Table of Contents

E7 Lab 9 Solutions 1
Question 1 1
Published Test Case 2
Additional Test Case 2
Question 2.1 2
Published Test Case 3
Additional Test Case 3
Question 2.2 3
Published Test Case 4
Additional Test Case 4
Question 2.3 5
Published Test Case 5
Additional Test Case 5
Question 3.1 6
Published Test Case 6
Question 3.2 7
Published Test Case 7
Question 4.1 8
Published Test Case 8
Question 4.2 9
Published Test Case 10
Question 4.3 10
Published Test Case 10
Question 4.4 10
Published Test Case 11

E7 Lab 9 Solutions

Spring 2016

```
format compact
format short
clear all
clc
close all
```

Question 1

```
type mySols
```

```
function x = mySols(A,b)
% solves a linear system of equations Ax = b, which could have 0,1,or
% infinitely many solutions
if rank([A b]) > rank(A)
    x = [];
    disp('There is no solution')
elseif rank([A b]) == rank(A) && rank(A) ~= size(A,2)
    x = pinv(A)*b;
```

```
    disp('There is an infinite number of solutions')
else
    x = A\b;
    disp('There is one solution')
end
end
```

Published Test Case

```
x = mySols([1 2 3;0 3 1;1 14 7],[1;2;3])
x = mySols([1 2 3;0 3 1; -1 14 7],[1;2;3])
x = mySols([1 2 3 4 5 6;2 3 4 5 6 7; 3 4 5 6 7 8],[21;27;33])
```

```
There is no solution
x =
     []
There is one solution
x =
    3.0000
    1.1429
   -1.4286
There is an infinite number of solutions
x =
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
```

Additional Test Case

```
x = mySols([2 4;1 2],[4;2])
x = mySols([2 4;1 2],[4;5])
x = mySols([2 4;1 3],[4;2])
```

```
There is an infinite number of solutions
x =
    0.4000
    0.8000
There is no solution
x =
     []
There is one solution
x =
     2
     0
```

Question 2.1

```
type cubicPolyDiff
```

```
function [cubicDf,cubicD] = cubicPolyDiff(f)
%differentiates an arbitrary 3rd degree polynomial
% f(x) = a0 + a1*x + a2*x^2 + a3*x^3
% f: f = [a0; a1; a2; a3]
% cubicDf: coefficient vector of polynomial f'(x)
% cubicD: differentiation matrix
cubicD = [zeros(3,1) diag(1:3); zeros(1,4)]; %create differentiation
 matrix
cubicDf = cubicD*f; %solve for Df
end
```

Published Test Case

```
f = [1;2;3;4];
[cubicDf,cubicD] = cubicPolyDiff(f)
```

```
cubicDf =
     2
     6
    12
     0
cubicD =
     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
```

Additional Test Case

```
f = [3;4;2;7];
[cubicDf,cubicD] = cubicPolyDiff(f)
```

```
cubicDf =
     4
     4
    21
     0
cubicD =
     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
```

Question 2.2

```
type polyDiff
```

```
function [Df,D] = polyDiff(f)
%differentiates an arbitrary nth degree polynomial
% f(x) = a0 + a1*x + a2*x^2 +...+ an*x^n
% f: f = [a0; a1; a2; ...; an]
% Df: coefficient vector of polynomial f'(x)
```

```
% D: differentiation matrix, such that D*f = Df
n = length(f);  %finds degree of polynomial
D = [zeros(n-1,1) diag(1:n-1); zeros(1,n)]; %creates differentiation
 matrix
Df = D*f;   %calculates the derivative
end
```

Published Test Case

```
f = [0;6;3;0;9;4];
[Df D] = polyDiff(f)
```

```
Df =
     6
     6
     0
    36
    20
     0
D =
     0     1     0     0     0     0
     0     0     2     0     0     0
     0     0     0     3     0     0
     0     0     0     0     4     0
     0     0     0     0     0     5
     0     0     0     0     0     0
```

Additional Test Case

```
f = [2;2;3;8];
[Df D] = polyDiff(f)

f = [2;5;0;0;2;7;3;4];
[Df D] = polyDiff(f)
```

```
Df =
     2
     6
    24
     0
D =
     0     1     0     0
     0     0     2     0
     0     0     0     3
     0     0     0     0
Df =
     5
     0
     0
     8
    35
    18
    28
```

```
     0
D =
     0     1     0     0     0     0     0     0
     0     0     2     0     0     0     0     0
     0     0     0     3     0     0     0     0
     0     0     0     0     4     0     0     0
     0     0     0     0     0     5     0     0
     0     0     0     0     0     0     6     0
     0     0     0     0     0     0     0     7
     0     0     0     0     0     0     0     0
```

Question 2.3

```
type polyDiffm
```

```
function [Dmf,Dm] = polyDiffm(f,m)
% finds the mth derivative of an arbitrary nth degree polynomial
% f(x) = a0 + a1*x + a2*x^2 +...+ an*x^n
% f: f = [a0; a1; a2; ...; an]
% Dmf: coefficient vector of polynomial mth derivative of f(x)
% Dm: differentiation matrix, such that Dm*f = Dmf

[~,D] = polyDiff(f); %get differentiation matrix for f

Dm = D^m; %create differentiation matrix that differentiate m times
Dmf = Dm*f; %solve for Dmf
end
```

Published Test Case

```
f = [0;6;3;0;9;4];
[Dfm Dm] = polyDiffm(f,2)
```

```
Dfm =
     6
     0
   108
    80
     0
     0
Dm =
     0     0     2     0     0     0
     0     0     0     6     0     0
     0     0     0     0    12     0
     0     0     0     0     0    20
     0     0     0     0     0     0
     0     0     0     0     0     0
```

Additional Test Case

```
f = [0;6;3;0;9;4];
[Dfm Dm] = polyDiffm(f,4)
```

```
f = [4;5;1;2;0];
[Dfm Dm] = polyDiffm(f,3)

Dfm =
   216
   480
     0
     0
     0
     0
Dm =
     0     0     0     0    24     0
     0     0     0     0     0   120
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
Dfm =
    12
     0
     0
     0
     0
Dm =
     0     0     0     6     0
     0     0     0     0    24
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Question 3.1

```
type myLinearRegression


function [a, RMSE] = myLinearRegression(x,y)
% given a set of points (x,y) and returns the estimation of the
 coefficient
% a and Root Mean Square Error (RMSE).
% a: coefficient of linear model y = a*x
% RMSE: RMSE = (1/m)*sum((yhat-y).^2)

a = x\y; %finds coefficient with backslash operator
m = length(y);

yhat = a*x; %estimation results
RMSE = sqrt((1/m)*sum((yhat-y).^2));
end
```

Published Test Case

```
load Ohm.mat
```

```
[G,RMSE] = myLinearRegression(V,I)

G =
    0.0627
RMSE =
    0.0261
```

Question 3.2

```
type myBestRegression


function [a,b,c,modelNum] = myBestRegression(x,y)
% estimates the coefficients a,b,c and computs the RMSE for three
 models
% derived from the general model y = a*f1(x) + b*f2(x) + c
%    1. f1(x) = sin(x), f2(x) = cos(x)
%    2. f1(x) = x^2, f2(x) = x
%    3. f1(x) = exp(x), f2(x) = 0
% a,b,c: coefficients for the best regression model
% type: 1,2,3 corresponding to the best model from list above

%type 1
yhat = [sin(x) cos(x) ones(numel(x),1)];
coeff(:,1) = yhat\y;
RMSE(1) = sqrt(sum((yhat*coeff(:,1)-y).^2)/numel(x));

%type 2
yhat = [x.^2 x ones(numel(x),1)];
coeff(:,2) = yhat\y;
RMSE(2) = sqrt(sum((yhat*coeff(:,2)-y).^2)/numel(x));

%type 3
yhat = [exp(x) ones(numel(x),1)];
coeff(1:2:3,3) = yhat\y;
coeff(2,3) = 0;
RMSE(3) = sqrt(sum((yhat*coeff(1:2:3,3)-y).^2)/numel(x));

%best model is determined by the one with the lowest RMSE
[~,modelNum] = min(RMSE); %find the index of the minimum
a = coeff(1,modelNum);
b = coeff(2,modelNum);
c = coeff(3,modelNum);


end
```

Published Test Case

```
load Projectile.mat
[a,b,c,modelNum] = myBestRegression(t_proj,y)
```

```
load Pendulum.mat
[a,b,c,modelNum] = myBestRegression(t_pend,theta)

a =
   -4.8961
b =
   50.0073
c =
    9.8852
modelNum =
     2
a =
    0.0051
b =
    2.9946
c =
    0.0263
modelNum =
     1
```

Question 4.1

```
type getDerivative


function [dQdt,avgQ] = getDerivative(discharge)
% returns a vector of values of the derivative, or time rate of
 change, of
% cischarge, and the discharge values corresponding to each computed
 value
% of the derivative. As an approximation for the derivative, the
 function
% computes the following: dQ(t)/dt = Q(t+1) - Q(t)
dQdt = discharge(2:end) - discharge(1:end-1);
avgQ = (discharge(2:end) + discharge(1:end-1))/2;
end
```

Published Test Case

```
load discharge.mat
[dQdt avgQ] = getDerivative(discharge)

dQdt =
  -421
  -346
  -210
  -121
   -65
   -40
   -29
   -26
   -15
```

```
  -10
   -6
   -6
   -5
   -5
   -5
   -2
   -2
   -3
   -1
   -2
avgQ =
   1.0e+03 *
    1.2095
    0.8260
    0.5480
    0.3825
    0.2895
    0.2370
    0.2025
    0.1750
    0.1545
    0.1420
    0.1340
    0.1280
    0.1225
    0.1175
    0.1125
    0.1090
    0.1070
    0.1045
    0.1025
    0.1010
```

Question 4.2

```
type getRecessionParams
```

```
function [a,b] = getRecessionParams(discharge)
% Solves for the parameters, a and b, of the recession model
% log(a) + b*log(avgQ) = log(-dQdt)

[dQdt,avgQ] = getDerivative(discharge);

%sets up the over-determined system of equations
y = log(-dQdt);
A = [ones(numel(discharge)-1,1) log(avgQ)];
x = A\y;

a = exp(x(1)); %a = exp(log(a))
b = x(2);
end
```

Published Test Case

```
[a,b] = getRecessionParams(discharge)
```

```
a =
   5.6376e-05
b =
    2.3699
```

Question 4.3

```
type rsquaredRecessionModel
```

```
function rsq = rsquaredRecessionModel(discharge)
% computes the R^2 goodness of fit of the streamflow recession model
 for
% the linear fit in log-log space.

[a,b] = getRecessionParams(discharge);
[dQdt,avgQ] = getDerivative(discharge);

yhat = log(a) + b*log(avgQ); %calculate model values for log(-dQdt)
y_mean = mean(log(-dQdt)); %calculate mean of y

SSE_lin = sum((yhat-log(-dQdt)).^2); %SSE linear
SSE_mean = sum((log(-dQdt) - y_mean).^2); %SSE mean

rsq = 1-(SSE_lin/SSE_mean);

end
```

Published Test Case

```
[rsq] = rsquaredRecessionModel(discharge)
```

```
rsq =
    0.9100
```

Question 4.4

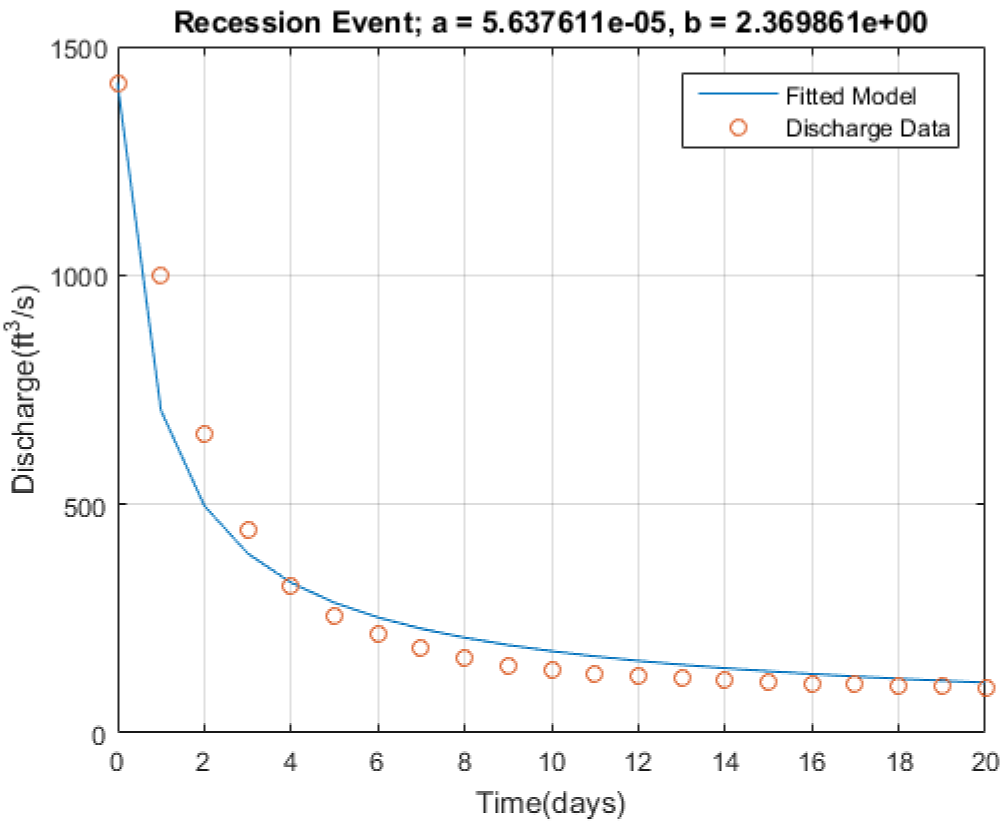
```
type plotRecessionModel
```

```
function [] = plotRecessionModel(a,b,discharge)
% plots the recession model on top of the discharge data

N = numel(discharge);
t = 0:N-1; %time
Q = ((b-1)*((discharge(1)^(1-b))/(b-1) + a.*t)).^(1/(1-b));

plot(t,Q,t,discharge,'o')
```

```
title(sprintf('Recession Event; a = %d, b = %d',a,b))
legend('Fitted Model','Discharge Data')
xlabel('Time(days)')
ylabel('Discharge(ft^3/s)')
grid on

end
```

Published Test Case

```
plotRecessionModel(a,b,discharge)
```



Published with MATLAB® R2015b