Exercises 2a: MAE Unit 2

Name: Josep Famadas Alsamora

Collaboration Info: I have used some information from mathworks.com.

Exercise 1. Evaluation of simulation time.

Explanation:

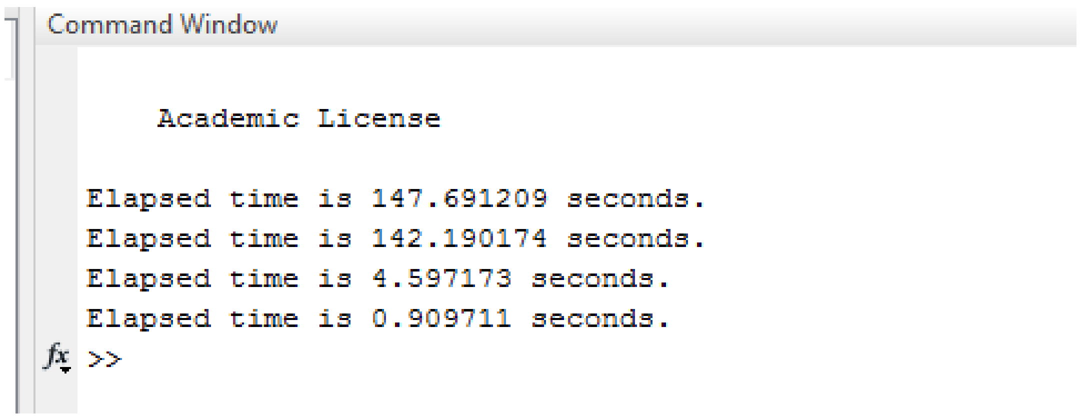
(1) I just used the command 'rund' to execute the Matlab file provided.

MATLAB Code:

```
%% (1)

run('Exec_time');
```

Results:

(1)



```
    Academic License

Elapsed time is 147.691209 seconds.
Elapsed time is 142.190174 seconds.
Elapsed time is 4.597173 seconds.
Elapsed time is 0.909711 seconds.
>>
```

Coments:

The first time it goes throw the whole B and C, so it is the longest.

In the second time, the matrix A has been previously created so has a constant size, which means a little save in time.

In the third, it only iterates C so it means a huge amount of time saving.

In the last one it does not iterate B nor C so, as expected, it is the shortest in time execution.

Exercise 2. Numerical integration.

Explanation:

(1) I create the function ‘Definite_integral’, which first makes the 2 comprovations. The amount of elements is odd if the size of the vector is not even, and the elements are equispaced if substracting the first element from everyone of them they turn into multiples of delta.
When both comprovations are done it just calculate the definite integral with the formula given.
(2) Here I create the function ‘Laplace_pdf’ which just applies the formula given.
(3) In this part I call both functions to calculate what has been asked using the formula given.

MATLAB Code:

(1)
```
function [ integral ] = Definite_integral( x, fx )
%Calculates the definite integral of f(x) between x(1) and x(end).

n = length(x);
delta = (x(end)-x(1))/(n-1);

comp = sum(mod((x-x(1)),delta));

if (comp~=0)
    disp('ERROR: Not equispaced X');
    return
elseif (mod(n,2)==0)
    disp('ERROR: The number of samples is not odd');
    return
end

v = 4*ones(1,n);
v(1:2:end) = 2;
v(1)= 1;
v(n)= 1;

integral = delta/3 * sum(fx .* v);

end
```

(2)

```
function [ laplace ] = Laplace_pdf( x, b, mu )
%Calculates the Laplace probability density function in 'x' with the
%parameters 'b' and 'mu'.
laplace = 1/(2*b).*exp(-abs(x-mu)/b);

end
```

(3)

```
%%(3)
x = 1:1:101;
b = 4;
mu = 2;

fx0 = Laplace_pdf(x, b, mu);
fx = fx0.*log(fx0);
entr = -1 * Definite_integral(x,fx)
```

<u>Results:</u>

-For x = 1:1:101

entr = 1.7942

<u>Comments:</u>

I am not sure that this result is correct because I have nothing to compare with.

Exercise 3. Mathematical equations.

Explanation:

Here I just used the function ‘inline’ to create the f(x) desired.

MATLAB Code:

```
%%(1)

g = inline('1-exp(-b*x).*cos(a*x)')
```

Results:

Inline function:

```
     g(a,b,x) = 1-exp(-b*x).*cos(a*x)
```

Comments:

Exercise 4. Mathematical equations.

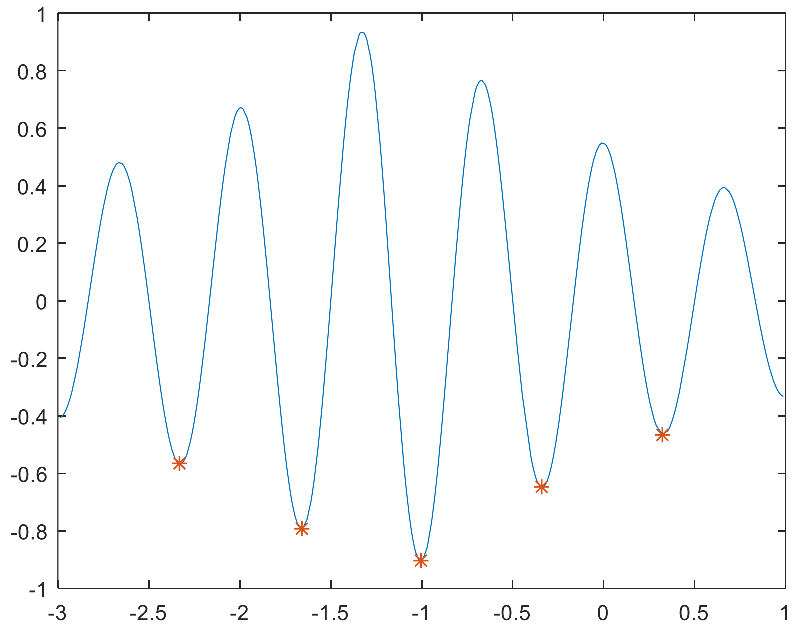
Explanation:

(1) First of all, I loaded the file ‘ex4_data.m’ with the function ‘load’.
(2) To search the minimums I create two copies of ‘f’, one moved 1 position to the left and the other to the right. Then I just find the points of ‘f’ that were smaller than f_left and f_right.
(3) I ploted the function ‘f’ and over it (with hold on) the function ‘f’ only evaluated in the mins and with *.

MATLAB Code:

```
%%(1)

load('ex4_data.mat')

%% (2)

f_left = [-inf f(1:end-1)];
f_right = [f(2:end) -inf];

mins = find(f<f_left & f<f_right);

%% (3)

plot(x,f)
hold on
plot(x(mins),f(mins),'*')
hold off
```

Results:



Exercise 5. Run length encoding.

Explanation:

(1) First of all, the function tries to be sure that there is not any integer out of the possible bit range. Then, I create a vector of 0s and I fulfill it with a for that iterates every 1st ,2nd ,3rd... n_bitsth position and puts the correspondent bit from the int_list.

(2) In this second function I use a similar technique to the one used to find minimums in ex4. I create a copy of the sequence but displaced 1 bit to the left and compare both to see where there is a change of bit so I can be able to compute the size of each burst. Finally, I use the previous function to compress the sequence.

(3) In this final part I just load the data, compute it using the function from part (2) and print the result.

MATLAB Code:

(1)
```
function [ bin_list ] = d2b_conversion( int_list, n_bits )
%transform an array of integers into blocks of n_bits


%comprovation
if (~isempty(find(int_list>2^n_bits)))
    disp('ERROR: Some bits are out of range');
    return
end


A = zeros(1,length(int_list)*n_bits);

for i = 0:1:(n_bits-1)

    A(n_bits-i:n_bits:end-i) = mod(floor((int_list-1)/(2^i)),2);


end

bin_list=A;
end
```

(2)

```
function [ fbv, burst_length, bin_compressed ] = compressing(
sequence, n_bits )
%Compress a sequence of bits in agrupation of bit bursts

fbv = sequence(1);

seq2 = [sequence(2:end), -1];
dif = find(sequence~=seq2);
```

```
burst_length = [dif(1) , dif(2:end)-dif(1:end-1)];

bin_compressed = [fbv, d2b_conversion(burst_length,n_bits)];


end
```

(3)

```
%% (3)

load('ex5_data.mat');

[fbv, burst_length, bin_compressed] = compressing(sequence,3);

bin_compressed
```

Results:

```
  Columns 1 through 20

    0    0    1    0    1    1    1    1    1    0    0    1    1    1    0    0    0    0    0    0

  Columns 21 through 40

    0    0    1    0    0    1    1    0    1    1    1    0    0    1    1    0    0    0    1    1

  Columns 41 through 46

    0    0    0    0    1    0
```

Comments:

Discussing with David Llaveria we found the way to do the part (2) without any loop.

Despite that, we found no way to do it in the part (1).