**ELECENG 3TP3 Signals and Systems**

Lab 1 Report

Section C01, Instructor: Prof. Jun Chen

Khaled Hassan

Hassak9

400203796

Section 2: Procedure

1. Discrete Time Signal Plotting
   1. x[n] = u[n] − 2u[n − 1] + u[n − 4]

|  |  |
| --- | --- |
| Graph: | Code:  %% Question 1  %% a)  n = -5:10;  x = unitstep(n) - 2.\*unitstep(n-1)+unitstep(n-4);    %%Graphing the function in a stem plot:  stem(n, x);  axis([-5 10 -5 5]);  ylabel("x[n]");  xlabel("n")  title("Khaled Hassan, 400203796") |

* 1. x[n] = (n + 2)u[n + 2] − 2u[n] − nu[n − 4]

|  |  |
| --- | --- |
| Graph: | Code:  %% b)  n = -5:10;  x = (n + 2).\*unitstep(n+2) - 2.\*unitstep(n)-n.\*unitstep(n-4);    %%Graphing the function in a stem plot:  stem(n, x);  axis([-5 10 -5 5]);  ylabel("x[n]");  xlabel("n")  title("Khaled Hassan, 400203796") |

* 1. x[n] = δ[n + 1] − δ[n] + u[n + 1] − u[n − 2]

|  |  |
| --- | --- |
| Graph: | Code:  %% c)  n = -5:10;  x = DT\_dirac(n+1)-DT\_dirac(n)+unitstep(n+1)-unitstep(n-2);  %% Graphing the function as a stem plot  stem(n, x);  axis([-5 10 -5 5]);  ylabel("x[n]");  xlabel("n")  title("Khaled Hassan, 400203796") |

* 1. x[n] = u[n + 1] + u[n]

|  |  |
| --- | --- |
| Graph: | Code:  %% d)  n = -5:10  x = exp(0.8 .\* n) .\* unitstep(n + 1) + unitstep(n);  %% Graphing the function as a stem plot  stem(n, x);  axis([-5 10 -10 100]);  ylabel("x[n]");  xlabel("n")  title("Khaled Hassan, 400203796") |

Question 1 Discussion:

The procedure for graphing these DT signals was fairly repetitive. A range of n values from -5 to 10 was used to allow for observation of function behavior across different ranges. To facilitate graphing the delta function, a new function DT\_dirac was defined, which modifies the built-in MATLAB version of this function, which outputs an infinite value representing the CT delta function. To modify this, the function iterates through the CT dirac to find the index at which the function is infinite and changes that value to 1. Other than that, the graphing function stem() was used, along with its attributes xlabel(), ylabel(), axis() and title().

1. Student Grades

Code:

%% Question 2

%% array to hold max possible grades in each grade component

grades\_out\_of = csvread("course\_grades.csv", 0, 1, [0, 1, 0, 11]);

%% Cell arrays to hold columns that hold each grade component

lab\_grade\_col = {2, 5};

midterm\_grade\_col = {6, 6};

exam\_questions\_col = {7, 12};

%% array to hold the student number column

student\_nums = csvread("course\_grades.csv", 1, 0, [1, 0, 20, 0]);

student\_grade\_matrix = {

csvread("course\_grades.csv", 1, 1, [1, 1, 1, 11]);

csvread("course\_grades.csv", 2, 1, [2, 1, 2, 11]);

csvread("course\_grades.csv", 3, 1, [3, 1, 3, 11]);

csvread("course\_grades.csv", 4, 1, [4, 1, 4, 11]);

csvread("course\_grades.csv", 5, 1, [5, 1, 5, 11]);

csvread("course\_grades.csv", 6, 1, [6, 1, 6, 11]);

csvread("course\_grades.csv", 7, 1, [7, 1, 7, 11]);

csvread("course\_grades.csv", 8, 1, [8, 1, 8, 11]);

csvread("course\_grades.csv", 9, 1, [9, 1, 9, 11]);

csvread("course\_grades.csv", 10, 1, [10, 1, 10, 11]);

csvread("course\_grades.csv", 11, 1, [11, 1, 11, 11]);

csvread("course\_grades.csv", 12, 1, [12, 1, 12, 11]);

csvread("course\_grades.csv", 13, 1, [13, 1, 13, 11]);

csvread("course\_grades.csv", 14, 1, [14, 1, 14, 11]);

csvread("course\_grades.csv", 15, 1, [15, 1, 15, 11]);

csvread("course\_grades.csv", 16, 1, [16, 1, 16, 11]);

csvread("course\_grades.csv", 17, 1, [17, 1, 17, 11]);

csvread("course\_grades.csv", 18, 1, [18, 1, 18, 11]);

csvread("course\_grades.csv", 19, 1, [19, 1, 19, 11]);

csvread("course\_grades.csv", 20, 1, [20, 1, 20, 11]);

}; %% matrix to hold the grade data from the csv file

%% arrays to hold averages of students for the labs, midterm and exams. The student\_average %% function is a the user defined function, to be explained in detail in the discussion/comments %% in the function file

lab\_grades = student\_average(student\_grade\_matrix, grades\_out\_of, lab\_grade\_col);

midterm\_grades = student\_average(student\_grade\_matrix, grades\_out\_of, midterm\_grade\_col);

exam\_grades = student\_average(student\_grade\_matrix, grades\_out\_of, exam\_questions\_col);

%% double variables to hold overall class average in each component

lab\_class\_average = 0;

midterm\_class\_average = 0;

exam\_class\_average = 0;

%% this for loop finds the lab class average by first summing all the student marks, then dividing %% by the number of students to get the class average.

for i = 1: length(lab\_grades)

lab\_class\_average = lab\_class\_average + lab\_grades(i);

end

lab\_class\_average = lab\_class\_average / length(lab\_grades); %% or length(student\_numbers);

%% this for loop does the same as the previous for loop, except for the exam instead of the labs

for i = 1:length(exam\_grades)

exam\_class\_average = exam\_class\_average + exam\_grades(i);

end

exam\_class\_average = exam\_class\_average / length(exam\_grades); %% or length(studentNumbers);

%% initialize the row array (1 row, multiple columns) to hold each student’s final course grade. %% Initially, the values in this array are zeros, to be replaced in the following for loop

final\_course\_grades = zeros(1,length(student\_grade\_matrix), 'double');

%% this for loop calculates each student’s final course grade using the weights outlined in the %% instructions of section 3(c)

for i=1:length(student\_grade\_matrix)

final\_course\_grades(i) = (lab\_grades(i) \* 0.4) + (midterm\_grades(i) \* 0.3) + (exam\_grades(i) \* 0.3);

end

%% This script outputs the average lab marks for each student, the class lab average, each %% student’s exam mark and the class exam average.

fprintf(" Individual Student Lab Averages: \n");

for i=1:length(student\_grade\_matrix)

fprintf(" Student ID: %d \n", student\_nums(i));

fprintf(" Student Mark: %f \n " , lab\_grades(i));

end

fprintf("\n Overall Class Lab Average: ");

fprintf(" %f " , lab\_class\_average);

fprintf( "\n\n Individual Student Exam Grades: \n");

for i=1:length(student\_grade\_matrix)

fprintf(" Student ID: %d \n", student\_nums(i));

fprintf(" Student Mark: %f \n " , exam\_grades(i));

end

fprintf( "\n Overall Class Exam Average: ");

fprintf( " %f " , exam\_class\_average);

%% To generate the stem plot

num\_students = zeros(1,length(student\_grade\_matrix),'double');

for i=1:length(student\_grade\_matrix)

%% num\_students(i) = i; %% Graph 1 is in descending order

num\_students(i) = student\_nums(i); %% Graph 2 is unordered

end

final\_grades = final\_course\_grades;

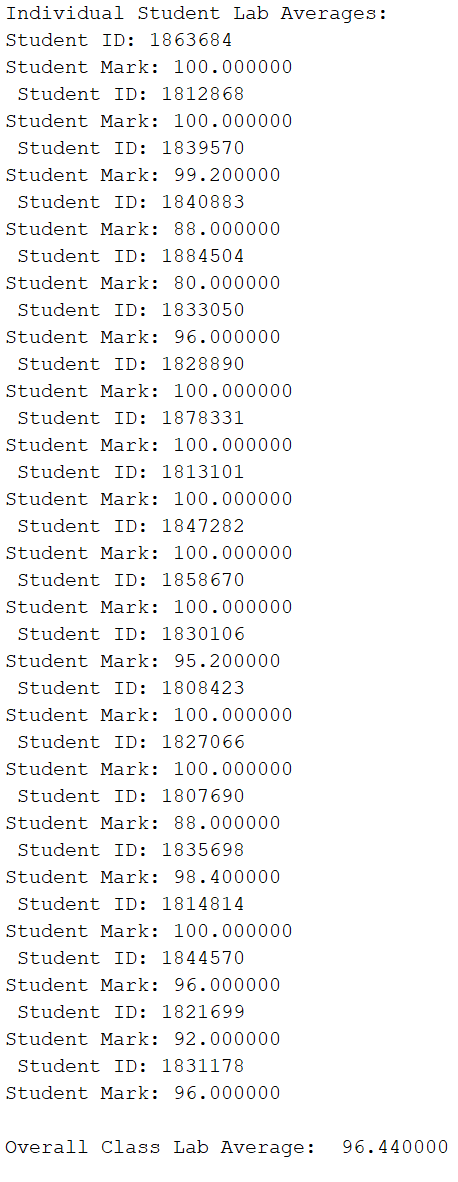
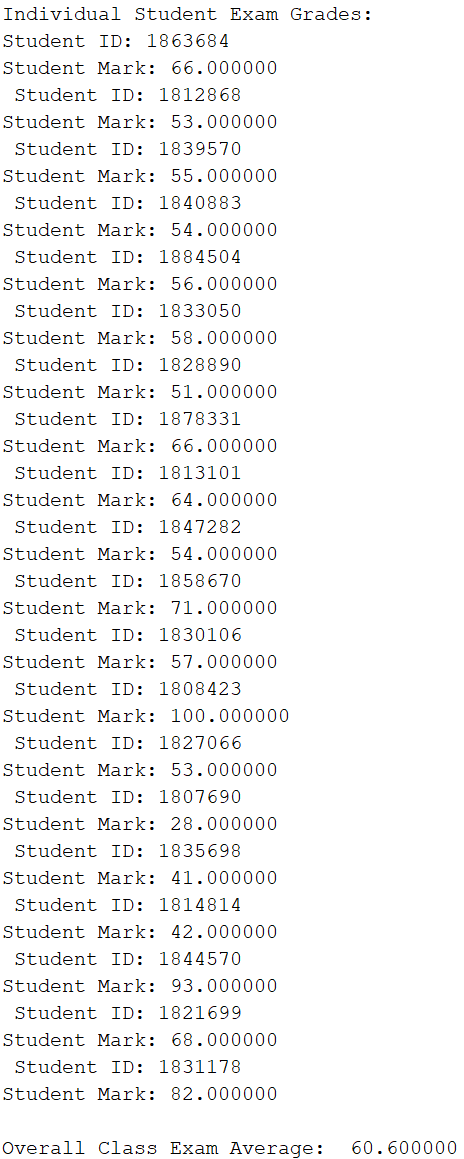
final\_grades = sort(final\_grades, "descend");

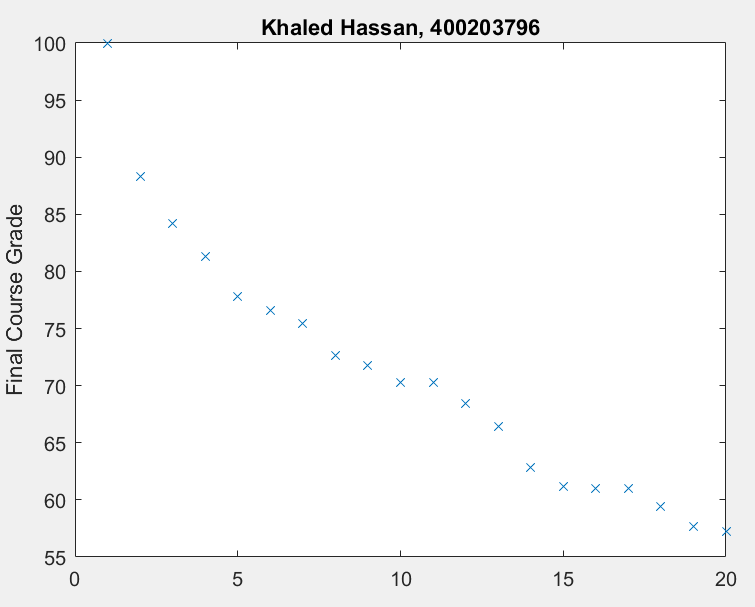
plot(num\_students, final\_grades, "x");

%% Not sure of this one xlabel("Number of Students");

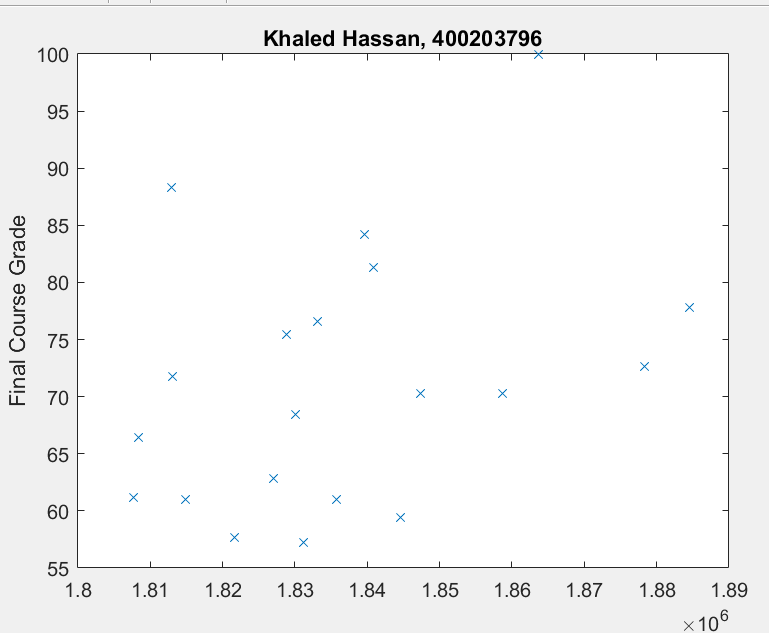
ylabel("Final Course Grade");

title("Khaled Hassan, 400203796");



Graph 1: (descending order of final grade)

Graph 2: (unordered, by student numbers)



Question 2 Discussion:

The first few lines of this code are used to define variables and arrays to hold various relevant groups of data. The user-defined function student\_average() is defined to get the individual averages of each student in an array format, indexed according to their student numbers. The function takes 3 inputs: the overall grades matrix, the maximum possible grade array and the relevant columns of the matrix to the labs/midterm/exam. Then, using nested for loops, the sum of marks achieved and maximum possible grades are separately calculated, with the former being stored in the array to be output. Once outside the inner for loop, the function divides the current sum at each index by the calculated sum of maximum possible grades to output each student’s mark as a percentage in the relevant course component.

1. Image Processing
2. 

|  |  |
| --- | --- |
| Graph 1:    Graph 2: | Code:  %% Question 3  %% a)  image = imread("ee3tp3picture2020.png");  image\_of\_doubles = double(image);  [n\_elements, centers] = hist(image\_of\_doubles(:), 20);  bar(centers, n\_elements);  xlim([0 255]); %% Graph 1  xlim([150 200]) %% Graph 2 is a zoomed in version of Graph 1 |

The problem with the image quality is that the pixel values are too close to each other to sufficiently distinguish one group of pixels from another. The result is a low-contrast image. As the histogram(s) in part a demonstrate, the overwhelming majority of pixels fell in the 160 – 180 range.

|  |  |
| --- | --- |
| Improved Image: | %% c)  Img\_size = size(image\_of\_doubles);  num\_Pixels = Img\_size(1) \* Img\_size(2) \* Img\_size(3);  alpha = 9.45;  beta = -1493;  new\_image = image\_of\_doubles;  for i = 1:num\_Pixels %% for loop to  new\_image(i) = alpha \* new\_image(i) + beta;  end  imshow(uint8(new\_image));  title(“Khaled Hassan, 400203796”); |

By approximating the lower limit and upper limit of the pixels in the original histogram, and by solving the system of linear equations: 158 (alpha) – (beta) = 0 and 185 (alpha) – (beta) = 255, we get alpha approximately equal to 9.45 and beta approximately equal to -1493.

|  |  |
| --- | --- |
| Final histogram: | Code:  [n\_elements, centers] = hist(new\_image(:), 20);  bar(centers, n\_elements);  xlim([0 255]); |

Question 3 Discussion:

Since the pixels in the original image were all within a very narrow range, the image had a low contrast and it appeared to be a very similar shade of grey. However, linear mapping of the pixels was done over a wider range to increase the variation in pixel values and therefore, widen the contrast in pixels in the image. After solving the system of linear equations mentioned in part (b) through approximation, an alpha and beta were found to sufficiently stretch out the pixels with negligible pixel loss outside the [0 255] range, as can be seen in the histogram in part (d).

M-Files:

**Unitstep.m:**

%% Code from Lab 1 Instructions Document

function y = unitstep(x)

% The unit step function, u(x).

if nargin ~= 1 %% this function needs exactly 1 input parameter

disp("unit step requires 1 argument!");

return

end

y = cast(x >= 0, class(x));

**DT\_dirac.m:**

function y = DT\_dirac(x)

%% Discrete Time Diract Delta function delta(x)

%% this function modifies the built-in MATLAB dirac() function, which is CT

if (nargin ~= 1) %% this function requires exactly 1 input parameter

disp("DT\_dirac function requires 1 input argument!"); %% error catching

return

end

%% The built-in dirac() function is CT, which means it outputs a value of Inf

%% since we're working with DT signals, the dirac function should output a 1 value

%% therefore, we modify that

y = dirac(x) %% built in dirac function

Ind = y == Inf; %% find the index for which the output value is Inf

y(Ind) = 1; %% replace the Inf with 1

**student\_average.m:**

function grade\_array = student\_average(student\_grade\_matrix, grades\_out\_of, relevant\_column) %% user-defined function to calculate student averages

grade\_array = zeros(1,length(student\_grade\_matrix),'double');

%% initialize the grade array as zeros initially (to be changed) as an array of 1 row, as many %% columns as there are in the student grade matrix. The data type to be stored is double to %% allow for decimal point calculations

for i = 1:length(student\_grade\_matrix) %% to iterate through every column in the student grade %% matrix; 1 for each student

max\_possible = 0; %% initialize the maximum grade for the lab, midterm or exam to 0

for j = relevant\_column{1}(1):relevant\_column{2}(1) %% to go through only the relevant %% columns to each course material

grade\_array(i) = grade\_array(i) + student\_grade\_matrix{i}(j-1); %% sum the achieved %% grades for the grade components and store it in the array for now

max\_possible = max\_possible + grades\_out\_of(j-1); %% get the maximum possible for that %% component

end

grade\_array(i) = (grade\_array(i)/max\_possible)\*100; %% divide the sum of the grade %% component by the maximum possible grade to get the average as a percentage, and store %% each percentage in its appropriate index in the array.

end

end