
	EigenVal = eye( Rows );
	EigenVec = matrix( Rows, Columns );
	EigenVec = EigenV_V( A, EigenVal );
	EigCondition = EigenVal( 0, 0 ) / EigenVal( Rows - 1, Rows - 1 );
	MatCondition = A.condition( );
	EigDecomposition = EigenVec*EigenVal*EigenVec.transpose( );

	printf( "Input Matrix '%s'\n", filename );
	PrintMatrix( A ); //prints the input matrix

	//This prints the condition of the matrix. It is calculated
	//in two different ways. The first uses the built in Condition() from
	//the Matrix.hpp file. The second uses out custom condition function.
	printf( "condition from object = %lf\n", MatCondition );
	printf( "condition from eigens = %lf\n\n", EigCondition );

	printf( "EigenValues Matrix\n" );
	PrintMatrix( EigenVal );
	printf( "\n" );

	printf( "EigenVector Matrix\n" );
	PrintMatrix( EigenVec );
	printf( "\n" );
```

```
	printf( "Test of Eigen Decomposition (V*D*V') = A\n" );
	PrintMatrix( EigDecomposition );
	printf( "\n" );


}

void main( )
{
	//part1( ); //call the function responsible for part 1
	part3( ); //call the functions for part 2 and 3 of the project
	getchar( );
}
```

EigenValues.cpp

```
//#include "stdafx.h"
//#define eps 2.22e-16
#include "EigenValues.h"

///Function for Normalizing a pointer to vector '*y' with length 'Length'
double *NormalizeVector( double *y, int Length )
{
	double Scale, *Local;
	int k;
	Local = y;
	Scale = ( *Local ) * ( *Local );
	for ( k = 1; k < Length; k++ )
	{
		Local++; // move to next entry
		Scale += ( *Local ) * ( *Local ); // Accumulate squared values;
	} // End of loop to compute length of the vector.
	Scale = 1.0 / sqrt( Scale );
	Local = y;
	*Local *= Scale;
	for ( k = 1; k < Length; k++ )
	{
		Local++; // move to next entry
		*Local *= Scale; // Normalize this entry.
	} // End of loop to normalize the entries in the vector.
	return y;
} // end of NormalizeVector

//Calculated the sum of the squares for the input vectors '*x' and '*y'
double SumSquaredDifference( double *x, double *y, int Length )
{
	int k;
	double Distance;
	Distance = ( *x - *y ) * ( *x - *y );
	for ( k = 1; k < Length; k++ )
	{
		x++;
		y++;
		Distance += ( *x - *y ) * ( *x - *y );
	} // end of loop through data.
	return sqrt( Distance );
} // End of SumSquaredDifference

//Returns the largest eigenvector and eigenvalue for 'A'
//Uses the power method and stops iterating once sum squared
//error in vectors is below tolerance.
matrix LargestEigenValue(matrix A, double *EigenValue, double Tolerance)
{
	int k, Rows, Columns;
	double D;
	matrix X, Y;
	Rows = A.wide();
	Columns = A.high();
	X = matrix(Rows);


	if(A.isValid())
```

```
	{
		for (k=0; k < Rows; k++)
		{
			X(k) = 1.0/sqrt(1.0*Rows);
		}

		k = 1000;
		D = 1.0;

		while ( D > Tolerance && --k )
		{
			Y = A*X;
			D = SumSquaredDifference(X.AsPointer(),
NormalizeVector(Y.AsPointer(), Rows), Rows);
			X = Y;
		}
	}
	Y = A*X;

	*EigenValue = (Y(0)/X(0));

	return X;
}

//Deflates the matrix 'A' by removing an eigenvalue vector set
matrix DeflateMatrix( matrix A, double *EigenValue, matrix EigenVector )
{
	int k, Rows, Columns, m;
	matrix DeflateMatrix, Y;

	Rows = A.wide( );
	Columns = A.high( );

	DeflateMatrix = matrix( Rows, Columns );

	DeflateMatrix = A - EigenVector*EigenVector.transpose( )*( *EigenValue );

	return DeflateMatrix;
}

//Computes all the eigenvalues and vecos for a matrix 'A'.
//Eigenvalues are returned as a matrix with each column being one of the vectors
matrix EigenV_V( matrix A, matrix &EigenValues )
{
	int R, C, m, k;
	double Tolerance, EigenValue;
	Tolerance = eps;
	matrix X, Y, X2, X3, EigVector, DeflateA;
	R = A.wide( );
	C = A.high( );
	EigVector = matrix( R, C );

	for ( k = 0; k < R; k++ )
	{
		X = LargestEigenValue( A, &EigenValue, Tolerance );
		EigenValues( k, k ) = EigenValue;

		for ( m = 0; m < C; m++ )
```

```
                {
                        EigVector( m, k ) = X( m );

                }
                DeflateA = DeflateMatrix( A, &EigenValue, X );

                A = DeflateA;
        }

        return EigVector;
}
```

MatrixRead.h

```
#include "matrix.hpp"
#include <stdio.h>
#include "ReadInCsvFile.h"

#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

//Even though this a header file, we will code these functions here
//since we want to use them in future projects

#ifndef MatrixRead_h
#define MatrixRead_h 0

//Reads in a matrix from a CSV file.
matrix ReadCsvMatrix( char *Name )
{
	matrix output;
	int j, k, counter = 0;
	int    Length = 0, Rows, Columns;
	double *Data, temp;

Need to check valid allocation of output -1
	Data = ReadInCSVFile( Name, &Rows, &Columns );
	//matrix output( Rows, Columns );
	output = matrix( Rows, Columns );

	// If invalid read
	if ( !Data )
	{
		// tell user and exit.
		printf( "Error reading %s\n", Name );
		return 0;
	}

	//loop through rows to transform the Data array into a matrix
	for ( j = 0; j < Rows; j++ )
	{
		//loop through the columns
		for ( k = 0; k < Columns; k++ )
		{
			//take this point (j, k) and copy the appropriate value of the Data
array in to it
			output( j, k ) = Data[ counter ];
			counter++; //increment the counter
		}
	}

	return output;
} // end ReadCsvMatrix

//Reads in a matrix file from a binary formatted matrix.
matrix ReadBinaryMatrix( char *Name )
{
	int Rows, Columns;
	FILE *fin;
	matrix output;
```

```
        fin = fopen( Name, "rb" );

        //Gets the rows/cols from the beginning of the file
        fread( &Rows, sizeof( int ), 1, fin );
        fread( &Columns, sizeof( int ), 1, fin );

        output = matrix( Rows, Columns ); //creates  the output matrix

        if ( fin ) //if successfully opened file for reading
        {
                fread( output.AsPointer( ), sizeof( double ), Rows*Columns, fin ); //Read
through the file for elements the size of a double
                fclose( fin );
        }
        return output;
} //end ReadBinaryMatrix

#endif
```

need to check for valid allocation of output
-1

```
Project 2

0) Coding
     Comments and its Relevancy to code .......   2/2
     Structure of loops .......................   1/1
     Variable naming ..........................   1/1
     Prototypes ...............................   1/1

1) Reading in Data
     a) Code
          Error Checks during read ............   0/1
          Error Checks on data read in ........   0/1
          Efficient-Clean code ................   2/2
     b) Timing
          Properly set up test (no prints) ....   3/3
          Properly measured time ..............   3/3
          Viable results ......................   1/2
     c) Opinion ...............................   3/3

2) Eigenvalue and EigenVector Code.
     a) Largest Eigen Value ...................   5/5
     b) Deflate ...............................   5/5
     c) All of the values and vectors .........   5/5

3) Eigenvalues from matrices.
     a) Correct Values and matrices ...........   6/6
     b) Correct Condition numbers .............   3/3
     c) Correct Condition for Scaled matrix ...   3/3
     d) Eigenvalues for scaled matrix .........   3/3

Total .....................................  47/50
 Late ..  -5 points (one letter grade)
```

Timing is off a little bit
-1