Table of Contents

Lab 4: Cross Correlation and Period Estimation

Author: Himanshu Sharma **Roll Number**: 1610110149 **Email**: hs583@snu.edu.in **Instructor**: Prof. Vijay K. Chakka

```
% create a main namespace
function a = main()
    % implement the function correlation.
    x = [1,2,3,4,-5];
    x_axis = 0:length(x)-1;
    y = [1,2,3,4,5];
    y_axis = 0:length(y)-1;
    [n, rxy] = correlation_array(x, y);

    X = [1,2,3,4];
    X_axis = 0:length(X)-1;
    Y = [4,3,2,1];
    Y_axis = 0:length(Y)-1;
    [n_dash, rXY] = correlation_array(X, Y);

    figure;
    suptitle('Cross-Correlation');
    subplot(2,3,1);
    stem(x_axis, x);
    grid on;
    ylabel('$$x(n)$$', 'interpreter', 'latex');
    xlabel('$$n$$', 'interpreter', 'latex');
    subplot(2,3,2);
    stem(y_axis, y);
    grid on;
    ylabel('$$y(n)$$', 'interpreter', 'latex');
    xlabel('$$n$$', 'interpreter', 'latex');
    subplot(2,3,3);
    stem(n, rxy);
    ylabel('$$r_{xy}(\tau)$$', 'interpreter', 'latex');
    xlabel('$$ \tau $$ (time delay)', 'interpreter', 'latex');
    grid on;
    subplot(2,3,4);
    stem(X_axis, X);
    xlabel('$$n$$', 'interpreter', 'latex');
    ylabel('$$ x(n) $$', 'interpreter', 'latex');
    grid on;
    subplot(2,3,5);
    stem(Y_axis, Y);
    xlabel('$$n$$', 'interpreter', 'latex');
```

```
    ylabel('$$ y(n) $$', 'interpreter', 'latex');
    grid on;
    subplot(2,3,6);
    stem(n_dash, rXY);
    ylabel('$$r_{xy}(\tau)$$', 'interpreter', 'latex');
    xlabel('$$ \tau $$ (time delay)', 'interpreter', 'latex');
    grid on;

    % Period Estimation of noiseData.mat
    % load the noiseData.mat
    noise = load('noiseData.mat', '-mat');
    noise = noise.noiseData;

    % find the auto-correlation of noise
    [out_index, output] = correlation_array(noise, noise);

    % the output of auto-correlation will contain zeros in front.
    % remove them using remove_initial_zeros.
    [new_indexes, non_zero_outputs] = remove_initial_zeros(out_index,
output);

    % detect peaks in the auto-correlation output now.
    [peaks, pos] = peak_detector(non_zero_outputs);

    % plot the auto-correlation function with peaks.
    figure;
    suptitle('Auto-Correlation of noiseData.mat');
    % first plot the noise itself.
    subplot(2,1,1);
    stem(noise);
    xlabel('$$n$$', 'interpreter', 'latex');
    ylabel('$$N_{n}$$', 'interpreter', 'latex');
    grid on;

    subplot(2,1,2);
    % enable hold so that peaks can also be plotted.
    hold on;
    stem(new_indexes, non_zero_outputs);
    % fix \tau array, since the output starts from -ve index but pos
contains positive numbers.
    pos = pos-50;
    scatter(pos, peaks);
    xlabel('$$ \tau $$', 'interpreter', 'latex');
    ylabel('$$r_{NN}(\tau)$$', 'interpreter', 'latex');
    grid on;

    % find the difference between indices of peaks.
    differences = [];
    % for this traverse through the len-1 of pos and take difference
of k+1 and kth elements
    for k = 1:length(pos)-1
        differences = [differences  pos(k+1) - pos(k)];
    end
```

```
   % the array differences contain the time periods. Take the average
to get the time period of noiseData.mat
   fprintf('The time period of noiseData.mat is %d.',
mean(differences));

   function [newIndex, newValues] = remove_initial_zeros(indexes,
values)
       % this function removes initial zeros from the array.
       % eg: [0,0,1,2,3,4,0,8] ---> [1,2,3,4,0,8]

       % the basic concept is to traverse the array until a non-zero
element is encountered and then
       % from that value of variable 'i', return the array.
       i = 1;
       while values(i) == 0
           i = i+1;
       end
       newIndex = indexes(i:length(indexes));
       newValues = values(i:length(values));

   end
   % create a correlation function.
```

Correlation Function

```
   function [n, rxy] = correlation_array(x, y)
      % this function reverses 'y' and 'x' remains untouched.

      % to reverse an array, we can flip it.
      y = y(length(y):-1:1);

      % the zeroth index of 'y' is now the last element.
      % take this example
      % x = 1,2,3,4,5 and y = 4,5,6
      % flip y ---> 6,5,4. But 4 is at index 0 ( 1 for MATLAB ).
      %     1 2 3 4 5
      % 6 5 4
      % add length(y) -  1 zeros to the front of x
      % 0 0 1 2 3 4 5
      % 6 5 4
      % take the difference in length and add that many zeros to the
end of y.
      % 0 0 1 2 3 4 5
      % 6 5 4 0 0 0 0
      % compute the convolution.
      front_zeros = length(y)-1;
      x = [zeros(1, front_zeros) x];
      y = [y zeros(1, abs(length(x) - length(y)))];
      rxy = lin_conv(x, y);
      % the cross correlation will contain -ve 'n' values but here
its starting
      % from 1 due to our convolution function. So make an array 'n'
which starts from -ve
```

```
        % indices.
        % in the above example, [0 0; 6 5] were the reason of getting
terms before n = 0.
        % Hence start indexing from -4.
        n = -front_zeros*2:length(rxy)-1-front_zeros*2;
    end

    % define a linear convolution from your side.
```

Linear Convolution Function

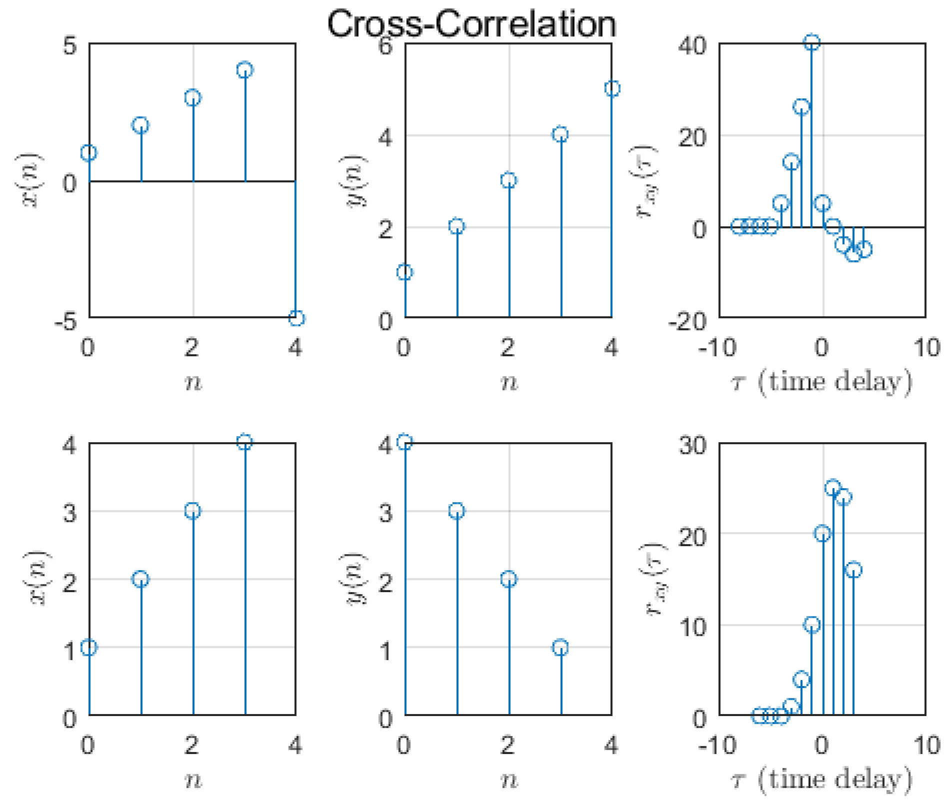
```
    function z = lin_conv(x, y)

        % make length of x and y equal.
        extra_zeros = zeros(1, abs(length(x) - length(y)));
        if length(x) < length(y)
            x = [x extra_zeros];
        else
            y = [y extra_zeros];
        end

        % define an empty output array.
        z = zeros(1, length(x) + length(y) -1);

        % since convolution is commutative, reverse 'y'.
        y = y(length(y):-1:1);

        for i = 1:length(y)
             out = y(i)*x;

             % add zeros at the end.
             out = [out zeros(1, i-1)];
             % add zeros at front.
             out = [zeros(1, length(z) - length(out)) out];

             % add this to z
             z = z + out;
        end
        % remove trailing zeros from z.
        z = z(length(z):-1:1);
        new_arr = [];
        counter = 1;

        for i = 1:length(z)
           if z(i) ~= 0
               break;
           else
               counter = counter + 1;
           end
        end

        % now collect the non zero values in z starting from counter.
        z = z(counter:length(z));
```

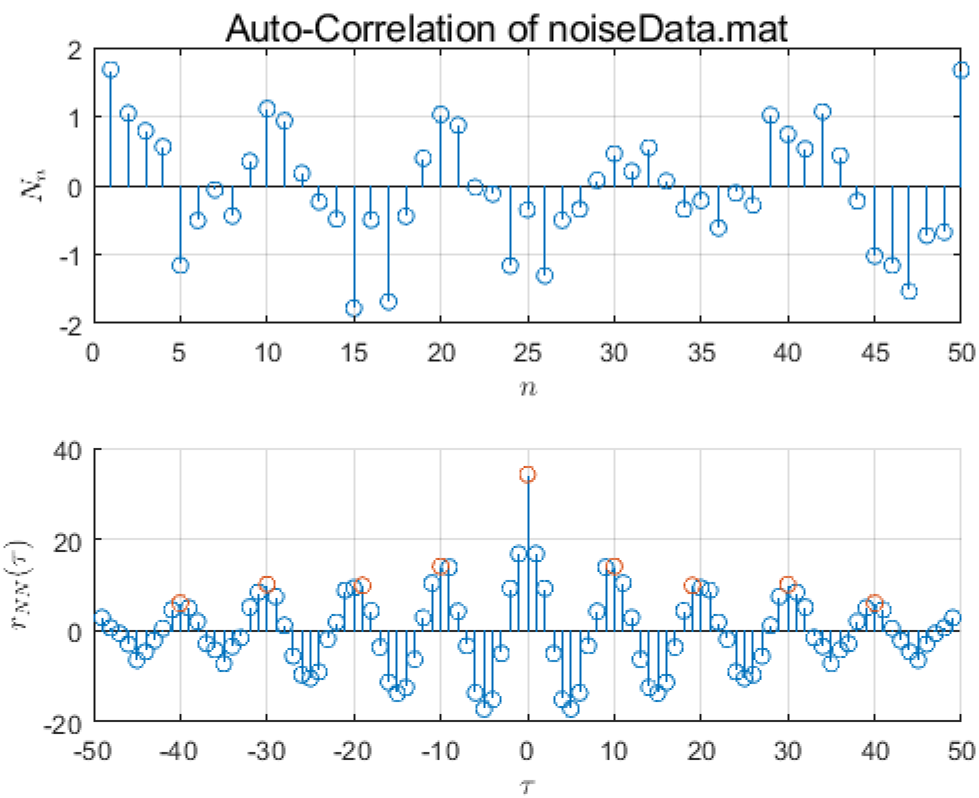
```
        z = z(length(z):-1:1);

    end
```

Peak Detector Function

```
    function [peaks, positions] = peak_detector(x)
        % returns the  peaks in the data and their positions (not the
index).
        l = length(x);

        % initialise 3 counters which will traverse the whole array. If
any one of
        % them is more than the length of the array, we stop the loop.
        c1 = 1;
        c2 = 2;
        c3 = 3;

        % initalise empty arrays which will store peaks and their
positions.
        peaks = [];
        positions = [];

        while (c1 < l & c2 < l & c3 < l)
            % we don't need negative peaks.
            if (x(c2) > x(c1) & x(c2) > x(c3) & x(c2) > 0)
                % since it is a positive peak, add it into peaks array.
                peaks = [peaks x(c2)];
                positions = [positions c2];

                % increase counters by 2, because the x(c3+1) might be
a peak but if x(c3+2) is taken as x(c2),
                % then x(c3+1) will not be detected.
                c1 = c1 + 2;
                c2 = c2 + 2;
                c3 = c3 + 2;
            else
                % increment counters by one, since peak is not
detected.
                c1 = c1 + 1;
                c2 = c2 + 1;
                c3 = c3 + 1;
            end

        end
    end
end
```

The time period of noiseData.mat is 10.


Cross-Correlation
$x(n)$
$y(n)$
$r_{xy}(\tau)$
n
τ (time delay)


Auto-Correlation of noiseData.mat
N_n
n
$r_{NN}(\tau)$
τ

Published with MATLAB® R2017b