Hans Guthrie
4/16/2014
ECE 540
Dr. Day

Project 3: Statistics

Part 1: Regression and Correlation

(a) We start this project by reading in a binary formatted file named `StatsData.mtx`. This file contains a 5 x 162,000 matrix. Using the header file `Stats_Support.h` we parsed this data into five separate matrices, and then computed the Linear Regression (LR) and Correlation Coefficient (CC) between each of the first four rows and the fifth row. The fifth row is the dependent data.
Here is the output we received.

```
LR Parameters & CC:
LR: Row 1
 Columns 1 to 1
|     0.079395,  |
|    -0.516183,  |
Abs of CC Row 1 & 5: 0.290579

LR: Row 2
 Columns 1 to 1
|    -0.081241,  |
|    -0.222331,  |
Abs of CC Row 2 & 5: -0.133197

LR: Row 3
 Columns 1 to 1
|     0.050018,  |
|    -0.411259,  |
Abs of CC Row 3 & 5: 0.758055

LR: Row 4
 Columns 1 to 1
|    -1.255861,  |
|     0.366336,  |
Abs of CC Row 4 & 5: -0.548706
```

(b) From the results in (a), we can see that Row 3 and Row 4 have the highest CC. We then performed a multivariable regression using Row 3 and Row 4 as the independent variables and Row 5 as the dependent variable. We also computed the Coefficient of Determination (CD) and its square root. Here is the output.

```
Multivariable Regression:
Parameters
 Columns 1 to 1
|     0.049931,  |
|    -1.251726,  |
|     0.216168,  |

Correlation of Determination (CoD) = 0.855502
Sqrt of CoD = 0.924933
Correlation Coefficient (CC) = 0.934743
```

c)

- The LR coefficient shows the relationship between the two variables. This is shown in the equation $y = ax + b$.
- The CC signifies how well the rows are related (correlated). As shown in by the output in (a), Row 1 and Row 2 have a small CC value, and thus poor correlation. Row 3 has the highest CC value.
- The weights are helpful in determining which variable is more important to the dependent variable because the higher the CC value, the higher the dependency.

Part 2: Histograms, PDF's and Confidence Intervals

(a) In this part we assumed the mean and variance of the entire row were also that of the hidden process. We calculated the 90% Confidence Interval (CI) for each row in `StatsData.mtx` assuming a sample size of 81. Here is the output from this.

```
Row 1 stats:
Mean: 3.190090
Stdev: 2.414344
90% Confidence of being between 2.750143 and 3.630038

Row 2 stats:
Mean: 0.499456
Stdev: 1.081552
90% Confidence of being between 0.302373 and 0.696539

Row 3 stats:
Mean: 2.965997
Stdev: 9.997825
90% Confidence of being between 1.144172 and 4.787823

Row 4 stats:
Mean: 0.501045
Stdev: 0.288221
90% Confidence of being between 0.448525 and 0.553566

Row 5 stats:
Mean: -0.262907
Stdev: 0.659670
90% Confidence of being between -0.383114 and -0.142701
```

(b) Then we calculated the Short Interval Mean (SIM), or mean for each of the 81 point subsections in each row. We calculated the mean and variance for each of the 2000 SIMs and compared this to the computed mean and variance from earlier. We found the mean to be the same as earlier, and the variance to be the variance of each row divided by the SIM size (81). This is shown here:

```
SIM:
Row 1:
Mean of SIMs: 3.190090
Stdev of SIMs: 0.274574
Number of times SIMs inside CI: 1805
ASD is 0.304947

Row 2:
Mean of SIMs: 0.499456
Stdev of SIMs: 0.046477
Number of times SIMs inside CI: 1797
ASD is 0.684052

Row 3:
Mean of SIMs: 2.965997
Stdev of SIMs: 1.113340
Number of times SIMs inside CI: 1784
ASD is 0.030578

Row 4:
Mean of SIMs: 0.501045
Stdev of SIMs: 0.031530
Number of times SIMs inside CI: 1809
ASD is 0.310675
```

(c) We counted the number of times each SIM was inside the 90% CI, and our data can be seen above. Comparing the data to 90% CI for Row 1 we have 1804/2000, which is about 90%. All of the other rows are very close to 90% CI as well.

(d) We found the data to be more clearly represented by histograms; we plotted the raw data, the probability density, and the Gaussian distribution of each row. The plots can be found on the following pages.

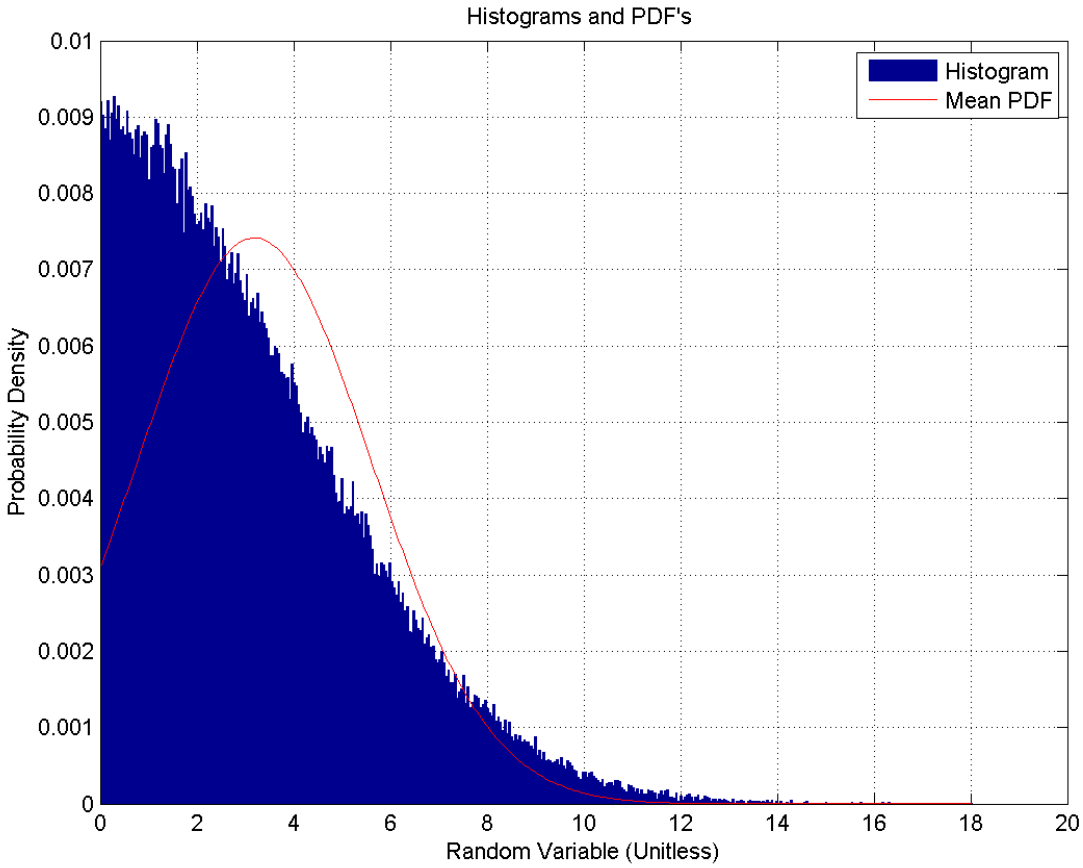
(e) We calculated the Absolute-Sum-Difference between the histogram estimate and the Gaussian Probability Density Function (PDF) using the following formula.

$$ASD = \sum_{n=1}^{N} |HE_n - PDF_n|$$

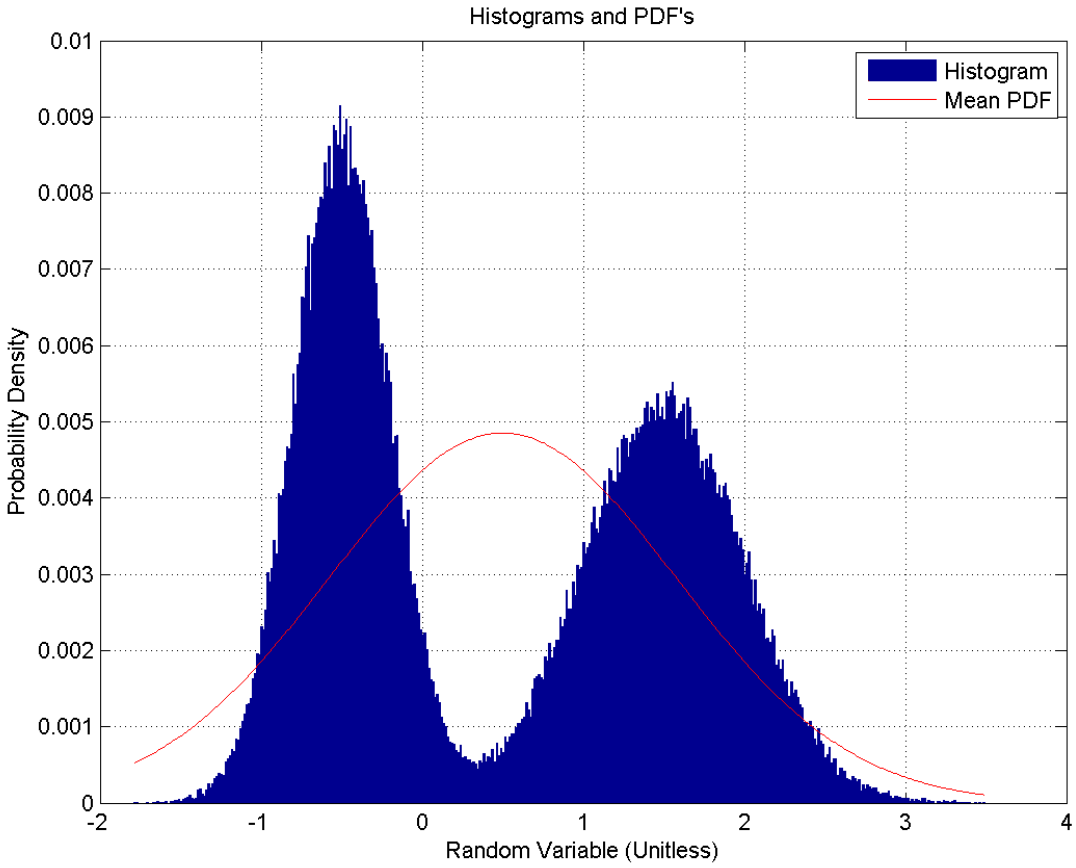
Where HE_n is the Histogram Estimate at bin n, and PDF_n is the PDF at bin n.

Plots:

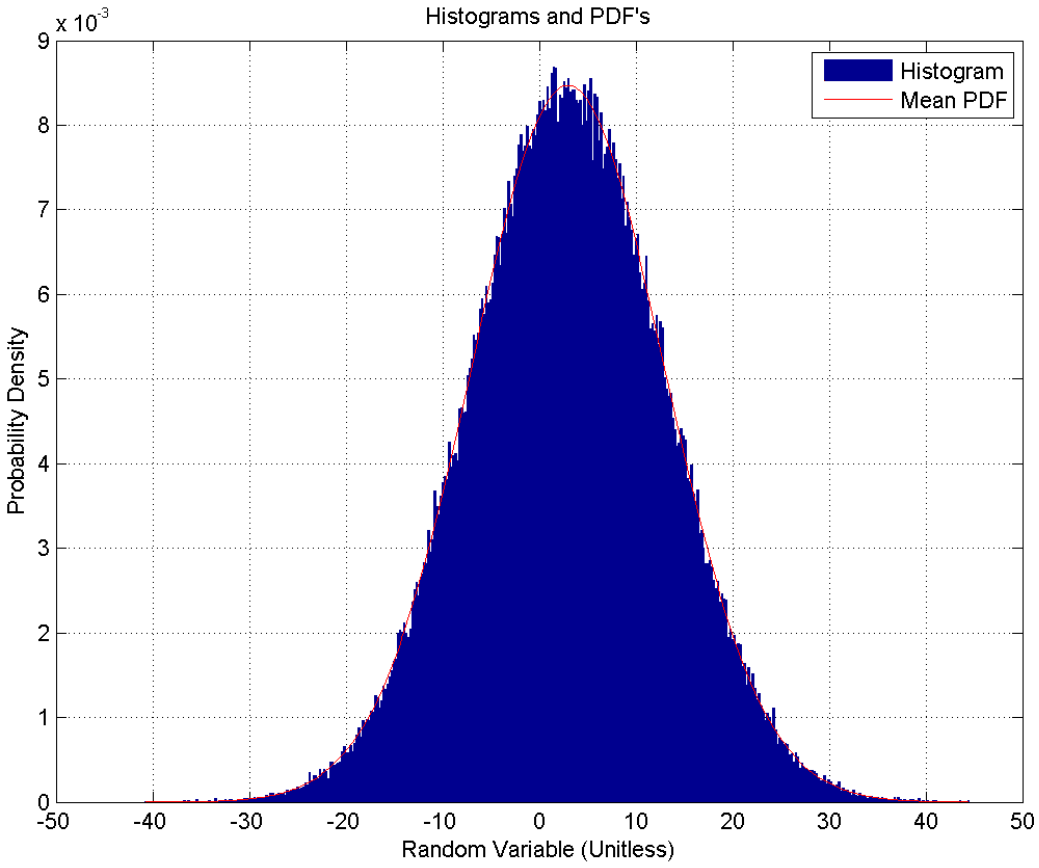
Row 1


Histograms and PDF's
Histogram
Mean PDF
Probability Density
Random Variable (Unitless)
0.01
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
0
2
4
6
8
10
12
14
16
18
20

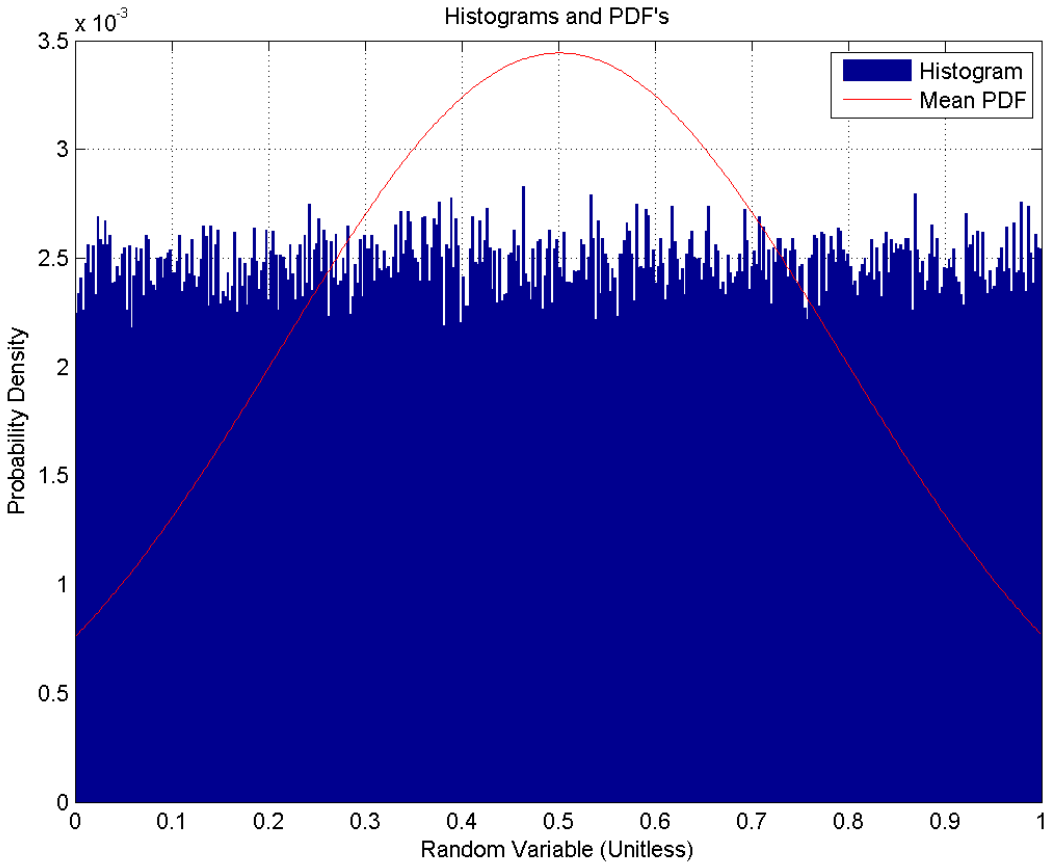
Row 2


Histograms and PDF's
Histogram
Mean PDF
Probability Density
Random Variable (Unitless)
0.01
0.009
0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
-2
-1
0
1
2
3
4

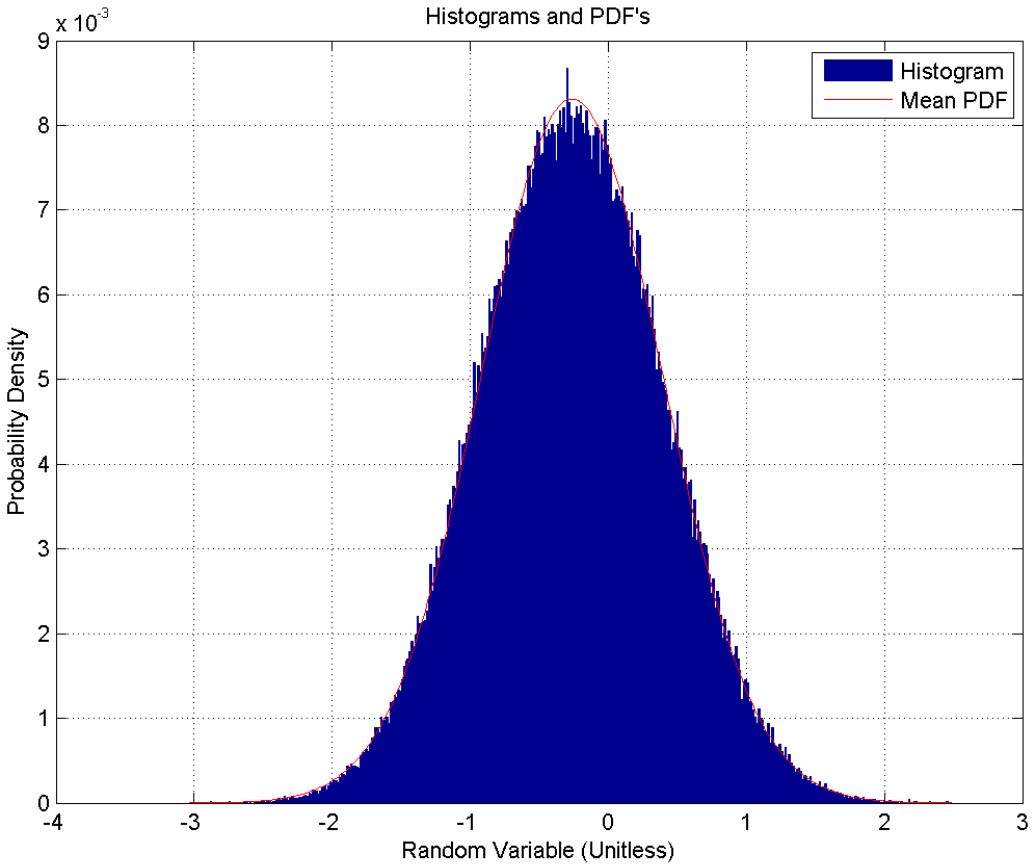
Row 3


Histograms and PDF's
x 10^-3
Probability Density
Random Variable (Unitless)
Histogram
Mean PDF
0
1
2
3
4
5
6
7
8
9
-50
-40
-30
-20
-10
0
10
20
30
40
50

Row 4


Histograms and PDF's
x 10^-3
3.5
3
2.5
2
1.5
1
0.5
0
Probability Density
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Random Variable (Unitless)
Histogram
Mean PDF

Row 5


Histograms and PDF's
x 10^-3
Probability Density
Random Variable (Unitless)
Histogram
Mean PDF
0
1
2
3
4
5
6
7
8
9
-4
-3
-2
-1
0
1
2
3

Code:

```
#include "matrix.hpp"
#include <stdio.h>
#include "MatrixRead.h"
#include "Stats_Support.h"
#include "MatrixOutputs.hpp"
#include <vector>
#include <math.h>
#include "Project3.h"
#include "RandomNumbers.h"

#ifdef _MSC_VER
#define _CRT_SECURE_NO_WARNINGS
#endif

///<Summary>
///Computes the Multivariable regression of matrix a, matrix b, and the dependent matrix.
///Requires input of the size of the matricies as an int
///</Summary>
void MultiVariableRegression( matrix a, matrix b, matrix dependent, int size )
{
	matrix In = matrix( size, 3 );

	matrix PIn;
	matrix p, x;
	double CoD, sqrootofCoD, CC;

	//create matrix In from Row 3, Row 4, and a lot of 1's.
	for ( int k = 0; k < size; k++ )
	{
		In( k, 0 ) = a( k ); // row 3
		In( k, 1 ) = b( k ); //row 4
		In( k, 2 ) = 1.0; //set all of last column to 1's
	}

	PIn = MatrixPseudoInverse( In ); //calculate the Pseudoinverse
	if ( PIn.isValid( ) )
	{
		// compute the pseudo inverse and check for valid operation.
		// Multiply pseudo inverse and dependent (row5), and check for valid
operation.
		p = PIn * dependent;
		if ( p.isValid( ) )
		{
			//prints p
			printf( "Parameters" );
			PrintMatrix( p );

			x = In * p; //x matrix is the product of In and p
			if ( x.isValid( ) )
			{
				CoD = CoefficientOfDetermination( x, dependent );
				sqrootofCoD = sqrt( CoD );
				CC = CorrelationCoefficient( x, dependent );
				printf( "Correlation of Determination (CoD) = %lg\n", CoD );
				printf( "Sqrt of CoD = %lg\n", sqrootofCoD );
				printf( "Correlation Coefficient (CC) = %lg\n", CC );
```

```
                        }// end of valid x = MtrxVector check
                } // end of valid p = MtrxVector check.
        }
} //end MultiVariableRegression

///<Summary>
///Calculates the confidence internal for the m matrix.
///Requires an int representing the number of items in the sample size (for us 81)
///</Summary>
///<param name="m">The matrix</param>
///<param name="samplesize">Num items in bin</param>
void ConfidenceInterval( matrix m, int samplesize )
{
        double Mean, Stdev, MeanLow, MeanHigh;
        int Length = m.high( );

        Mean = ComputeMean( m.AsPointer( ), Length );
        Stdev = ComputeStdev( m.AsPointer( ), Length, Mean );
        MeanLow = Mean - 1.64*Stdev / sqrt( (double)samplesize );
        MeanHigh = Mean + 1.64*Stdev / sqrt( (double)samplesize );

        printf( "Mean: %lf\nStdev: %lf\n", Mean, Stdev );
        printf( "90%% Confidence of being between %lf and %lf\n\n", MeanLow, MeanHigh );
} //end ConfidenceInterval

///<summary>
///Computes the mean and the standard deviation of a pointer to a row array.
///Prints the Mean, Standard Deviation, Number of times SIMs inside CI:
///</summary>
///<param name = "*row">Pointer to the row's data (double array)</param>
void IntervalMeanAndDev( double *row )
{
        const int binsize = 2000; //the number of subsections in the matrix
        double mean[ binsize ], stdev[ binsize ]; //arrays to hold the mean and stdev for
what we will compute
        double ComputedMean, ComputedStdev, CIHigh, CILow;
        int NumTimesInsideCI = 0;

        //loop through each bin and compute the mean & stdev for the subset
        for ( int i = 0; i < binsize; i++ )
        {
                mean[ i ] = ComputeMean( row, 162000 / binsize );
                stdev[ i ] = ComputeStdev( row, 162000 / binsize, mean[ i ] );
                row += 81; //increment to the next set of 81
        }
        ComputedMean = ComputeMean( mean, binsize ); //cache this here so we don't have to
compute it more than once
        ComputedStdev = ComputeStdev( mean, binsize, ComputedMean );

        //2c

        //Confidence intervals to use in checking number of times SIM is inside 90% CI
range
        CILow = ComputedMean - ( 1.64*sqrt( 81 * pow( ComputedStdev, 2 ) ) ) / 9;
        CIHigh = ComputedMean + ( 1.64*sqrt( 81 * pow( ComputedStdev, 2 ) ) ) / 9;

        //Dumb code that computes the number of times the SIM falls in the 90% CI range
        for ( int i = 0; i < binsize; i++ )
```

```
	{
		if ( ( mean[ i ]> CILow ) && ( mean[ i ] < CIHigh ) )
		{
			NumTimesInsideCI++;
		}
	}

	//print the results
	printf( "Mean of SIMs: %-lf\n", ComputedMean );
	printf( "Stdev of SIMs: %-lf\n", ComputeStdev( mean, binsize, ComputedMean ) );
	printf( "Number of times SIMs inside CI: %d\n", NumTimesInsideCI );
} //end IntervalMeanAndDev

///<summary>
///Computes and outputs two CSV files (Histogram_<paramref name="rownumber"/>.csv, and
HistogramPDF_ <paramref name="rownumber"/>.csv)
///</summary>
///<param name="rownumber">The row number to use in generating the filename</param>
void Histogram( double *row, int size, int rownumber )
{
	const int NUMBINS = 2000;
	const int LENGTH = (int)sqrt( size );
	const double PI = 4.0 * atan( 1.0 );
	char Filename[ 64 ];
	int Histogram[ NUMBINS ];
	double Bins[ NUMBINS ], PDF[ NUMBINS ], GaussPDF[ NUMBINS ];
	double Max, Min; //the max and min of the array
	double scale; //for the gauss stuff
	double computedMean, computedStdev;
	double ASD = 0, sum = 0;

	SearchForMaxMin( row, size, &Max, &Min );
	LoadHistogramFromVector( Histogram, LENGTH, row, size, Max, Min );
	ComputeHistogramBins( Bins, int( LENGTH ), Max, Min );
	sprintf( Filename, "Histogram%d.csv", rownumber ); //copy everything into a
pointer to the filename, getting it ready for the WriteHistogram()
	WriteHistogram( Filename, Histogram, Bins, LENGTH ); //write out the raw data

	//Convert Histogram to PDF
	for ( int m = 0; m < LENGTH; m++ )
	{
		PDF[ m ] = (double)Histogram[ m ] / (double)size;
	}
	sprintf( Filename, "HistogramPDF%d.csv", rownumber );//copy everything into a
pointer to the filename, getting it ready for the WriteHistogram()
	WritePDF( Filename, PDF, Bins, LENGTH ); //write out the PDF

	//Gauss PDF
	computedMean = ComputeMean( row, size ); //precompute stdev for efficiency
	computedStdev = ComputeStdev( row, size, computedMean );
	double Scale = ( Bins[ 1 ] - Bins[ 0 ] ) / ( sqrt( 2 * PI ) * computedStdev );
	for ( int m = 0; m < LENGTH; m++ )
	{
		GaussPDF[ m ] = Scale * exp( -( Bins[ m ] - computedMean ) *( Bins[ m ] -
computedMean ) / ( 2.0*pow( computedStdev, 2 ) ) );
	}
	sprintf( Filename, "PDF%d.csv", rownumber );
	WritePDF( Filename, GaussPDF, Bins, LENGTH );
```

```
	//ASD
	for ( int i = 0; i < NUMBINS; i++ )
	{
		sum += fabs( GaussPDF[ i ] - PDF[ i ] );
	}
	printf( "ASD is %lf\n\n", sum );
} //end Histogram

///<Summary>
/// Function to write a histogram to a file.
/// Note the histogram is an array of integers, which contain the number of times
/// the data being histogrammed fell within a bin.
/// Also included with the counts, is a double precision number that is the
/// center value for each bin.
///</Summary>
void WriteHistogram( char *name, int *Histo, double *Bins, int bins )
{
	// Open the file.
	FILE *fout = fopen( name, "w" );
	// Check for valid file open.
	if ( fout )
	{
		// Loop through the data.
		while ( bins-- )
		{
			fprintf( fout, "%d,%18.16lg\n", // writeout and
				*Histo++, *Bins++ ); // move to next entries.
		} // End of loop through bins.
		fclose( fout );
	} // End of valid file open test.
} // End of WriteHistogram

/// <summary>Function to write a Probablity Distribution to a file.
/// This is similar to the histogram write about, except
/// instead of the integer counts, a double precision probablity is given.
/// </summary>
void WritePDF( char *name, double *Pdf, double *Bins, int bins )
{
	// Open the file.
	FILE *fout = fopen( name, "w" );
	// Check for valid file open.
	if ( fout )
	{
		// Loop through the data.
		while ( bins-- )
		{
			fprintf( fout, "%18.16lg,%18.16lg\n", // writeout and
				*Pdf, *Bins ); // move to next entries.
			Pdf++; Bins++;
		} // End of loop through bins.
		fclose( fout );
	} // End of valid file open test.
} // End of WriteHistogram

///<summary>The main method.</summary>
int main( )
{
```

```
	matrix InputMatrix; //the main matrix that we're reading in
	matrix row1, row2, row3, row4, row5; //separate matricies for each row
	matrix FaultTol1, FaultTol2, FaultTol3, FaultTol4;
	double CorrelCoeff1, CorrelCoeff2, CorrelCoeff3, CorrelCoeff4;

	//1a)

	InputMatrix = ReadBinaryMatrix( "StatsData.mtx" ); //read in the matrix from file
	//calculate the boundaries for the matricies so we don't have to calculate it
multiple times
	const int NUMCOLS = InputMatrix.wide( );
	const int SAMPLESIZE = 81; //SampleSize for Confidence Interval

	//Allocate the space for the rows
	row1 = matrix( NUMCOLS, 1 );
	row2 = matrix( NUMCOLS, 1 );
	row3 = matrix( NUMCOLS, 1 );
	row4 = matrix( NUMCOLS, 1 );
	row5 = matrix( NUMCOLS, 1 );

	//matrix is created in this format (rows, cols). Don't forget row1 is InputMatrix
row0
	for ( int i = 0; i < NUMCOLS; i++ )
	{
		row1( i ) = InputMatrix( 0, i );
		row2( i ) = InputMatrix( 1, i );
		row3( i ) = InputMatrix( 2, i );
		row4( i ) = InputMatrix( 3, i );
		row5( i ) = InputMatrix( 4, i );
	}

	//Compute Linear Regression parameters (FaultTol)
	FaultTol1 = FaultTolerantRegression( row1, row5 );
	FaultTol2 = FaultTolerantRegression( row2, row5 );
	FaultTol3 = FaultTolerantRegression( row3, row5 );
	FaultTol4 = FaultTolerantRegression( row4, row5 );

	//Compute Correlation Coeffcient
	CorrelCoeff1 = CorrelationCoefficient( row1, row5 );
	CorrelCoeff2 = CorrelationCoefficient( row2, row5 );
	CorrelCoeff3 = CorrelationCoefficient( row3, row5 );
	CorrelCoeff4 = CorrelationCoefficient( row4, row5 );

	//Print what we have computed

	printf( "Linear Regression (LR) Parameters & Correlation Coefficient (CC): \n" );
	printf( "LR: Row 1" );
	PrintMatrix( FaultTol1 );
	printf( "CC Row Row 1 & 5: %lg \n\n\n", CorrelCoeff1 );

	printf( "LR: Row 2" );
	PrintMatrix( FaultTol2 );
	printf( "CC Row Row 2 & 5: %lg \n\n\n", CorrelCoeff2 );

	printf( "LR: Row 3" );
	PrintMatrix( FaultTol3 );
	printf( "CC Row Row 3 & 5: %lg \n\n\n", CorrelCoeff3 );
```

```
        printf( "LR: Row 4" );
        PrintMatrix( FaultTol4 );
        printf( "CC Row Row 4 & 5: %lg \n\n\n", CorrelCoeff4 );

        //printf( "Correlation Coefficients\n" );

        //1b)

        //Row 3 and Row 4 have the highest CC, so perform a multivariable regression on
them
        printf( "Multivariable Regression:\n" );
        MultiVariableRegression( row3, row4, row5, NUMCOLS );

        //2a)
        printf( "\n" );
        printf( "Row 1 stats:\n" );
        ConfidenceInterval( row1, SAMPLESIZE );
        printf( "Row 2 stats:\n" );
        ConfidenceInterval( row2, SAMPLESIZE );
        printf( "Row 3 stats:\n" );
        ConfidenceInterval( row3, SAMPLESIZE );
        printf( "Row 4 stats:\n" );
        ConfidenceInterval( row4, SAMPLESIZE );
        printf( "Row 5 stats:\n" );
        ConfidenceInterval( row5, SAMPLESIZE );

        //2b - d)
        printf( "SIMs:\n" );
        printf( "Row 1:\n" );
        IntervalMeanAndDev( row1.AsPointer( ) );
        Histogram( row1.AsPointer( ), NUMCOLS, 1 );
        printf( "Row 2:\n" );
        IntervalMeanAndDev( row2.AsPointer( ) );
        Histogram( row2.AsPointer( ), NUMCOLS, 2 );
        printf( "Row 3:\n" );
        IntervalMeanAndDev( row3.AsPointer( ) );
        Histogram( row3.AsPointer( ), NUMCOLS, 3 );
        printf( "Row 4:\n" );
        IntervalMeanAndDev( row4.AsPointer( ) );
        Histogram( row4.AsPointer( ), NUMCOLS, 4 );
        printf( "Row 5:\n" );
        IntervalMeanAndDev( row5.AsPointer( ) );
        Histogram( row5.AsPointer( ), NUMCOLS, 5 );

        //Generate histogram CSV files (both raw data and PDF)

        getchar( );

        return 0;
}
```