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 Technoand University of Gothenburg, SE-42196 Gothenburg, Sweden, e-mail: maitreya@student.chalmers.se

Contents

List of Figures

List of Tables

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

$$y = A * e^{\left(\frac{E}{T-T_0}\right)} \tag{1a}$$
$$\ln y = \ln(A * e^{\left(\frac{E}{T-T_0}\right)}) \tag{1b}$$
$$\ln y = \ln A + \ln(e^{\left(\frac{E}{T-T_0}\right)}) \tag{1c}$$
$$\ln y = \ln A + \frac{E}{T-T_0} \tag{1d}$$
$$\ln\left(\frac{y}{A}\right) = \frac{E}{T-T_0} \tag{1e}$$
$$E = +(\ln y - \ln A) * (T - T_0) \tag{1f}$$
$$E = +(\ln y)T - (\ln A)T - (\ln y)T_0 + (\ln A)T_0 \tag{1g}$$
$$(\ln y)T = +(\ln y)T_0 + (\ln A)T + E - (\ln A)T_0 \tag{1h}$$

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

$$c_1 = T_0 \tag{2a}$$
$$c_2 = \ln A \tag{2b}$$
$$c_3 = E - (T_0)(\ln A) \tag{2c}$$
$$(\ln y)T = (\ln y)c_1 + (T)c_2 + (1)c_3 \tag{2d}$$

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:-

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

and we have the following system to be solved:-

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{4a}$$

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

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:-

$$y_\delta(T) = y(T)\,(1 + \delta\alpha)\,, \alpha\,\epsilon\,(-1, 1)\,, \delta\,\epsilon\,[0, 1] \tag{5}$$

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:- $[0.0 \quad 0.1 \quad 0.2 \quad \dots \quad 0.9 \quad 1.0]^T$

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

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

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

$$\nabla f(x) = 0 \implies 2A^TAx - 2A^Tb = 0 \implies x = (A^TA)^{-1}A^Tb \quad (7)$$

2.5.2 QR Factorization

We define $A = QR$, $Q \epsilon R^{mxn}$, $R \epsilon R^{nxn}$ and $Q^TQ = I_n$ Assuming rank$(A) = n$, then using the normal equations we get:-

$$R^TQ^TQRx = R^TQ^Tb \implies Rx = Q^Ty \implies x = R^{-1}Q^Tb \quad (8)$$

2.5.3 Singular Value Decomposition (SVD)

We define $A = U\Sigma V$, $A \epsilon R^{mxn}$, $U \epsilon R^{mxm}$, $\Sigma \epsilon R^{mxn}$, $V \epsilon R^{nxn}$.

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

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

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

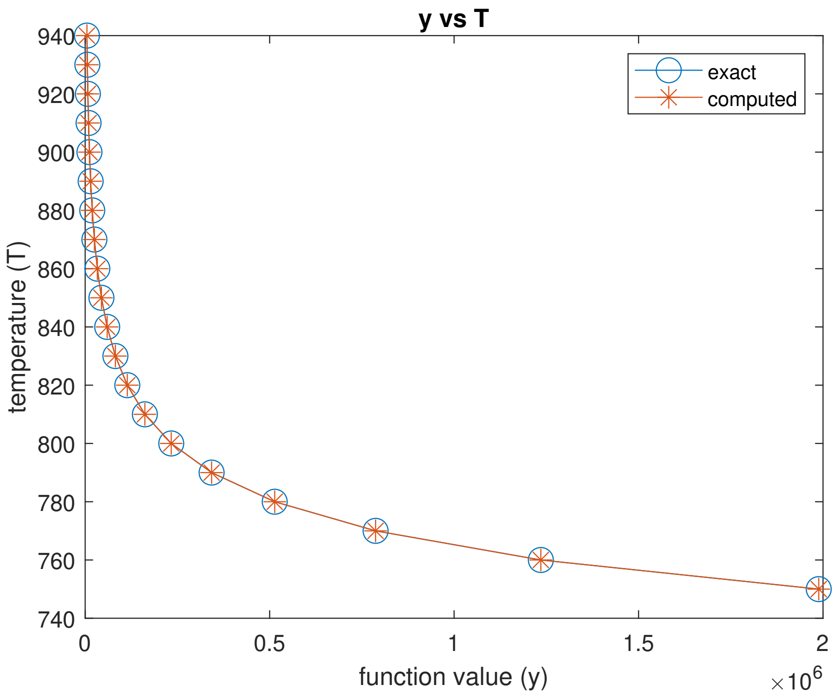


Figure 1: Recovery of Exact Parameters using $x = A^{-1}b$

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	$1.4211 \times 10^{-16} \approx 0$

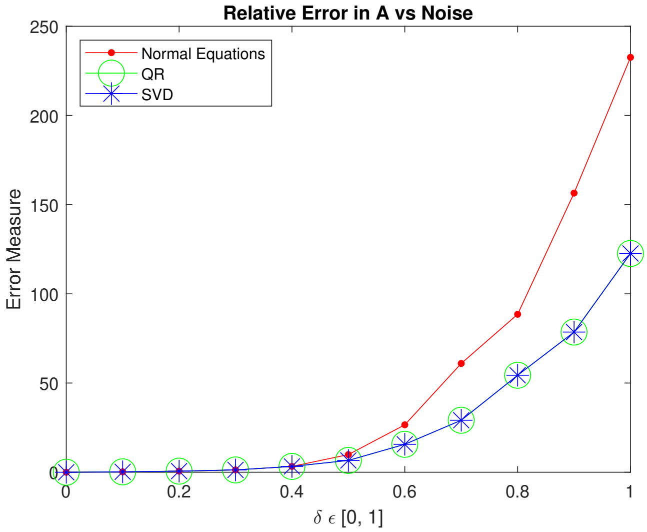
Table 1: Relative Error's for Recovery of Exact Parameters

3.2 Analyzing relative errors for different noise levels γ

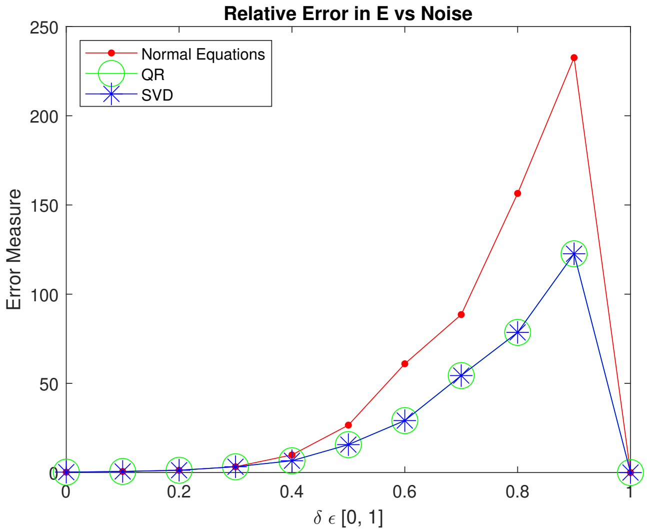
Relevant Code: Appendix {B}

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.

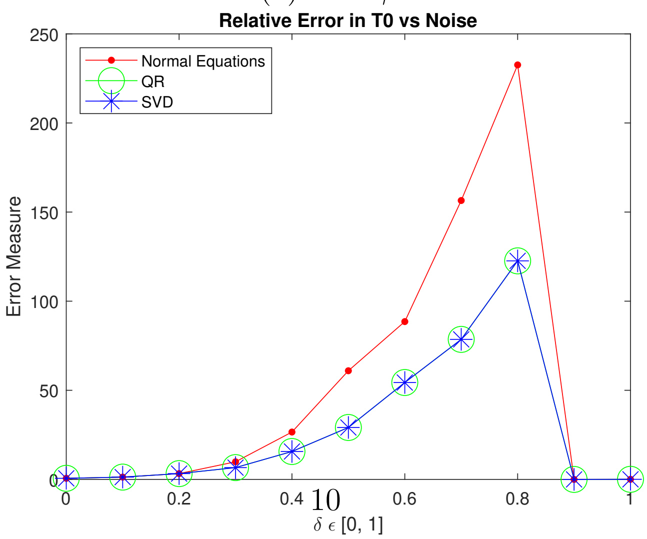
The very first point to note from Figure:2 is that, a visible change in relative error for each parameter is visible only for $\delta \geq 0.3$. Apart from that it is quite evident that, SVD and QR Factorization approximately gives the same result since their relative error's overlap and that their error's 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 $N = 125$

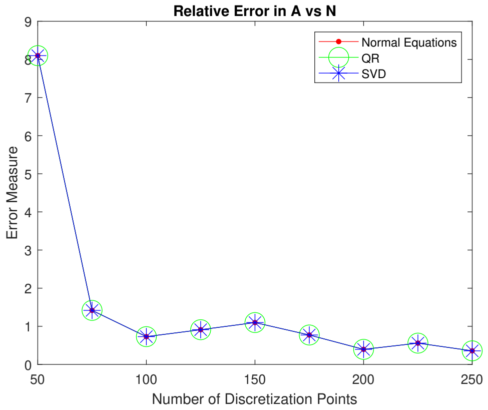
3.3 Analyzing relative errors for different no. of discretization points N

Relevant Code: Appendix {B}

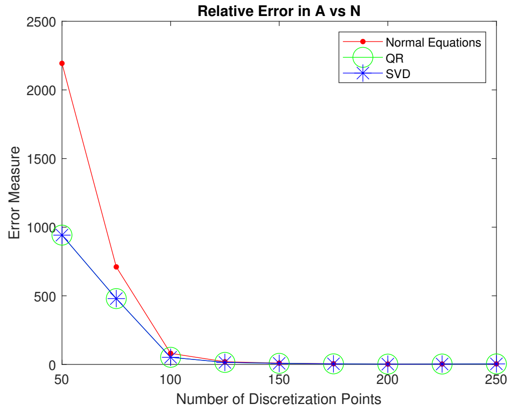
Figure:3 represents the effect of varying N for a constant δ. Using the results of the previous section, we decided to use $\delta = 0.5$ as the constant value. From the graphs it is quite evident that the relative error for all parameters starts reducing significantly only after $N = 100$. It is important 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.

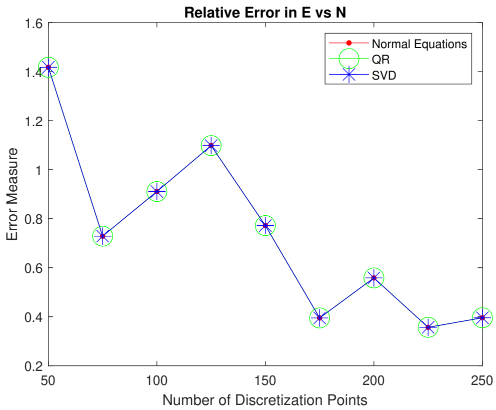
To see this effect one can compare the following figures:-
Figure: (3a vs 3b), Figure: (3c vs 3d), Figure: (3e vs 3f)



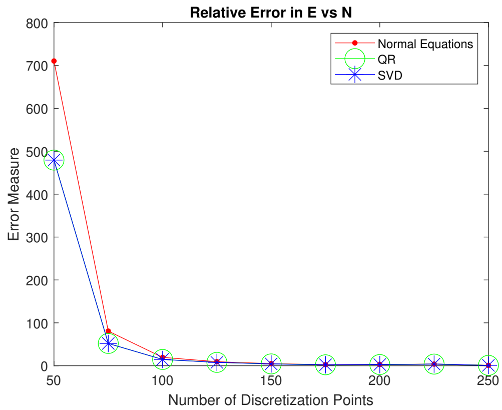
(a) A vs N, $\delta = 0.35$



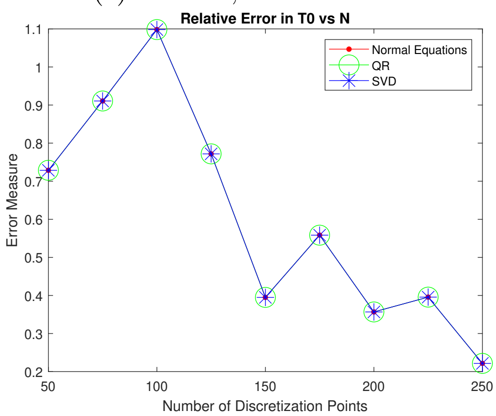
(b) A vs N, $\delta = 0.5$



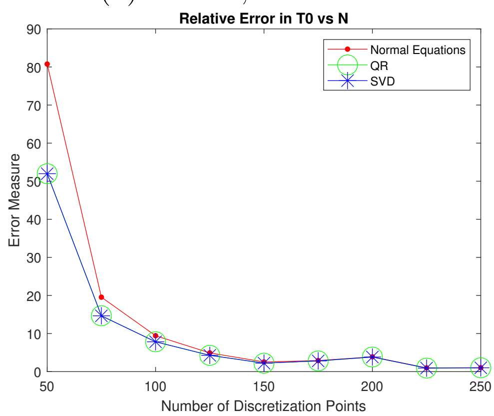
(c) E vs N, $\delta = 0.35$



(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:-

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

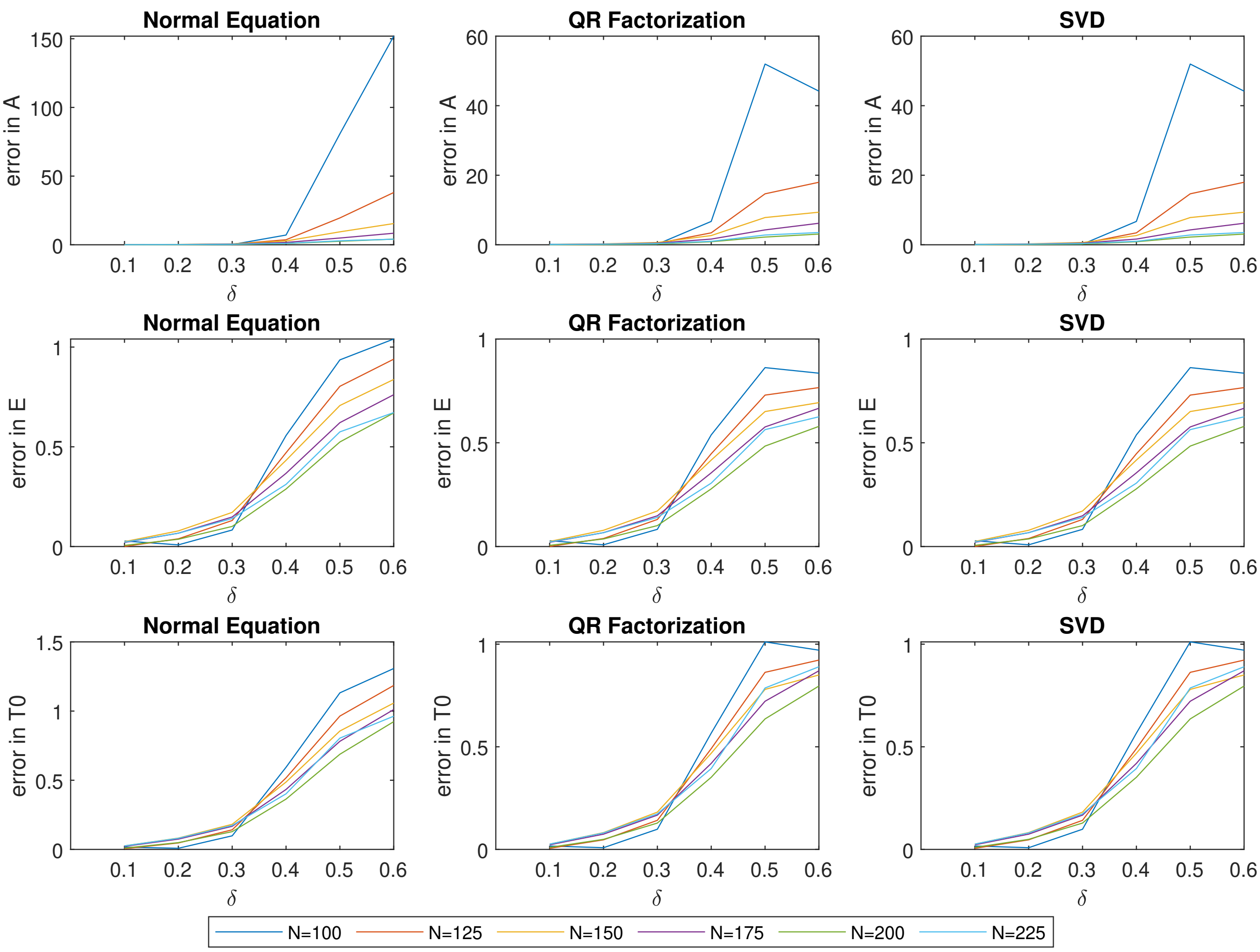


Figure 4: Relative Errors for **N** vs δ

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”

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Computer Lab 1 : Code 1
% Linearized Model : zT = c1*z + c2*T +c3*1
% c1 = T_0, c2 = log A, c3 = E-T_0*(log A)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Fresh Start
clc;
clear all;
close all;

%% Linearized Model Derivation
% log y = log(A) + E/(T - T_0)
% (log y) = (log A) + E/(T-T_0)
% (log y) - (log A) = E/(T-T_0)
% (log y)T - (log A)T - (log y)T_0 + (log A)T_0 = E
% (log y)T = + (log y)T_0 + (log A)T + E - (log A)T_0
%
% c1 = T_0
% c2 = log A
% c3 = E-T_0*(log A)
% zT = c1*z + c2*T +c3*1

%% Generate Data
E_star = 6 * 1000;
A_star = exp(-2.64);
T_0_star = 400;
N = 20;
T = 750:10:(750+(10*(N-1)));
y = A_star * exp(E_star./(T-T_0_star));

%% Simple Linear Least Squares
% zT = c1*z + c2*T +c3*1
% c1=T_0, c2=log A, c3=E-T_0(log A)=E-c1*c2
z = log(y);
b = (z.*T)';
A = [z' T' (T.^0)'];
x_hat = A\b; % A*x = b => x = inv(A)*b

T_0_hat = x_hat(1);
A_hat = exp(x_hat(2));
E_hat = x_hat(3) + (x_hat(1) * x_hat(2));
y_hat = A_hat * exp(E_hat./(T-T_0_hat));

Error_A = relative_error(A_hat,A_star);
```

```
Error_E = relative_error(E_hat,E_star);
Error_T0 = relative_error(T_0_hat,T_0_star);
fprintf('Relative Error Table:\n');
table(Error_A,Error_E,Error_T0)

figure(1)
plot(y,T,'Marker','o','MarkerSize',11);
hold on;
plot(y_hat,T,'Marker','*','MarkerSize',10);
xlabel('function value (y)')
ylabel('temperature (T)')
legend('exact', 'computed')
title('y vs T')
saveas(gcf, 'lab10_a', 'epsc')
saveas(gcf, 'lab10_a', 'png')

function y = relative_error(x, x_star)
    y = abs(x-x_star)/abs(x_star);
end

% %% Using Optimization Solver
% % zT = c1*z + c2*T +c3*1
% % c1=T_0, c2=log A, c3=E-T_0(log A)
% c = optimvar('c',3);
% func = c(1)*z + c(2)*T + c(3);
% obj = sum((z.*T - func).^2);
% lsqproblem = optimproblem("Objective",obj);
%
% x0.c(1) = 100;
% x0.c(2) = log(1);
% x0.c(3) = 1000-(x0.c(1)*x0.c(2));
% [sol,fval] = solve(lsqproblem,x0)
% sol.c
%
% T_0_hat_op = sol.c(1);
% A_hat_op = exp(sol.c(2));
% E_hat_op = sol.c(3) + (sol.c(1) * sol.c(2));
% y_hat_op = A_hat * exp(E_hat_op./(T-T_0_hat_op));
%
% sprintf('Difference A_hat-A_star:\t\t%d\nDifference T_0_hat-T_0_star:\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)
%
% figure
% p=plot(y,T,y_hat,T, y_hat_op,T)
% p(1).LineWidth = 1;
% p(2).LineWidth = 1;
% p(3).LineWidth = 1;
```

```
% p(1).Marker = 'diamond';
% p(1).MarkerIndices = 1:2:length(T);
% p(2).Marker = '*';
% p(3).Marker = 'o';
% p(1).Color = 'r';
% p(2).Color = 'g';
% p(3).Color = 'b';
% legend('original', 'linearized ','linearized
    optimization prob')
```

B MATLAB Code 2: "lab11.m"

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Computer Lab 1 : Code 2
% Linearized Model : zT = c1*z + c2*T +c3*1
% c1 = T_0, c2 = log A, c3 = E-T_0*(log A)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Fresh Start
clc; clf;
clear all;
close all;

%% Parameters
E_star = 6 * 1000;
A_star = exp(-2.64);
T_0_star = 400;

%% Vary Delta, Constant N
delta = 0:0.1:1;
len_d = length(delta);

ea = zeros(len_d,3);
ee = zeros(len_d,3);
et = zeros(len_d,3);

N = 125;

for j = 1:len_d
    [y, T] = genOriginalData(N, A_star, E_star, T_0_star)
    ;

    rng(3,'philox');
    alpha = randn(1, length(y));

    z = log(y + y.*(delta(j)*alpha));
```

```
    b = (z.*T)';
    A = [z' T' (T.^0)'];
    x = compute(A, b);

    for i=1:3                                   % i = 1 : NE
        x_hat = x(:,i);                         % i = 2 : QR
        T_0_hat = x_hat(1);                     % i = 3 : SVD
        A_hat = exp(x_hat(2));
        E_hat = x_hat(3) + (x_hat(1) * x_hat(2));

        ea(j,i) = relative_error(A_hat, A_star);
        ee(j,i) = relative_error(E_hat, E_star);
        et(j,i) = relative_error(T_0_hat, T_0_star);
    end
    %table(ea, ee, et)
end

errors = [ea; ee; et];
errornames = ["A vs Noise", "E vs Noise", "T0 vs Noise"];
xtitle = '{\delta} {\epsilon} [0, 1]';

for i=1:3
    e = errors(i:i+len_d-1, :);
    plotter(i, delta, e(:,1), e(:,2), e(:,3), xtitle,
   errornames(i), 'Northwest');
end

%% Vary N, Constant Delta
N = 50:25:250;
len_N = length(N);

ea = zeros(len_N,3);
ee = zeros(len_N,3);
et = zeros(len_N,3);

delta = 0.35;

for j = 1:len_N
    [y, T] = genOriginalData(N(j), A_star, E_star,
   T_0_star);

    rng(2,'philox');
    alpha = randn(1, length(y));

    z = log(y + y.*(delta*alpha));
    b = (z.*T)';
    A = [z' T' (T.^0)'];
    x = compute(A, b);

```

```
    for i=1:3                                   % i = 1 : NE
        x_hat = x(:,i);                         % i = 2 : QR
        T_0_hat = x_hat(1);                     % i = 3 : SVD
        A_hat = exp(x_hat(2));
        E_hat = x_hat(3) + (x_hat(1) * x_hat(2));

        ea(j,i) = relative_error(A_hat, A_star);
        ee(j,i) = relative_error(E_hat, E_star);
        et(j,i) = relative_error(T_0_hat, T_0_star);
    end
    %table(ea, ee, et)
end

errors = [ea; ee; et];
errornames = ["A vs N", "E vs N", "T0 vs N"];
xtitle = 'Number of Discretization Points';

for i=1:3
    e = errors(i:i+len_N-1, :);
    plotter(i+3, N, e(:,1), e(:,2), e(:,3), xtitle, errornames(i), 'Northeast');
end

fprintf('All Plots are saved to disk!\n');

%% Plot Everything
function p = plotter(i, x, y_ne, y_qr, y_svd, xtitle, ytitle, loc)
    h = figure(i);
    set(h,'visible','off')
    p = plot(x, y_ne, x, y_qr, x, y_svd);

    p(1).Marker = '.';
    p(2).Marker = 'o';
    p(3).Marker = '*';

    p(1).MarkerSize = 12;
    p(2).MarkerSize = 15;
    p(3).MarkerSize = 13;

    p(1).Color = 'r';
    p(2).Color = 'g';
    p(3).Color = 'b';

    xticks('auto');
    xticklabels('auto');
    xlabel(xtitle);

    ylabel('Error Measure');
```

```
    yticks('auto');
    yticklabels('auto');

    title(sprintf('Relative Error in %s', ytitle));
    legend('Normal Equations', 'QR', 'SVD', 'Location',
    loc);
  saveas(gcf, 'lab11_'+strrep(ytitle, ' ', ''), 'epsc');
    saveas(gcf, 'lab11_'+strrep(ytitle, ' ', ''), 'png');
    clf;
end

%% Compute Function
function x = compute(A, b)
    x_ne = LLS_NE(A,b);
    x_qr = LLS_QR(A,b);
    x_svd = LLS_SVD(A,b);
    x = [x_ne x_qr x_svd];
end

%% Generate Data
function [y, T] = genOriginalData(N, A_star, E_star,
    T_0_star)
    T = 750:10:(750+(10*(N-1)));
    y = A_star * exp(E_star./(T-T_0_star));
end

%% Solution Method: Normal Equations
function x = LLS_NE(A,b)
    ATb = A'*b;
    ATA = A'*A;
    n = length(A(1,:));
    lowerChol = zeros(n);

    %Cholesky factorization
    for j = 1:1:n
        s1 = 0;
        for k = 1:1:j-1
            s1 = s1 + lowerChol(j,k)*lowerChol(j,k);
        end
        lowerChol(j,j) = (ATA(j,j)-s1)^(1/2);
        for i = j+1:1:n
            s2 = 0;
            for k = 1:1:j-1
                s2 = s2 + lowerChol(i,k)*lowerChol(j,k);
            end
            lowerChol(i,j) = (ATA(i,j)-s2)/lowerChol(j,j)
    ;
        end
    end
```

```

    % Solver for LL^T x = A^Tb:
    % Define z=L^Tx, then solve
    % Lz=A^T b to find z.
    % After by known z we get x.

    % forward substitution Lz=A^T b to obtain z

    for i = 1:1:n
        for k = 1:1:i-1
            ATb(i) = ATb(i) - ATb(k)*lowerChol(i,k);
        end
        ATb(i) = ATb(i)/lowerChol(i,i);
    end

    % Solution of L^Tx=z , backward substitution

    for i = n:-1:1
        for k = n:-1:i+1
            ATb(i) = ATb(i) - ATb(k)*lowerChol(k,i);
        end
        ATb(i) = ATb(i)/lowerChol(i,i);
    end

    % Obtained solution
    x = ATb;
end

%% Solution Method: QR Factorization
function x = LLS_QR(A,b)
    n = length(A(1,:));
    q = [];
    r = [];

    for i = 1:1:n
        q(:,i) = A(:,i);
        for j = 1:1:i-1
          r(j,i) = q(:,j)'*A(:,i);
          q(:,i) = q(:,i) - r(j,i)*q(:,j);
        end
        r(i,i) = norm(q(:,i));
        q(:,i) = q(:,i)/r(i,i);
    end

    % compute right hand side in the equation
    Rx = q'*b;

    % compute solution via backward substitution
    for i = n:-1:1
```

```
        for k = n:-1:i+1
            Rx(i)=Rx(i)-Rx(k)*r(i,k);
        end
        Rx(i) = Rx(i)/r(i,i);
    end

    x = Rx;
end

%% Solution Method: Singular Value Decomposition
function x = LLS_SVD(A,b)
    [U, S, V]=svd(A);

    UTb = U'*b;

    % choose tolerance
    tol = max(size(A))*eps(S(1,1));
    s = diag(S);
    n = length(A(1,:));

    % compute number of singular values > tol
    r = sum(s > tol);

    w = [(UTb(1:r)./s(1:r))' zeros(1,n-r)]';

    x = V*w;
end

%% Relative Error
function y = relative_error(x, x_star)
    y = abs(x-x_star)/abs(x_star);
end
```

C MATLAB Code 3: "lab12.m"

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Computer Lab 1 : Code 3
% Linearized Model : zT = c1*z + c2*T +c3*1
% c1 = T_0, c2 = log A, c3 = E-T_0*(log A)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Fresh Start
clc; clf;
clear all;
close all;

```

```
%% Parameters
E_star = 6 * 1000;
A_star = exp(-2.64);
T_0_star = 400;

%% Vary N, Vary Delta
N = 100:25:225;
delta = 0.1:0.1:0.6;

len_N = length(N);
len_d = length(delta);

l = len_N*len_d;

ea = zeros(l,3);
ee = zeros(l,3);
et = zeros(l,3);

l = 1;

best_N = 200;
delta_range = [0 0.1 0.2 0.3];

for k = 1:len_N
    for j = 1:len_d
        [y, T] = genOriginalData(N(k), A_star, E_star, T_0_star);

        rng(2,'philox');
        alpha = randn(1, length(y));

        z = log(y + y.*(delta(j)*alpha));
        b = (z.*T)';
        A = [z' T' (T.^0)'];
        x = compute(A, b);

        for i=1:3                              % i = 1 : NE
            x_hat = x(:,i);                    % i = 2 : QR
            T_0_hat = x_hat(1);                % i = 3 : SVD
            A_hat = exp(x_hat(2));
            E_hat = x_hat(3) + (x_hat(1) * x_hat(2));

            ea(l,i) = relative_error(A_hat, A_star);
            ee(l,i) = relative_error(E_hat, E_star);
            et(l,i) = relative_error(T_0_hat, T_0_star);
        end
        l = l+1;
    end
end
```

```
plotter(delta, l-1, len_d, ea, ee, et, N);
check(best_N, delta_range, A_star, E_star, T_0_star);

%% Checker Function
function check(N, delta, A_star, E_star, T_0_star)
    [y, T] = genOriginalData(N, A_star, E_star, T_0_star);

    rng(2,'philox');
    alpha = randn(1, length(y));

    len_d = length(delta);

    t = zeros(len_d*3,3); k = 1;
    ea = zeros(len_d,3);
    ee = zeros(len_d,3);
    et = zeros(len_d,3);

    figure;

    for j = 1:len_d
        z = log(y + y.*(delta(j)*alpha));
        b = (z.*T)';
        A = [z' T' (T.^0)'];
        x = compute(A, b);

        for i=1:3                           % i = 1 : NE
            x_hat = x(:,i);                 % i = 2 : QR
            T_0_hat = x_hat(1);             % i = 3 : SVD
            A_hat = exp(x_hat(2));
            E_hat = x_hat(3) + (x_hat(1) * x_hat(2));
            y_hat = A_hat * exp(E_hat./(T-T_0_hat));
            t(k:k+2,i) = [A_hat, E_hat, T_0_hat];
            plot(y_hat,T);
            hold on;
            ea(j,i) = relative_error(A_hat, A_star);
            ee(j,i) = relative_error(E_hat, E_star);
            et(j,i) = relative_error(T_0_hat, T_0_star);
        end
        k=k+3;
    end
    fprintf('All Values:-\n'); table(t)
    fprintf('\n[Rounded 7 Digits] Relative Error in A:-\n'); table(round(ea,7))
    fprintf('\n[Rounded 7 Digits] Relative Error in E:-\n'); table(round(ee,7))
    fprintf('\n[Rounded 7 Digits] Relative Error in T0:-\n'); table(round(et,7))
```

```
    xlabel('function value (y)');
    ylabel('temperature (T)');
    legend(cellstr(num2str(delta', 'delta=%-.2f')));
    title(sprintf('y vs T for N=%d',N));
    hold off;
    saveas(gcf, 'lab12_check', 'epsc');
    saveas(gcf, 'lab12_check', 'png');
end

%% Plot Everything
function plotter(delta, l, n, ea, ee, et, N)
    figure('Position', [50 50 900 600]);
    for j=1:3
        for i = 1:n:l
            subplot(3,3,j)
            plot(delta, ea(i:i+n-1,j));
            xlabel('{\delta}'); ylabel('error in A');
   xticks(delta); xticklabels('auto'); yticks('auto');
   yticklabels('auto');
            if j == 1
                title("Normal Equation");
            elseif j == 2
                title("QR Factorization");
            elseif j == 3
                title('SVD');
            end
            hold on;
            subplot(3,3,j+3)
            plot(delta, ee(i:i+n-1,j));
            xlabel('{\delta}'); ylabel('error in E');
   xticks(delta); xticklabels('auto'); yticks('auto');
   yticklabels('auto');
            if j == 1
                title("Normal Equation");
            elseif j == 2
                title("QR Factorization");
            elseif j == 3
                title('SVD');
            end
            hold on;
            subplot(3,3,j+6)
            plot(delta, et(i:i+n-1,j));
            if j == 1
                title("Normal Equation");
            elseif j == 2
                title("QR Factorization");
            elseif j == 3
                title('SVD');
            end
```

```
            xlabel('{\delta}'); ylabel('error in T0'); xticks(delta); xticklabels('auto'); yticks('auto'); yticklabels('auto');
            hold on;
        end
    end
    legend(cellstr(num2str(N', 'N=%-d')), 'Orientation', 'horizontal', 'Location', 'none', 'Position', [0.5 0.035 0 0]);
    hold off;
    saveas(gcf, 'lab12_compare', 'epsc');
    saveas(gcf, 'lab12_compare', 'png');
end

%% Compute Function
function x = compute(A, b)
    x_ne = LLS_NE(A,b);
    x_qr = LLS_QR(A,b);
    x_svd = LLS_SVD(A,b);
    x = [x_ne x_qr x_svd];
end

%% Generate Data
function [y, T] = genOriginalData(N, A_star, E_star, T_0_star)
    T = 750:10:(750+(10*(N-1)));
    y = A_star * exp(E_star./(T-T_0_star));
end

%% Solution Method: Normal Equations
function x = LLS_NE(A,b)
    ATb = A'*b;
    ATA = A'*A;
    n = length(A(1,:));
    lowerChol = zeros(n);

    %Cholesky factorization
    for j = 1:1:n
        s1 = 0;
        for k = 1:1:j-1
            s1 = s1 + lowerChol(j,k)*lowerChol(j,k);
        end
        lowerChol(j,j) = (ATA(j,j)-s1)^(1/2);
        for i = j+1:1:n
            s2 = 0;
            for k = 1:1:j-1
                s2 = s2 + lowerChol(i,k)*lowerChol(j,k);
            end
            lowerChol(i,j) = (ATA(i,j)-s2)/lowerChol(j,j)
```

```
    ;
          end
      end

      % Solver for LL^T x = A^Tb:
      % Define z=L^Tx, then solve
      % Lz=A^T b to find z.
      % After by known z we get x.

      % forward substitution Lz=A^T b to obtain z

      for i = 1:1:n
          for k = 1:1:i-1
              ATb(i) = ATb(i) - ATb(k)*lowerChol(i,k);
          end
          ATb(i) = ATb(i)/lowerChol(i,i);
      end

      % Solution of L^Tx=z , backward substitution

      for i = n:-1:1
          for k = n:-1:i+1
              ATb(i) = ATb(i) - ATb(k)*lowerChol(k,i);
          end
          ATb(i) = ATb(i)/lowerChol(i,i);
      end

      % Obtained solution
      x = ATb;
end

%% Solution Method: QR Factorization
function x = LLS_QR(A,b)
      n = length(A(1,:));
      q = [];
      r = [];

      for i = 1:1:n
          q(:,i) = A(:,i);
          for j = 1:1:i-1
            r(j,i) = q(:,j)'*A(:,i);
            q(:,i) = q(:,i) - r(j,i)*q(:,j);
          end
          r(i,i) = norm(q(:,i));
          q(:,i) = q(:,i)/r(i,i);
      end

      % compute right hand side in the equation
      Rx = q'*b;
```

```
    % compute solution via backward substitution
    for i = n:-1:1
        for k = n:-1:i+1
            Rx(i)=Rx(i)-Rx(k)*r(i,k);
        end
        Rx(i) = Rx(i)/r(i,i);
    end

    x = Rx;
end

%% Solution Method: Singular Value Decomposition
function x = LLS_SVD(A,b)
    [U, S, V]=svd(A);

    UTb = U'*b;

    % choose tolerance
    tol = max(size(A))*eps(S(1,1));
    s = diag(S);
    n = length(A(1,:));

    % compute number of singular values > tol
    r = sum(s > tol);

    w = [(UTb(1:r)./s(1:r))' zeros(1,n-r)]';

    x = V*w;
end

%% Relative Error
function y = relative_error(x, x_star)
    y = abs(x-x_star)/abs(x_star);
end
```