ELECENG 3TP3 Signals and Systems

Lab 1 Report

Section C01, Instructor: Prof. Jun Chen

Khaled Hassan

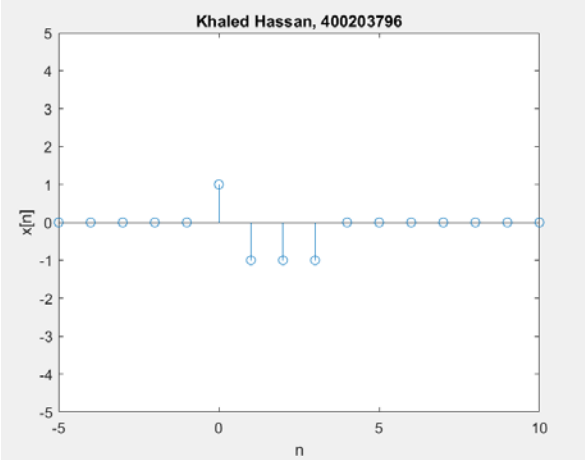
Hassak9

400203796

Section 2: Procedure

1. Discrete Time Signal Plotting
    a. $x[n] = u[n] - 2u[n-1] + u[n-4]$

Graph:

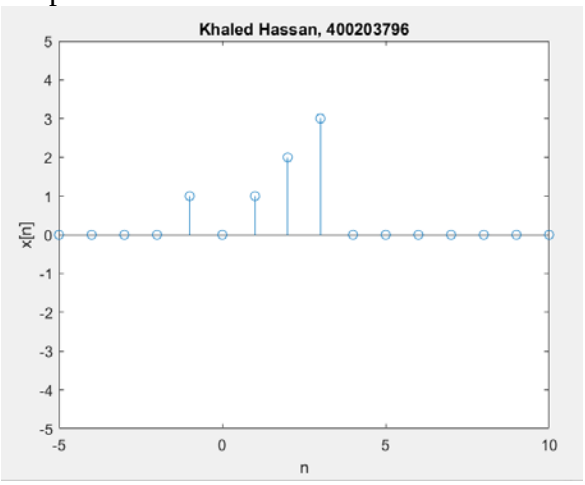


Code:

```
%% 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")
```

    b. $x[n] = (n+2)u[n+2] - 2u[n] - nu[n-4]$

Graph:

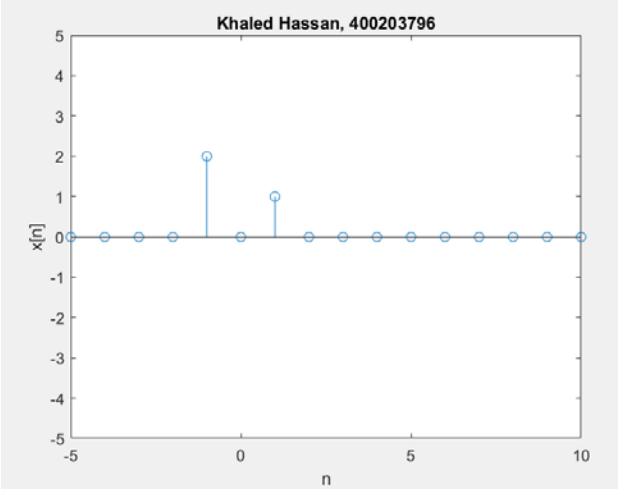


Code:

```
%% 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")
```

c. $x[n] = \delta[n + 1] - \delta[n] + u[n + 1] - u[n - 2]$

Graph:

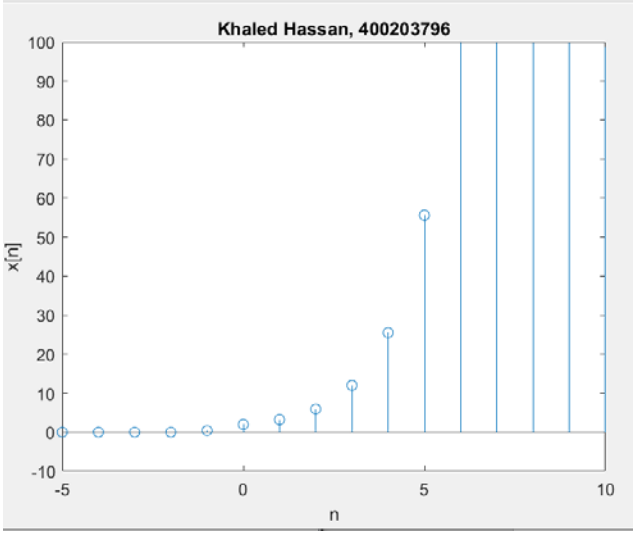


Code:

```
%% 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")
```

d. $x[n] = e^{0.8n}u[n + 1] + u[n]$

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().

2. Student Grades

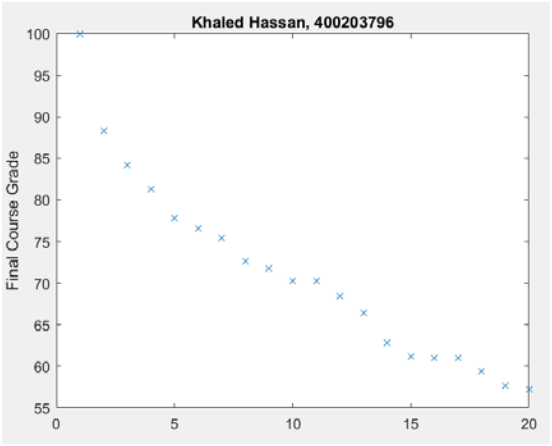
Code:

```
%% 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);
```

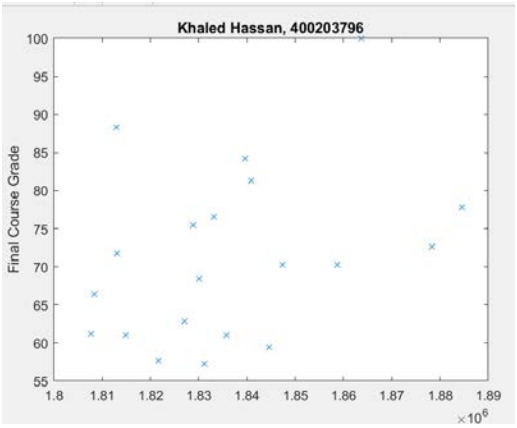
```
%% double variables to hold overall class average in each component
lab_class_average = 0;
midterm_class_average = 0;
exam_class_average = 0;

%% 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)




```
Individual Student Exam Grades:
Student ID: 1863684
Student Mark: 66.000000
 Student ID: 1812868
Student Mark: 53.000000
 Student ID: 1839570
Student Mark: 55.000000
 Student ID: 1840883
Student Mark: 54.000000
 Student ID: 1884504
Student Mark: 56.000000
 Student ID: 1833050
Student Mark: 58.000000
 Student ID: 1828890
Student Mark: 51.000000
 Student ID: 1878331
Student Mark: 66.000000
 Student ID: 1813101
Student Mark: 64.000000
 Student ID: 1847282
Student Mark: 54.000000
 Student ID: 1858670
Student Mark: 71.000000
 Student ID: 1830106
Student Mark: 57.000000
 Student ID: 1808423
Student Mark: 100.000000
 Student ID: 1827066
Student Mark: 53.000000
 Student ID: 1807690
Student Mark: 28.000000
 Student ID: 1835698
Student Mark: 41.000000
 Student ID: 1814814
Student Mark: 42.000000
 Student ID: 1844570
Student Mark: 93.000000
 Student ID: 1821699
Student Mark: 68.000000
 Student ID: 1831178
Student Mark: 82.000000

Overall Class Exam Average:  60.600000
```

```
Individual Student Lab Averages:
Student ID: 1863684
Student Mark: 100.000000
 Student ID: 1812868
Student Mark: 100.000000
 Student ID: 1839570
Student Mark: 99.200000
 Student ID: 1840883
Student Mark: 88.000000
 Student ID: 1884504
Student Mark: 80.000000
 Student ID: 1833050
Student Mark: 96.000000
 Student ID: 1828890
Student Mark: 100.000000
 Student ID: 1878331
Student Mark: 100.000000
 Student ID: 1813101
Student Mark: 100.000000
 Student ID: 1847282
Student Mark: 100.000000
 Student ID: 1858670
Student Mark: 100.000000
 Student ID: 1830106
Student Mark: 95.200000
 Student ID: 1808423
Student Mark: 100.000000
 Student ID: 1827066
Student Mark: 100.000000
 Student ID: 1807690
Student Mark: 88.000000
 Student ID: 1835698
Student Mark: 98.400000
 Student ID: 1814814
Student Mark: 100.000000
 Student ID: 1844570
Student Mark: 96.000000
 Student ID: 1821699
Student Mark: 92.000000
 Student ID: 1831178
Student Mark: 96.000000

Overall Class Lab Average:  96.440000
```

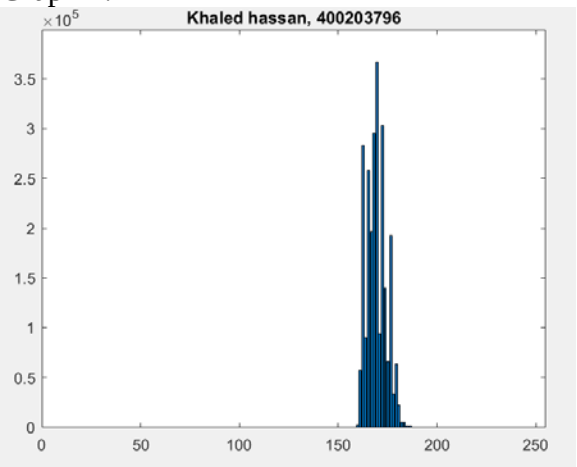
Question 2 Discussion:

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.

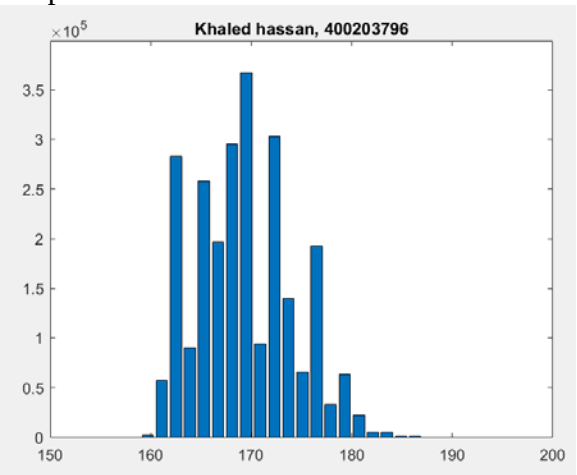
3. Image Processing

(a)

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
```

(b)

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.

(c)

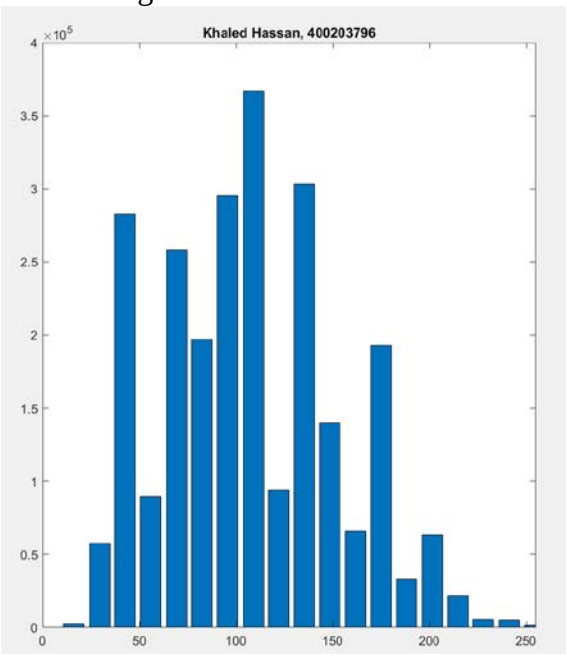
Improved Image:


```
%% 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.

(d)

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
```