**Department of Electrical Engineering**

|  |  |
| --- | --- |
| **Faculty Member:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Dated: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
|  |  |
| **Course/Section:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | **Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
|  |  |

**EE-330 Digital Signal Processing**

**Lab #8 DFT Properties and Block Convolution Methods**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4-CLO4** | | **PLO5-CLO5** | **PLO8-CLO6** | **PLO9-CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Team Work** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| **Abdul Ahad Butt** | **262517** |  |  |  |  |  |
| **Talha Rehman** | **243575** |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**Lab8: DFT Properties and Block Convolution Methods**

**Objectives**

The objective of this lab is to implement the block convolution methods:

* Overlap and Add
* Overlap Save

**Lab Instructions**

* The students should perform and demonstrate each lab task separately for step-wise evaluation (please ensure that course instructor/lab engineer has signed each step after ascertaining its functional verification)
* Each group shall submit one lab report on LMS within 6 days after lab is conducted. Lab report submitted via email will not be graded.

. Students are however encouraged to practice on their own in spare time for enhancing their

**Lab Report Instructions**

All questions should be answered precisely to get maximum credit. Lab report must ensure following items:

* Lab objectives
* MATLAB codes
* Results (graphs/tables) duly commented and discussed
* Conclusion

**Lab8: DFT Properties and Block Convolution Methods**

**Introduction**

Two block convolution methods were discussed during the class:

* Overlap and Add Method
* Overlap Save Method

During this lab, we want to implement these methods using DFT and circular convolution. As intermediate steps to arrive at your final implementation, you will thus need to implement the following nested functions:

1. Circular Convolution *cconv\_bee()*
2. Circular Flip *cflip\_bee()*
3. Circular Shift *cshift\_bee()*

**Lab Tasks**

**Lab Task 1**

**Implementation of Circular Flipping**

Write a simple function in MATLAB, with the name *cflip\_bee()* that implements circular flipping of an input array ***x***.

The format of the function should be

function y = cflip\_bee(x,N)

%where x = input array

% N = the number of points for circular flipping (DFT points)

%y = output that should be Modulo N circularly flipped version of x

Note your function should take care of the fact that *N* can be greater than the length of ***x***. So put a check on the length of ***x***, in case it is less than *N*, you need to append zeros in ***x*** before flipping.

Note you need to play with the indices, so define an index vector at the start as:

>> indx = zeros (N,1);

Check your function for different lengths of input and values of *N*.

|  |
| --- |
| Function |
| function y = cflip\_bee (x, N)  if (N > length(x))    x = [x, zeros(1, N - length(x))]  end  x\_s = x(2:end);  x\_s = flip(x\_s, 2);  y = [x(1), x\_s];  endfunction |
| For x = [5, 4, 3, 2, 1] |
|  |

**Implementation of Circular Shifting**

Write a function in MATLAB, with the name *cshift\_bee()* that implements circular shifting of an input array *x*.

The format of the function should be

function y = cshift\_bee(x,r,N)

%where x = input array

* r = amount of shift (in samples), left shift r > 0 and right shift r < 0
* N = the number of points for circular flipping (DFT points)

%y = output that should be Modulo N circularly shifted version of x by an amount r

Note your function should take care of the fact that *N* can be greater than the length of ***x***. So put a check on the length of ***x***, in case it is less than *N*, you need to append zeros in ***x*** before shifting.

Note you need to play with the indices, so define an index vector.at the start as: >> indx = zeros (1, N);

Consider two cases with if statement in your code, depending on the input value of ‘r’. if (r>0)

% implement the left shift

if(r<0)

%implement the right shift

Verify your result for different values of *x*, *r*, and *N*.

|  |
| --- |
| Function |
| function y = cshift\_bee (x, r, N)  r = mod(r, N);  if (N > length(x))    x = [x, zeros(1, N - length(x))];  end  if (r >= 0)    y = [x( r+1: length(x) ), x(1:r)];  else    y = [x(length(x) + r + 1: length(x)), x(1: length(x) + r)];  end  endfunction |
| Verification |
|  |

**Implementation of Circular Convolution**

Write a function in MATLAB, with the name *cconv\_bee()* that implements circular convolution of an input array *x* with an array *h*.

The format of the function should be

function y = cconv\_bee(x,h,N)

%where x, h = input arrays

% N = the number of DFT points

%y = N point output of circular convolution

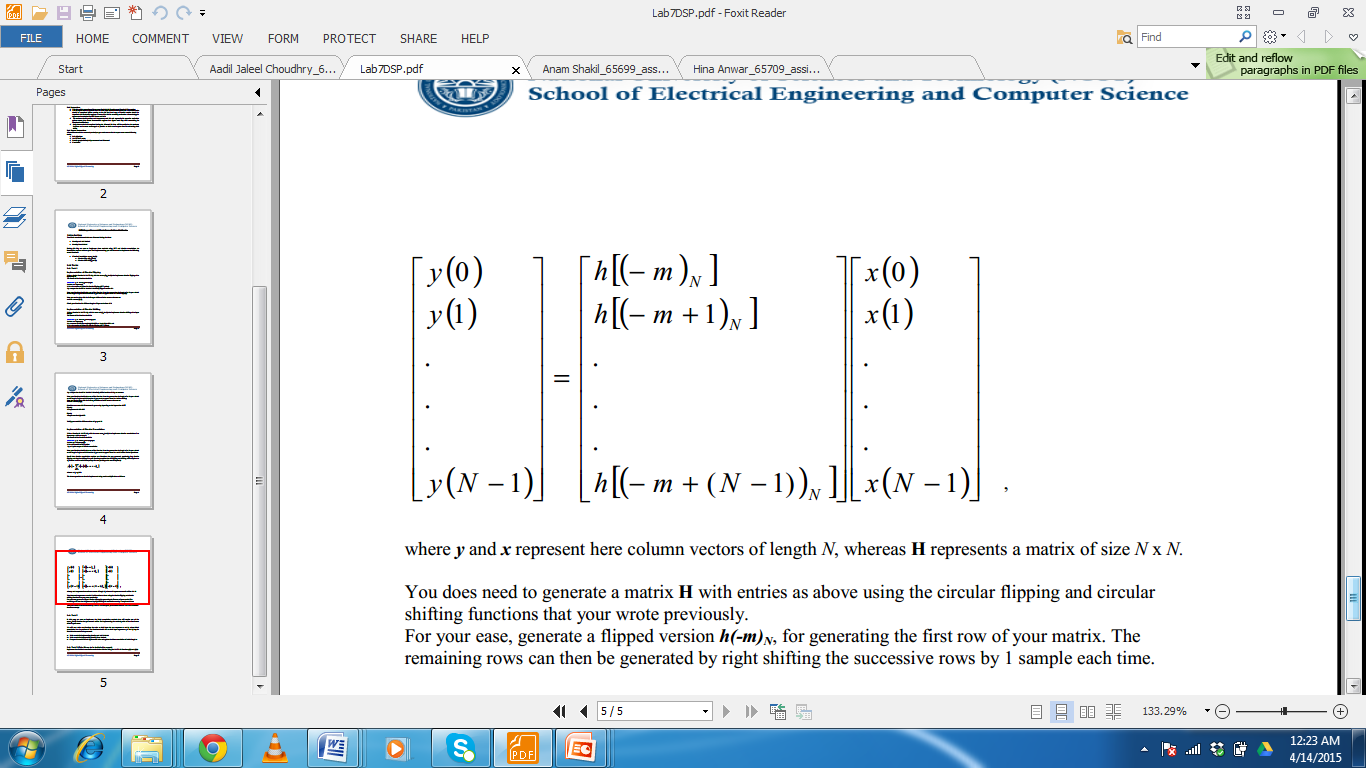
Note your function should take care of the fact that *N* can be greater than the length of ***x***. So put a check on the length of ***x***, in case it is less than *N*, you need to append zeros in ***x*** and ***h*** before further operations.

Recall that circular convolution requires two functions that you generated previously, i.e., circular flipping and circular shifting. For your function, consider that the flipping and shifting will take place on *N*, whereas***x***will remain intact (except for zero padding at the start if required).

|  |  |  |
| --- | --- | --- |
| *N* 1 |  |  |
| *y* *n* ∑ *x**m* *h* *m*  *n* *N* |  |
| *m* 0 |  |  |

where *n = 0, …, N-1.*

The above operation can thus be implemented using matrix multiplication as follows:



Where ***y*** and ***x*** represent here column vectors of length *N*, whereas **H** represents a matrix of size *N* x *N.*

You does need to generate a matrix **H** with entries as above using the circular flipping and circular shifting functions that your wrote previously.

For your ease, generate a flipped version ***h(-m)N***, for generating the first row of your matrix. The remaining rows can then be generated by right shifting the successive rows by 1 sample each time.

**Test your result on some values of *x*, *h* and *N* and compare your results with the MATLAB built-in function cconv().**

|  |
| --- |
| Function |
| function y = cconv\_bee (x, h, N)  x = [x, zeros(1, N - length(x))];  h = [h, zeros(1, N - length(h))];  for n = 1:N    y(n) = 0;    for i = 1:N      j = n - i + 1;      if (j <= 0)        j = N + j;      end     y(n) = [y(n) + x(i)\*h(j)];    end   end  endfunction |

**Lab Task 2**

In this part, we want to implement the block convolution methods that will require use of the *cconv\_bee()* function that you created earlier. For implementing the methods, refer to the class lectureslides or your notes.

We will now write two functions that take at their input the two sequences ***x*** and ***h***, whose block convolution is to be performed. The functions should take as their input arguments ***x***, ***h*** and ***L***, i.e., the block size that needs to be processed.

1. Write a code for implementing Overlap and Add Method.

|  |
| --- |
| function [ y ] = overlap\_add(x,h,L)    x\_l = length(x);  h\_l = length(h);    %Dividing 'x'  into its chunks and then convolving  x\_chunks = [];    mcconv = [];  y = [];    for i = 1:x\_l/L      x\_chunks = [x\_chunks ; x((1+((i-1)\*L)) : (i\*L))];  end  [r,c] = size(x\_chunks);    chunk\_L = c;    N = chunk\_L + h\_l - 1;    for i = 1:r      mcconv = [mcconv ; cconv\_bee(x\_chunks(i,:),h,N)];  end    y = (mcconv(1,:))  for i = 1:(r-1)      y = [y zeros(1,(L))] + [zeros(1,(i\*L)) mcconv((i+1),:)];  end    end |

1. Write a code for implementing Overlap Save Method.

|  |
| --- |
| function [y] = overlap\_save(x,h,L)    x\_l = length(x);  h\_l = length(h);    x = [zeros(1,h\_l-1),x];    loop\_point = 1;  sections = [];  index = 1;    while(loop\_point == 1)        if(length(x(index:x\_l))<L)          x = [x,zeros(1,L-length(x(index:x\_l)))];          sections = [sections;x(index:index+L-1)];          index = (index+L-1)  - length(h)+1;            loop\_point = 0;      else          sections = [sections;x(index:index+L-1)];          index = index + 1;          index = (index+L-1) - length(h) + 1;          index;      end        if(index+L-1 == x\_l)              sections = [sections;x(index:index+L-1)];              loop\_point = 0;      end  end    [r,c] = size(sections);  z = r;    section\_convolved = [];    for i = 1:z      lone\_wolf = cconv\_bee(sections(i,:),h,L);      section\_convolved = [section\_convolved;lone\_wolf];  end    discard = [];  for i=1:z      discard=[discard;section\_convolved(i,h\_l:L)];  end    y = [];    for i=1:z     y = [y,discard(i,:)];  end  end |

1. Compare the results of both against each other and against the direct convolution of whole length ***x***.

|  |  |
| --- | --- |
| Overlap And Add |  |
| Overlap And Save |  |
| FFT |  |

**Lab Task 3**

Implement the two block convolution methods but this time using the MATLAB functions *fft()* and *ifft()*.

|  |
| --- |
| function y = Conv\_fft(x,h,N)    if(N > length(x))   x = [x, zeros(1, N-length(x))];  end    if(N > length(h))   h = [h, zeros(1, N-length(h))];  end    x\_fft = fft(x);  h\_fft = fft(h);    dot\_product = x\_fft.\*x\_fft;    y = ifft(dot\_product);    end |