**For the octave online**

**Consider a discrete linear system for which we know its impulse response g(n) (the so-called convolution kernel). In the case where we feed a known sequence u(n) to the input of this system, we can calculate the response of the system y(n) using the convolution sum.**

**Convolution sum:**

**y(n)=∑∞k=−∞u(k)g(n−k)=u(k)∗g(k)**

**y(n)=∑N+M−1n=1u(k)g(n−k+1)**

**Index k is from 1 to (N+M-1), where N is the length of the impulse response, M is the length of the input sequence.**

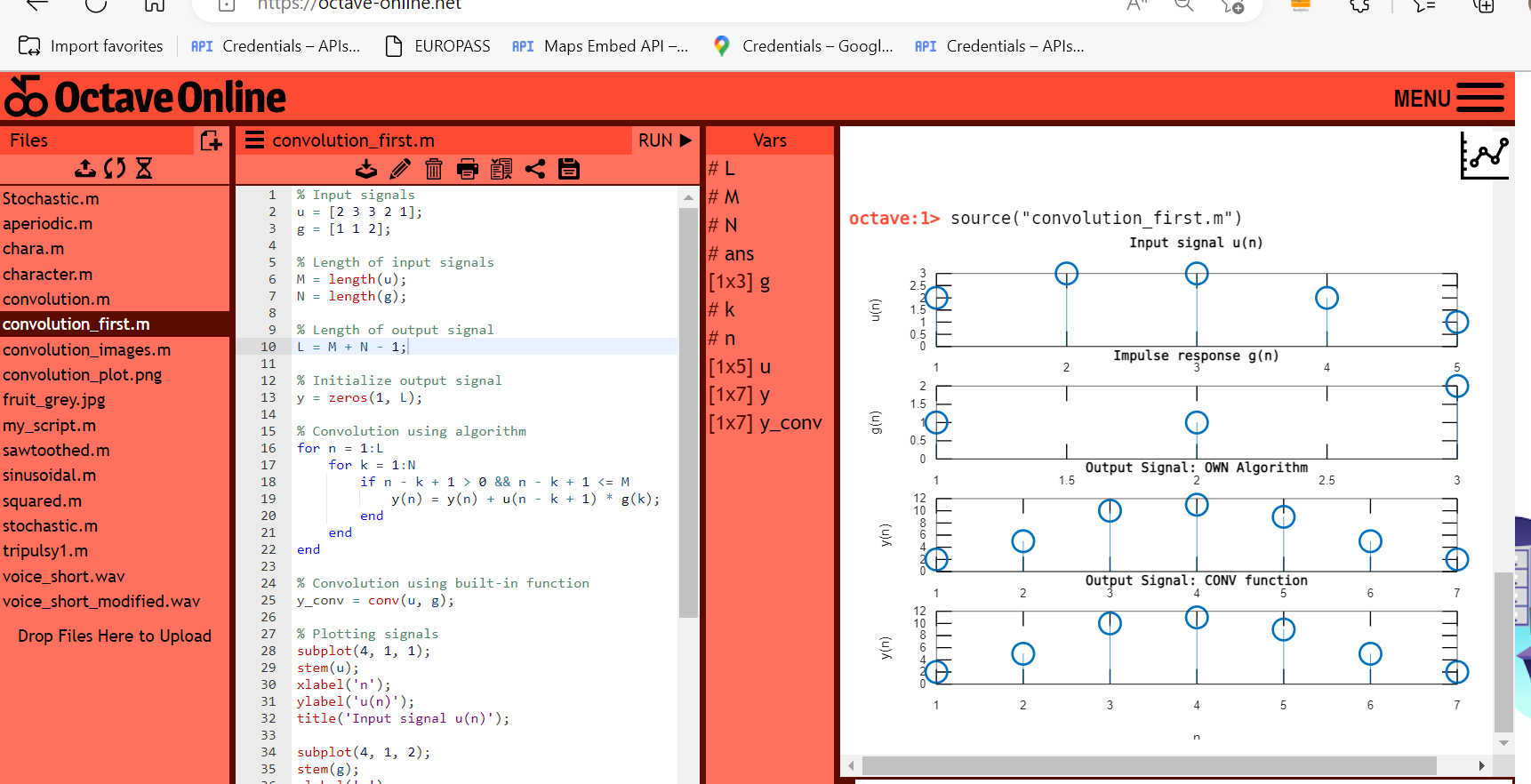
**Create your own algorithm for calculating discrete linear convolution. Verify your algorithm and compare it with the built-in function (in Matlab, the "conv" function) for this case:**

**u(n) = [2 3 3 2 1]**

**g(n) = [1 1 2]**

**Display the calculated system response y(n) as well as the signals u(n) and g(n) in one window (in Matlab using the "subplot" function):**

**Screenshots:**



**Code in Octave online:**

% Input signals

u = [2 3 3 2 1];

g = [1 1 2];

% Length of input signals

M = length(u);

N = length(g);

% Length of output signal

L = M + N - 1;

% Initialize output signal

y = zeros(1, L);

% Convolution using algorithm

for n = 1:L

for k = 1:N

if n - k + 1 > 0 && n - k + 1 <= M

y(n) = y(n) + u(n - k + 1) \* g(k);

end

end

end

% Convolution using built-in function

y\_conv = conv(u, g);

% Plotting signals

subplot(4, 1, 1);

stem(u);

xlabel('n');

ylabel('u(n)');

title('Input signal u(n)');

subplot(4, 1, 2);

stem(g);

xlabel('n');

ylabel('g(n)');

title('Impulse response g(n)');

subplot(4, 1, 3);

stem(y);

xlabel('n');

ylabel('y(n)');

title('Output Signal: OWN Algorithm');

xticks(1:L); % Set x-axis ticks to cover the entire length of y

subplot(4, 1, 4);

stem(y\_conv);

xlabel('n');

ylabel('y(n)');

title('Output Signal: CONV function');

xticks(1:L); % Set x-axis ticks to cover the entire length of y

**Comment:**

This Octave code demonstrates how to perform convolution of two signals using both an algorithmic approach and a built-in function. The input signals "u" and "g" are defined, and their lengths are calculated. The output signal "y" is initialized as an array of zeros with a length determined by the lengths of "u" and "g". The convolution operation is then performed using a nested loop that iterates over the elements of "u" and "g" to compute the convolution sum and store the results in "y". Additionally, the built-in "conv" function is used to perform the convolution and store the results in "y\_conv". The code also includes plotting of the input signals, the output signals obtained from the algorithmic approach and the built-in function, with appropriate labels and titles. This code provides a visual comparison of the output signals obtained from the two different methods of convolution.