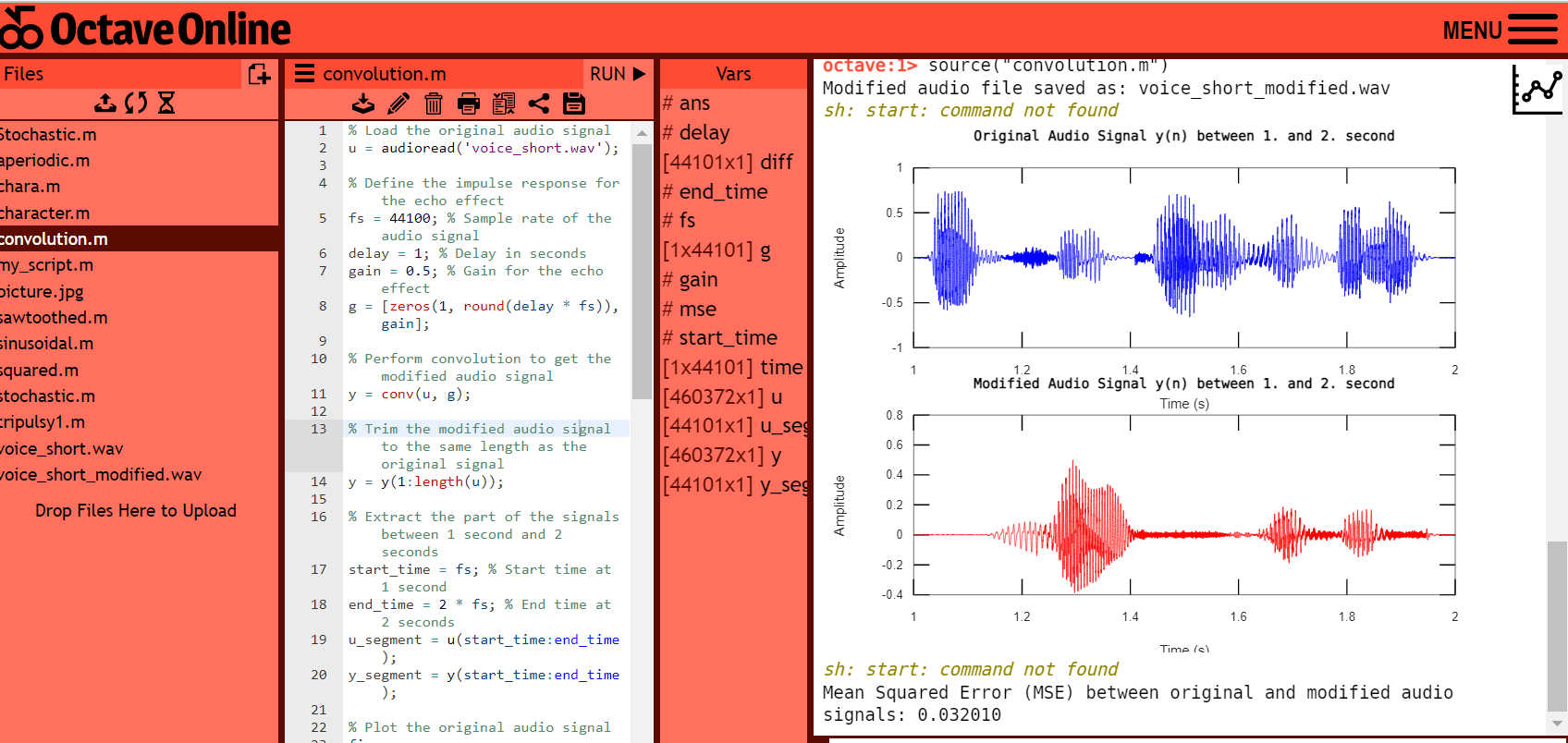
**1) We have an audio signal u(n) ("voice\_short.wav") that we want to modify to achieve an echo effect that starts after 1 second of recording and has half the intensity (volume) of the original.**

**Using the convolution (in Matlab, the conv function) of the original audio signal u(n) with a suitably designed impulse response (filter) g(n), we calculate the modified signal y(n). Plot the original and modified audio signal (plot only the part of the signal between the first and second second second) and listen to both signals (in Matlab, the "sound" function, or the "audioplayer" and "play" functions). Evaluate the difference between the original audio signal u(n) and the modified audio signal y(n).**

**y(n)=u(k)∗g(k)**

**Picture of the Solution in Octave Online**



**Code for octave online:**

% Load the original audio signal

u = audioread('voice\_short.wav');

% Define the impulse response for the echo effect

fs = 44100; % Sample rate of the audio signal

delay = 1; % Delay in seconds

gain = 0.5; % Gain for the echo effect

g = [zeros(1, round(delay \* fs)), gain];

% Perform convolution to get the modified audio signal

y = conv(u, g);

% Trim the modified audio signal to the same length as the original signal

y = y(1:length(u));

% Extract the part of the signals between 1 second and 2 seconds

start\_time = fs; % Start time at 1 second

end\_time = 2 \* fs; % End time at 2 seconds

u\_segment = u(start\_time:end\_time);

y\_segment = y(start\_time:end\_time);

% Plot the original audio signal

figure;

subplot(2, 1, 1);

time = linspace(start\_time/fs, end\_time/fs, length(u\_segment)); % Update x-axis limits

plot(time, u\_segment, 'b');

xlabel('Time (s)');

ylabel('Amplitude');

title('Original Audio Signal y(n) between 1. and 2. second');

% Plot the modified audio signal

subplot(2, 1, 2);

time = linspace(start\_time/fs, end\_time/fs, length(y\_segment)); % Update x-axis limits

plot(time, y\_segment, 'r');

xlabel('Time (s)');

ylabel('Amplitude');

title('Modified Audio Signal y(n) between 1. and 2. second');

% Save the modified audio signal to a new audio file

audiowrite('voice\_short\_modified.wav', y, fs);

% Print a message with the file location

fprintf('Modified audio file saved as: voice\_short\_modified.wav\n');

% Play the original and modified audio signals using the default media player on your system

system('start voice\_short.wav'); % Play the original audio signal

pause(length(u\_segment)/fs + 1); % Wait for the original signal to finish playing

system('start voice\_short\_modified.wav'); % Play the modified audio signal

% Evaluate the difference between the original and modified audio signals

diff = u\_segment - y\_segment;

mse = mean(diff.^2);

fprintf('Mean Squared Error (MSE) between original and modified audio signals: %f\n', mse);

**Comment:**

This Octave code demonstrates how to apply an echo effect to an audio signal using convolution. The original audio signal is loaded from a WAV file using the "audioread" function. The impulse response for the echo effect is defined by specifying the delay (in seconds) and the gain. The "conv" function is then used to perform convolution between the original audio signal and the impulse response, resulting in the modified audio signal "y".The modified audio signal is then trimmed to the same length as the original signal using array indexing. A segment of the original and modified audio signals, between 1 and 2 seconds, is plotted using the "plot" function, with appropriate labels and titles.

The modified audio signal is saved to a new WAV file using the "audiowrite" function, and a message with the file location is printed. The original and modified audio signals are played using the default media player on the system using the "system" function.Finally, the difference between the original and modified audio signals is evaluated by calculating the Mean Squared Error (MSE) between the two signals, and the result is printed using the "fprintf" function. The MSE is a measure of the similarity between the original and modified audio signals, with a lower value indicating a closer match.

**2) For the image signal (fruit\_grey.jpg), modify the image by using 2D convolution ("conv2" function in Matlab) and the impulse responses (kernels or masks) listed below - blurring and edge highlighting. In Matlab, use the "imread" and "imshow" functions to load and display images.**

**To accentuate the edges, try these impulse responses:**

**h1 = [1 0 -1; 0 0 0; -1 0 1];**

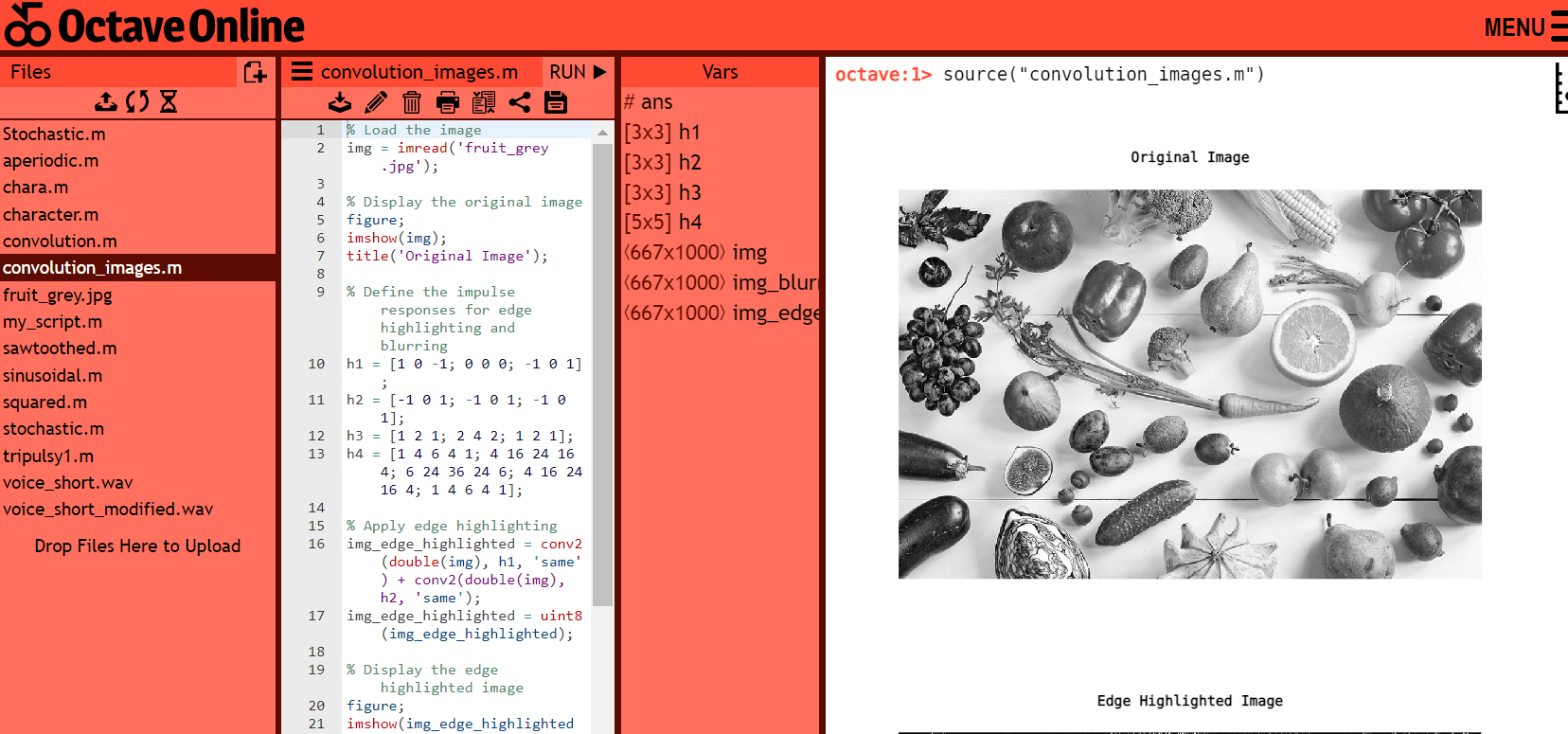
**h2 = h = [-1 0 1; -1 0 1; -1 0 1];**

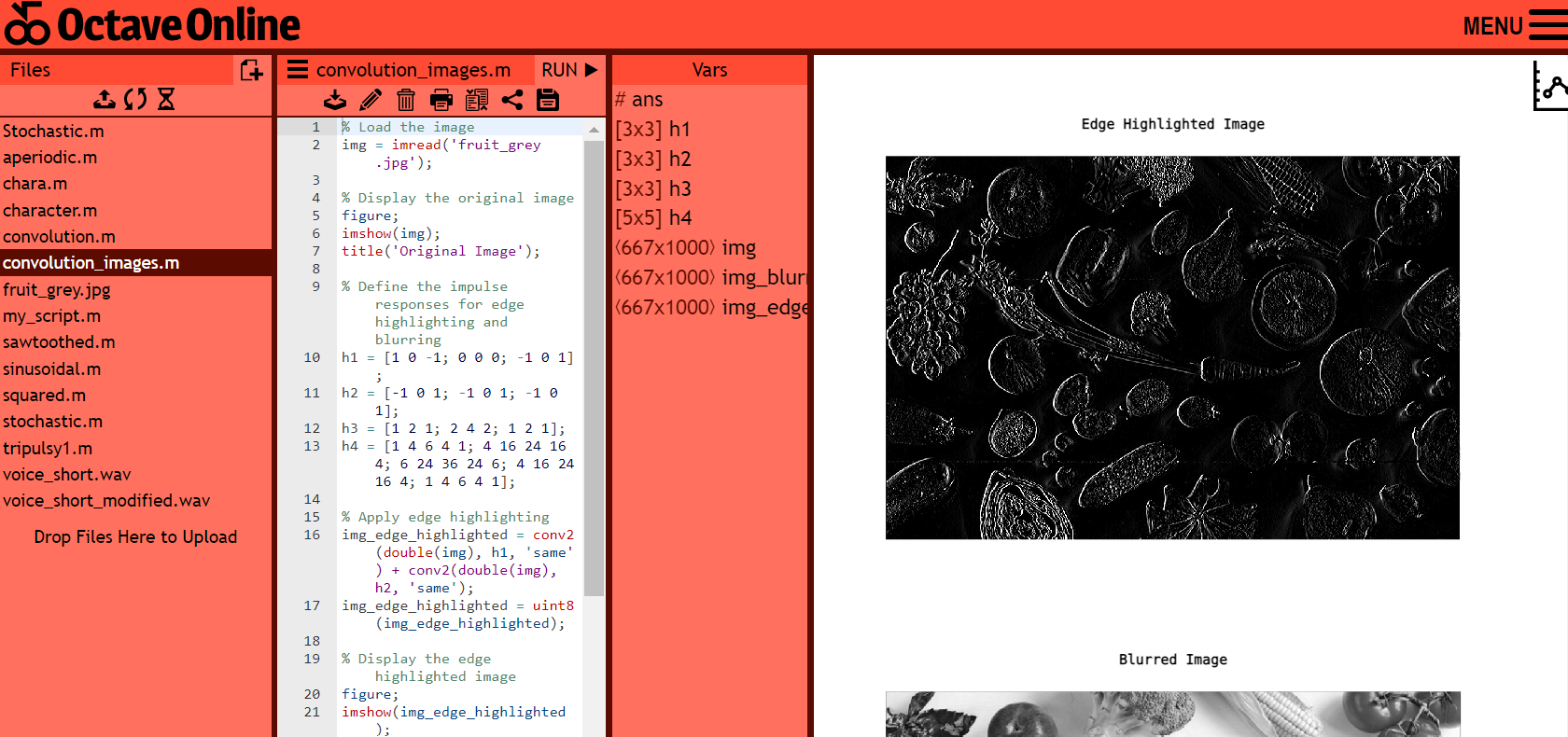
**To blur image, try these impulse responses:**

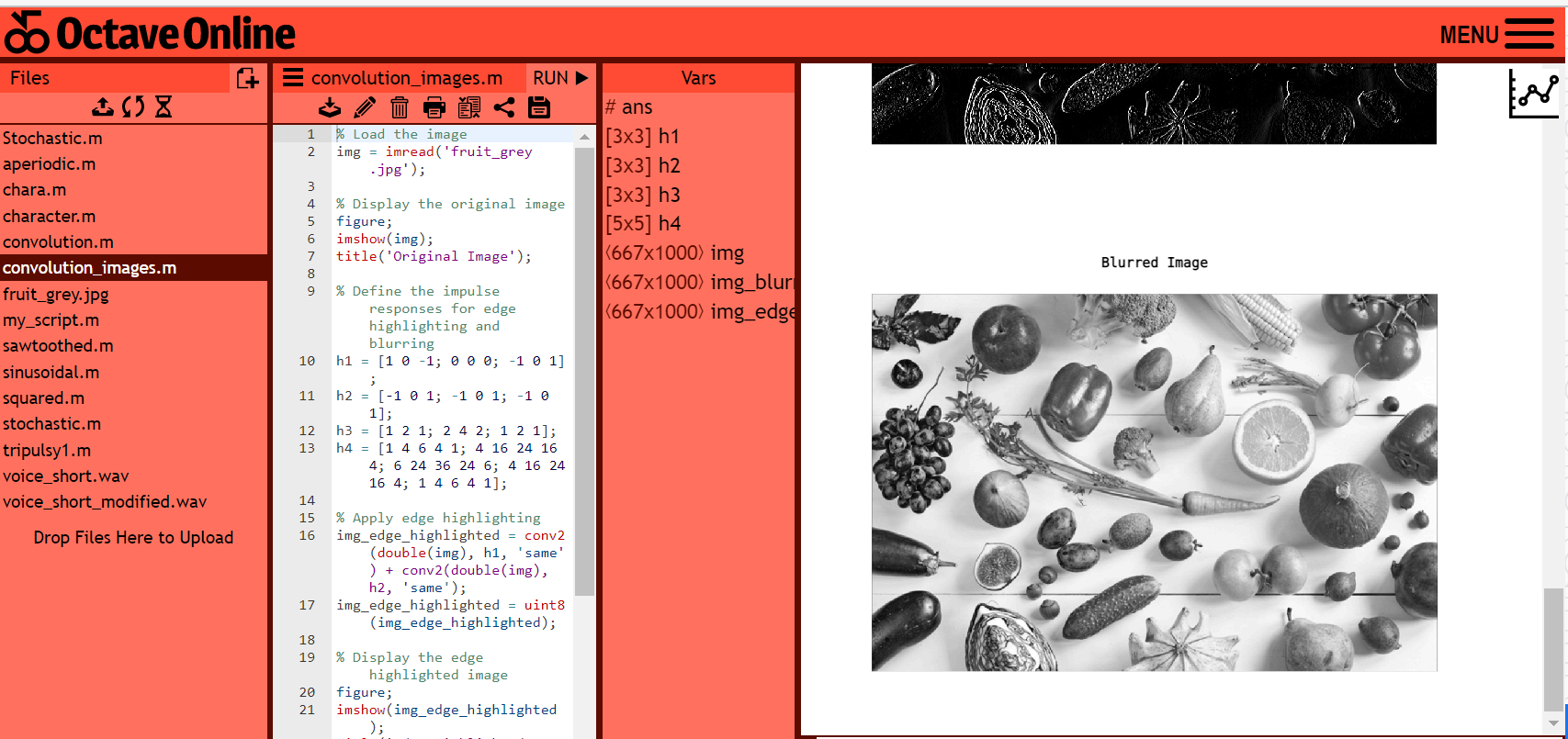
**h3 = [1 2 1; 2 4 2; 1 2 1];**

**h4= [1 4 6 4 1; 4 16 24 16 4; 6 24 36 24 6; 4 16 24 16 4; 1 4 6 4 1];**

**Pictures of the solution in the ocatave online:**







**Code for octave online:**

% Load the image

img = imread('fruit\_grey.jpg');

% Display the original image

figure;

imshow(img);

title('Original Image');

% Define the impulse responses for edge highlighting and blurring

h1 = [1 0 -1; 0 0 0; -1 0 1];

h2 = [-1 0 1; -1 0 1; -1 0 1];

h3 = [1 2 1; 2 4 2; 1 2 1];

h4 = [1 4 6 4 1; 4 16 24 16 4; 6 24 36 24 6; 4 16 24 16 4; 1 4 6 4 1];

% Apply edge highlighting

img\_edge\_highlighted = conv2(double(img), h1, 'same') + conv2(double(img), h2, 'same');

img\_edge\_highlighted = uint8(img\_edge\_highlighted);

% Display the edge highlighted image

figure;

imshow(img\_edge\_highlighted);

title('Edge Highlighted Image');

% Apply blurring

img\_blurred = conv2(double(img), h3, 'same') / sum(h3(:)); % Normalizing by sum of impulse response for blurring

img\_blurred = uint8(img\_blurred);

% Display the blurred image

figure;

imshow(img\_blurred);

title('Blurred Image');

Comment:

This Octave code demonstrates how to apply edge highlighting and blurring to an image using convolution. The original image is loaded from a JPEG file using the "imread" function and displayed using the "imshow" function with an appropriate title.Impulse responses for edge highlighting and blurring are defined as matrices "h1", "h2", "h3", and "h4". These matrices represent the convolution kernels that will be applied to the image. Edge highlighting is applied by using the "conv2" function to convolve the original image with "h1" and "h2" kernels separately, and then adding the results together to obtain the edge highlighted image. The "double" function is used to convert the image to double precision for convolution, and the "uint8" function is used to convert the resulting image back to unsigned 8-bit integer format for display. The edge highlighted image is displayed using the "imshow" function with an appropriate title.

Blurring is applied by using the "conv2" function to convolve the original image with the "h3" kernel, and then normalizing the result by dividing by the sum of all elements in the "h3" kernel. The resulting image is converted back to unsigned 8-bit integer format using the "uint8" function. The blurred image is displayed using the "imshow" function with an appropriate title.