

TMA265/MMA600 - Numerical Linear Algebra

Computer Exercise 1

Maitreya Dave * Alan Ali Doosti

October 5, 2020

Abstract

We have studied and understood the implementation of different methods to solve a Linear Least Squares problem. The exact parameters were completely recovered when there was no noise added to the data and recovered to certain extent with Gaussian noise added to the data.

*Department of Mathematical Sciences, Chalmers University of Technology and University of Gothenburg, SE-42196 Gothenburg, Sweden, e-mail: maitreya@student.chalmers.se

Contents

| | | |
|----------|-----------------------------------------------------------------------------------------|-----------|
| 1 | Introduction | 4 |
| 2 | Theory and Methods | 5 |
| 2.1 | Linearization | 5 |
| 2.2 | Formulating Linear Least Squares (LLS) Problem | 5 |
| 2.3 | Data Generation | 6 |
| 2.4 | Noise Generation | 6 |
| 2.5 | Solution Methods | 7 |
| 2.5.1 | Method of Normal Equations | 7 |
| 2.5.2 | QR Factorization | 7 |
| 2.5.3 | Singular Value Decomposition (SVD) | 7 |
| 3 | Numerical Examples | 8 |
| 3.1 | Recovery of exact parameters | 8 |
| 3.2 | Analyzing relative errors for different noise levels γ . . . | 9 |
| 3.3 | Analyzing relative errors for different no. of discretiza- tion points N | 11 |
| 3.4 | Minimum N Required | 13 |
| 4 | Conclusion | 15 |
| A | MATLAB Code 1: "lab10.m" | 17 |
| B | MATLAB Code 2: "lab11.m" | 19 |
| C | MATLAB Code 3: "lab12.m" | 24 |

List of Figures

| | | |
|---|--------------------------------------------------------------------------------------|----|
| 1 | Recovery of Exact Parameters using $x = A^{-1}b$ | 8 |
| 2 | Relative Errors in Exact Parameters for varying noise levels and $N = 125$ | 10 |
| 3 | Relative Errors in Exact Parameters for varying N | 12 |
| 4 | Relative Errors for \mathbf{N} vs δ | 13 |

List of Tables

| | | |
|---|-------------------------------------------------------------|----|
| 1 | Relative Error's for Recovery of Exact Parameters | 8 |
| 2 | Relative Error in A for $N = 200$ | 14 |
| 3 | Relative Error in E for $N = 200$ | 14 |
| 4 | Relative Error in T_0 for $N = 200$ | 14 |

1 Introduction

Here I present the condensed form of the given problem statement and the outline of how I decided to carryout the assignment.

1. Linearize the non-linear equation.
2. Write a data generation function.
3. Add a method to generate noisy data.
4. Implement method of normal equations, QR Factorization and SVD.
5. Add a method to calculate relative error's.
6. Study the plots of Relative Errors vs Random Noise for each method.
7. Study the plots of Relative Errors vs Number of discretization points for each method.
8. Suggest a minimal N within a noise level σ to recover the original parameters A , E , T_0 .

2 Theory and Methods

2.1 Linearization

$$y = A * e^{\left(\frac{E}{T-T_0}\right)} \quad (1a)$$

$$\ln y = \ln(A * e^{\left(\frac{E}{T-T_0}\right)}) \quad (1b)$$

$$\ln y = \ln A + \ln(e^{\left(\frac{E}{T-T_0}\right)}) \quad (1c)$$

$$\ln y = \ln A + \frac{E}{T - T_0} \quad (1d)$$

$$\ln \left(\frac{y}{A} \right) = \frac{E}{T - T_0} \quad (1e)$$

$$E = +(\ln y - \ln A) * (T - T_0) \quad (1f)$$

$$E = +(\ln y)T - (\ln A)T - (\ln y)T_0 + (\ln A)T_0 \quad (1g)$$

$$(\ln y)T = +(\ln y)T_0 + (\ln A)T + E - (\ln A)T_0 \quad (1h)$$

Using the following substitutions we can re-write equation:(1h) as equation:(2d)

$$c_1 = T_0 \quad (2a)$$

$$c_2 = \ln A \quad (2b)$$

$$c_3 = E - (T_0)(\ln A) \quad (2c)$$

$$(\ln y)T = (\ln y)c_1 + (T)c_2 + (1)c_3 \quad (2d)$$

2.2 Formulating Linear Least Squares (LLS) Problem

Since we have obtained a linear equation, formulating the LLS problem is easy now. The LLS problem is written in the following way:-

$$\min_c \sum_{i=1}^N (T_i \ln y_i - f(c, y_i, T_i))^2 \quad (3)$$

and we have the following system to be solved:-

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \quad (4a)$$

$$\begin{bmatrix} \ln y_1 & T_1 & 1 \\ \ln y_2 & T_2 & 1 \\ \vdots & \vdots & \vdots \\ \ln y_N & T_N & 1 \end{bmatrix} \cdot \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} (\ln_1)T_1 \\ (\ln_2)T_2 \\ \vdots \\ (\ln_N)T_N \end{bmatrix} \quad (4b)$$

2.3 Data Generation

The data has been generated using the original equation:(1a). The exact parameter's to generate the data are:-

| Parameter | Value |
|-----------|----------------------------|
| A^* | $e^{-2.64} \approx 0.0714$ |
| E^* | $6 * 10^3$ |
| T_0^* | 400 |

2.4 Noise Generation

To generate noise I used the MATLAB function *randn()* which generates normally distributed random variables. I also changed the internal MATLAB controller for random number generator from default i.e. seed: 0 and algorithm: 'twister' to seed: 2 and algorithm: 'philox'. The random noise was added in the following way:-

$$y_\delta(T) = y(T) (1 + \delta\alpha), \alpha \in (-1, 1), \delta \in [0, 1] \quad (5)$$

Where α is a normally distributed random number and δ is the signal-to-noise ratio (SNR) or noise level. While generating α was straightforward (using *randn()*), for δ I generated an evenly space vector such as:- $[0.0 \ 0.1 \ 0.2 \ \dots \ 0.9 \ 1.0]^T$

2.5 Solution Methods

We shortly describe the three solution methods used in solving this assignment.

2.5.1 Method of Normal Equations

We define our problem by saying

$$f(x) = \|Ax - b\|_2^2 = (Ax - b)^T (Ax - b) = x^T A^T A x - 2x^T A^T b + b^T b \quad (6)$$

Then minimization of $f(x)$ gives us the solution:-

$$\nabla f(x) = 0 \implies 2A^T A x - 2A^T b = 0 \implies x = (A^T A)^{-1} A^T b \quad (7)$$

2.5.2 QR Factorization

We define $A = QR$, $Q \in R^{m \times n}$, $R \in R^{n \times n}$ and $Q^T Q = I_n$. Assuming $\text{rank}(A) = n$, then using the normal equations we get:-

$$R^T Q^T Q R x = R^T Q^T b \implies R x = Q^T b \implies x = R^{-1} Q^T b \quad (8)$$

2.5.3 Singular Value Decomposition (SVD)

We define $A = U \Sigma V$, $A \in R^{m \times n}$, $U \in R^{m \times m}$, $\Sigma \in R^{m \times n}$, $V \in R^{n \times n}$.

U and V are orthogonal matrices where as Σ is a diagonal matrix with $\sigma_{11} \geq \sigma_{22} \geq \dots \geq \sigma_{\min(m,n)} \geq 0$

Then if A has full-rank, then solution of $\|Ax - b\|_2$ is given by $x = V \Sigma^{-1} U^T b$.

3 Numerical Examples

3.1 Recovery of exact parameters

Relevant Code: Appendix {A}

We have to recover the exact parameters A^* , E^* and T_0^* . In the simple case we consider that there is no noise in the data and we have full rank (rank = 3) for matrix A . As one can notice, the parameter's

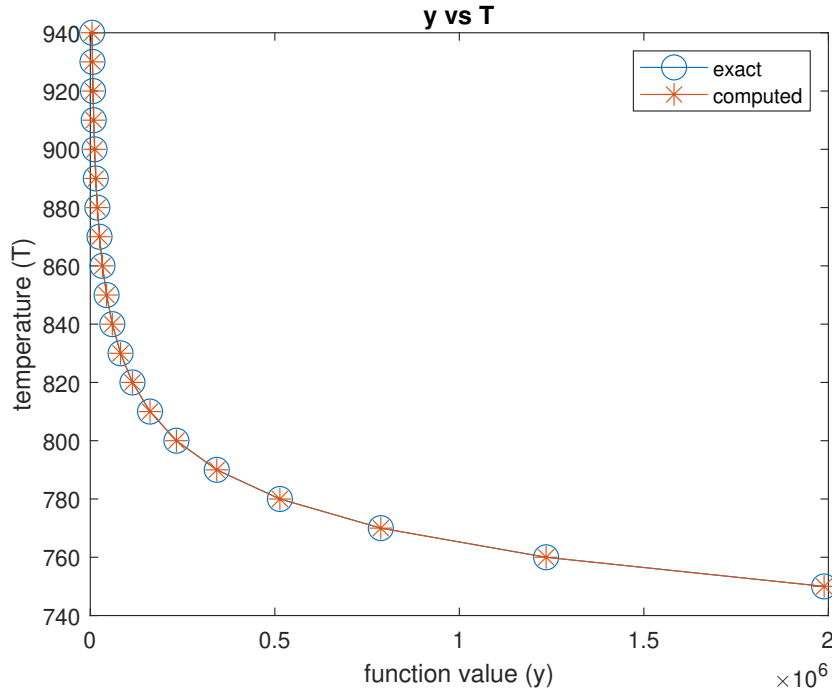


Figure 1: Recovery of Exact Parameters using $x = A^{-1}b$

are recovered without any significant difference between them and the actual values. The relative error's of the recovered parameter's are presented in table below.

| Name | Exact | Computed | Relative Error |
|-------|--------|----------|------------------------------------|
| A | 0.0714 | 0.0714 | $2.3337 \times 10^{-15} \approx 0$ |
| E | 6000 | 6000 | $3.0316 \times 10^{-16} \approx 0$ |
| T_0 | 400 | 400 | $1.4211 \times 10^{-16} \approx 0$ |

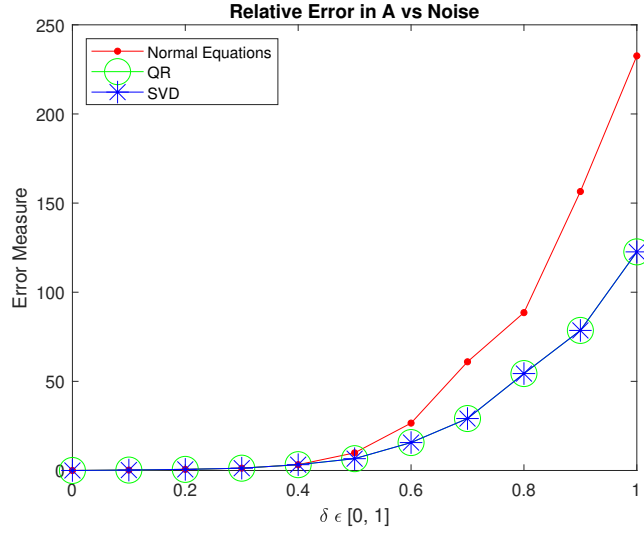
Table 1: Relative Error's for Recovery of Exact Parameters

3.2 Analyzing relative errors for different noise levels γ

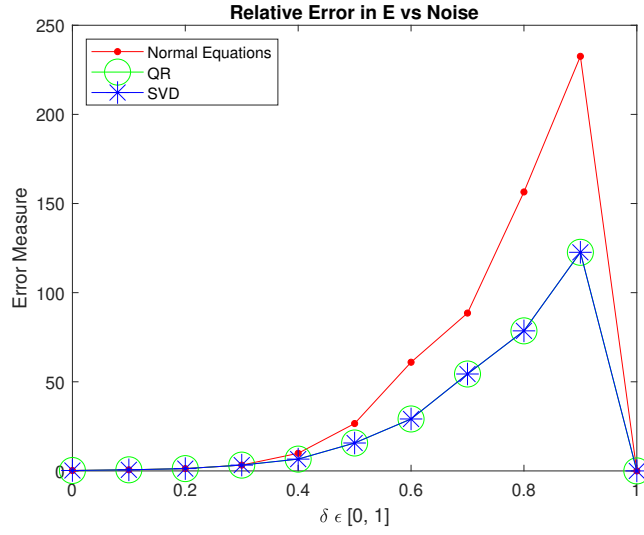
Relevant Code: Appendix {B}

To investigate the effect of different noise levels, we decided to keep N constant at a reasonable value of 125 instead of changing it at every iteration. In this we could interpret on how a different δ would cause variation in the final result. It would also demonstrate the accuracy and precision of each solution method.

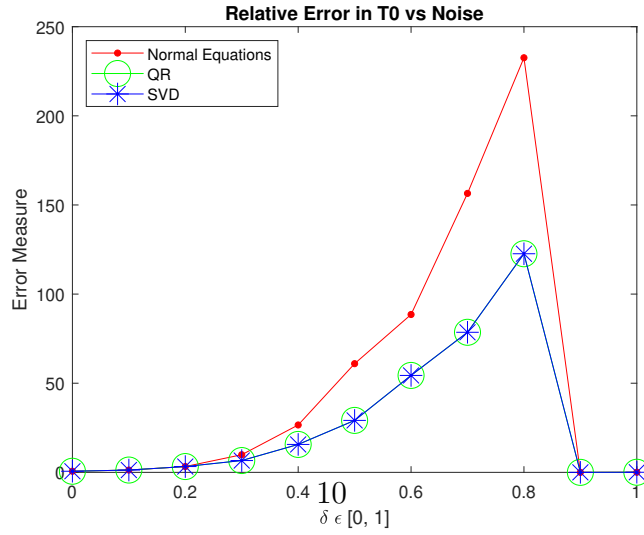
The very first point to note from Figure:2 is that, a visible change in relative error for each parameter is visible only for $\delta \geq 0.3$. Apart from that it is quite evident that, SVD and QR Factorization approximately gives the same result since their relative error's overlap and that their error's are lower than that of method of normal equations.



(a) A vs γ



(b) E vs γ



(c) T_0 vs γ

Figure 2: Relative Errors in Exact Parameters for varying noise levels and $N = 125$

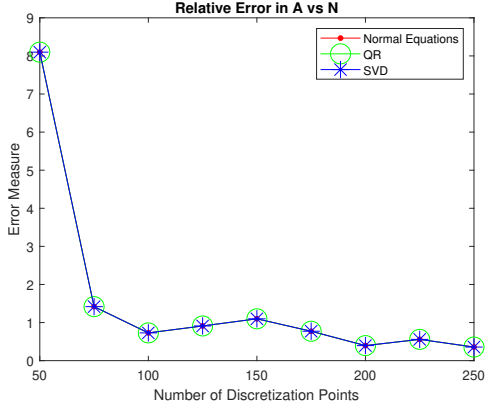
3.3 Analyzing relative errors for different no. of discretization points N

Relevant Code: Appendix {B}

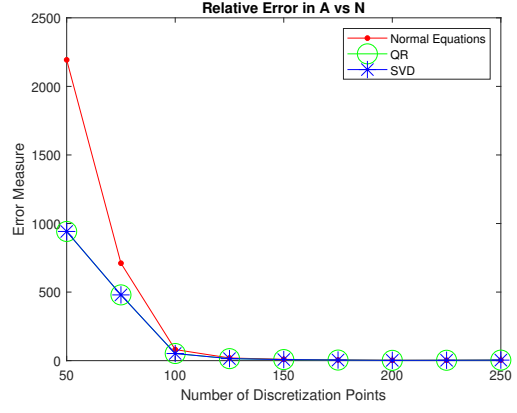
Figure:3 represents the effect of varying N for a constant δ . Using the results of the previous section, we decided to use $\delta = 0.5$ as the constant value. From the graphs it is quite evident that the relative error for all parameters starts reducing significantly only after $N = 100$. It is important to note that the error measure of E is directly dependant on A and T_0 , we shall talk more about this in conclusion.

As expected, SVD and QR Factorization again deliver better results than method of normal equations. In Figure:3 we have plotted N in the range of $[50, 250]$ with an increment of 25 along with this we chose 2 values of $\delta = [0.35, 0.5]$. The motivation to choose these two high values was to show that the given problem was being solved with same accuracy by all 3 solution methods. The superiority of SVD and QR Factorization is visible only when $\delta = 0.5$ indicating one must have around 50% signal to noise ratio to actually see these two methods perform better than method of normal equations. It is quite interesting that one has to introduce such a high level of noise to obtain such results; this is due to the fact that the original matrix has a small condition number. By adding random noise, we make the condition number larger, hence making SVD and QR as more viable methods to solve the problem.

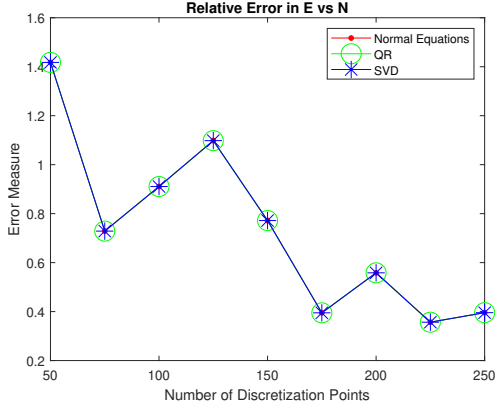
To see this effect one can compare the following figures:-
Figure: (3a vs 3b), Figure: (3c vs 3d), Figure: (3e vs 3f)



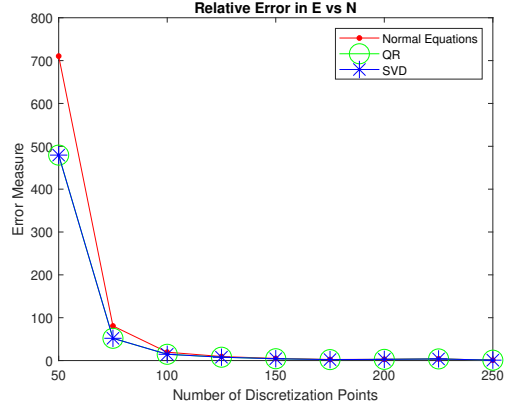
(a) A vs N , $\delta = 0.35$



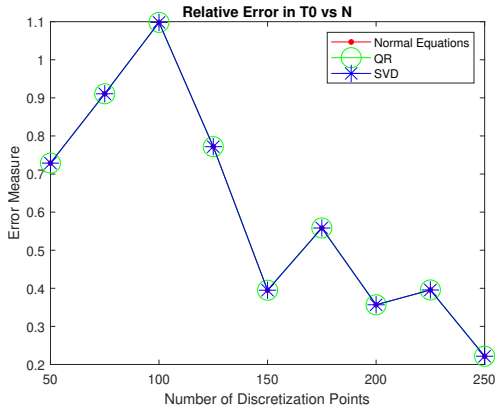
(b) A vs N , $\delta = 0.5$



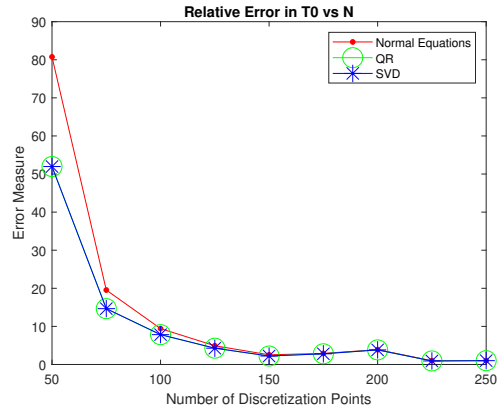
(c) E vs N , $\delta = 0.35$



(d) E vs N , $\delta = 0.5$



(e) T_0 vs N , $\delta = 0.35$



(f) T_0 vs N , $\delta = 0.5$

Figure 3: Relative Errors in Exact Parameters for varying N

3.4 Minimum N Required

Relevant Code: Appendix {C}

Interpretation of the previous two results tells us that the relative error increases considerably in every parameter as the $\delta \geq 0.3$ (for $N = 125$) and it decreases considerably as $N \geq 100$ (for $\delta = 0.5$). So it makes sense for us to test N and δ in the following range:-

$$100 \leq N \leq 225, 0 \leq \delta \leq 0.6$$

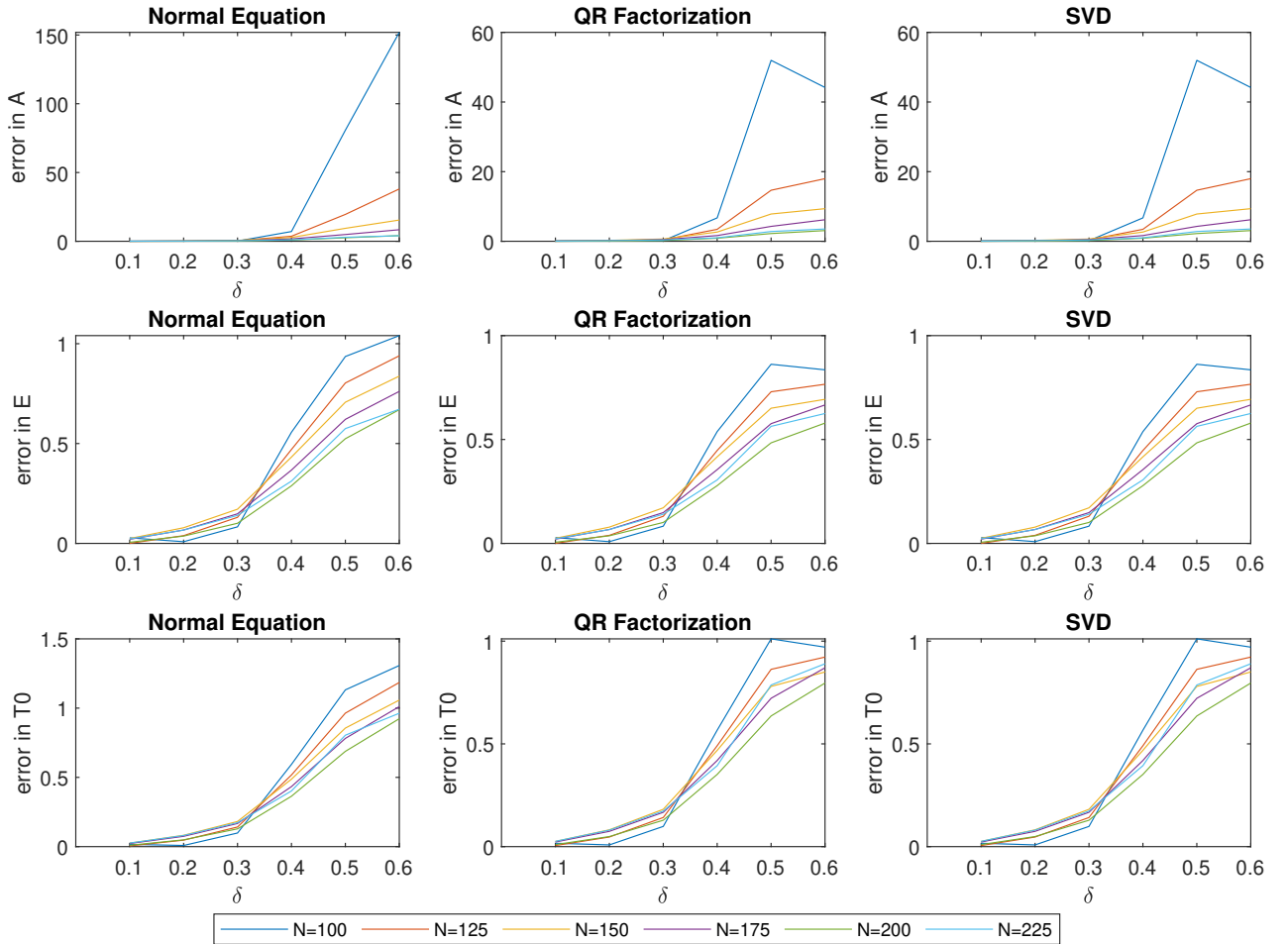


Figure 4: Relative Errors for \mathbf{N} vs δ

From Figure:4 we can see that the relative error for each solution method is under 1.0 for the parameters E and T_0 for $\delta \leq 0.5$. Only for A the relative error reaches values above 10 but that occurs at $\delta \geq 0.35$.

From the colour of the plot lines we can say that lower the value N higher the error in parameters for an increasing δ .

Selecting $N = 200$ and $\delta = [0, 0.1, 0.2, 0.3]$, we obtain the following results:-

| δ | Normal Equations | QR Factorization | SVD |
|----------|------------------|------------------|----------|
| 0.0 | 0 | 0 | 0 |
| 0.1 | 0.006223 | 0.006223 | 0.006223 |
| 0.2 | 0.06813 | 0.06813 | 0.06813 |
| 0.3 | 0.22293 | 0.22293 | 0.22293 |

Table 2: Relative Error in A for $N = 200$

| δ | Normal Equations | QR Factorization | SVD |
|----------|------------------|------------------|-----------|
| 0.0 | 0 | 0 | 0 |
| 0.1 | 0.0056475 | 0.0056475 | 0.0056475 |
| 0.2 | 0.03646 | 0.03646 | 0.03646 |
| 0.3 | 0.10085 | 0.10085 | 0.10085 |

Table 3: Relative Error in E for $N = 200$

| δ | Normal Equations | QR Factorization | SVD |
|----------|------------------|------------------|-----------|
| 0.0 | 0 | 0 | 0 |
| 0.1 | 0.0099242 | 0.0099242 | 0.0099242 |
| 0.2 | 0.04911 | 0.04911 | 0.04911 |
| 0.3 | 0.12844 | 0.12844 | 0.12844 |

Table 4: Relative Error in T_0 for $N = 200$

From Table:2, Table:3 and Table:4 we can see the relative error's for every parameter. Considering the relative error as a measure of accuracy we can say that by choosing $N = 200$ and $\delta = [0, 0.1, 0.2, 0.3]$ would give us a maximum error of 10% when recovering E and T_0 and a maximum error of 22% when recovering A .

4 Conclusion

1. In conclusion we would like to say that SVD, QR Factorization are much more powerful methods than method of normal equations and this can be seen effectively when the data is noisy or matrix A in $A \cdot x = b$ has a high condition number.
2. From equations:(2a)-(2d) one can see that computation of E directly depends on A and T_0 . A acts as a scaling variable for T_0 and T_0 acts as translation variable for E . This evident when one looks at the graphs Figure:2, Figure:(3a,3c,3e) and Figure:(3b,3d,3f).
3. For future work, one could use inbuilt MATLAB functions like *timeit()* to more accurately find how different values of N and δ actual affect the computational time and hence can come up with a reasonably good value of minimum N required to recover exact parameters for a range of δ .

References

- [1] G.S. Fulcher, *Analysis of recent measurements of the viscosity of glasses*, Journal of the American Ceramic Society, volume 8, issue 6, 1925.
- [2] MATLAB and C++/PetSc Codes of Numerical Linear Algebra Theory by Larisa Beilina, Evgenii Karchevskii, and Mikhail

A MATLAB Code 1: "lab10.m"

```
1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %
3 % Computer Lab 1 : Code 1
4 % Linearized Model :  $zT = c1*z + c2*T + c3*1$ 
5 %  $c1 = T_0$ ,  $c2 = \log A$ ,  $c3 = E - T_0 * (\log A)$ 
6 %
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8
9 %% Fresh Start
10 clc;
11 clear all;
12 close all;
13
14 %% Linearized Model Derivation
15 %  $\log y = \log(A) + E/(T - T_0)$ 
16 %  $(\log y) = (\log A) + E/(T - T_0)$ 
17 %  $(\log y) - (\log A) = E/(T - T_0)$ 
18 %  $(\log y)T - (\log A)T - (\log y)T_0 + (\log A)T_0 = E$ 
19 %  $(\log y)T = + (\log y)T_0 + (\log A)T + E - (\log A)T_0$ 
20 %
21 %  $c1 = T_0$ 
22 %  $c2 = \log A$ 
23 %  $c3 = E - T_0 * (\log A)$ 
24 %  $zT = c1*z + c2*T + c3*1$ 
25
26 %% Generate Data
27 E_star = 6 * 1000;
28 A_star = exp(-2.64);
29 T_0_star = 400;
30 N = 20;
31 T = 750:10:(750+(10*(N-1)));
32 y = A_star * exp(E_star./(T-T_0_star));
33
34 %% Simple Linear Least Squares
35 %  $zT = c1*z + c2*T + c3*1$ 
36 %  $c1=T_0$ ,  $c2=\log A$ ,  $c3=E-T_0(\log A)=E-c1*c2$ 
37 z = log(y);
38 b = (z.*T)';
39 A = [z' T' (T.^0)'];
40 x_hat = A\b; %  $A*x = b \Rightarrow x = \text{inv}(A)*b$ 
41
42 T_0_hat = x_hat(1);
43 A_hat = exp(x_hat(2));
44 E_hat = x_hat(3) + (x_hat(1) * x_hat(2));
45 y_hat = A_hat * exp(E_hat./(T-T_0_hat));
46
47 Error_A = relative_error(A_hat,A_star);
```

```

48 Error_E = relative_error(E_hat,E_star);
49 Error_T0 = relative_error(T_0_hat,T_0_star);
50 fprintf('Relative Error Table:\n');
51 table(Error_A,Error_E,Error_T0)
52
53 figure(1)
54 plot(y,T,'Marker','o','MarkerSize',11);
55 hold on;
56 plot(y_hat,T,'Marker','*','MarkerSize',10);
57 xlabel('function value (y)')
58 ylabel('temperature (T)')
59 legend('exact', 'computed')
60 title('y vs T')
61 saveas(gcf, 'lab10_a', 'epsc')
62 saveas(gcf, 'lab10_a', 'png')
63
64 function y = relative_error(x, x_star)
65     y = abs(x-x_star)/abs(x_star);
66 end
67
68 %% Using Optimization Solver
69 %% zT = c1*z + c2*T +c3*1
70 %% c1=T_0, c2=log A, c3=E-T_0(log A)
71 % c = optimvar('c',3);
72 % func = c(1)*z + c(2)*T + c(3);
73 % obj = sum((z.*T - func).^2);
74 % lsqproblem = optimproblem("Objective",obj);
75 %
76 % x0.c(1) = 100;
77 % x0.c(2) = log(1);
78 % x0.c(3) = 1000-(x0.c(1)*x0.c(2));
79 % [sol,fval] = solve(lsqproblem,x0)
80 % sol.c
81 %
82 % T_0_hat_op = sol.c(1);
83 % A_hat_op = exp(sol.c(2));
84 % E_hat_op = sol.c(3) + (sol.c(1) * sol.c(2));
85 % y_hat_op = A_hat * exp(E_hat_op./(T-T_0_hat_op));
86 %
87 % sprintf('Difference A_hat-A_star:\t\t%d\nDifference
    T_0_hat-T_0_star:\t\t%d\nDifference E_hat-E_star:\t\t%d
    ', A_hat_op-A_star, T_0_hat_op-T_0_star, E_hat_op-
    E_star)
88 %
89 % figure
90 % p=plot(y,T,y_hat,T, y_hat_op,T)
91 % p(1).LineWidth = 1;
92 % p(2).LineWidth = 1;
93 % p(3).LineWidth = 1;

```

```

94 % p(1).Marker = 'diamond';
95 % p(1).MarkerIndices = 1:2:length(T);
96 % p(2).Marker = '*';
97 % p(3).Marker = 'o';
98 % p(1).Color = 'r';
99 % p(2).Color = 'g';
100 % p(3).Color = 'b';
101 % legend('original', 'linearized ', 'linearized
      optimization prob')

```

B MATLAB Code 2: "lab11.m"

```

1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2 %
3 % Computer Lab 1 : Code 2
4 % Linearized Model :  $zT = c1*z + c2*T + c3*1$ 
5 %  $c1 = T_0$ ,  $c2 = \log A$ ,  $c3 = E - T_0 * (\log A)$ 
6 %
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8
9 %% Fresh Start
10 clc; clf;
11 clear all;
12 close all;
13
14 %% Parameters
15 E_star = 6 * 1000;
16 A_star = exp(-2.64);
17 T_0_star = 400;
18
19 %% Vary Delta, Constant N
20 delta = 0:0.1:1;
21 len_d = length(delta);
22
23 ea = zeros(len_d,3);
24 ee = zeros(len_d,3);
25 et = zeros(len_d,3);
26
27 N = 125;
28
29 for j = 1:len_d
30     [y, T] = genOriginalData(N, A_star, E_star, T_0_star)
31     ;
32
33     rng(3, 'philox');
34     alpha = randn(1, length(y));
35
36     z = log(y + y.*(delta(j)*alpha));

```

```

36     b = (z.*T)';
37     A = [z' T' (T.^0)'];
38     x = compute(A, b);
39
40     for i=1:3                                % i = 1 : NE
41         x_hat = x(:,i);                      % i = 2 : QR
42         T_0_hat = x_hat(1);                  % i = 3 : SVD
43         A_hat = exp(x_hat(2));
44         E_hat = x_hat(3) + (x_hat(1) * x_hat(2));
45
46         ea(j,i) = relative_error(A_hat, A_star);
47         ee(j,i) = relative_error(E_hat, E_star);
48         et(j,i) = relative_error(T_0_hat, T_0_star);
49     end
50     %table(ea, ee, et)
51 end
52
53 errors = [ea; ee; et];
54 errornames = ["A vs Noise", "E vs Noise", "T0 vs Noise"];
55 xtitle = '\{\delta\} \{\epsilon\} [0, 1]';
56
57 for i=1:3
58     e = errors(i:i+len_d-1, :);
59     plotter(i, delta, e(:,1), e(:,2), e(:,3), xtitle,
60         errornames(i), 'Northwest');
61 end
62 %% Vary N, Constant Delta
63 N = 50:25:250;
64 len_N = length(N);
65
66 ea = zeros(len_N,3);
67 ee = zeros(len_N,3);
68 et = zeros(len_N,3);
69
70 delta = 0.35;
71
72 for j = 1:len_N
73     [y, T] = genOriginalData(N(j), A_star, E_star,
74         T_0_star);
75
76     rng(2, 'philox');
77     alpha = randn(1, length(y));
78
79     z = log(y + y.*(delta*alpha));
80     b = (z.*T)';
81     A = [z' T' (T.^0)'];
82     x = compute(A, b);

```

```

83     for i=1:3                                % i = 1 : NE
84         x_hat = x(:,i);                        % i = 2 : QR
85         T_0_hat = x_hat(1);                    % i = 3 : SVD
86         A_hat = exp(x_hat(2));
87         E_hat = x_hat(3) + (x_hat(1) * x_hat(2));
88
89         ea(j,i) = relative_error(A_hat, A_star);
90         ee(j,i) = relative_error(E_hat, E_star);
91         et(j,i) = relative_error(T_0_hat, T_0_star);
92     end
93     %table(ea, ee, et)
94 end
95
96 errors = [ea; ee; et];
97 errornames = ["A vs N", "E vs N", "T0 vs N"];
98 xtitle = 'Number of Discretization Points';
99
100 for i=1:3
101     e = errors(i:i+len_N-1, :);
102     plotter(i+3, N, e(:,1), e(:,2), e(:,3), xtitle,
103         errornames(i), 'Northeast');
104 end
105 fprintf('All Plots are saved to disk!\n');
106
107 %% Plot Everything
108 function p = plotter(i, x, y_ne, y_qr, y_svd, xtitle,
109     ytitle, loc)
110     h = figure(i);
111     set(h, 'visible', 'off')
112     p = plot(x, y_ne, x, y_qr, x, y_svd);
113
114     p(1).Marker = '.';
115     p(2).Marker = 'o';
116     p(3).Marker = '*';
117
118     p(1).MarkerSize = 12;
119     p(2).MarkerSize = 15;
120     p(3).MarkerSize = 13;
121
122     p(1).Color = 'r';
123     p(2).Color = 'g';
124     p(3).Color = 'b';
125
126     xticks('auto');
127     xticklabels('auto');
128     xlabel(xtitle);
129
130     ylabel('Error Measure');

```

```

130     yticks('auto');
131     yticklabels('auto');
132
133     title(sprintf('Relative Error in %s', ytitle));
134     legend('Normal Equations', 'QR', 'SVD', 'Location',
135           loc);
136     saveas(gcf, 'lab11_'+strrep(ytitle, ' ', ''), 'epsc');
137     saveas(gcf, 'lab11_'+strrep(ytitle, ' ', ''), 'png');
138     clf;
139 end
140
141 %% Compute Function
142 function x = compute(A, b)
143     x_ne = LLS_NE(A,b);
144     x_qr = LLS_QR(A,b);
145     x_svd = LLS_SVD(A,b);
146     x = [x_ne x_qr x_svd];
147 end
148
149 %% Generate Data
150 function [y, T] = genOriginalData(N, A_star, E_star,
151                                   T_0_star)
152     T = 750:10:(750+(10*(N-1)));
153     y = A_star * exp(E_star./(T-T_0_star));
154 end
155
156 %% Solution Method: Normal Equations
157 function x = LLS_NE(A,b)
158     ATb = A'*b;
159     ATA = A'*A;
160     n = length(A(1,:));
161     lowerChol = zeros(n);
162
163     %Cholesky factorization
164     for j = 1:1:n
165         s1 = 0;
166         for k = 1:1:j-1
167             s1 = s1 + lowerChol(j,k)*lowerChol(j,k);
168         end
169         lowerChol(j,j) = (ATA(j,j)-s1)^(1/2);
170         for i = j+1:1:n
171             s2 = 0;
172             for k = 1:1:j-1
173                 s2 = s2 + lowerChol(i,k)*lowerChol(j,k);
174             end
175             lowerChol(i,j) = (ATA(i,j)-s2)/lowerChol(j,j);
176         end
177     end
178
179     ;
180 end
181
182 end

```

```

176
177 % Solver for  $LL^T x = A^T b$ :
178 % Define  $z=L^T x$ , then solve
179 %  $Lz=A^T b$  to find  $z$ .
180 % After by known  $z$  we get  $x$ .
181
182 % forward substitution  $Lz=A^T b$  to obtain  $z$ 
183
184 for i = 1:1:n
185     for k = 1:1:i-1
186         ATb(i) = ATb(i) - ATb(k)*lowerChol(i,k);
187     end
188     ATb(i) = ATb(i)/lowerChol(i,i);
189 end
190
191 % Solution of  $L^T x=z$  , backward substitution
192
193 for i = n:-1:1
194     for k = n:-1:i+1
195         ATb(i) = ATb(i) - ATb(k)*lowerChol(k,i);
196     end
197     ATb(i) = ATb(i)/lowerChol(i,i);
198 end
199
200 % Obtained solution
201 x = ATb;
202 end
203
204 %% Solution Method: QR Factorization
205 function x = LLS_QR(A,b)
206     n = length(A(1,:));
207     q = [];
208     r = [];
209
210     for i = 1:1:n
211         q(:,i) = A(:,i);
212         for j = 1:1:i-1
213             r(j,i) = q(:,j)'*A(:,i);
214             q(:,i) = q(:,i) - r(j,i)*q(:,j);
215         end
216         r(i,i) = norm(q(:,i));
217         q(:,i) = q(:,i)/r(i,i);
218     end
219
220 % compute right hand side in the equation
221 Rx = q'*b;
222
223 % compute solution via backward substitution
224 for i = n:-1:1

```

```

225         for k = n:-1:i+1
226             Rx(i)=Rx(i)-Rx(k)*r(i,k);
227         end
228         Rx(i) = Rx(i)/r(i,i);
229     end
230
231     x = Rx;
232 end
233
234 %% Solution Method: Singular Value Decomposition
235 function x = LLS_SVD(A,b)
236     [U, S, V]=svd(A);
237
238     UTb = U'*b;
239
240     % choose tolerance
241     tol = max(size(A))*eps(S(1,1));
242     s = diag(S);
243     n = length(A(1,:));
244
245     % compute number of singular values > tol
246     r = sum(s > tol);
247
248     w = [(UTb(1:r)./s(1:r))' zeros(1,n-r)]';
249
250     x = V*w;
251 end
252
253 %% Relative Error
254 function y = relative_error(x, x_star)
255     y = abs(x-x_star)/abs(x_star);
256 end

```

C MATLAB Code 3: "lab12.m"

```

1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %
3  % Computer Lab 1 : Code 3
4  % Linearized Model : zT = c1*z + c2*T +c3*1
5  % c1 = T_0, c2 = log A, c3 = E-T_0*(log A)
6  %
7  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8
9  %% Fresh Start
10 clc; clf;
11 clear all;
12 close all;
13

```



```

14 %% Parameters
15 E_star = 6 * 1000;
16 A_star = exp(-2.64);
17 T_0_star = 400;
18
19 %% Vary N, Vary Delta
20 N = 100:25:225;
21 delta = 0.1:0.1:0.6;
22
23 len_N = length(N);
24 len_d = length(delta);
25
26 l = len_N*len_d;
27
28 ea = zeros(1,3);
29 ee = zeros(1,3);
30 et = zeros(1,3);
31
32 l = 1;
33
34 best_N = 200;
35 delta_range = [0 0.1 0.2 0.3];
36
37 for k = 1:len_N
38     for j = 1:len_d
39         [y, T] = genOriginalData(N(k), A_star, E_star,
40             T_0_star);
41
42         rng(2, 'philox');
43         alpha = randn(1, length(y));
44
45         z = log(y + y.*(delta(j)*alpha));
46         b = (z.*T)';
47         A = [z' T' (T.^0)'];
48         x = compute(A, b);
49
50         for i=1:3
51             x_hat = x(:,i);
52             T_0_hat = x_hat(1);
53             A_hat = exp(x_hat(2));
54             E_hat = x_hat(3) + (x_hat(1) * x_hat(2));
55
56             ea(1,i) = relative_error(A_hat, A_star);
57             ee(1,i) = relative_error(E_hat, E_star);
58             et(1,i) = relative_error(T_0_hat, T_0_star);
59         end
60         l = l+1;
61     end
62 end

```

```

62
63 plotter(delta, l-1, len_d, ea, ee, et, N);
64 check(best_N, delta_range, A_star, E_star, T_0_star);
65
66 %% Checker Function
67 function check(N, delta, A_star, E_star, T_0_star)
68     [y, T] = genOriginalData(N, A_star, E_star, T_0_star)
69     ;
70
71     rng(2, 'philox');
72     alpha = randn(1, length(y));
73
74     len_d = length(delta);
75
76     t = zeros(len_d*3,3); k = 1;
77     ea = zeros(len_d,3);
78     ee = zeros(len_d,3);
79     et = zeros(len_d,3);
80
81     figure;
82
83     for j = 1:len_d
84         z = log(y + y.*(delta(j)*alpha));
85         b = (z.*T)';
86         A = [z' T' (T.^0)'];
87         x = compute(A, b);
88
89         for i=1:3 % i = 1 : NE
90             x_hat = x(:,i); % i = 2 : QR
91             T_0_hat = x_hat(1); % i = 3 : SVD
92             A_hat = exp(x_hat(2));
93             E_hat = x_hat(3) + (x_hat(1) * x_hat(2));
94             y_hat = A_hat * exp(E_hat./(T-T_0_hat));
95             t(k:k+2,i) = [A_hat, E_hat, T_0_hat];
96             plot(y_hat,T);
97             hold on;
98             ea(j,i) = relative_error(A_hat, A_star);
99             ee(j,i) = relative_error(E_hat, E_star);
100             et(j,i) = relative_error(T_0_hat, T_0_star);
101         end
102         k=k+3;
103     end
104     fprintf('All Values:-\n'); table(t)
105     fprintf('\n[Rounded 7 Digits] Relative Error in A:-\n'); table(round(ea,7))
106     fprintf('\n[Rounded 7 Digits] Relative Error in E:-\n'); table(round(ee,7))
107     fprintf('\n[Rounded 7 Digits] Relative Error in T0:-\n'); table(round(et,7))

```

```

107     xlabel('function value (y)');
108     ylabel('temperature (T)');
109     legend(cellstr(num2str(delta', 'delta=%.2f')));
110     title(sprintf('y vs T for N=%d',N));
111     hold off;
112     saveas(gcf, 'lab12_check', 'epsc');
113     saveas(gcf, 'lab12_check', 'png');
114 end
115
116 %% Plot Everything
117 function plotter(delta, l, n, ea, ee, et, N)
118     figure('Position', [50 50 900 600]);
119     for j=1:3
120         for i = 1:n:1
121             subplot(3,3,j)
122             plot(delta, ea(i:i+n-1,j));
123             xlabel('\delta'); ylabel('error in A');
124             xticks(delta); xticklabels('auto'); yticks('auto');
125             yticklabels('auto');
126             if j == 1
127                 title("Normal Equation");
128             elseif j == 2
129                 title("QR Factorization");
130             elseif j == 3
131                 title('SVD');
132             end
133             hold on;
134             subplot(3,3,j+3)
135             plot(delta, ee(i:i+n-1,j));
136             xlabel('\delta'); ylabel('error in E');
137             xticks(delta); xticklabels('auto'); yticks('auto');
138             yticklabels('auto');
139             if j == 1
140                 title("Normal Equation");
141             elseif j == 2
142                 title("QR Factorization");
143             elseif j == 3
144                 title('SVD');
145             end
146             hold on;
147             subplot(3,3,j+6)
148             plot(delta, et(i:i+n-1,j));
149             if j == 1
150                 title("Normal Equation");
151             elseif j == 2
152                 title("QR Factorization");
153             elseif j == 3
154                 title('SVD');
155             end

```

```

152         xlabel('\delta'); ylabel('error in T0');
153         xticks(delta); xticklabels('auto'); yticks('auto');
154         yticklabels('auto');
155         hold on;
156         end
157     end
158     legend(cellstr(num2str(N', 'N=%-d')), 'Orientation',
159     'horizontal', 'Location', 'none', 'Position', [0.5
160     0.035 0 0]);
161     hold off;
162     saveas(gcf, 'lab12_compare', 'eps');
163     saveas(gcf, 'lab12_compare', 'png');
164 end
165
166 %% Compute Function
167 function x = compute(A, b)
168     x_ne = LLS_NE(A,b);
169     x_qr = LLS_QR(A,b);
170     x_svd = LLS_SVD(A,b);
171     x = [x_ne x_qr x_svd];
172 end
173
174 %% Generate Data
175 function [y, T] = genOriginalData(N, A_star, E_star,
176     T_0_star)
177     T = 750:10:(750+(10*(N-1)));
178     y = A_star * exp(E_star./(T-T_0_star));
179 end
180
181 %% Solution Method: Normal Equations
182 function x = LLS_NE(A,b)
183     ATb = A'*b;
184     ATA = A'*A;
185     n = length(A(1,:));
186     lowerChol = zeros(n);
187
188     %Cholesky factorization
189     for j = 1:1:n
190         s1 = 0;
191         for k = 1:1:j-1
192             s1 = s1 + lowerChol(j,k)*lowerChol(j,k);
193         end
194         lowerChol(j,j) = (ATA(j,j)-s1)^(1/2);
195         for i = j+1:1:n
196             s2 = 0;
197             for k = 1:1:j-1
198                 s2 = s2 + lowerChol(i,k)*lowerChol(j,k);
199             end
200             lowerChol(i,j) = (ATA(i,j)-s2)/lowerChol(j,j)

```

```

;
196     end
197 end
198
199 % Solver for  $LL^T x = A^T b$ :
200 % Define  $z=L^T x$ , then solve
201 %  $Lz=A^T b$  to find  $z$ .
202 % After by known  $z$  we get  $x$ .
203
204 % forward substitution  $Lz=A^T b$  to obtain  $z$ 
205
206 for i = 1:1:n
207     for k = 1:1:i-1
208         ATb(i) = ATb(i) - ATb(k)*lowerChol(i,k);
209     end
210     ATb(i) = ATb(i)/lowerChol(i,i);
211 end
212
213 % Solution of  $L^T x=z$  , backward substitution
214
215 for i = n:-1:1
216     for k = n:-1:i+1
217         ATb(i) = ATb(i) - ATb(k)*lowerChol(k,i);
218     end
219     ATb(i) = ATb(i)/lowerChol(i,i);
220 end
221
222 % Obtained solution
223 x = ATb;
224 end
225
226 %% Solution Method: QR Factorization
227 function x = LLS_QR(A,b)
228     n = length(A(1,:));
229     q = [];
230     r = [];
231
232     for i = 1:1:n
233         q(:,i) = A(:,i);
234         for j = 1:1:i-1
235             r(j,i) = q(:,j)'*A(:,i);
236             q(:,i) = q(:,i) - r(j,i)*q(:,j);
237         end
238         r(i,i) = norm(q(:,i));
239         q(:,i) = q(:,i)/r(i,i);
240     end
241
242 % compute right hand side in the equation
243 Rx = q'*b;

```

```

244
245 % compute solution via backward substitution
246 for i = n:-1:1
247     for k = n:-1:i+1
248         Rx(i)=Rx(i)-Rx(k)*r(i,k);
249     end
250     Rx(i) = Rx(i)/r(i,i);
251 end
252
253 x = Rx;
254 end
255
256 %% Solution Method: Singular Value Decomposition
257 function x = LLS_SVD(A,b)
258     [U, S, V]=svd(A);
259
260     UTb = U'*b;
261
262     % choose tolerance
263     tol = max(size(A))*eps(S(1,1));
264     s = diag(S);
265     n = length(A(1,:));
266
267     % compute number of singular values > tol
268     r = sum(s > tol);
269
270     w = [(UTb(1:r)./s(1:r))' zeros(1,n-r)]';
271
272     x = V*w;
273 end
274
275 %% Relative Error
276 function y = relative_error(x, x_star)
277     y = abs(x-x_star)/abs(x_star);
278 end

```