HW7 - Math5610

Andrew Sheridan

December 13, 2016

Problem 1 : Least Squares via Normal Equations

By following the algorithm on page 145 of the textbook I created the following Least Squares algorithm using Normal Equations. I also tested the code using Example 6.1 and assuring the result matched. I've also printed various steps in the algorithm to assure our results match that of the example.

Because my Vector and Matrix classes will be required on every problem in this assignment, and their class declarations are each one page in length, I have included them at the end of this assignment to refrain from printing an extra twenty pages. Their declarations are not the focus of the assignment, rather the implementation of the algorithms. These implementations, and all functions required in their execution, are included on a per problem basis.

```
1  //Andrew Sheridan
2  //Math 5610
3  //Written in C++
4  //main.cpp
5
6  #include "Matrix.h"
7  #include "Vector.h"
8  #include "MatrixOperations.h"
9  #include "MatrixFactory.h"
10 #include <iostream>
11
12 int main() {
13   //Problem 1
14   Matrix matrix1(5, 3);
15
16   matrix1[0][0] = 1;
17   matrix1[0][1] = 0;
18   matrix1[0][2] = 1;
19   matrix1[1][0] = 2;
20   matrix1[1][1] = 3;
21   matrix1[1][2] = 5;
22   matrix1[2][0] = 5;
23   matrix1[2][1] = 3;
24   matrix1[2][2] = -2;
25   matrix1[3][0] = 3;
26   matrix1[3][1] = 5;
27   matrix1[3][2] = 4;
28   matrix1[4][0] = -1;
29   matrix1[4][1] = 6;
30   matrix1[4][2] = 3;
31   Vector vector1 = Vector(5);
32   vector1[0] = 4;
33   vector1[1] = -2;
34   vector1[2] = 5;
```

```
35    vector1[3] = -2;
36    vector1[4] = 1;
37
38    Vector result1 = LeastSquares(matrix1, vector1);
39    result1.Print();
40
41    return 0;
42  }
```

```
B:
40          30          10
30          79          47
10          47          55

Y:
18          5           -21

Result:
0.347226    0.399004    -0.785917
```

```
1  /// A Least Squares algorithm via Normal Equations
2  /// Requires a matrix A and a vector b
3  Vector LeastSquares(Matrix A, Vector b) {
4    Matrix AT = A.Transpose();
5    Matrix B = AT * A;
6    Vector y = AT * b;
7    std::cout << "A:" << std::endl;
8    A.Print();
9    std::cout << "B: " << std::endl;
10   B.Print();
11   std::cout << "Y: " << std::endl;
12   y.Print();
13
14   Matrix G = CholeskyDecomposition(B);
15   Vector z = ForwardSubstitution(G, y);
16   Vector x = BackSubstitution(G.Transpose(), z);
17
18   return x;
19 }
20
21 ///Computes the Cholesky Decomposition of an n by n matrix A
22 /// Returns NULL if the matrix is not SPD
23 Matrix CholeskyDecomposition(Matrix& A) {
24   if (A.IsSymmetric() == false)
25     return NULL;
26
27   Matrix L(A.GetRows(), A.GetColumns()); //Initialize the new matrix
28   for (unsigned i = 0; i < A.GetRows(); i++) {
29     for (unsigned j = 0; j < A.GetColumns(); j++) {
30       L[i][j] = 0;
31     }
32   }
33
34   for (unsigned i = 0; i < A.GetRows(); i++) {
35     for (unsigned j = 0; j < (i + 1); j++) {
36       double entry = 0;
37       for (unsigned k = 0; k < j; k++) {
38         entry += L[i][k] * L[j][k];
39       }
```

```
40        double sqrtValue = A[i][i] - entry;
41        if (sqrtValue < 0)
42          return NULL;
43
44        // Conditional assignment. If the entry is diagonal, assign to ↩
              the sqare root of the previous value.
45        // Otherwise, Do computation for a nondiagonal entry.
46        L[i][j] = i == j ? std::sqrt(sqrtValue) : (1.0 / L[j][j] * (A[i↩
              ][j] - entry));
47      }
48    }
49
50    return L;
51  }
52
53  /// Solves a set of linear equations using forward substitution
54  /// Does not reduce matrix A
55  //A: The matrix to be reduced
56  //b: right-hand-side
57  Vector ForwardSubstitution(Matrix A, Vector b) {
58    if (A.GetRows() != b.GetSize()) return NULL;
59
60    Vector x(b.GetSize());
61
62    x[0] = b[0];
63    for (unsigned i = 0; i < A.GetRows(); i++) {
64      x[i] = b[i];
65      for (unsigned j = 0; j < i; j++) {
66        x[i] = x[i] - (A[i][j] * x[j]);
67      }
68      x[i] = x[i] / A[i][i];
69    }
70
71    return x;
72  }
73
74  /// Solves a set of linear equations using back substitution
75  /// Does not reduce matrix A
76  //A: The matrix to be reduced
77  // b: Right-Hand-Side
78  Vector BackSubstitution(Matrix A, Vector b) {
79    if (A.GetRows() != b.GetSize()) return NULL;
80
81    Vector x(b.GetSize());
82
83    for(int i = b.GetSize() - 1; i >= 0; i--)
84    {
85      x[i] = b[i];
86      for (int j = i + 1; j < b.GetSize(); j++) {
87        x[i] -= A[i][j] * x[j];
88      }
89      x[i] /= A[i][i];
90    }
91    return x;
92  }
```

```
1
2  //Andrew Sheridan
3  //Math 5610
4  //Written in C++
5  //Matrix.cpp
6
```

```
7  /// Returns the transpose of the n by n matrix A
8  Matrix Matrix::Transpose() {
9    Matrix matrix(columns, rows);
10
11   for (unsigned i = 0; i < rows; i++) {
12     for (unsigned j = 0; j < columns; j++) {
13       matrix[j][i] = entries[i][j];
14     }
15   }
16   return matrix;
17 }
18
19 ///Checks to see if matrix is symmetric
20 bool Matrix::IsSymmetric() {
21   for (unsigned int i = 0; i < rows; i++) {
22     for (unsigned int j = 0; j <= i; j++) {
23       if (entries[i][j] != entries[j][i])
24         return false;
25     }
26   }
27   return true;
28 }
```

Problem 2: Gram-Schmidt Orthogonalization

I modeled my implementation of the Gram-Schmidt method after the algorithm given on page 159 of the textbook. Using the transpose of A and Q as matrices proved unsuccessful, so I instead used a more iterative method with various loops, which proved successful. Below I have a test case where we create a diagonally dominant 5x5 matrix and compute the QR Factorization, then find the One Norm $||A - QR||$, where A is our starting matrix, and QR is the QR factorization.

The class declarations of Matrix and Vector are included in the end of this assignment, as explained in Problem 1. Because they are not the focus of the problem, I have placed them there.

```
//Andrew Sheridan
//Math 5610
//Written in C++
//main.cpp

#include "Matrix.h"
#include "Vector.h"
#include "MatrixOperations.h"
#include "MatrixFactory.h"
#include <iostream>

int main() {
    //Problem 2
  Matrix matrix2 = MatrixFactory::Instance()->DiagonallyDominant(5, 5);
  Matrix* QR = GramSchmidt(matrix2);
  Matrix Q = QR[0];
  Matrix R = QR[1];

  std::cout << "Starting Matrix: " << std::endl;
  matrix2.Print();
  std::cout << "Q: " << std::endl;
  Q.Print();
  std::cout << "R: " << std::endl;
  R.Print();

  std::cout << "Q * R" << std::endl;
  Matrix TestResult = Q * R;
  TestResult.Print();

  Matrix Norm = matrix2 - TestResult;
  double normValue = Norm.OneNorm();
  std::cout << "||A - (Q * R)|| : " << normValue << std::endl;

  return 0;
}
```

```
Starting Matrix:
50.713      0.428471    0.690885    0.71915     0.491119
0.780028    50.4109     0.579694    0.139951    0.401018
0.627317    0.324151    50.2448     0.694755    0.593902
0.631792    0.440257    0.0837265   50.7123     0.427863
0.29778     0.492085    0.740296    0.357729    50.4172

Q:
0.99971     -0.0156152  -0.0123673  -0.0123431  -0.00545983
0.0153768   0.999777    -0.00647483 -0.00868187 -0.00956081
0.0123664   0.00613281  0.999798    -0.00149318 -0.0145344
```

```
0.0124546     0.00843422    0.00118936    0.999863      -0.00674982
0.00587018    0.00962131    0.0144128     0.00657804    0.999811

R:
50.7276       1.21588       1.32633       1.36339       0.805775
0             50.4034       0.884747      0.564112      0.88559
0             0             50.2331       0.750286      1.31227
0             0             0             50.6966       0.749021
0             0             0             0             50.3896

Q * R
50.713        0.428471      0.690885      0.71915       0.491119
0.780028      50.4109       0.579694      0.139951      0.401018
0.627317      0.324151      50.2448       0.694755      0.593902
0.631792      0.440257      0.0837265     50.7123       0.427863
0.29778       0.492085      0.740296      0.357729      50.4172

||A - (Q * R)|| : 7.21645e-15
```

```cpp
//Andrew Sheridan
//Math 5610
//Written in C++
//MatrixOperations.h

#pragma once
#include "Matrix.h"
#include "Vector.h"
#include <iostream>
#include <cmath>

///Computes the QR factorization of Matrix A
///Returns a pair of matrices. The first is Q, the second, R.
Matrix* GramSchmidt(Matrix A) {
  if (A.GetRows() != A.GetColumns()) return NULL;

  //Creates a Matrix with the same number of rows and columns as A
  Matrix r(A.GetRows(), A.GetColumns());
  Matrix q(A.GetRows(), A.GetColumns());

  for (int k = 0; k < A.GetRows(); k++) {
    r[k][k] = 0;
    for (int i = 0; i < A.GetRows(); i++)
      r[k][k] = r[k][k] + A[i][k] * A[i][k];

    r[k][k] = sqrt(r[k][k]);

    for (int i = 0; i < A.GetRows(); i++)
      q[i][k] = A[i][k] / r[k][k];

    for (int j = k + 1; j < A.GetColumns(); j++) {
      r[k][j] = 0;
      for (int i = 0; i < A.GetRows(); i++)
        r[k][j] += q[i][k] * A[i][j];

      for (int i = 0; i < A.GetRows(); i++)
        A[i][j] = A[i][j] - r[k][j] * q[i][k];
    }
  }
```

```
  Matrix* QR = new Matrix[2];
  QR[0] = q;
  QR[1] = r;
  return QR;
}
```

```
//Andrew Sheridan
//Math 5610
//Written in C++
//Matrix.cpp

#include "Matrix.h"
#include "Vector.h"
#include <random>
#include <iostream>
#include <iomanip>

///Computes the 1-norm of the matrix
double Matrix::OneNorm() {
  double columnMax = 0;
  for (unsigned i = 0; i < rows; i++) {
    double columnSum = 0;
    for (unsigned j = 0; j < columns; j++) {
      columnSum += std::abs(entries[i][j]);
    }
    if (columnSum > columnMax)
      columnMax = columnSum;
  }
  return columnMax;
}
```

```
//Andrew Sheridan
//Math 5610
//Written in C++
//MatrixFactory.cpp

#include "MatrixFactory.h"
#include <random>

///Returns the instance of the MatrixFactory singleton
MatrixFactory* MatrixFactory::Instance() {
  if (!m_instance)
    m_instance = new MatrixFactory();

  return m_instance;
}

///Creates a Diagonally Dominant matrix
Matrix MatrixFactory::DiagonallyDominant(unsigned rows, unsigned ↩
    columns) {
  Matrix m(rows, columns);
  m.InitializeDiagonallyDominant();
  return m;
}
```

Problem 3 : Testing Gram-Schmidt

To test my implementation of the Gram-Schmidt algorithm I sought out examples on the Internet to compare my results with. I found two small systems at the following sites which I used to test:

http://www.math.ucla.edu/ yanovsky/Teaching/Math151B/handouts/GramSchmidt.pdf

http://web.mit.edu/18.06/www/Essays/gramschmidtmat.pdf

After setting up the matrices, I immediately compute the values of Q and R. Once those are computed, I take the product of Q and R to assure that they match the original matrix. I also checked the results against the values given at the sites to make sure that Q and R are exact matches. Below is my implementation and results, and my results match the expected values perfectly.

```
//Andrew Sheridan
//Math 5610
//Written in C++
//main.cpp

#include "Matrix.h"
#include "Vector.h"
#include "MatrixOperations.h"
#include "MatrixFactory.h"
#include <iostream>

int main() {
    //Problem 3
  Matrix matrix3(2);
  matrix3[0][0] = 4;
  matrix3[0][1] = -2;
  matrix3[1][0] = 3;
  matrix3[1][1] = 1;
  Matrix* QR = GramSchmidt(matrix3);
  Matrix Q = QR[0];
  Matrix R = QR[1];
  matrix3.Print();
  Q.Print();
  R.Print();

  Matrix Product = Q * R;
  Product.Print();

  Matrix matrix3b(3);
  matrix3b[0][0] = 1;
  matrix3b[0][1] = 1;
  matrix3b[0][2] = 0;
  matrix3b[1][0] = 1;
  matrix3b[1][1] = 0;
  matrix3b[1][2] = 1;
  matrix3b[2][0] = 0;
  matrix3b[2][1] = 1;
  matrix3b[2][2] = 1;

    Matrix* QRb = GramSchmidt(matrix3b);
  Matrix Qb = QRb[0];
  Matrix Rb = QRb[1];

  std::cout << "Matrix two" << std::endl;
  matrix3b.Print();
  std::cout << "Q:" << std::endl;
```

```
47    Qb.Print();
48    std::cout << "R:" << std::endl;
49    Rb.Print();
50    Matrix secondProduct = Qb*Rb;
51    std::cout << "QR:" << std::endl;
52    secondProduct.Print();
53
54    return 0;
55 }
```

```
First Matrix
4               -2
3               1

Q:
0.8             -0.6
0.6             0.8

R:
5               -1
0               2

QR
4               -2
3               1

Matrix two
1               1               0
1               0               1
0               1               1

Q:
0.707107        0.408248        -0.57735
0.707107        -0.408248       0.57735
0               0.816497        0.57735

R:
1.41421         0.707107        0.707107
0               1.22474         0.408248
0               0               1.1547

QR:
1               1               0
1               0               1
0               1               1
```

Problem 4 : Solving System using QR Factorization

Some of the sytems used in previous problems were trivial, so I've implemented a method which will create a random 5 by 5 matrix A which is diagonally dominant. It also creates a vector b which contains random values. We then solve the equation $Ax = b$ for x using both Gaussian Elimination and QR Factorization. We then compare the results by taking the L2 norm of their difference.

```
//Andrew Sheridan
//Math 5610
//Written in C++
//main.cpp

#include "Matrix.h"
#include "Vector.h"
#include "MatrixOperations.h"
#include "MatrixFactory.h"
#include <iostream>

int main() {
    ///Problem 4
  const int size4 = 5;
  //Creates a diagonally dominant matrix
  Matrix matrix4 = MatrixFactory::Instance()->DiagonallyDominant(size4, size4);
  Vector vector4(size4);
  vector4.InitializeRandomEntries();
  Vector vector4a = vector4;
  Vector vector4b = vector4;
  Matrix matrix4a = matrix4;
  Matrix matrix4b = matrix4;

  //The first entry in this array is Q, the second, R
  Matrix* QR4 = GramSchmidt(matrix4a);
  Vector QTy = QR4[0].Transpose() * vector4a;
  Vector x = BackSubstitution(QR4[1], QTy);

  GaussianElimination(matrix4b, vector4b);
  Vector gaussianEliminationResults = BackSubstitution(matrix4b, vector4b);

  std::cout << "Test System: " << std::endl;
  matrix4.PrintAugmented(vector4);

  std::cout << "Results of gaussian elimination and back substitution:" << std::endl;
  gaussianEliminationResults.Print();

  std::cout << "Results of QR factorization and back substitution: " << std::endl;
  x.Print();

  Vector difference = gaussianEliminationResults - x;
  std::cout << "L2 norm of the difference:" << difference.L2Norm() << std::endl;

  return 0;
}
```

```
Test System:
50.713      0.428471    0.690885    0.71915     0.491119    | 0.712955
0.780028    50.4109     0.579694    0.139951    0.401018    | 0.428471
```

```
0.627317      0.324151      50.2448       0.694755      0.593902      | 0.690885
0.631792      0.440257      0.0837265     50.7123       0.427863      | 0.71915
0.29778       0.492085      0.740296      0.357729      50.4172       | 0.491119

Results of gaussian elimination and back substitution:
0.0135243     0.00802584    0.0132285     0.0138426     0.00929043

Results of QR factorization and back substitution:
0.0135243     0.00802584    0.0132285     0.0138426     0.00929043

L2 norm of the difference:4.24919e-18
```

```cpp
//Andrew Sheridan
//Math 5610
//Written in C++
//MatrixOperations.h

#pragma once
#include "Matrix.h"
#include "Vector.h"
#include <iostream>
#include <cmath>

///Computes the QR factorization of Matrix A
///Returns a pair of matrices. The first is Q, the second, R.
Matrix* GramSchmidt(Matrix A) {
  if (A.GetRows() != A.GetColumns()) return NULL;

  //Creates a Matrix with the same number of rows and columns as A
  Matrix r(A.GetRows(), A.GetColumns());
  Matrix q(A.GetRows(), A.GetColumns());

  for (int k = 0; k < A.GetRows(); k++) {
    r[k][k] = 0;
    for (int i = 0; i < A.GetRows(); i++)
      r[k][k] = r[k][k] + A[i][k] * A[i][k];

    r[k][k] = sqrt(r[k][k]);

    for (int i = 0; i < A.GetRows(); i++)
      q[i][k] = A[i][k] / r[k][k];

    for (int j = k + 1; j < A.GetColumns(); j++) {
      r[k][j] = 0;
      for (int i = 0; i < A.GetRows(); i++)
        r[k][j] += q[i][k] * A[i][j];

      for (int i = 0; i < A.GetRows(); i++)
        A[i][j] = A[i][j] - r[k][j] * q[i][k];
    }
  }
  Matrix* QR = new Matrix[2];
  QR[0] = q;
  QR[1] = r;
  return QR;
}
```

```cpp
//Andrew Sheridan
```

```cpp
//Math 5610
//Written in C++
//Matrix.cpp

#include "Matrix.h"
#include "Vector.h"
#include <random>
#include <iostream>
#include <iomanip>

/// Returns the transpose of the n by n matrix A
Matrix Matrix::Transpose() {
  Matrix matrix(columns, rows);

  for (unsigned i = 0; i < rows; i++) {
    for (unsigned j = 0; j < columns; j++) {
      matrix[j][i] = entries[i][j];
    }
  }
  return matrix;
}

/// Solves a set of linear equations using back substitution
/// Does not reduce matrix A
//A: The matrix to be reduced
// b: Right-Hand-Side
Vector BackSubstitution(Matrix A, Vector b) {
  if (A.GetRows() != b.GetSize()) return NULL;

  Vector x(b.GetSize());

  for(int i = b.GetSize() - 1; i >= 0; i--)
  {
    x[i] = b[i];
    for (int j = i + 1; j < b.GetSize(); j++) {
      x[i] -= A[i][j] * x[j];
    }
    x[i] /= A[i][i];
  }
  return x;
}

///Outputs an augmented coefficient matrix to the console
void Matrix::PrintAugmented(Vector v) {
  if (v.GetSize()  != rows) {
    std::cout << "Vector and Matrix do not match sizes" << std::endl;
    return;
  }

  for (unsigned i = 0; i < rows; i++) {
    for (unsigned j = 0; j < columns; j++) {
      std::cout << std::setw(13) << std::left << entries[i][j];
    }
    std::cout << "| " << v[i] << std::endl;
  }
  std::cout << std::endl;
}

///Multiplies an nxn matrix A by the vector x
Vector operator *(const Matrix& A, Vector& x) {
  if (A.columns != x.GetSize()) return NULL;

  Vector result(A.rows);
  for (unsigned i = 0; i < A.rows; i++) {
```

```
66      result[i] = 0;
67      for (unsigned j = 0; j < A.columns; j++) {
68        result[i] += A.entries[i][j] * x[j];
69      }
70    }
71    return result;
72  }
```

```
1  //Andrew Sheridan
2  //Math 5610
3  //Written in C++
4  //Vector.cpp
5
6  #pragma once
7  #include "Vector.h"
8  #include <random>
9  #include <iostream>
10 #include <iomanip>
11
12 ///Computes the L2 norm of the vector
13 double Vector::L2Norm() {
14   double sum = 0;
15   for (int i = 0; i < size; i++) {
16     sum += (entries[i] * entries[i]);
17   }
18   return std::sqrt(sum);
19 }
```

```
1  //Andrew Sheridan
2  //Math 5610
3  //Written in C++
4  //MatrixFactory.cpp
5
6  #include "MatrixFactory.h"
7  #include <random>
8
9  ///Returns the instance of the MatrixFactory singleton
10 MatrixFactory* MatrixFactory::Instance() {
11   if (!m_instance)
12     m_instance = new MatrixFactory();
13
14   return m_instance;
15 }
16
17 ///Creates a Diagonally Dominant matrix
18 Matrix MatrixFactory::DiagonallyDominant(unsigned rows, unsigned ↩
       columns) {
19   Matrix m(rows, columns);
20   m.InitializeDiagonallyDominant();
21   return m;
22 }
```

Problem 5 : Solving large systems using QR Factorization

Using a similar implementation to that of HW4, I used a for loop which will solve a system of equations ranging in size $n = 10$ to $n = 160$. Once it is solved, I compare the results to the ones vector to get the absolute error. Because the code outside of *main.cpp* is identical to the code found in problems 3 and 4, I have not included it.

```
//Andrew Sheridan
//Math 5610
//Written in C++
//main.cpp

#include "Matrix.h"
#include "Vector.h"
#include "MatrixOperations.h"
#include "MatrixFactory.h"
#include <iostream>

int main() {
    ///Problem 5
  for (int size5 = 10; size5 <= 160; size5 *= 2) {
    //Creates a diagonally dominant matrix
    Matrix matrix5 = MatrixFactory::Instance()->DiagonallyDominant(size5, size5);
    Vector onesVector(size5);
    Vector vector5(size5);
    onesVector.InitializeAllOnes();
    vector5 = matrix5 * onesVector;

    //The first entry in this array is Q, the second, R
    Matrix* QR5 = GramSchmidt(matrix5);
    Vector QTy = QR5[0].Transpose() * vector5;
    Vector Result = BackSubstitution(QR5[1], QTy);

    /*std::cout << "Results with size " << size5 << std::endl;
    Result.Print();*/

    Vector difference = Result - onesVector;
    std::cout << "Current size: " << size5 << std::endl;
    std::cout << "L2 Error : " << difference.L2Norm() << std::endl;
  }
  return 0;
}
```

```
Current size: 10
L2 Error : 1.15378e-15
Current size: 20
L2 Error : 1.19058e-15
Current size: 40
L2 Error : 2.95202e-15
Current size: 80
L2 Error : 5.20977e-15
Current size: 160
L2 Error : 1.18347e-14
```

Problem 6 : Creating Least Squares Test Data

Below is my implementation of the function for best approximation by low order polynomials using normal equations. To ensure that the Vander-monde matrix A is generated correctly, I have used the function to create one and included it here.

```
//Andrew Sheridan
//Math 5610
//Written in C++
//main.cpp

#include "Matrix.h"
#include "Vector.h"
#include "MatrixOperations.h"
#include "MatrixFactory.h"
#include <iostream>

int main() {
    ///Problem 6
    Vector v6(6);
    Vector b6(6);
    for (int i = 0; i < 6; i++) {
      v6[i] = i * 0.2;
      b6[i] = std::rand();
    }
    LSFit(v6, b6, 4);


  return 0;
}
```

```
1           0           0           0
1           0.2         0.04        0.008
1           0.4         0.16        0.064
1           0.6         0.36        0.216
1           0.8         0.64        0.512
1           1           1           1
```

```
//Andrew Sheridan
//Math 5610
//Written in C++
//MatrixOperations.h

#pragma once
#include "Matrix.h"
#include "Vector.h"
#include <iostream>
#include <cmath>

///Creates a vector of data which can be used to test least squares methods
///Takes two vectors, t and b, and an int n (the size of the sample space)
Vector LSFit(Vector t, Vector b, int n) {
  int m = t.GetSize();

  Matrix A = MatrixFactory::Instance()->Ones(m, n);
  for (int j = 0; j < n - 1; j++) {
```

```
    for (int i = 0; i < m; i++) {
      A[i][j + 1] =  A[i][j] * t[i];
    }
  }
  A.Print();
  Matrix AT = A.Transpose();
  Matrix B = AT * A;
  Vector y = AT * b;

  return B / y;
}
```

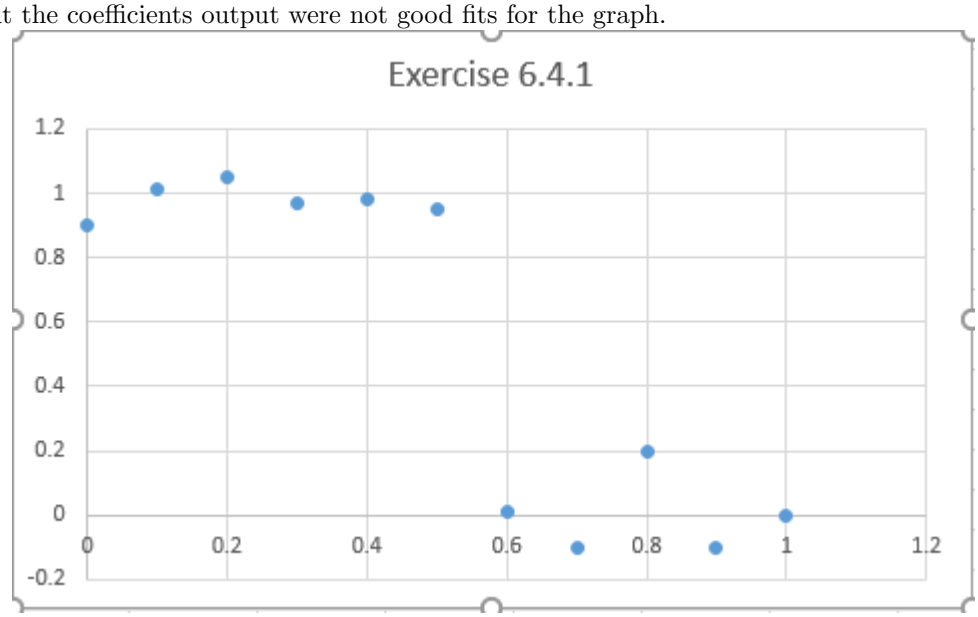
```
/// Returns the transpose of the n by n matrix A
Matrix Matrix::Transpose() {
  Matrix matrix(columns, rows);

  for (unsigned i = 0; i < rows; i++) {
    for (unsigned j = 0; j < columns; j++) {
      matrix[j][i] = entries[i][j];
    }
  }
  return matrix;
}

///Multiplies an matrix A by matrix B
Matrix operator* (Matrix A, Matrix B) {
  if (A.columns != B.rows) throw "Incompatible sizes";

  Matrix matrix(A.rows, B.columns);
  Matrix bTranspose = B.Transpose();

  //std::cout << "Starting multiplication. A:" << std::endl;
  /*A.Print();
  std::cout << "B: " << std::endl;
  B.Print();
  std::cout << "Bt: " << std::endl;
  bTranspose.Print();*/

  for (unsigned i = 0; i < A.rows; i++) {
    for (unsigned j = 0; j < B.columns; j++) {
      //matrix[i][j] = DotProduct(A[i], bTranspose[j], A.columns);
      matrix[i][j] = A[i] * bTranspose[j];
    }
  }
  return matrix;
}


///Multiplies an nxn matrix A by the vector x
Vector operator *(const Matrix& A, Vector& x) {
  if (A.columns != x.GetSize()) return NULL;

  Vector result(A.rows);
  for (unsigned i = 0; i < A.rows; i++) {
    result[i] = 0;
    for (unsigned j = 0; j < A.columns; j++) {
      result[i] += A.entries[i][j] * x[j];
    }
  }
  return result;
}

```

```
51 ///Divides an nxn matrix A by the vector x
52 Vector operator /(const Matrix& A, Vector& x) {
53   if (A.columns != x.GetSize()) return NULL;
54
55   Vector result(A.rows);
56   for (unsigned i = 0; i < A.rows; i++) {
57     result[i] = 0;
58     for (unsigned j = 0; j < A.columns; j++) {
59       result[i] += A.entries[i][j] / x[j];
60     }
61   }
62   return result;
63 }
```

```
1  //Andrew Sheridan
2  //Math 5610
3  //Written in C++
4  //MatrixFactory.cpp
5
6  #include "MatrixFactory.h"
7  #include <random>
8
9  ///Returns the instance of the MatrixFactory singleton
10 MatrixFactory* MatrixFactory::Instance() {
11   if (!m_instance)
12     m_instance = new MatrixFactory();
13
14   return m_instance;
15 }
16
17 ///Creates a matrix where every entry is a 1
18 Matrix MatrixFactory::Ones(unsigned rows, unsigned columns) {
19   Matrix m(rows, columns);
20   for (int i = 0; i < rows; i++) {
21     for (int j = 0; j < columns; j++) {
22       m[i][j] = 1;
23     }
24   }
25   return m;
26 }
```

Problem 7: Exercise 6.1

After plotting the data given in the exercise, I have decided that the break point should be at approximately $t = 0.55$, for this is where the data changes in magnitude dramatically. I followed the examples on page 149 for creating the approximation alogrithm, and my matrix A in the LSFit function looks exactly as it should, but the coefficients output were not good fits for the graph.




```
//Andrew Sheridan
//Math 5610
//Written in C++
//main.cpp

#include "Matrix.h"
#include "Vector.h"
#include "MatrixOperations.h"
#include "MatrixFactory.h"
#include <iostream>
#include <fstream>
#include <iomanip>

int main() {
    ///Problem 7
  std::ofstream output("output.txt");

  Vector t(11);
  Vector b(11);
  t[0] = 0.0;
  t[1] = 0.1;
  t[2] = 0.2;
  t[3] = 0.3;
  t[4] = 0.4;
  t[5] = 0.5;
  t[6] = 0.6;
  t[7] = 0.7;
  t[8] = 0.8;
  t[9] = 0.9;
  t[10] = 1;

  b[0] = 0.9;
```

```
33   b[1] = 1.01;
34   b[2] = 1.05;
35   b[3] = .97;
36   b[4] = .98;
37   b[5] = .95;
38   b[6] = 0.01;
39   b[7] = -.1;
40   b[8] = .02;
41   b[9] = -.1;
42   b[10] = 0;
43
44   const int N = 5;
45   Vector* coefs = new Vector[N];
46   for (int i = 0; i <= N; i++) {
47     coefs[i] = LSFit(t, b, i);
48     coefs[i].Print();
49   }
50   int size = 1 / .01;
51   Vector T(size);
52   for (int i = 0; i < size; i++) {
53     T[i] = .01 * i;
54   }
55
56   Matrix Z = MatrixFactory::Instance()->Ones(N, size);
57
58   for (int n = 0; n < N; n++) {
59     for (int i = 0; i < size; i++) {
60       Z[n] = Z[n] * coefs[n][n];
61       for (int j = n - 2; j >= 0; j--) {
62         Z[n][j] = (Z[n] * T) + coefs[n][j];
63       }
64     }
65   }
66
67   output << std::setw(13) << "t" << std::setw(13) << "p1" << std::setw↩
         (13) << "p2"
68     << std::setw(13) << "p1" << std::setw(13) << "p3" << std::setw(13)↩
           << "p4"
69     << std::setw(13) << "p5" << std::endl;
70   for (int j = 0; j < size; j++) {
71     output << std::setw(13) << T[j];
72     for (int n = 0; n < N; n++) {
73       output << std::setw(13) << Z[n][j];
74     }
75     output << std::endl;
76   }
77   output.close();
78
79   return 0;
80 }
```

```
Vander-monde matrices
1
1
1
1
1
1
1
1
1
```

```
1
1

1           0
1           0.1
1           0.2
1           0.3
1           0.4
1           0.5
1           0.6
1           0.7
1           0.8
1           0.9
1           1

1           0           0
1           0.1         0.01
1           0.2         0.04
1           0.3         0.09
1           0.4         0.16
1           0.5         0.25
1           0.6         0.36
1           0.7         0.49
1           0.8         0.64
1           0.9         0.81
1           1           1

1           0           0           0
1           0.1         0.01        0.001
1           0.2         0.04        0.008
1           0.3         0.09        0.027
1           0.4         0.16        0.064
1           0.5         0.25        0.125
1           0.6         0.36        0.216
1           0.7         0.49        0.343
1           0.8         0.64        0.512
1           0.9         0.81        0.729
1           1           1           1

1           0           0           0           0
1           0.1         0.01        0.001       0.0001
1           0.2         0.04        0.008       0.0016
1           0.3         0.09        0.027       0.0081
1           0.4         0.16        0.064       0.0256
1           0.5         0.25        0.125       0.0625
1           0.6         0.36        0.216       0.1296
1           0.7         0.49        0.343       0.2401
1           0.8         0.64        0.512       0.4096
1           0.9         0.81        0.729       0.6561
1           1           1           1           1
```

```
1  //MatrixOperations.h
2
3  ///Creates a vector of data which can be used to test least squares ↩
       methods
4  ///Takes two vectors, t and b, and an int n (the size of the sample ↩
       space)
5  Vector LSFit(Vector t, Vector b, int n) {
6    int m = t.GetSize();
7
8    Matrix A = MatrixFactory::Instance()->Ones(m, n);
9    for (int j = 0; j < n - 1; j++) {
10     for (int i = 0; i < m; i++) {
11       A[i][j + 1] =  A[i][j] * t[i];
12     }
13   }
14   A.Print();
15   Matrix AT = A.Transpose();
16   Matrix B = AT * A;
17   Vector y = AT * b;
18
19   return B / y;
20 }
```

Appendix

Class Header Files

Here I have included the header files for the classes Vector, Matrix, and MatrixFactory Because they are not relevant to the actually, but the code depends on these declarations, I have included them here.

```
//Andrew Sheridan
//Math 5610
//Written in C++
//Matrix.h

#pragma once
#include "Vector.h"

class Matrix{
public:
  //Initialization and Deconstruction
  Matrix() = default;
  Matrix(unsigned size);
  Matrix(unsigned rowCount, unsigned columnCount);
  Matrix(const Matrix &m);
  Matrix operator = (const Matrix& m);
  ~Matrix();

  void InitializeIdentityMatrix();
  void InitializeRandom();
  void InitializeRange(double minValue, double maxValue);
  void InitializeDiagonallyDominant();

  //Overloaded Operators
  Vector &operator[] (unsigned row) { return entries[row]; }
  friend bool operator == (const Matrix& A, const Matrix& B);
  friend bool operator != (const Matrix& A, const Matrix& B);
  friend Vector operator * (const Matrix& A, Vector& x);
  friend Matrix operator * (Matrix A, Matrix B);

  //Basic Algorithms
  bool IsSymmetric();
  Matrix Transpose();
  double OneNorm();
  double InfinityNorm();

  //Getters and Setters
  unsigned GetRows() { return rows; }
  unsigned GetColumns() { return columns; }
  void SetRows(unsigned r) { rows = r; }
  void SetColumns(unsigned c) { columns = c; }

  //Output
  void Print();
  void PrintAugmented(Vector v);

private:
  Vector* entries; //The entries of the matrix

  unsigned rows;
  unsigned columns;
};
```

```cpp
#pragma once
//Andrew Sheridan
//Math 5610
//Written in C++
//Vector.h

class Vector {
public:
  //Initialization and Destruction
  Vector();
  Vector(unsigned n);
  Vector(const Vector &v);
  Vector(double* v, unsigned size);
  Vector operator=(const Vector& v);
  ~Vector();

  void InitializeRandomEntries();
  void InitializeAllOnes();

  //Overloaded Operators
  double& operator[] (unsigned x) { return entries[x]; }
  friend double operator*( Vector& a, Vector& b);
  friend Vector operator*(Vector& a, double constant);
  friend Vector operator-(Vector& a, Vector& b);
  friend Vector operator/(Vector& a, double constant);

  //Basic Algorithms
  double FindMaxMagnitudeStartingAt(unsigned start);
  unsigned FindMaxIndex();
  double L2Norm();

  //Accessing Data
  void Print();
  unsigned GetSize() { return size; }
  void SetSize(unsigned newSize) { size = newSize; }

protected:
  unsigned size;

private:
  double* entries; //The stored values of the vector
};
```

```cpp
//Andrew Sheridan
//Math 5610
//Written in C++
//MatrixFactory.h
#pragma once
#include "Matrix.h"

///A singleton which creates new matrices
class MatrixFactory {
public:
  static MatrixFactory* Instance();
  Matrix Identity(unsigned rows, unsigned columns);
  Matrix Random(unsigned rows, unsigned columns);
  Matrix RandomRange(unsigned rows, unsigned columns, double min, ↩
      double max);
  Matrix DiagonallyDominant(unsigned rows, unsigned columns);
  Matrix Symmetric(unsigned size);
  Matrix SPD(unsigned size);

```

```
19 private:
20   MatrixFactory() {};
21   MatrixFactory(MatrixFactory const&) {};
22   MatrixFactory& operator=(MatrixFactory const&) {};
23   static MatrixFactory* m_instance;
24 };
```