148

UNIVERSITY OF COLOMBO, SRI LANKA

UCSC

UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

Academic Year 2016/2017 – Second Year Examination – Semester II – 2019

SCS2211– Laboratory II – Part B

TWO (2) HOURS (for A + B)

To be completed by the candidate

Examination Index No: .....

Important Instructions to candidates:

1. The medium of instruction and question is **English**.
2. **Write your answers** in **English.**
3. If a page or a part of this question paper is not printed, please inform the supervisor immediately.
4. Note that questions appear on both sides of the paper. If a page is not printed, please inform the supervisor immediately.
5. Write your index number on each and every page of the question paper.
6. This paper has **04** questions and **18** pages.
7. Answer **ALL** questions. All questions carry equal marks (25 marks).
8. **This paper consists of two parts, Part A (Question No 1 and Question No 2) and Part B (Question No 3 and Question No 4) and submit separately.**
9. Any electronic device capable of storing and retrieving text including electronic dictionaries and mobile phones are **not allowed**.
10. **Non-Programmable** calculators are **allowed**.

For Examiner's use only	
Question No	**Marks**
1	
2	
3	
4	
Total	

Part B

Question 3

(i) A prime number (or a prime) is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers.

Fill-in the blanks of the ***Code Listing 1*** in Octave/Matlab to create a row vector `p` with prime numbers from `1` to any given number '`n`'.

Steps are listed below:

1) Initialize a row vector `Z` with the values from `1` to `n` in ascending order. The index of each element should be its content.
2) Use two `for`-loops such that one nested inside the other.
3) The outer `for`-loop's increment variable '`i`' should be used to traverse all elements of `Z` until the n^{th}.
4) The inner for-loop's increment variable '`j`' should be used to generate multiples of '`i`'.
5) Set an element of the row vector `Z` to zero if the containing value (or the element's index) is not a prime number.
6) Suppose `k = i*j` where `i, j` are defined in above steps. Set k^{th} element of `Z` to be zero since k is **not a prime** and `k` is a multiple of `i` and `j`.
7) Initially set the first element of `Z` to be zero since `1` is not a prime number.
8) Note that `k` should always be less than or equal to `n`. Use an `if` condition for this.
9) Identify the limits of iterators `i, j` properly.
10) Use `find()` to obtain all the non-zero values of `Z` and assign to a variable `p` which is a row vector. This is the vector that contains prime numbers. Display the row vector `p`.

An example output of the programme in ***Code Listing 1*** that is showing prime numbers from 1-20 is listed below:

```
Prime numbers are given below:
p =
    2    3    5    7   11   13   17   19
```

Code Listing 1: *Fill-in the blanks to obtain a vector p of prime numbers.*

```
n = 20;

Z = ..... % 1 mark.

Z(1) = ..... ; % 1 mark.

for i = 2 : ..... % 1 mark.

  for j=2 : ..... % 1 mark.

    k =..... ; % 1 mark.

    if( .....) % 1 mark.

      ..... = 0;  % 2 marks.

    endif

  endfor

endfor

p = ..... ; % Use find(). 2 marks.

disp('prime numbers are given below:');
disp(p);
```

[10 Marks]

(ii) The Kullback-Liebler (KL) distance function is used to identify the **similarity/dissimilarity between two probability distributions**. The two distributions **p, q** are similar if the KL distance is a low value. It is zero if the two distributions are identical.

Suppose two probability distributions are represented by two row vectors **p, q** of same number of elements. The sum of the elements of each vector is one (1) since they represent probabilities.

KL distance function $\kappa(\cdot,\cdot)$ is defined as following equation (1):

$$\kappa(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n} \underbrace{p_i \log \underbrace{\left(\frac{p_i}{q_i}\right)}_{a_i}}_{b_i} \tag{1}$$

$$\mathbf{p} = [p_1, p_2, ..., p_n] \tag{2}$$

$$\mathbf{q} = [q_1, q_2, ..., q_n] \tag{3}$$

$$\mathbf{a} = [a_1, a_2, ..., a_n] = [\frac{p_1}{q_1}, \frac{p_2}{q_2}, ..., \frac{p_n}{q_n}] \tag{4}$$

$$\mathbf{b} = [b_1, b_2, ..., b_n] = [p_1 \log \frac{p_1}{q_1}, p_2 \log \frac{p_2}{q_2}, ..., p_n \log \frac{p_n}{q_n}] \tag{5}$$

Fill-in the blanks in ***Code Listing 2*** to calculate the KL distance function for any two vectors **p, q** of the same length. Assume that all elements in **p** and **q** are non-zero.

Hint: Consider element-wise matrix operations and standard matrix operations.

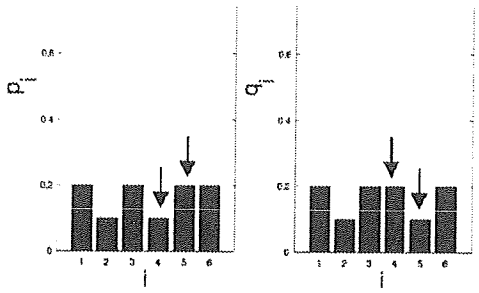

Figure 1: An example for **p, q** vectors that represent two distributions. Kullback-Liebler distance can be used to calculate the dissimilarity between the two distributions. Only the fourth and the fifth elements of **p, q** differ. The file `demo.m` in ***Code Listing 2*** contains the definition in Octave for **p, q** probability vectors.

Code Listing 2: *Fill-in the blanks to calculate the KL distance between* **p, q** *row vectors.*

kl_distance.m

```
function [ k ] = kl_distance ( p, q )

  a = p [      ] q;        % This blank is an operator.

  b = p [      ] log( [            ] );% 1st blank is an operator.

  k = [                  ]

endfunction
```

demo.m

```
p = [0.2, 0.1, 0.2, 0.1, 0.2, 0.2];
q = [0.2, 0.1, 0.2, 0.2, 0.1, 0.2];

kld_value = kl_distance ( p, q )
```

[2 × 4 = 8 Marks]

(iii)

(a) Fill-in the blanks in ***Code Listing 3*** in Octave to plot a sine signal in Figure 2 (a) between -2π to 2π with 0.01 radians resolution.

(b) Fill-in the blanks in ***Code Listing 3*** to convert the negative half of the sine wave to positive to obtain a signal in Figure 2 (b) between -2π to 2π with 0.01 radians resolution.

(c) Fill-in the blanks in ***Code Listing 3*** to convert the negative half of the sine wave to zero to obtain a signal indicated in Figure 2 (c) between -2π to 2π with 0.01 radians resolution.

Code Listing 3: *Fill-in the blanks to obtain the three signals in Figure 2 (a) to (c).*

```
clear;
close all;
clc;

x = .....; % 1 mark.

y = .....; % 1 mark.
subplot(1,3,1);
plot(x,y,'linewidth',4);
ylim([-1,+1]);

y2 = .....; % 2 marks.
plot(x,y2,'linewidth',4);
ylim([-1,+1]);
y3 = y;

y3(find(.....)) = .....; %3 marks.
subplot(1,3,3);
plot(x,y3,'linewidth',4);
ylim([-1,+1]);
```

[7 marks]

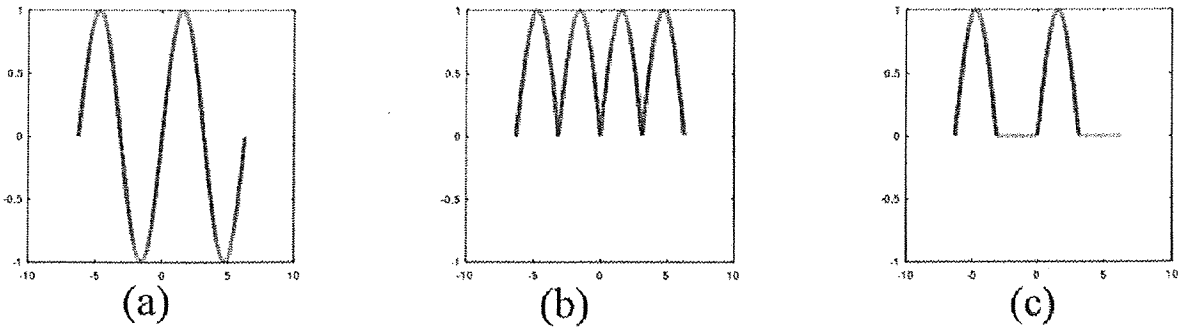

Figure 2: (a) Sine signal between -2π and 2π. (b) Negative half of sine wave is converted to positive values between -2π and 2π. (c) Negative half of the sine wave is converted to zero between -2π and 2π. Fill-in the blanks in ***Code Listing 3*** to answer question (3) (iii) – (a) to (c).

Question 4

[25 Marks]

(i) The triangle ABC should be rotated by an angle of $\pi/2$ around the Z axis. Rotation around Z axis by an angle θ is defined by the following equation (6):

$$R_Z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (6)$$

Write Octave code to obtain the resultant triangle A'B'C' for the following A, B, C coordinates: A ≡ [1, 0], B ≡ [1, 1], C ≡ [0, 0]

Note that `sin(.), cos(.)` functions take radian inputs, not degrees. Octave has **`pi`** keyword to get the value of π. Use a variable **`theta`** instead of hardcoded matrix values.

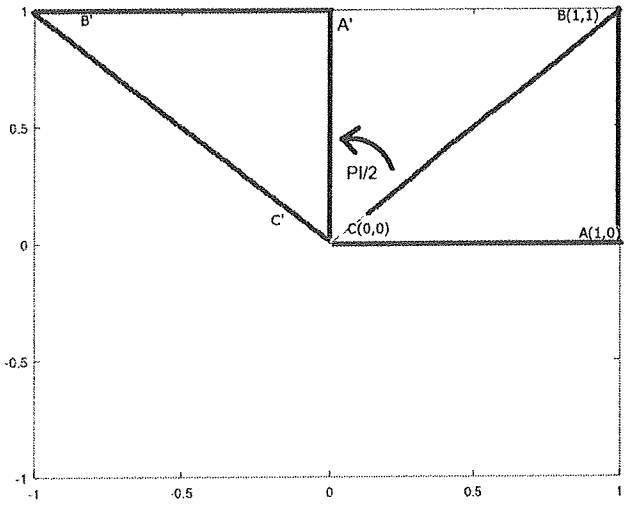

Figure 3: Write Octave code to rotate the triangle ABC by $\pi/2$ to obtain triangle A'B'C'.

Code Listing 4: *Rotation of ABC around Z axis to obtain A'B'C' triangle.*

```
clear; clc; close all;
-------------------------------------------------------------------
theta =                                          % 0.5 mark.
-------------------------------------------------------------------
R_z = [                  ,                ,                  ; ...
-------------------------------------------------------------------
                         ,                ,                  ; ...
-------------------------------------------------------------------
                         ,                ,          ]; % 3 marks.
```

```
A = [1; 0; .....];    B = [1; 1; .....];    C = [0; 0; .....]; % 1 Mark.
A_dash =
B_dash =
C_dash =                                                  % 1.5 marks.
```

[6 Marks]

(ii) Write Octave code to calculate the x_1 and x_2 coordinates of P. P is the crossing point of the two lines L_1 and L_2 in Figure 4.

L_1 : $-2 = x_1 - x_2$

L_2 : $-3 = 3 * x_1 - x_2$

Hint: *Represent the two lines as two linear equations in a linear system Ax=b. The point P is the solution for x_1, x_2 for the set of these two linear equations.*

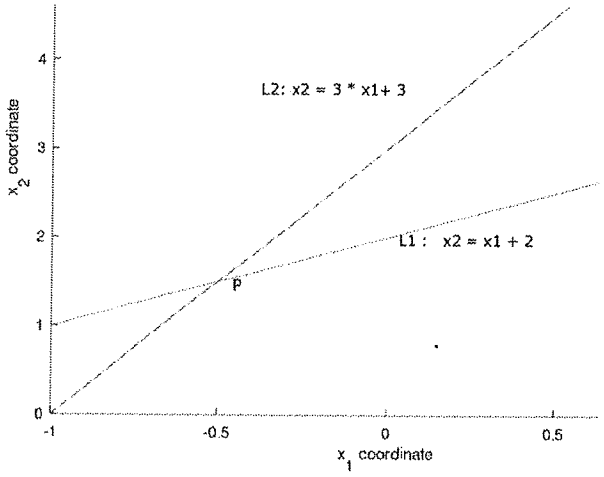

Figure 4: Write Octave code to solve the two linear systems to obtain the x_1, x_2 coordinates of P.

Code Listing 5: *Solve the linear systems L_1, L_2 to obtain the point P in Figure 4.*

```
clear; clc; close all;
A = [      ,      ;      ,      ]; % 2 marks.
b = [      ,      ];                          % 1 mark.
P =                                           % 2 marks.
```

[5 Marks]

(iii) A convex function is a curved down function with minimum/minima (see Figure 5). Consider two points $f(x_i)$ and $f(x_j)$ on the curve of any function $f(x)$. A line segment called "code" can be drawn connecting these two points $f(x_i)$ and $f(x_j)$. This function $f(x)$ is convex if this "code" lies above the function's curve for all the points between x_i and x_j (see Figure 6 in the last page).

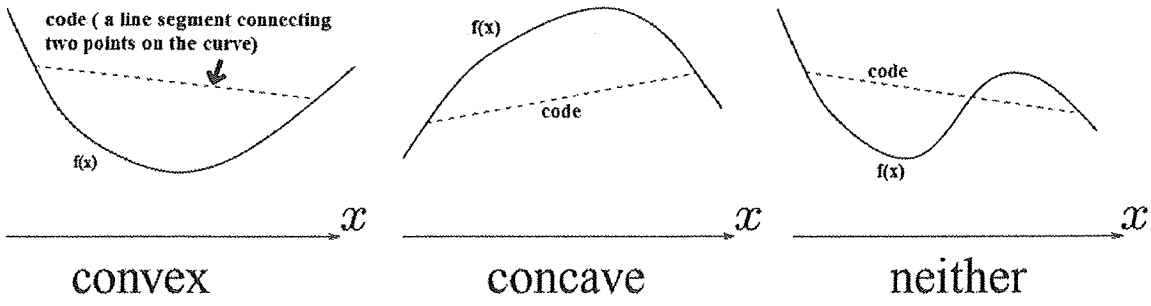

Figure 5: A convex function, a concave function, and a function that is neither convex nor concave. Convex functions are curved down. Concave functions are curved up. Consider how the "code" (that is, the line segment connecting any two points on the function curve $f(x)$) is situated.

Suppose α, β are two positive values that sum up to one (1) defined as equation (9). A function $f(x)$ is convex if the following inequality (7) is true:

$$f(x_k) \leq \alpha f(x_i) + \beta f(x_j) \quad (7)$$

$$x_k = \alpha x_i + \beta x_j \quad (8)$$

$$\beta = (1 - \alpha) \text{ where } \alpha \in [0, ..., 1] \quad (9)$$

On an XY coordinate system, the y-coordinate of any point P on the "code" for an x_k value between x_i and x_j can be obtained by $\alpha f(x_i) + \beta f(x_j)$. This refers to the right hand side of inequality (7). The meaning of this inequality (7) is that an arbitrary point P on the "code" is always above the function at x_k, that is $f(x_k)$. This is shown in Figure 6 (last page). Here, α and β represent the ratio of distances from an arbitrary point in the middle called x_k to the two points x_i and x_j .

Suppose an example function $f(x)$ is defined as below.

$$f(x) = 4x^4 + x^2 + 2 \quad (10)$$

Fill-in the blanks in ***Code Listing 6*** to **numerically check whether the function $f(x)$ in eq. (10) is a convex function**.

Hint: This is similar to checking whether a matrix is positive definite by obtaining random values, but in this case, the random values for α and β are between 0 and 1.

Steps are given below:

1. First, represent $f(x)$ in the standard polynomial format.
   Use the **`polyval(coeffients, xValues)`** function with the necessary coefficients to calculate the function values $f(x)$ for `x = [-2:0.05:2]` range.
2. Display $f(x)$ for x values from -2 to +2.
3. Set x_i = -2 and x_j = +2. For x_i and x_j, calculate $f(x_i)$ and $f(x_j)$.
4. Set N = 100.
5. Create a row vector `alpha` with N number of random values between 0 and 1.
6. Calculate `beta = 1 - alpha`
7. For n^{th} element in the row vector `alpha`, repeat the following steps:
   (i) Calculate `x_k = alpha[n] * x_i + beta[n] * x_j`
   (ii) Calculate $f(x_k)$ by using **`polyval( )`**.
   (iii) Calculate the point on the code at x_k,
   $$P\_y = \texttt{alpha[n]} * f(x_i) + \texttt{beta[n]} * f(x_j).$$
   (iv) If $f(x_k) > P\_y$, set flag to indicate that $f(x)$ is not a convex function and break the loop. Otherwise, keep iterating.
8. After the loop, check the flag whether $f(x)$ is convex or not, and print the output.

Code Listing 6: *Fill-in the blanks in the Octave code to numerically check whether the function $f(x)$ in equation (10) is convex or not.*

```
c = .....;

x = -2: ..... :2;

f_x = polyval(c,x);

plot(....., .....); % 0.5 marks x 2 = 1 mark.
x_i = -2;
x_j = 2;
```

```
N = 100;

alpha = rand(1,.....);
beta = 1 - alpha;

f_x_i = polyval(c, x_i);
f_x_j = polyval(c, x_j);

isConvex = 1;

for n = 1:.....

  x_k = alpha(n) * x_i + ..... * .....;

  f_x_k = .....;

  P_y = alpha(n) * f_x_i + ..... * .....;

  if(..... > .....)

          isConvex = .....;
          break;
  endif

endfor

if(.....)
  disp('convex function');
else
  disp('not convex function');
endif
```

[14 Marks]

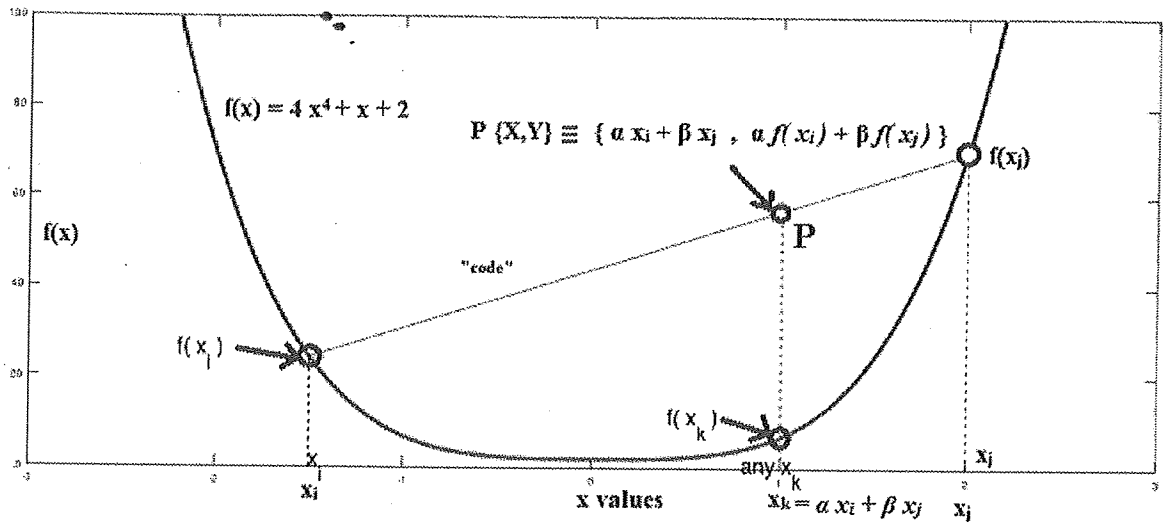

Figure 6: Verify that *f(x) in eq. (10)* is a convex function by considering random α values between 0 and 1 and checking whether the arbitrary point P on the code lies above the function value $f(x_k)$.

****