University of Antwerp

Scientific Programming

Data Smoothing Exercise 4

Armin Halilovic - s0122210

November 13, 2015

Contents

1 Problem

We are given the following set of data points:

x	0.635	1.435	2.235	3.035	3.835	4.635	5.435
y	7.50	4.35	2.97	2.20	1.70	1.28	1.00

We will use data smoothing techniques to create an approximating function that attempts to capture important patterns in the data. We assume that the following is true:

$$\lambda_1 f_1(x_i) + ... + \lambda_n f_n(x_i) = y_i \;\; where \;\; i = 1, ..., m \gg n$$

From this, we can for construct the $m * n$ linear system $A\lambda = y$.
We will use QR factorization to decompose and solve this system.

This will be done using C++ and the GNU Scientific Library. In section 3, we will describe how we reached each solution, using the most important parts from the code. Some basic knowledge of the GSL is assumed.

2 Using the program

All of the C++ code for the program can be found in the main.cpp and in appendix A of this document. main.cpp comes accompanied by data_smoothing.sh, which contains all of the necessary UNIX commands to generate the graph images. This file relies on the graph program in the GNU plotutils package to plot graphs, so make sure that it is installed.

To compile and run the program, execute the following commands in the build/ directory:

```
cmake ..
make
chmod +x ./data_smoothing.sh
./data_smoothing.sh
```

Do not forget the .sh extension. All of the graphs will be present in the build/images/ directory. Output files which give some extra information about the solutions can be found in build/output/ and in appendix B.

3 Solutions

3.1 Datapoints plot

We begin by plotting and examining the given datapoints.

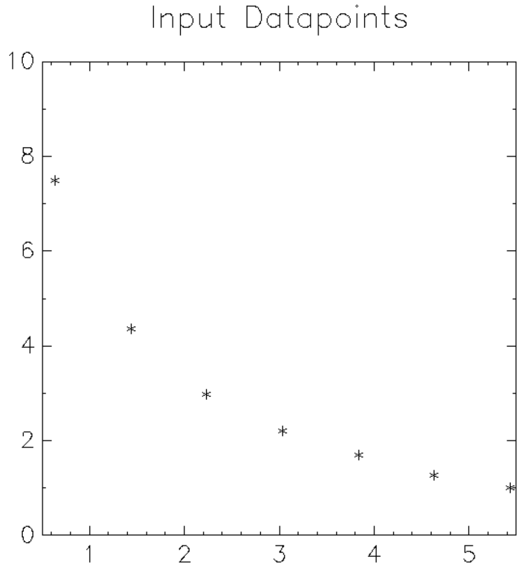

Figure 1: The given data points

We notice that the plot of the points look like they follow a decreasing exponential function. We will plot the points $(x_i, \log(y_i))$ next, because we expect it might give us an easier relationship between x and y to work with.

If we find that using the logarithmic function makes the relationship easier to work with, then we can create an approximating model $g(x)$ for the points $(x_i, \log(y_i))$. Afterwards, we can use $\exp(g(x))$ as an approximation for the originial data points.

3.2 Plot of $\log(y_i)$

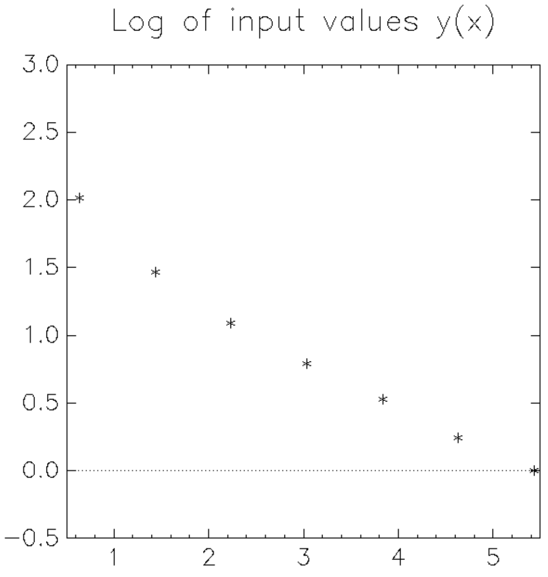

Figure 2: log of the given data points

As expected, we find a more pleasing relationship to work with. The graph of $(x_i, \log(y_i))$ seems to be following a linear equation, which gives us a good idea of which base function $f_k(x)$ where $k = 1, ..., n$ to use in

$$\lambda_1 f_1(x_i) + ... + \lambda_n f_n(x_i) = y_i \quad \text{where} \quad i = 1, ..., m \gg n$$

Since figure 2 seems to follow a linear equation, we will use $f_k(x) = x^{k-1}$ where $k = 1, 2$ to create an approximating model. We then then get

$$\lambda_1 f_1(x_i) + \lambda_2 f_2(x_i) = y_i$$

and can start constructing the overdetermined system of equations we will use to find the approximating model.

3.3 Setting up the system of equations

Now that we have a base function, we can fill in the system $A\lambda = y$ we will work with. The system looks like this:

$$\begin{bmatrix} f_1(x_i) & f_2(x_i) \end{bmatrix} \times \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} y_i \end{bmatrix}$$

A and y are constructed in the code as follows:

```
gsl_matrix *A = gsl_matrix_alloc(size, n);
gsl_vector *Y = gsl_vector_alloc(size);

for (int i = 0; i < size; i++) {    //size = amount of points
    for (int j = 0; j < n; j++) {   //n = 2
        gsl_matrix_set(A, i, j, gsl_pow_int(xa1[i], j));
    }
    gsl_vector_set(Y, i, ya2[i]);
}
```

Later, we will also construct the system with higher values for n (which we will call the degree) to see what happens to the approximation.

3.4 QR factorization

Since the system $A\lambda = y$ is overdetermined, we cannot solve it exactly, so we will find the least squares solution for it. The least squares solution will minimize the Euclidean norm of the residual, $||A\lambda - y||$.
Before we do that, QR factorization will be used to decompose the matrix A into a product $A = QR$ of an orthogonal matrix Q and an upper triangular matrix R.

This is done very easily using GSL:

```
gsl_linalg_QR_decomp(QR, tau);
gsl_linalg_QR_lssolve(QR, tau, Y, X, R);
```

3.5 Results

We get the following solution for the system:

```
Vector X, solution found by solving after QR decomposition:
 2.10317965707954
-0.403985459719526
```

These are the coefficients of the model $g(x)$ for the points $(x_i, \log(y_i))$. Thus, we have found that

$$g(x) = 2.10317965707954 - 0.403985459719526x$$

The graph for g(x) along with the points $(x_i, \log(y_i))$ can be seen on the left side in figure 3.

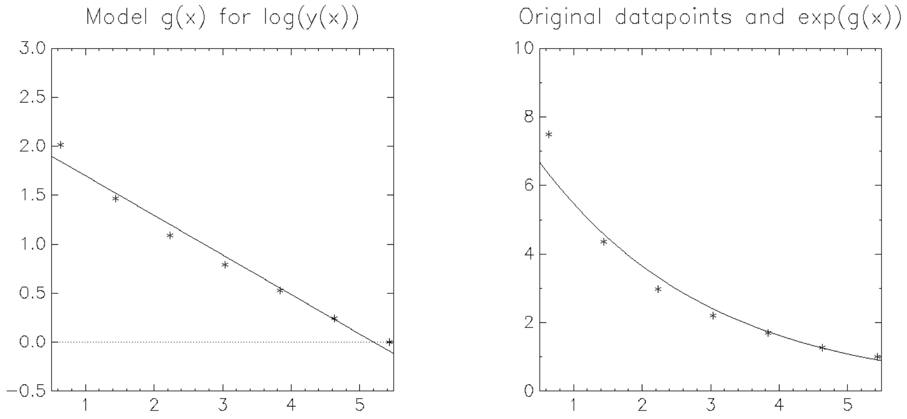

Figure 3: The resulting models

Now that we have the model $g(x)$, we can create an approximating model for the original data points. On the right side in figure 3, we plot $exp(g(x))$ to find that model.
We now have what we were looking for: we can see the trend which the original data approximately follows. However, the experiment does not stop here.

3.5.1 Condition number of A

We will calculate the condition number of matrix A to have some kind of measure for the accuracy of our solutions. The condition number is not a direct representation of the accuracy of the solutions, but it plays a big role in it. A condition number with a high order of magniture means that a small change in the input matrix A *could* cause a significant change in the solutions, whereas the opposite counts for condition numbers with smaller

orders of magnitude.
To calculate the condition number of A, we will use the following definition:

$$\kappa(A) = \frac{|max(S)|}{|min(S)|}$$

where S is the vector of singular values of A. In GSL, we can find S by using gsl_linalg_SV_decomp.

```
gsl_linalg_SV_decomp(U, V, S, work);
double condNumber, minS, maxS;
minS = gsl_vector_get(S, 0);
maxS = gsl_vector_get(S, 0);
for (int j = 0; j < n; j++) {
    if (gsl_vector_get(S, j) < minS) minS = gsl_vector_get(S, j);
    if (gsl_vector_get(S, j) > maxS) maxS = gsl_vector_get(S, j);
}
condNumber = fabs(maxS) / fabs(minS);
```

After doing this, we find a condition number of about 8 (see section B.1). Thus, we can conclude that our solution

$$\exp(g(x)) = \exp(2.10317965707954 - 0.403985459719526x)$$

is a pretty good approximation model for the given datapoints. However, if we look at the solution when we use higher degrees n, things become very different.

3.6 Higher degrees n

In figures 4 and 5 the graphs for the models where $n = 4$ and $n = 7$ can be seen. We notice that the graphs seem to display the trend which the data follows better than the case where $n = 2$. More specifically, they have minimized the sum of the squares of the residuals better. We can confirm this by looking at the residual vector $r = y - A\lambda$ in the output files in section B, they decrease as n increases. However, this comes at the cost of more processing power, since the matrix A increases in size as n increases.

In the output files we can also see that the condition number becomes very large as n increases, up to around 6E6 for $n = 7$. This means that a small change in the input matrix A could cause significant changes in the results. This makes sense, because matrix A contains elements like x^n, and these could multiply potential errors in the matrix by a big margin.

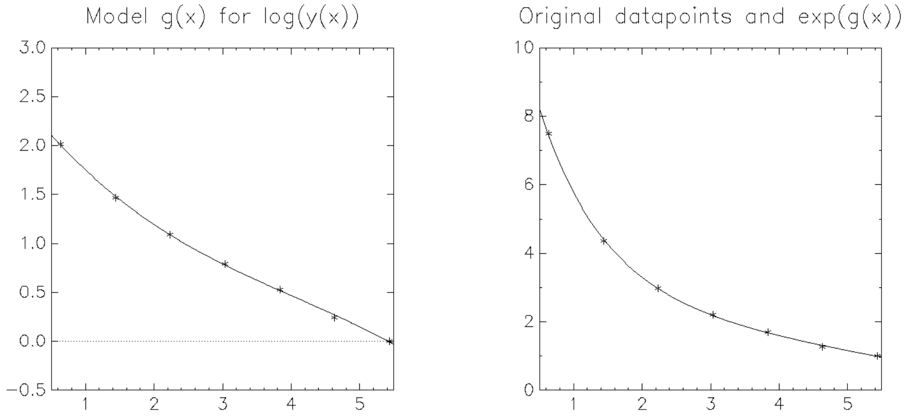

Figure 4: Graphs for n = 4

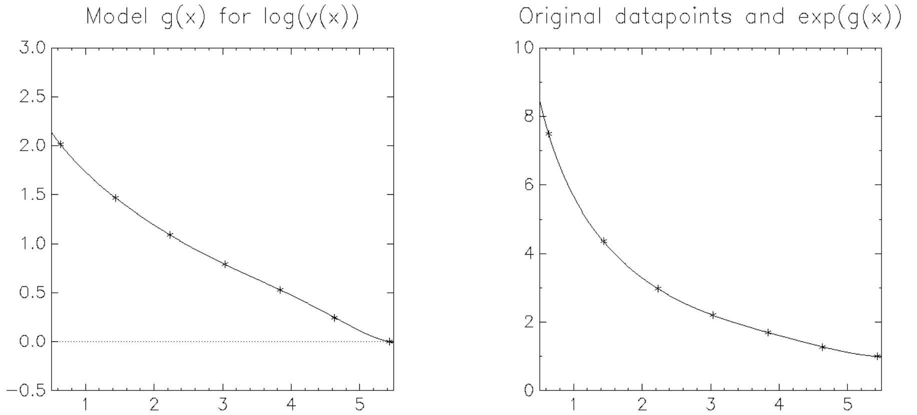

Figure 5: Graphs for n = 7

3.7 Improving the solutions

In order to get more reliable results, we will rescale the given datapoints to fit in the interval $[-1, 1]$. This will make the values x^n in the matrix A much safer to use. To rescale the the points, we use the following formula:

$$x' = (b-a)\frac{x - \min(x_i)}{\max(x_i) - \min(x_i)} + a = (1+1)\frac{x - 0.635}{5.435 - 0.635} - 1 = \frac{2x - 1.27}{4.8} - 1$$

In figures 6 and 7, we see the the graphs after rescaling the points; they look exactly the same as before rescaling. If we examine the output of the program (sections B.4 and further), we find that the condition number has indeed decreased. It decreased from 7.9 to 1.5 for n = 2, and from around 6E6 to 189.8, which is a much more favorable condition number to work with. Another thing to note, is that the residual vectors also decreased after rescaling the data points.

Thus, we can conclude that rescaling the data points to fit in the interval $[-1, 1]$ has indeed improved our results. Another way to improve the results would be to use Legendre Polynomials as base functions instead of $f_k(x) = x^k$, but they are not used in these solutions.

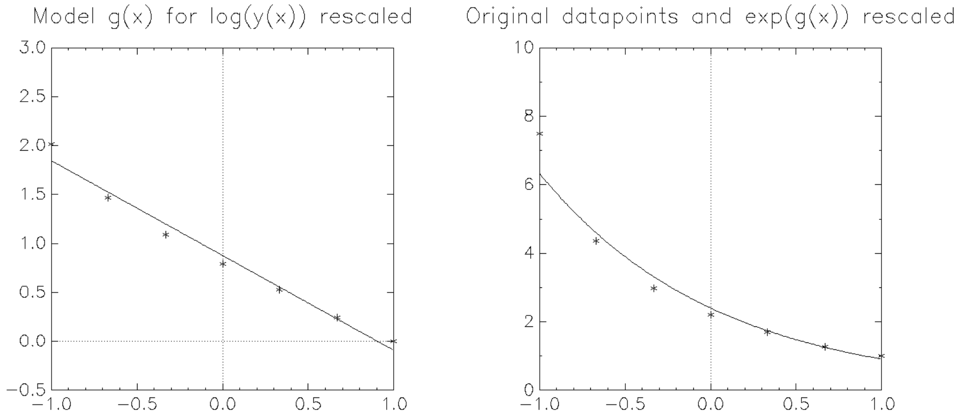

Figure 6: Graphs for n = 2

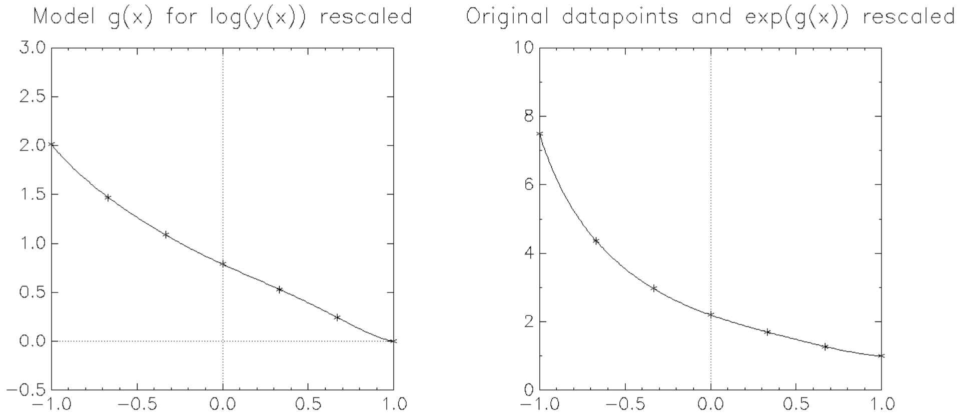

Figure 7: Graphs for n = 7

Appendices

A Code

A.1 main.cpp

```
#include <fstream>
#include <iostream>
#include <iomanip>
#include <sstream>
#include <gsl/gsl_math.h>
#include <gsl/gsl_sf_log.h>
#include <gsl/gsl_linalg.h>

void printVector(const gsl_vector * v, std::string string, std::ostream &out) {
    out << "Vector " << string << ":\n";
    for (unsigned int i = 0; i < v->size; i++) {
        out << std::setw(21) << std::setprecision(15) << gsl_vector_get(v, i) << "\n";
    }
    out << "\n";
}

void printMatrix(const gsl_matrix *m, std::string string, std::ostream &out) {
    out << "Matrix " << string << ":\n";
    for (unsigned int i = 0; i < m->size1; i++) {
        for (unsigned int j = 0; j < m->size2; j++) {
            out << std::setw(21) << std::setprecision(15) << gsl_matrix_get(m, i, j);
        }
        out << "\n";
    }
    out << "\n";
}

int main (int argc, char *argv[]) {
    if (argc < 2 or argc > 3) {
        std::cout << "Wrong amount of arguments given." << std::endl;
        std::cout << "\tUsage: " << argv[0] << " n: Creates an approximating model for
     the data points" << std::endl;
        std::cout << "\t        " << argv[0] << " n rescale: Creates an approximating model
     for the data points, after rescaling the x values" << std::endl;
        return -1;
    }
    std::stringstream ss(argv[1]);

    //size = amount of points given
    int const size = 7;

    int n;
    if (!(ss >> n) or (n > size)) {
        std::cerr << "Invalid number for degree n: " << argv[1] << std::endl;
        std::cerr << "n has to lie between 1 and 7" << std::endl;
        return -1;
    }

    /* Input data points
     *          x | 0.635 1.435 2.235 3.035 3.835 4.635 5.435
     *  f(x) = y | 7.50  4.35  2.97  2.20  1.70  1.28  1.00
     */
    double xa1[size] = {0.635, 1.435, 2.235, 3.035, 3.835, 4.635, 5.435},
            ya1[size] = {7.50, 4.35, 2.97, 2.20, 1.70, 1.28, 1.00};

    //points for model g(x) for log(y(x))
    double ya2[size];

    std::string scale = "";
    //rescale the x values we will work with
    if (argc == 3) {
        for (int i = 0; i < size; i++) {
            xa1[i] = (2 * xa1[i] - 1.27) / 4.8 - 1;
        }
        scale = "_rescaled";
    }

    //Files to write data points for graphs to
    std::ofstream output, datapoints, log_datapoints, gx, exp_gx;
    output.open("output/output_degree" + ss.str() + scale + ".txt");
    datapoints.open("images/datapoints.dat");
    log_datapoints.open("images/log_datapoints.dat");
    gx.open("images/gx_degree" + ss.str() + scale + ".dat");
    exp_gx.open("images/exp_gx_degree" + ss.str() + scale + ".dat");

    if (!datapoints.is_open()) {
        std::cout << "Could not open file 'datapoints.dat', make sure the images folder
     exists" << std::endl;
        return 0;
    }

    //Mark the following points on the graph with a plus sign
    datapoints << "#m=0,S=3\n";
    log_datapoints << "#m=0,S=3\n";
    gx << "#m=0,S=3\n";
    exp_gx << "#m=0,S=3\n";

    //Output given data points to files
    for (int i = 0; i < size; i++) {
        datapoints << xa1[i] << " " << ya1[i] << std::endl;
        exp_gx     << xa1[i] << " " << ya1[i] << std::endl;

        ya2[i] = gsl_sf_log(ya1[i]);
        log_datapoints << xa1[i] << " " << ya2[i] << std::endl;
        gx             << xa1[i] << " " << ya2[i] << std::endl;
    }

    // Initialize matrices A and Y for the equation A lamda = Y
    gsl_matrix *A = gsl_matrix_alloc(size, n);
    gsl_vector *Y = gsl_vector_alloc(size);

    //Put the input data into matrix A and vector Y
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < n; j++) {
            gsl_matrix_set(A, i, j, gsl_pow_int(xa1[i], j));
        }
        gsl_vector_set(Y, i, ya2[i]);
    }

    gsl_vector *tau = gsl_vector_alloc(n),
               *X = gsl_vector_alloc(n),
               *R = gsl_vector_alloc(size),
               *S = gsl_vector_alloc(n),
               *work = gsl_vector_alloc(n);
    gsl_matrix *QR = gsl_matrix_alloc(size, n),
               *U = gsl_matrix_alloc(size, n),
               *V = gsl_matrix_alloc(n, n);
    gsl_matrix_memcpy(QR, A);
    gsl_matrix_memcpy(U, A);

    gsl_linalg_QR_decomp(QR, tau);
    gsl_linalg_QR_lssolve(QR, tau, Y, X, R);

    printMatrix(A, "input A", output);
    printVector(Y, "input Y", output);
    //printMatrix(QR, "QR, received by QR decomposition", output);
    printVector(X, "X, solution found by solving after QR decomposition", output);
    printVector(R, "residual R = y - Ax", output);

    //Connect the following data points with a line
    gx << "#m=1,S=0\n";
    exp_gx << "#m=1,S=0\n";

    //Write new data points to files
    double leftLimit = 0.5, rightLimit = 5.5;
    if (argc == 3) {
        leftLimit = -1;
        rightLimit = 1;
    }
    for (double x = leftLimit; x < rightLimit; x = x + 0.01) {
        double y = 0;
        for (int i = 0; i < n; i++) {
            y += gsl_vector_get(X, i) * gsl_pow_int(x, i);
        }
        gx     << x << " " << y      << std::endl;
        exp_gx << x << " " << exp(y) << std::endl;
    }

    //The condition number we will use is max(S) / min(S)
    gsl_linalg_SV_decomp(U, V, S, work);
    double condNumber, minS, maxS;
    minS = gsl_vector_get(S, 0);
    maxS = gsl_vector_get(S, 0);
    for (int j = 0; j < n; j++) {
        if (gsl_vector_get(S, j) < minS) minS = gsl_vector_get(S, j);
        if (gsl_vector_get(S, j) > maxS) maxS = gsl_vector_get(S, j);
    }
    condNumber = fabs(maxS) / fabs(minS);
    output << "Calculating condition number by: abs(max(singular values)) /
     abs(min(singular values)):\n\t";
    output << "Condition number: " << condNumber << std::endl;

    //Free the memory and close the files
    gsl_matrix_free(A);
    gsl_matrix_free(QR);
    gsl_matrix_free(U);
    gsl_matrix_free(V);
    gsl_vector_free(Y);
    gsl_vector_free(tau);
    gsl_vector_free(X);
    gsl_vector_free(R);
    gsl_vector_free(S);
    gsl_vector_free(work);
    output.close();
    datapoints.close();
    log_datapoints.close();
    gx.close();
```

```
    exp_gx.close();
    return 0;
}
```

A.2 data_smoothing.sh

```
#!/bin/bash

DIR=”images”

for n in 2 4 7
do
    ./data_smoothing.bin $n rescale
    graph −T png −x −1 1 −y −0.5 3  −F HersheySans −L ”Model g(x) for log(y(x))
     rescaled” −−bitmap−size 820x820 < $DIR/gx_degree$n”_rescaled.dat” >
     $DIR/gx_degree$n”_rescaled.png”
    graph −T png −x −1 1 −y 0 10  −F HersheySans −L ”Original datapoints and exp(g(x))
     rescaled” −−bitmap−size 820x820 < $DIR/exp_gx_degree$n”_rescaled.dat” >
     $DIR/exp_gx_degree$n”_rescaled.png”

    ./data_smoothing.bin $n
    graph −T png −x 0.5 5.5 −y −0.5 3  −F HersheySans −L ”Model g(x) for log(y(x))”
     −−bitmap−size 820x820 < $DIR/gx_degree$n.dat > $DIR/gx_degree$n.png
    graph −T png −x 0.5 5.5 −y 0 10  −F HersheySans −L ”Original datapoints and
     exp(g(x))” −−bitmap−size 820x820 < $DIR/exp_gx_degree$n.dat >
     $DIR/exp_gx_degree$n.png

done

graph −T png −x 0.5 5.5 −y 0 10 −F HersheySans −L ”Input Datapoints” −−bitmap−size
     820x820 < $DIR/datapoints.dat > $DIR/datapoints.png
graph −T png −x 0.5 5.5 −y −0.5 3 −F HersheySans −L ”Log of input values y(x)”
     −−bitmap−size 820x820 < $DIR/log_datapoints.dat > $DIR/log_datapoints.png
```

B Output

B.1 output_degree2.txt

```
Matrix input A:
                    1                 0.635
                    1                 1.435
                    1                 2.235
                    1                 3.035
                    1                 3.835
                    1                 4.635
                    1                 5.435

Vector input Y:
     2.01490302054226
     1.47017584510059
     1.08856195281461
     0.78845736036427
     0.53062825106217
    0.246860077931526
                    0

Vector X, solution  found by solving  after  QR decomposition:
     2.10317965707954
   −0.403985459719526

Vector  residual  R = y − Ax:
    0.168254130384629
  −0.0532846772814243
   −0.111710201791788
  −0.0886264264665059
  −0.0232671679929855
   0.0161530266519902
   0.0924813164960849

Calculating condition  number by: abs(max(singular values)) / abs(min(singular values)):
   Condition number: 7.85470336979176
```

B.2 output_degree4.txt

```
Matrix input A:
                    1                 0.635              0.403225           0.256047875
                    1                 1.435              2.059225           2.954987875
                    1                 2.235              4.995225          11.164327875
                    1                 3.035              9.211225          27.956067875
                    1                 3.835             14.707225          56.402207875
                    1                 4.635             21.483225          99.574747875
                    1                 5.435             29.539225         160.545687875

Vector input Y:
     2.01490302054226
     1.47017584510059
     1.08856195281461
     0.78845736036427
     0.53062825106217
    0.246860077931526
                    0
```

```
Vector X, solution found by solving after QR decomposition:
      2.54045519739697
    −0.930438126050449
     0.153796078494285
  −0.0126765164724805

Vector residual R = y − Ax:
    0.0065074045416499
  −0.0143424186779639
 0.000914737155382888
  0.00961714732510905
    0.0114732537472656
  −0.0227892319514701
  0.00861910786002648

Calculating condition number by: abs(max(singular values)) / abs(min(singular values)):
    Condition number: 842.570124157451
```

B.3 output_degree7.txt

```
Matrix input A:
                     1                 0.635              0.403225           0.256047875
      0.162590400625   0.103244904396875  0.0655605142920156
                     1                 1.435              2.059225           2.954987875
      4.240407600625    6.08498490689688    8.73195334139702
                     1                 2.235              4.995225          11.164327875
     24.952272800625    55.7683297093969    124.642216900502
                     1                 3.035              9.211225          27.956067875
     84.846666000625    257.509631311897    781.541731031607
                     1                 3.835             14.707225          56.402207875
    216.302467200625    829.519961714397    3181.20905317471
                     1                 4.635             21.483225          99.574747875
    461.528956400625     2139.1867129169    9915.13041436981
                     1                 5.435             29.539225         160.545687875
    872.565813600625    4742.39519691939    25774.9178952569

Vector input Y:
      2.01490302054226
      1.47017584510059
      1.08856195281461
      0.78845736036427
      0.53062825106217
     0.246860077931526
                     0

Vector X, solution found by solving after QR decomposition:
      2.82184898462098
     −1.79017226707901
      1.09426860787786
    −0.519263507385884
     0.145404637042612
   −0.0211970695038202
    0.0012259659937529

Vector residual R = y − Ax:
                     0
                     0
                     0
                     0
```

```
                        0
                        0
                        0

Calculating condition number by: abs(max(singular values)) / abs(min(singular values)):
    Condition number: 5579987.35768587
```

B.4 output_degree2_rescaled.txt

```
Matrix input A:
                          1                    −1
                          1    −0.666666666666667
                          1    −0.333333333333333
                          1 2.22044604925031e−16
                          1     0.333333333333333
                          1     0.666666666666667
                          1                     1

Vector input Y:
      2.01490302054226
      1.47017584510059
      1.08856195281461
      0.78845736036427
      0.53062825106217
     0.246860077931526
                     0

Vector X, solution found by solving after QR decomposition:
     0.877083786830776
    −0.969565103326861

Vector residual R = y − Ax:
     0.168254130384628
  −0.0532846772814237
    −0.111710201791788
  −0.0886264264665057
  −0.0232671679929853
    0.0161530266519902
    0.0924813164960847

Calculating condition number by: abs(max(singular values)) / abs(min(singular values)):
    Condition number: 1.5
```

B.5 output_degree4_rescaled.txt

```
Matrix input A:
                          1                    −1                    1                    −1
                          1    −0.666666666666667    0.444444444444444    −0.296296296296296
                          1    −0.333333333333333    0.111111111111111    −0.037037037037037
                          1 2.22044604925031e−16 4.93038065763132e−32 1.09476442525376e−47
                          1     0.333333333333333    0.111111111111111    0.0370370370370371
                          1     0.666666666666667    0.444444444444445     0.296296296296296
                          1                     1                    1                     1

Vector input Y:
      2.01490302054226
      1.47017584510059
      1.08856195281461
      0.78845736036427
      0.53062825106217
     0.246860077931526
                     0

Vector X, solution found by solving after QR decomposition:
     0.778840213039162
    -0.833267198214751
     0.221048041031133
     -0.17524016371557

Vector residual R = y - Ax:
   0.00650740454164981
   -0.0143424186779636
  0.000914737155382776
   0.00961714732510894
    0.0114732537472656
   -0.0227892319514699
   0.00861910786002637

Calculating condition number by: abs(max(singular values)) / abs(min(singular values)):
    Condition number: 6.8930865345215
```

B.6 output_degree7_rescaled.txt

```
Matrix input A:
                     1                    -1                     1                    -1
                1                    -1                     1
                     1    -0.666666666666667     0.444444444444444    -0.296296296296296
       0.197530864197531    -0.131687242798354    0.0877914951989026
                     1    -0.333333333333333     0.111111111111111   -0.0370370370370370
      0.0123456790123457 -0.00411522633744856   0.00137174211248285
                     1 2.22044604925031e-16 4.93038065763132e-32 1.09476442525376e-47
    2.43086534291451e-63 5.39760534693403e-79  1.1985091468012e-94
                     1     0.333333333333333     0.111111111111111    0.0370370370370371
      0.0123456790123457   0.00411522633744857   0.00137174211248286
                     1     0.666666666666667     0.444444444444445     0.296296296296296
       0.197530864197531     0.131687242798354    0.0877914951989027
                     1                     1                     1                     1
                1                     1                     1

Vector input Y:
      2.01490302054226
      1.47017584510059
      1.08856195281461
      0.78845736036427
      0.53062825106217
     0.246860077931526
                     0

Vector X, solution found by solving after QR decomposition:
     0.788457360364271
    -0.805603884744018
     0.212677114686377
    -0.291647809141079
    -0.227968714660536
    0.0898001836139637
    0.234285749881022

Vector residual  R = y − Ax:
                   0
                   0
                   0
                   0
                   0
                   0
                   0

Calculating condition number by: abs(max(singular values)) / abs(min(singular values)):
   Condition number: 189.814113008495
```
