EEE 391
Basics of Signal and Systems
Fall 2019-2020
Computer Assignment 2

First name: *Fuad*
Last name: *Aghazada*
Student ID: *21503691*
Section: *1*

1)

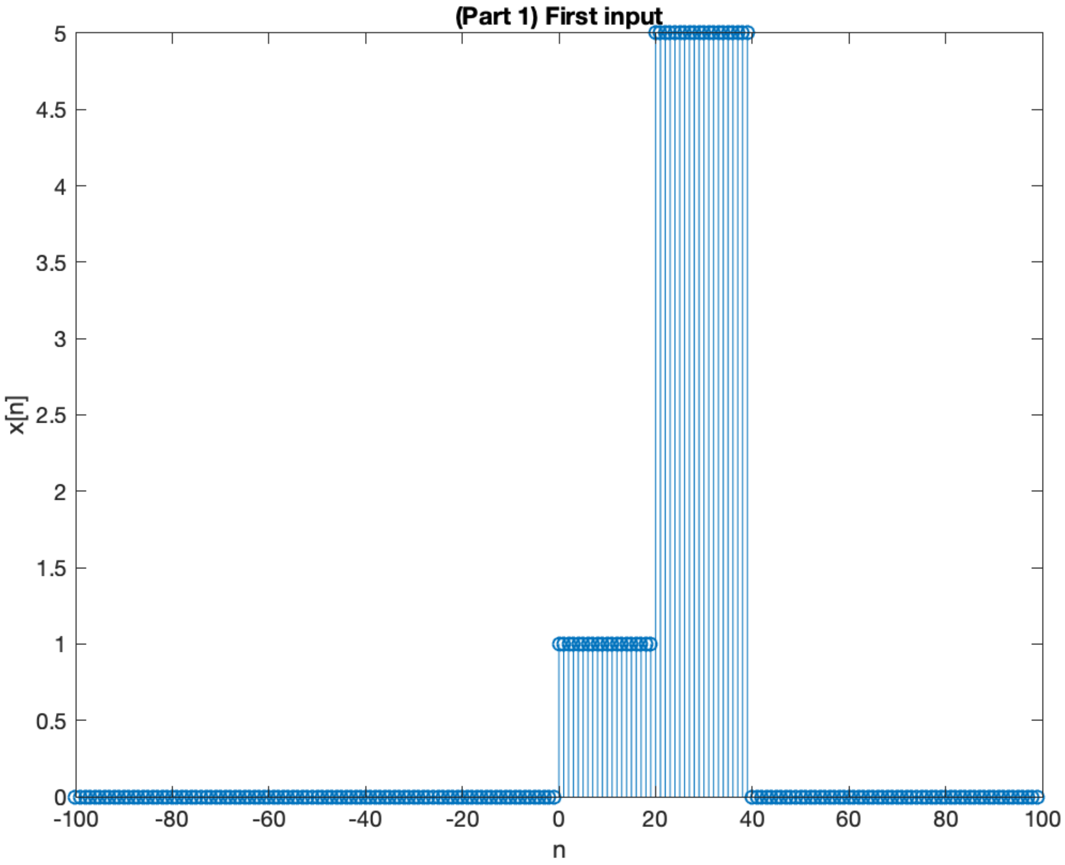

Figure 1. First input in Part 1

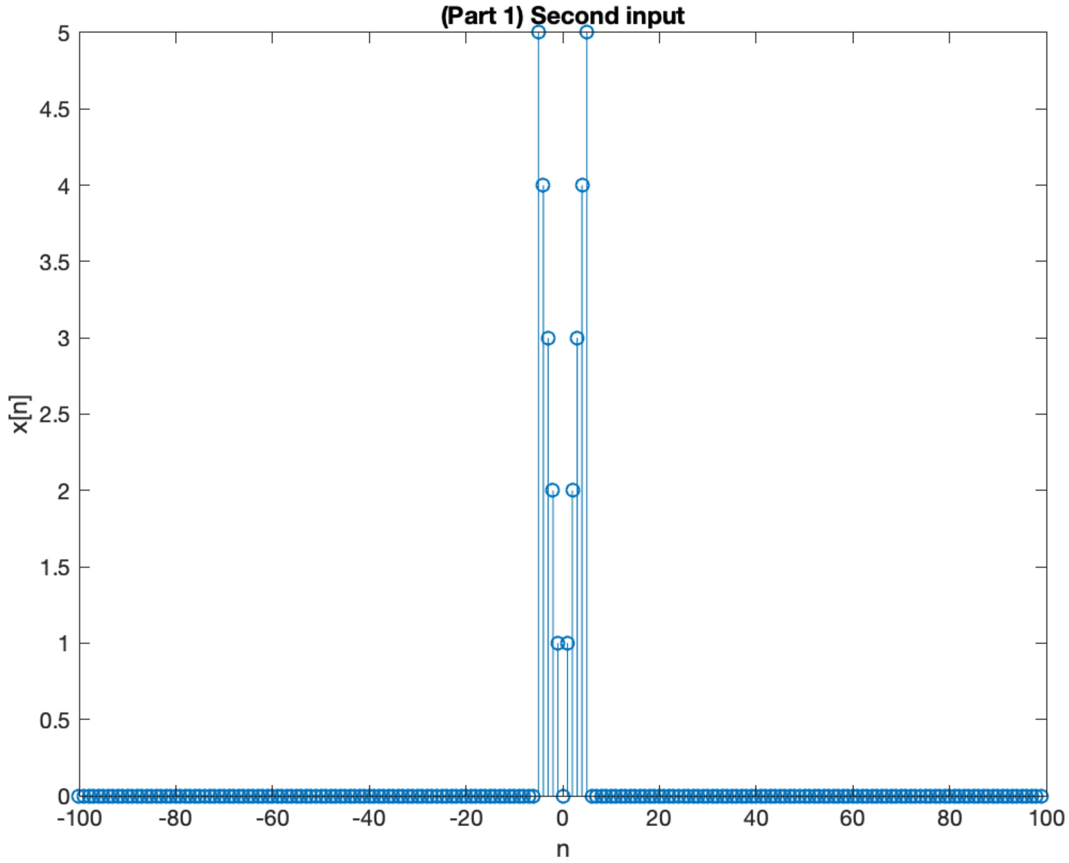

Figure 2. Second input in Part 1

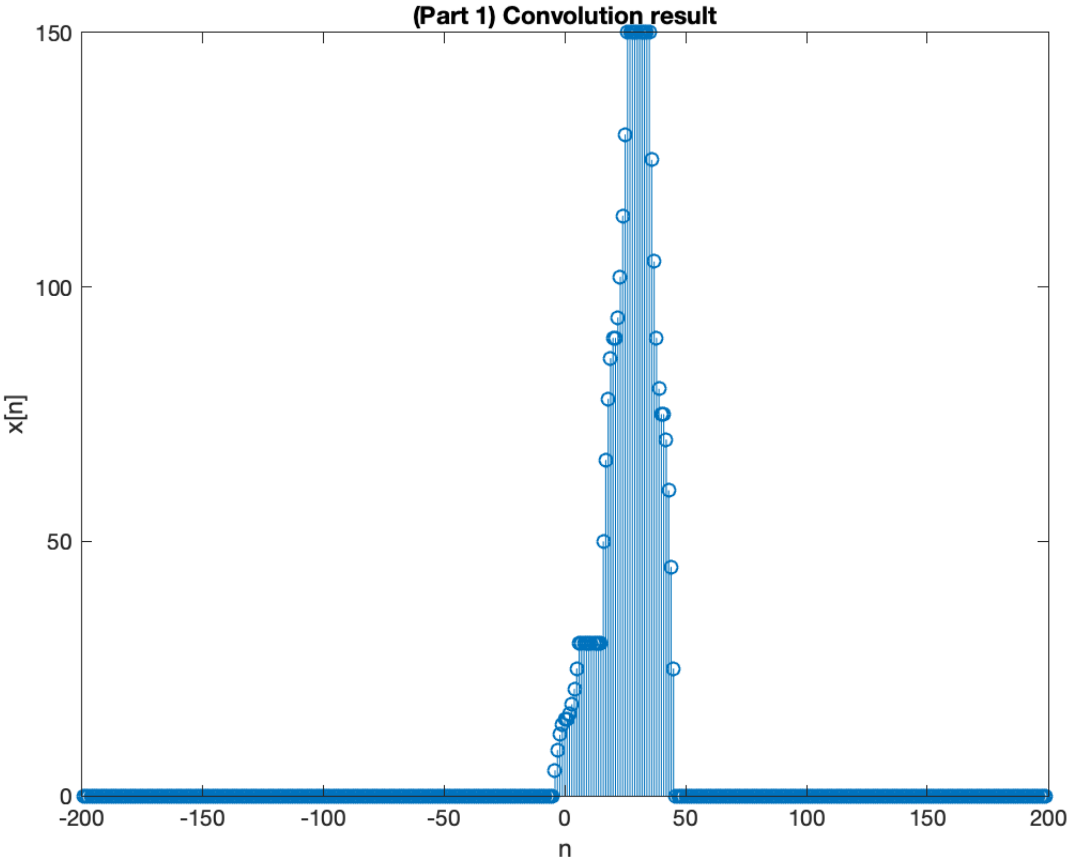

Figure 3. The result of convolution in Part 1

2)

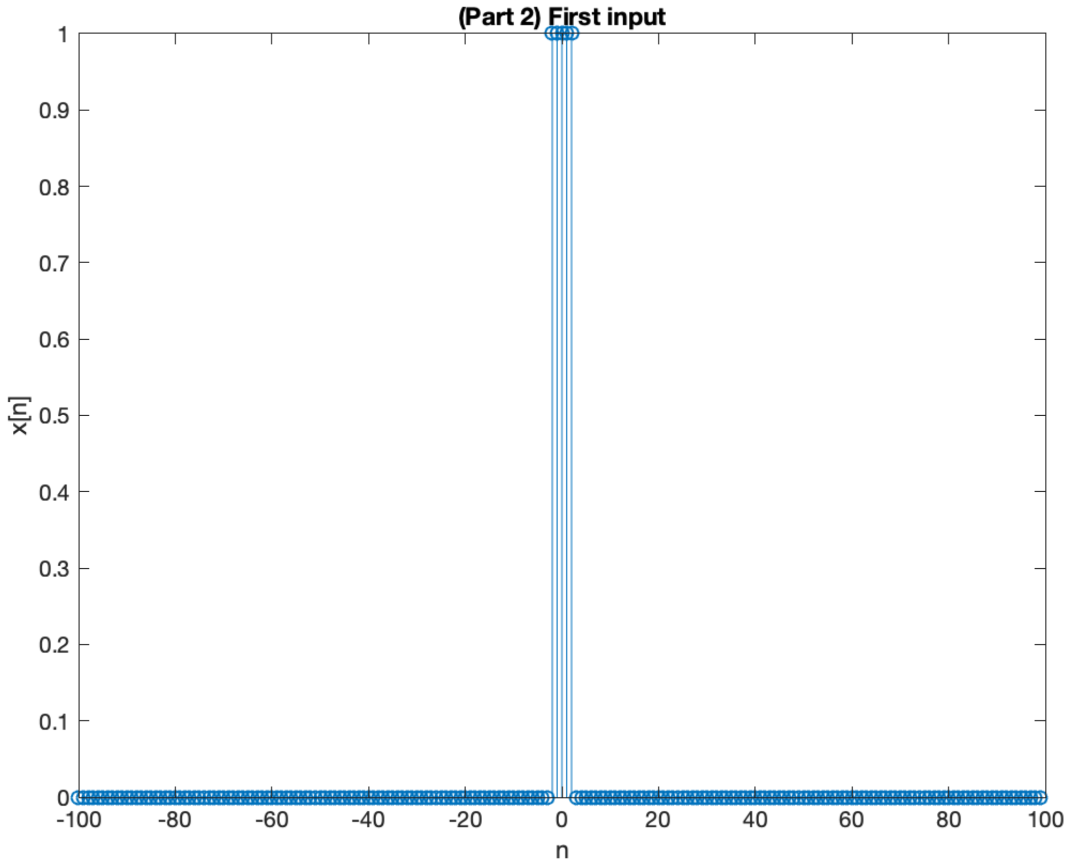

Figure 4. First input in Part 2

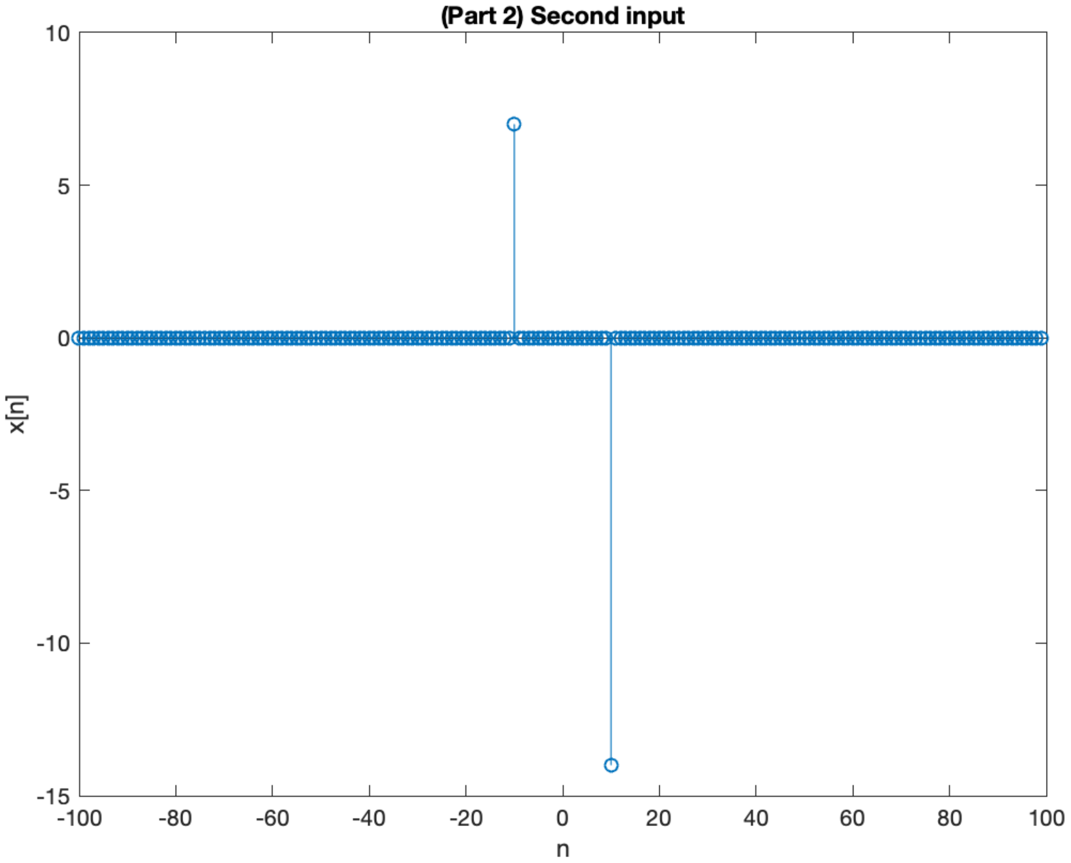

Figure 5. Second input in Part 2

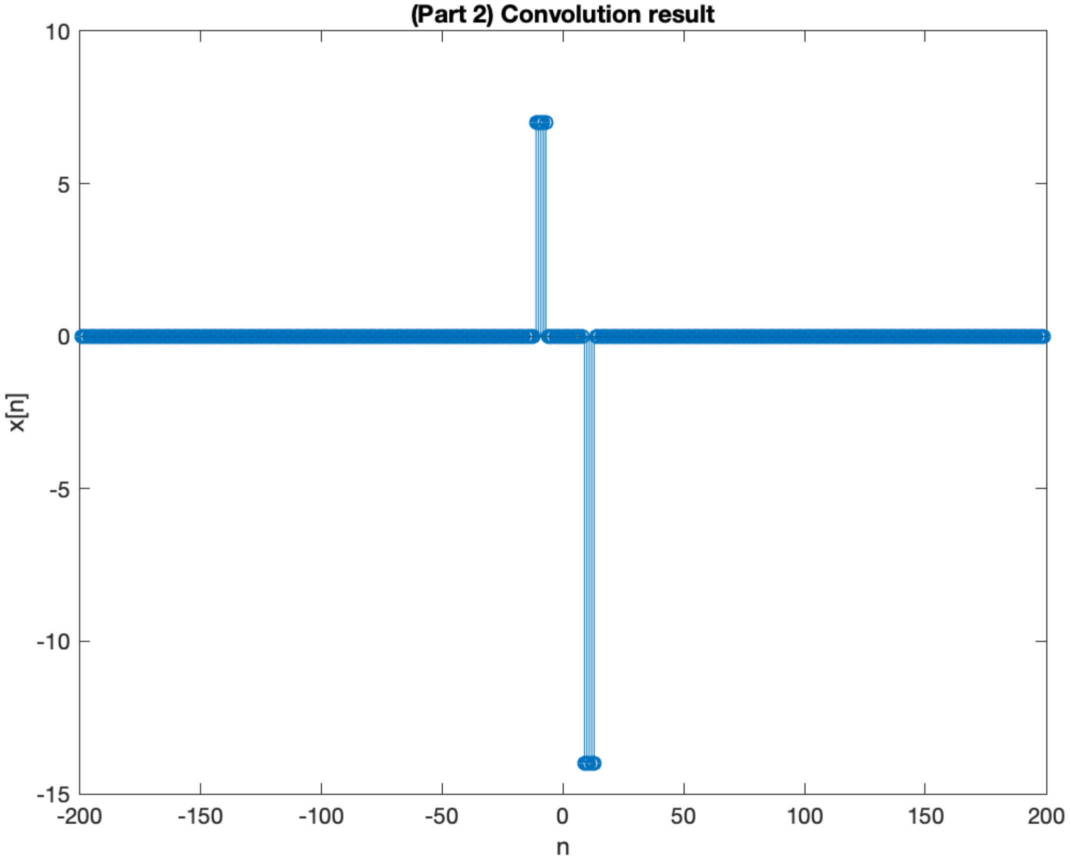

Figure 6. Result of convolution in Part 2

3)

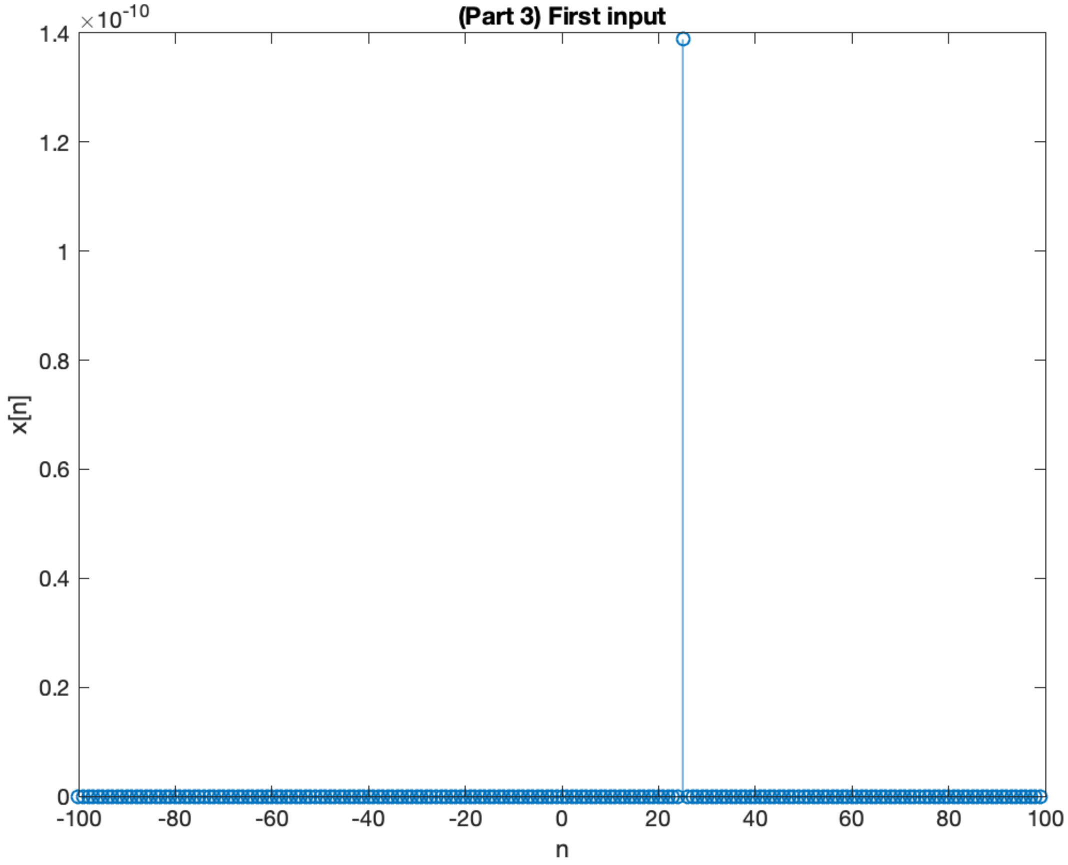

Figure 7. First input in Part 3

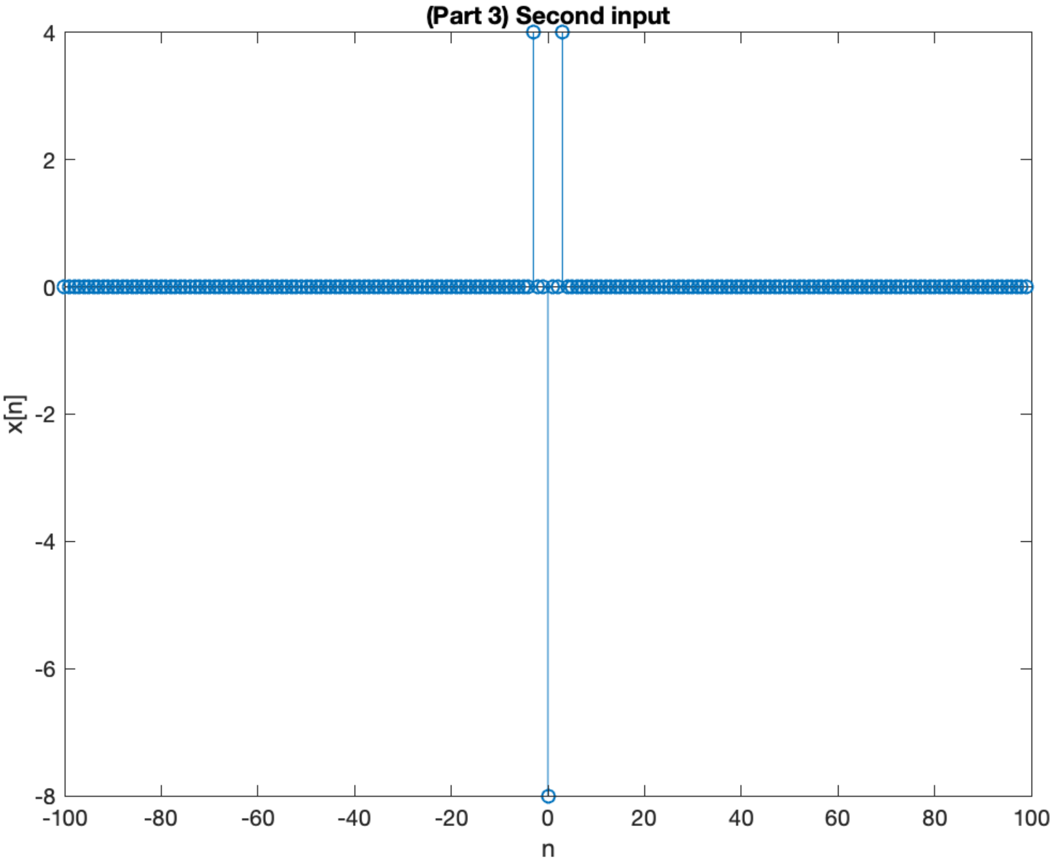

Figure 8. Second input in Part 3

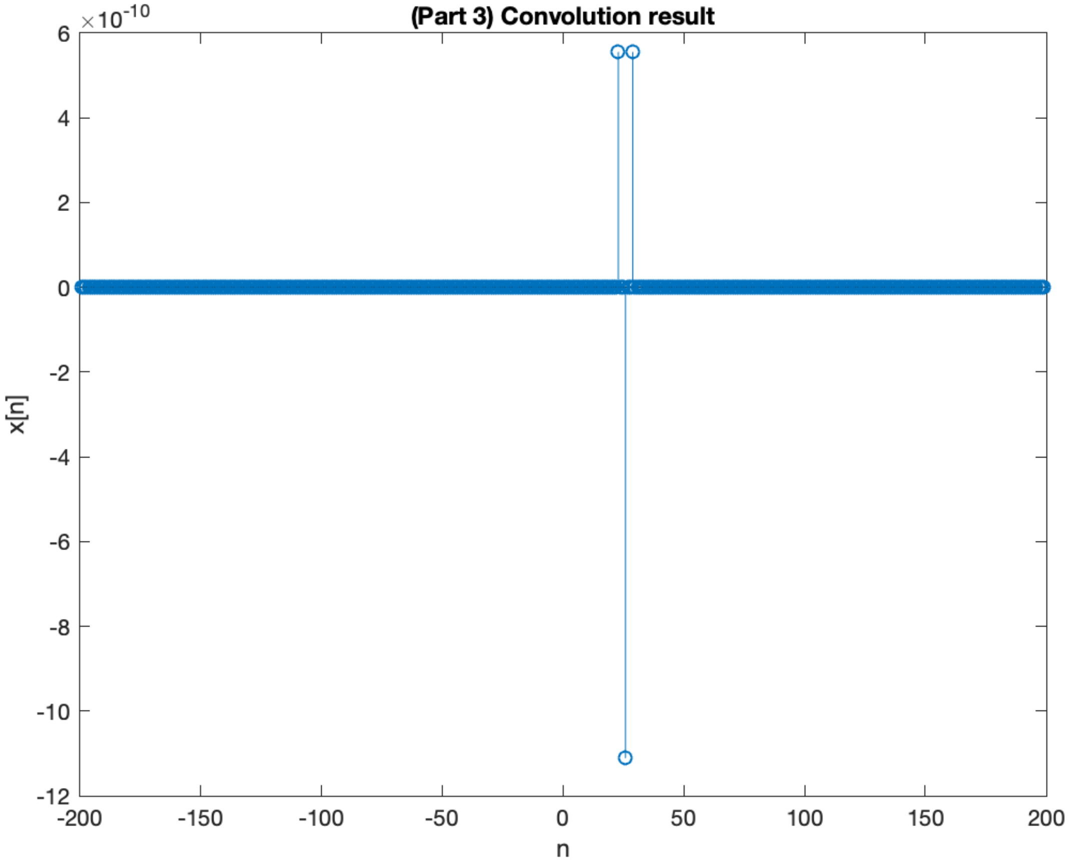

Figure 9. Result of convolution in Part 3

MATLAB code:

```
%{
    Basics of Signals and Systems (Fall 2019-2020)
    Computer Assignment 2

    @author: Fuad Aghazada
    @id: 21503691
    @date: 07.12.2019
%}

% Limit
global IN_LIMIT;
IN_LIMIT = 100;

% --- Part 1 ---
[x1, x2] = generate_part1_inputs();

my_result = disc_conv1d(x1,x2);
check_result = conv(x1, x2);

plot_results(x1, x2, my_result, "Part 1");
figure
% plot_results(x1, x2, check_result, "Part 1 (with conv)");

% --- Part 2 ---
[x1, x2] = generate_part2_inputs();

my_result = disc_conv1d(x1,x2);
check_result = conv(x1, x2);

plot_results(x1, x2, my_result, "Part 2");
figure
% plot_results(x1, x2, check_result, "Part 2 (with conv)");

% --- Part 3 ---
[x1, x2] = generate_part3_inputs();

my_result = disc_conv1d(x1,x2);
check_result = conv(x1, x2);

plot_results(x1, x2, my_result, "Part 3");
% figure
% plot_results(x1, x2, check_result, "Part 3 (with conv)");



%
% 1-D Discrete convolution sum function
%
% @param (vector) x - first sequence for convolving
% @param (vecotr) h - second sequence for convolving
%
% @return (vector) y - convolved sequence of the two input
%                             discrete sequences
%
```

```
function [y] = disc_conv1d(x, h)

    % Constants / Limits
    max_in_index = length(x);
    max_out_index = 2 * max_in_index;

    % Start and end indices for final vector
    start_index = 1;
    end_index = max_out_index;

    % Empty result array
    y = zeros(1, max_out_index);

    % Sliding the vectors across each other:
    % Using the discrete sum formula:
    % sum x[k] * h[n - k] for k and n
    for n = -max_out_index:max_out_index

        sum = 0;
        intersect = false;

        for k = 1:max_in_index
            if n - k >= 1 && n - k <= max_in_index
                % Getting the corresponding sequence values
                x_k = x(k);
                h_k = h(n - k);

                % Taking the sum
                sum = sum + x_k * h_k;
                intersect = true;

                % Setting start and end indices for slicing final vector
                if start_index == 1
                    start_index = n;
                end
                end_index = n;
            end
        end

        % Taking the only intersected values
        if intersect == true
            y(n) = sum;
        end
    end

    % Slicing the convolved vector
    y = y(start_index: end_index);
end


%
% Generates the input sequences in Part 1
%
function [x1, x2] = generate_part1_inputs()
    global IN_LIMIT;
    % Initializing the empty inputs
    x1 = zeros(1, 2 * IN_LIMIT);
    x2 = zeros(1, 2 * IN_LIMIT);

    % x1 - first input
    for n = -IN_LIMIT:IN_LIMIT - 1
        if n >= 0 && n <= 19
            x1(n + IN_LIMIT + 1) = 1;
        elseif n >= 20 && n <= 39
            x1(n + IN_LIMIT + 1) = 5;
        end
    end

    % x2 - second input
    for n = -IN_LIMIT:IN_LIMIT - 1
        if n >= -5 && n <= 5
            x2(n + IN_LIMIT + 1) = abs(n);
        else
            x2(n + IN_LIMIT + 1) = 0;
        end
    end
end


%
% Generates the input sequences in Part 2
%
function [x1, x2] = generate_part2_inputs()
    global IN_LIMIT;

    % Initializing the empty inputs
    x1 = zeros(1, 2 * IN_LIMIT);
    x2 = zeros(1, 2 * IN_LIMIT);

    % x1 - first input
    for n = -IN_LIMIT:IN_LIMIT - 1
        x1(n + IN_LIMIT + 1) = unit_step(-2 * n + 4) - unit_step(-n - 3);
    end

    % x2 - second input
    for n = -IN_LIMIT:IN_LIMIT - 1
        x2(n + IN_LIMIT + 1) = 7 * unit_impulse(-n - 10) - 14 *
unit_impulse(-n + 10);
    end
end


%
% Generates the input sequences in Part 2
%
function [x1, x2] = generate_part3_inputs()
    global IN_LIMIT;

    % Initializing the empty inputs
    x1 = zeros(1, 2 * IN_LIMIT);
    x2 = zeros(1, 2 * IN_LIMIT);

    % x1 - first input
    for n = -IN_LIMIT:IN_LIMIT - 1
        if n <= 25 && n >= 25
            x1(n + IN_LIMIT + 1) = 10 * exp(-abs(n));
        end
    end

    % x2 - second input
    for n = -IN_LIMIT:IN_LIMIT - 1
        x2(n + IN_LIMIT + 1) = 4 * unit_impulse(n + 3) - 8 * unit_impulse(n)
+ 4 * unit_impulse(n - 3);
    end
end


%
% Unit impulse input function
%
function [value] = unit_impulse(n)
    if n == 0
        value = 1;
    else
        value = 0;
    end
end


%
% Unit step input function
%
function [value] = unit_step(n)
    if n < 0
        value = 0;
    else
        value = 1;
    end
end


%
% Plotting the results
%
function plot_results(x1, x2, res, plot_name)
    global IN_LIMIT;

    n = -IN_LIMIT:1:IN_LIMIT - 1;

    % First input
    stem(n, x1)
    title(sprintf("(%s) First input", plot_name))
    xlabel("n")

    figure
    % Second input
    stem(n, x2)
    title(sprintf("(%s) Second input", plot_name))
    xlabel("n")

    n = -2 * IN_LIMIT + 1:1:2 * IN_LIMIT - 1;

    figure

    % Convolution result
    stem(n, res)
    title(sprintf("(%s) Convolution result", plot_name))
    xlabel("n")
end
```