# BMET4790:BIOMECHATRONICS

# TUTORIAL: Sensory Substitution Processing

Name: Hunter Tambimuttu SID: 510448271

Name: Caleb Nazareth SID: 530372811

Name: Gabriel Ralph SID: 470205736

Note: Activity needs to be completed and submitted before the start of the next tutorial

THIS TUTORIAL SHOULD TAKE A TOTAL OF 4 HOURS TO COMPLETE

Notes:

* Answer questions briefly and clearly within the blocks provided
* A signed plagiarism coversheet must be included with the submission
* A soft copy (including MATLAB listings) must be submitted through Canvas
* Tutorials submitted after the due date will lose 5% per day (or part thereof) unless accompanied by a valid doctor’s certificate.
* Use this document as a template for your submission

**Background**

The vOICe system converts an image to a sound file that can be interpreted by a blind person. It substitutes an acoustic image for a visual one using the process illustrated in Figure 1.

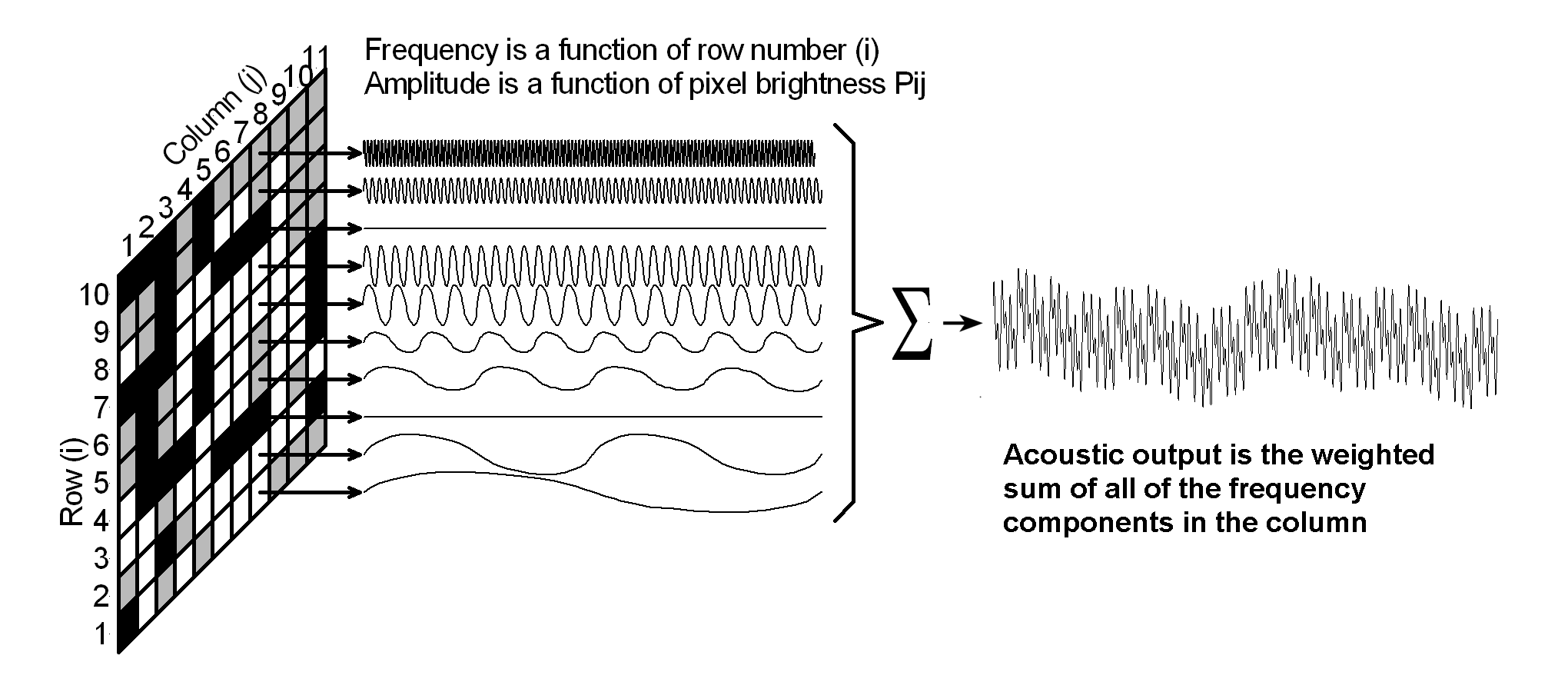


Figure 1: Sensory substitution from video to audio

* The resolution of the image is reduced significantly, typically to 8×8 pixels
* The number of grayscale shades is reduced to 3 or 4
* A composite sound signal is produced for each column by summing the sound components
  + The base frequency of each sound is scaled by the row number
  + The amplitude of each sound is weighed by the shade value of each pixel
  + The overall amplitude of the signal in each ear is weighted by the angle off the midline to produce a stereo effect
* The composite sound file for each column is played in turn to build up a complete sound “image”

**Tutorial**

Copy the greyscale bitmap file, **face.bmp,** from the Biomechatronics site on WebCT

Use the following MATLAB code to read the file

**[z]=imread('face2','bmp');**

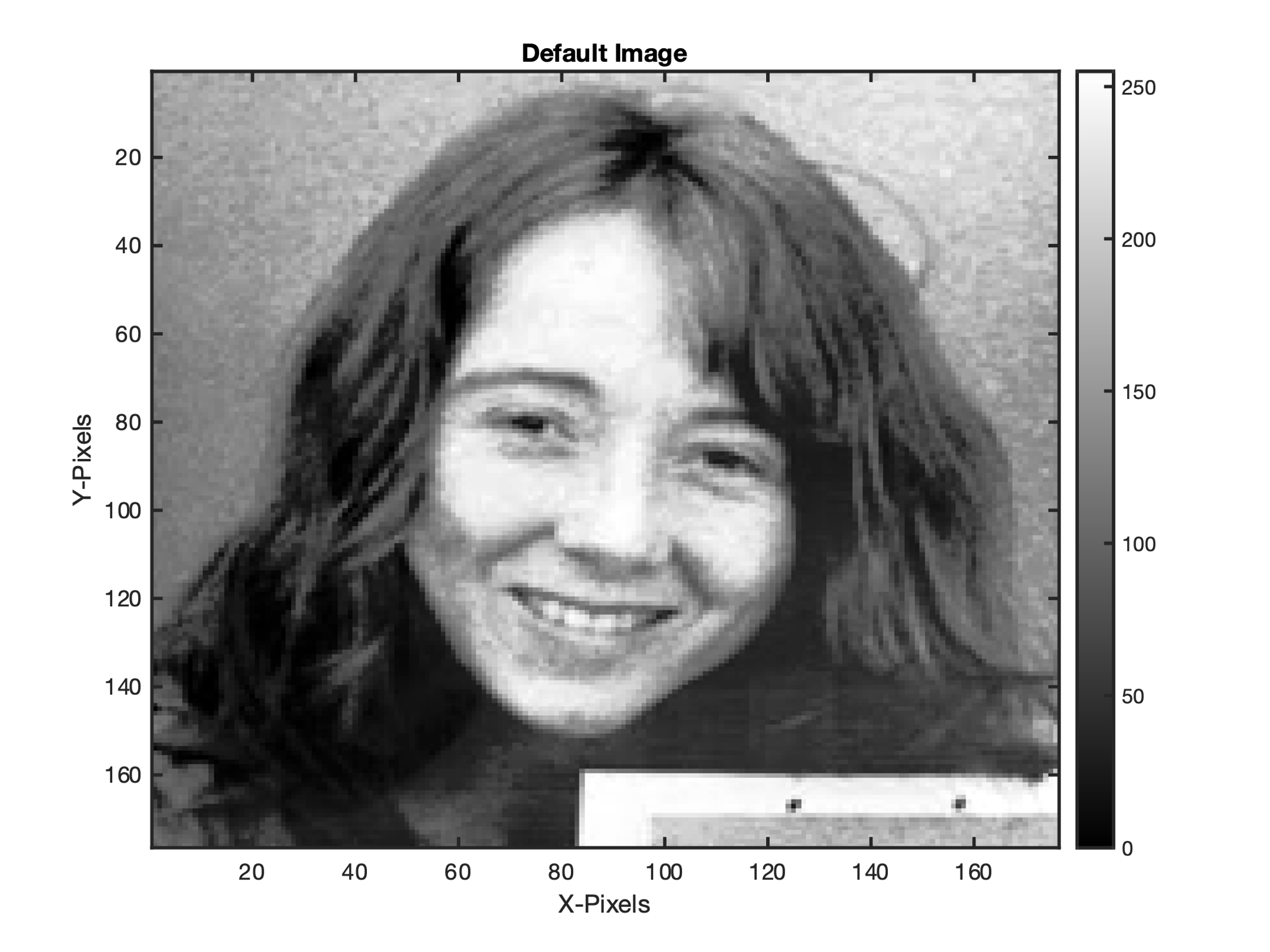
**zf=double(z); % convert the image data to floating point**

**Question 1: Plotting the Raw Image**

Note that there are various different ways that MATLAB can be used to generate a 256 shade greyscale image. If you choose not to use the process described below, please justify the alternative. Note that it may affect all of your subsequent answers.

* Use MATLAB help to find out about producing scaled images **IMAGESC**, making sure that the pixels are square using **AXIS** and using the correct colouring using **COLORMAP**
* Plot the raw image (6)
* Generate axes labels using the **XLABEL** and **YLABEL** commands (2)
* Generate a **COLORBAR** (2)
* The result should look something like this





**Question 2: Spatial Filtering**

It is possible to sub-sample the image space to reduce the resolution to 8×8 by displaying 1 pixel in N. However, this loses information. A superior method is to produce a pixel which is the average of all of the surrounding pixels

* Determine the size of the original image using the **SIZE** command. (2)

176x176

* To produce an 8×8 array, how many pixels must be skipped between samples in the x and the y directions? (2)

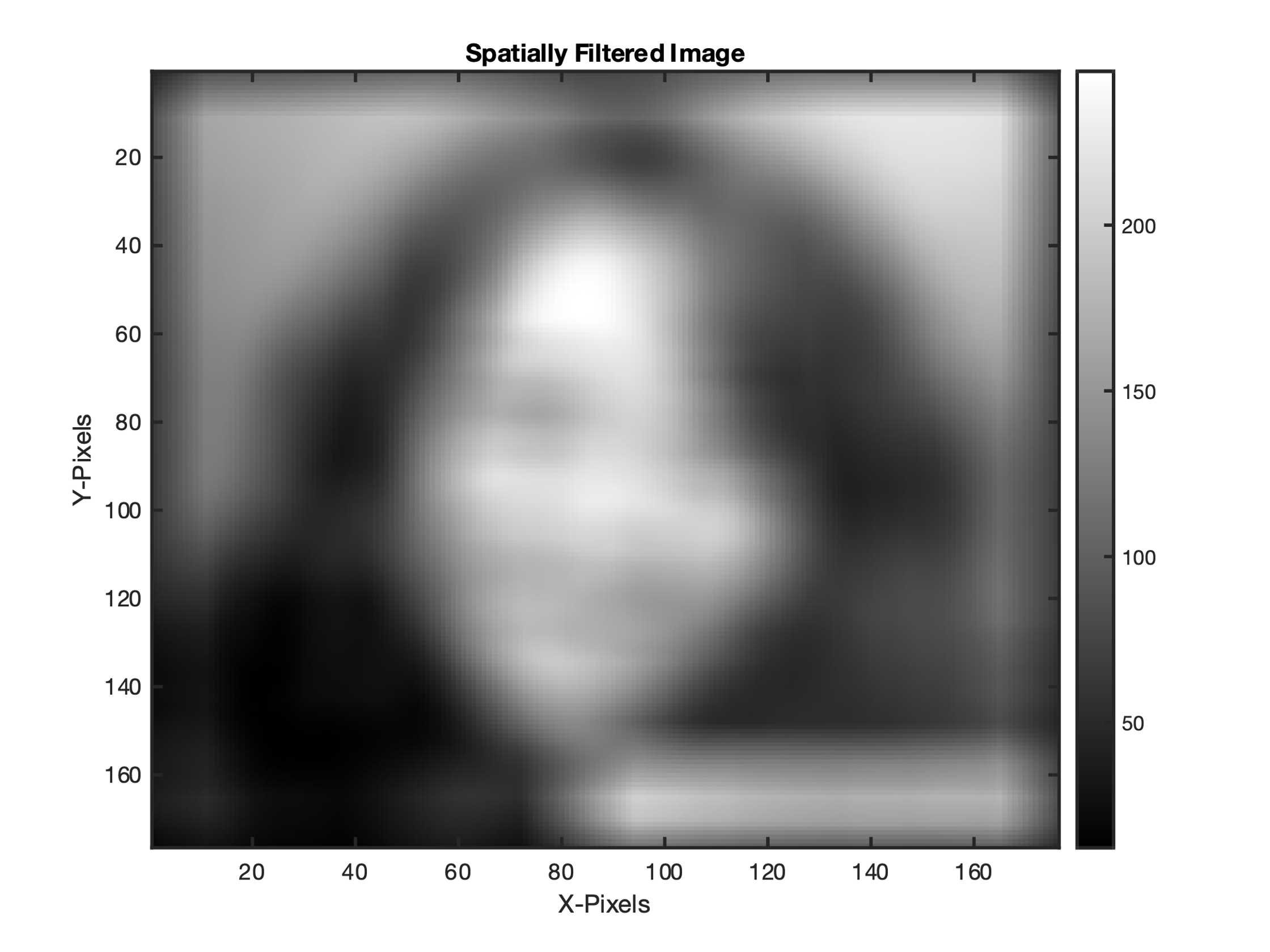
Start at 11 and then take every 22nd from there.

* Use **ONES** to generate the basis of an N×N matrix of the correct size to average all of the cells that will be missed during the sub-sampling process. Read the help file to find out how **FILTER2** uses this “kernel” to generate a 2D low-pass filter. What will the value of each element be to ensure that the average is generated? Explain your answer (4)

The value of the kernel is 1/N where N is the number of pixels in the kernel. For N pixels, the sum of these will add to one, making it the average of the kernel region in the image.

* Use the **FILTER2** command to produce an average image of the same size as the original that performs this averaging process
* Plot the result – it should look like this (9)

****

****

**Question 3: Sub-sampling**

You can use **FOR** loops to sub-sample the image down to 8×8, but MATLAB has good techniques to resample the 2D image array. You should have encountered these when you worked through the introductory MATLAB tutorials

* Sub-sample the image to produce the following 8 × 8 matrix (9)

163.9050 179.6033 182.5847 136.0124 113.1322 191.5331 223.7355 218.2004

149.9215 164.3822 99.4050 148.4112 133.1384 102.5661 172.9607 199.7252

138.7025 108.1818 73.6178 239.6632 196.1529 80.5269 94.9773 173.4525

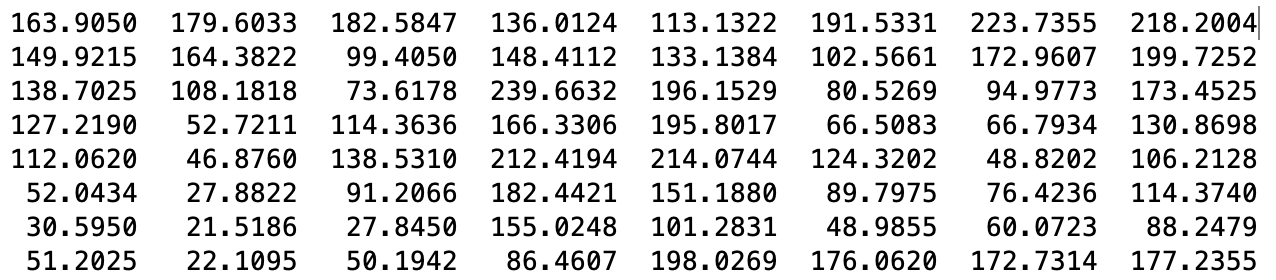
127.2190 52.7211 114.3636 166.3306 195.8017 87.5083 66.7934 130.8698

112.0620 46.8760 138.5310 212.4194 214.0744 124.3202 48.8202 106.2128

52.0434 27.8822 91.2066 182.4421 151.1880 89.7975 76.4236 114.3740

30.5950 21.5186 27.8450 155.0248 101.2831 48.9855 60.0723 88.2479

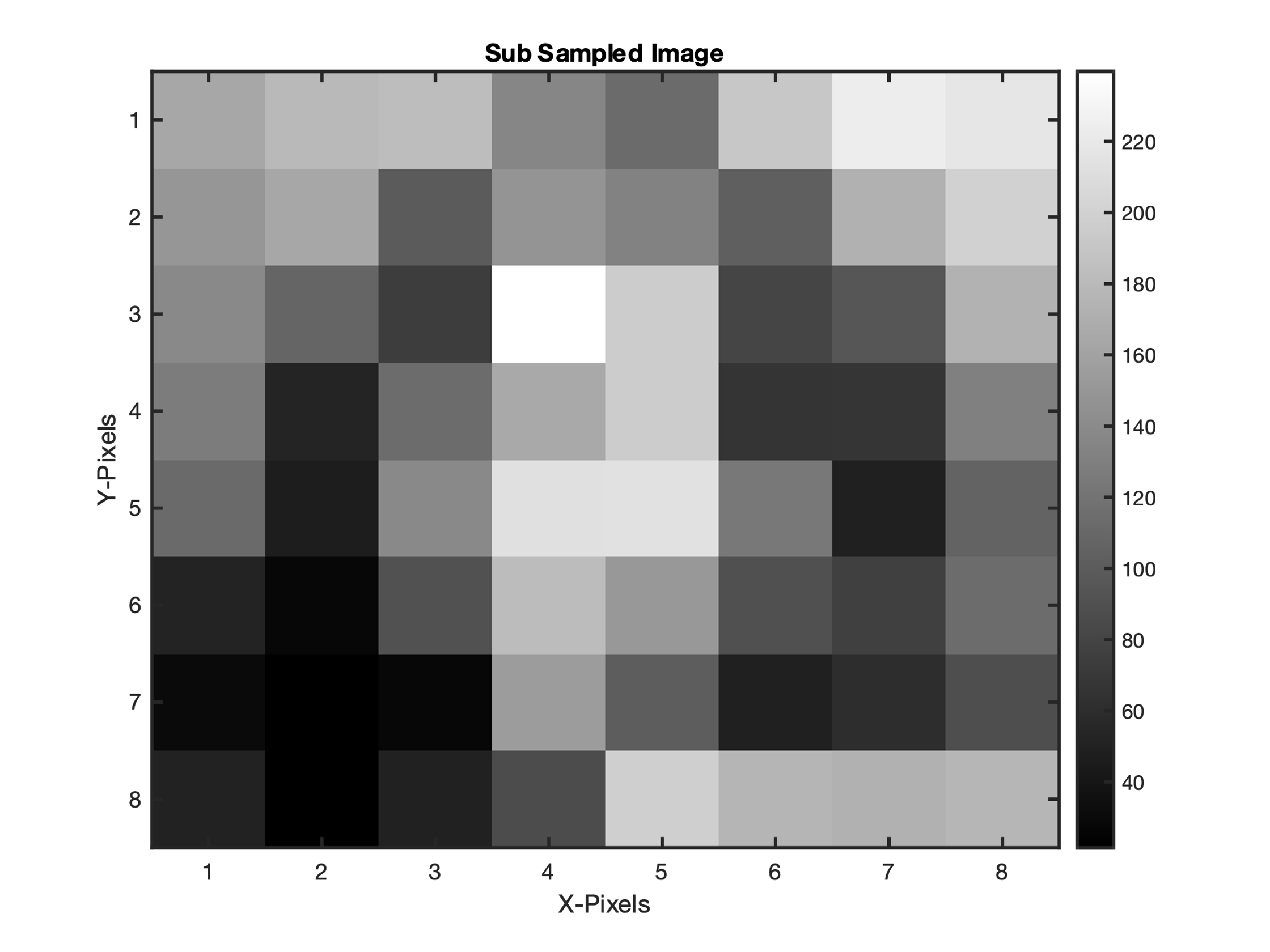
51.2025 22.1095 50.1942 86.4607 198.0269 176.0620 172.7314 177.2355





* Note that there is a deliberate error in one of the numbers. Highlight the correct value in your answer (2)
* Plot the reduced resolution 8 × 8 image. It should look like this (4)





* Use the help to find out about **MAX**, **MIN** and **FLOOR** to scale the image values to produce five quantised gray scale values of 0,1,2,3,4. Justify your algorithm for rounding and include the section of code that you used to perform this quantisation function (6)

min\_val = min(zf3, [], 'all'); % finds the absolute minimum

max\_val = max(zf3, [], 'all'); % finds the absolute maximum

zf4 = floor(4\*(zf3 - min\_val)/(max\_val - min\_val));

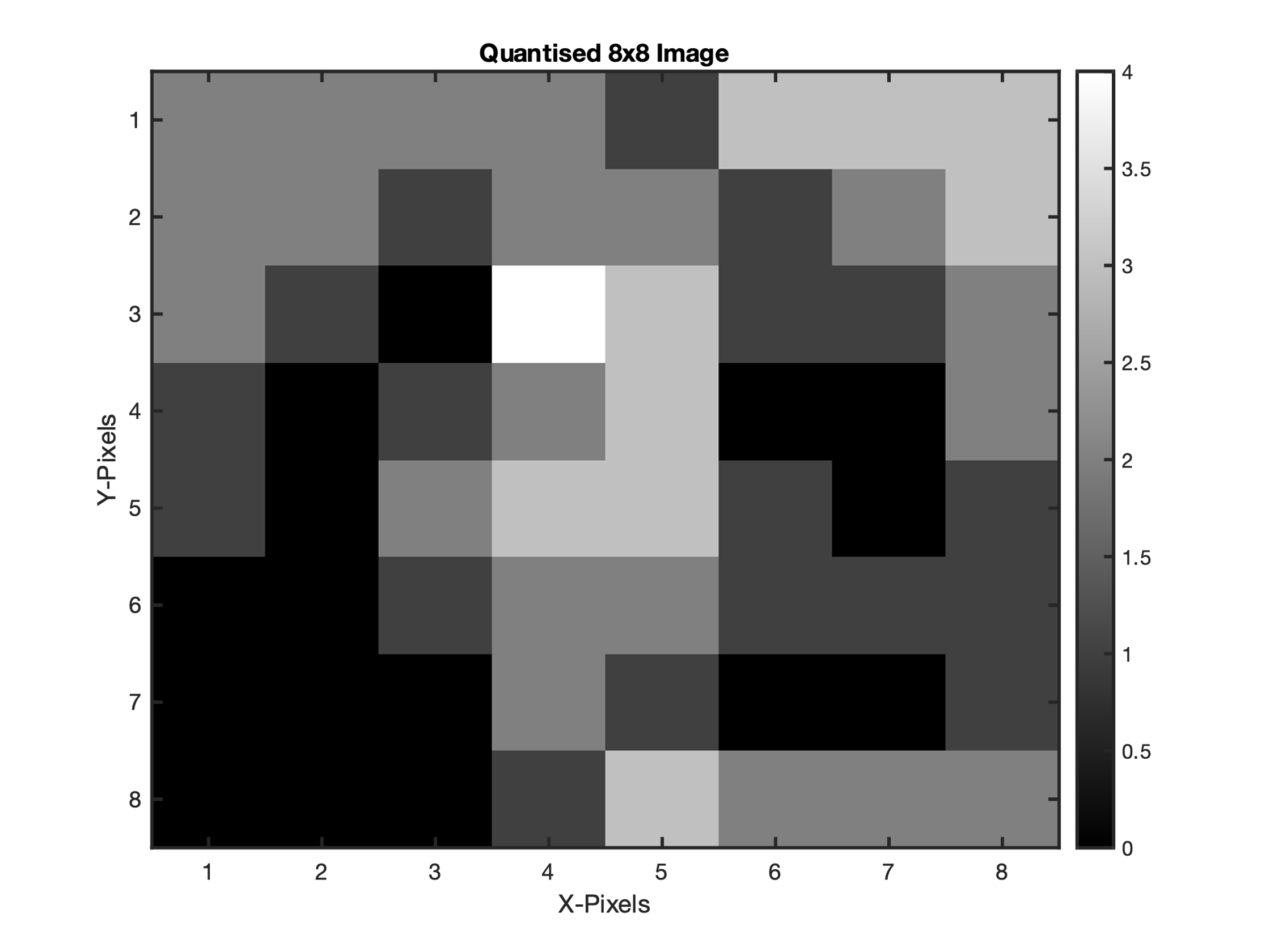
% subtract the min to make zero the minimum

% divide by the range to normalise

% multiply by 4 to get four quantised values

* Plot the quantised 8 × 8 image. It should look like this (4)





**Question 4: Sensory Substitution Process**

* Use **LINSPACE** to generate a time signal from 0 to 0.5 seconds sampled at 8kHz (it will contain 4000 samples). Include your code snippet (2)

Fs = 8000; % frequncy of signals

n = size(zf4, 2); % number of signals

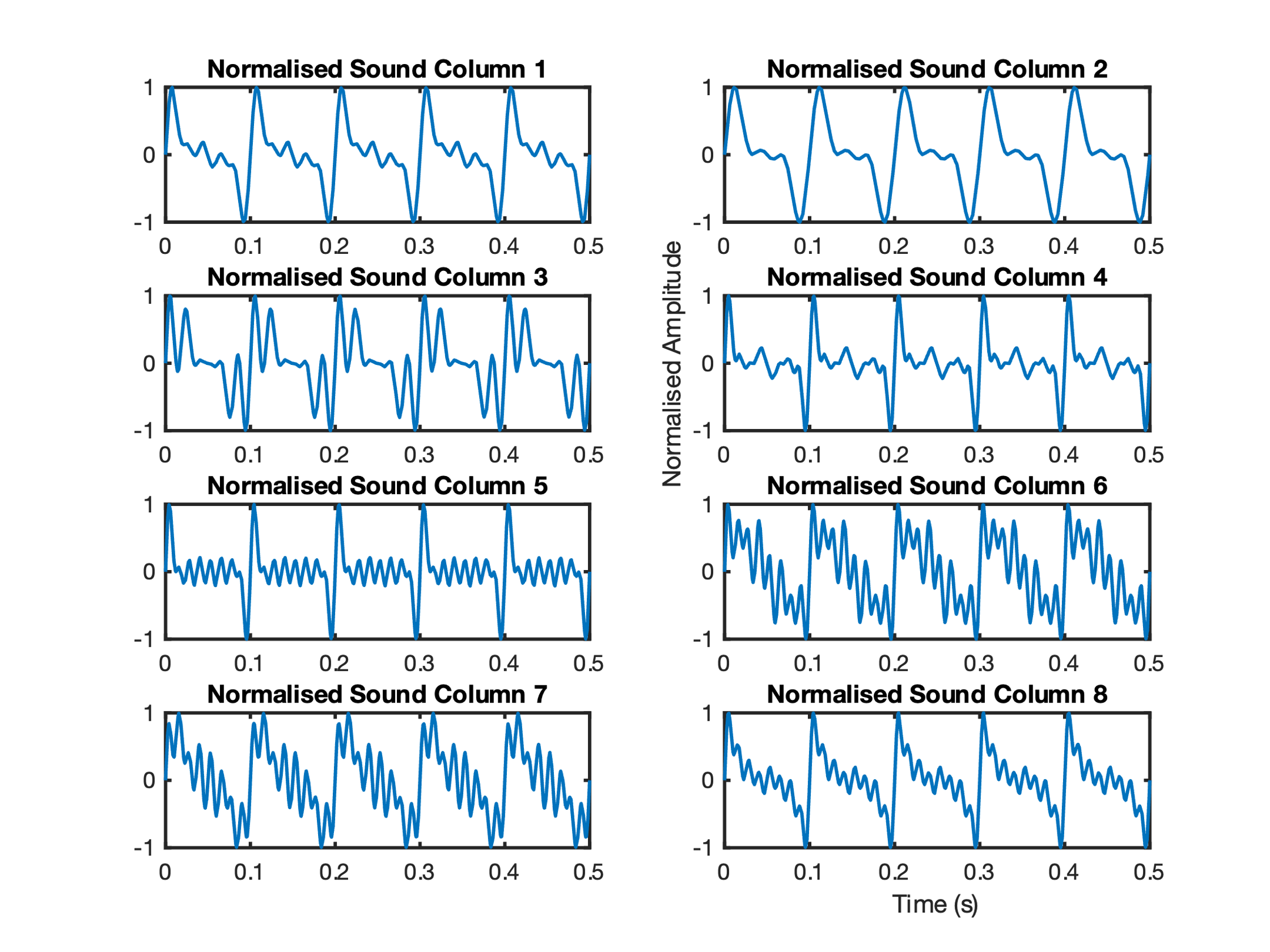
L = 4000; % points in signals

fl = 10\*(1:n); % frequency levels

t = linspace(0,L/Fs,L)'

* Use **SIN** to generate eight sinusoidal signals with frequencies corresponding to the row number (1 to 8). Starting at a base signal frequency of *f1* = 10Hz ,  etc. Store these so that you can use them to build up the sound file without having to regenerate them for each column
* For each column produce a composite signal being the sum of all of the frequencies in that column, weighted by the gray scale value of each quantised pixel
* Display the normalised sound file for each column. The first should look like this (8)





Use the **FFT** to confirm that the elements in the fourth column correspond to the frequencies generated in that sound file.

* List the quantised amplitudes of the pixels in the fourth column (2)

2 2 2 x 1 3 3 3

2 2 1 x 2 1 2 3

2 1 0 x 3 1 1 2

1 0 1 x 3 0 0 2

1 0 2 x 3 1 0 1

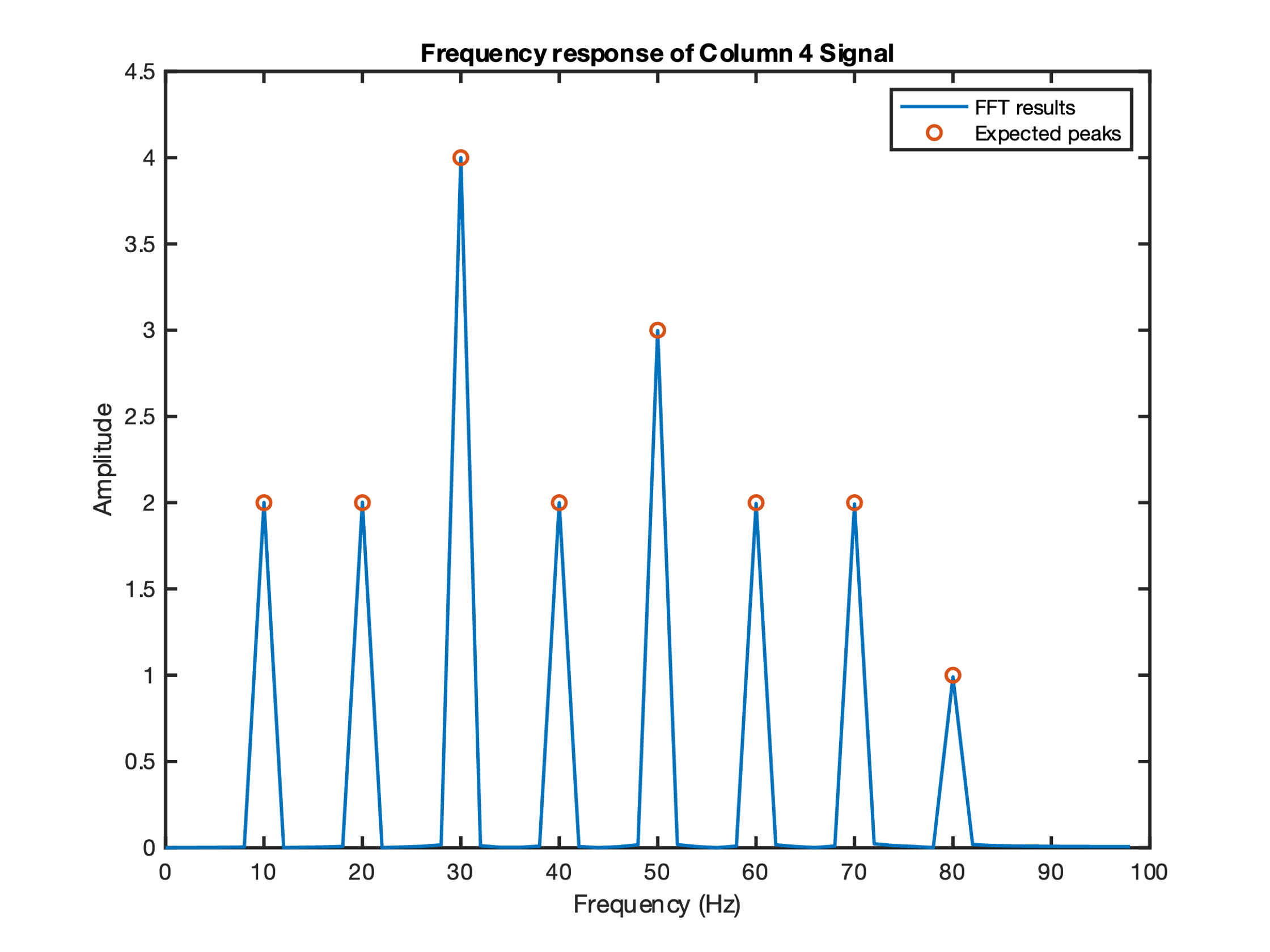
0 0 1 x 2 1 1 1

0 0 0 x 1 0 0 1

0 0 0 x 3 2 2 2

2,2,4,2,3,2,2,1

* Plot the section of the FFT that corresponds to these spectral components (6)



* Justify any differences in the frequency and amplitude of the spectral components (2)

All can be seen on the plot

**Question 5: Listen to the Sound File**

The frequency generated starting at 10Hz is good to display the waveforms, but it is too low to be heard.

* Regenerate the sinusoidal signals:
  + First frequency = f1 = 293.66Hz (The note D4)
  + Generate a vector: f\_ratios = [0 4 7 12 16 19 24 28] / 12  
    (For those musically inclined this is relating to semitones in a scale.  
    0, 12 and 24 are octaves or the tonic.  
    4, 16 and 28 are thirds.  
    7 and 19 are fifths.
  + Generate a frequency vector by raising f1 to the power of the vector.  
    (This contains 8 different frequencies to use for each column)
* Regenerate the 8 sinusoidal signals by the frequencies.  
  ,  and so on.
* Regenerate the composite signals for every column
* Normalise the signal for each to lie between -1 and 1
* Split the signal into a vector with two columns to create a stereo file
* Weight the signal for each of the two stereo channels according to its offset from the centre line of the image. [7 1],[6 2],[5 3],[4 4],[4 4],[3 5],[2 6],[1 7]
* Normalise each channel again with respect to the peak value of the combined signal  so that the louder signal lies between -1 and 1
* Using the **SOUND** function generate the individual sounds for each column (note you will need a set of speakers or a pair of headphones to hear this)
* Verify that the sounds appear to move from the left, through the centre to the right (if your headphones are on correctly)
* **Copy and paste your code underneath.**
* **Export and upload the file as a .wav file to Canvas**

Bonus 10

Total (72+10)