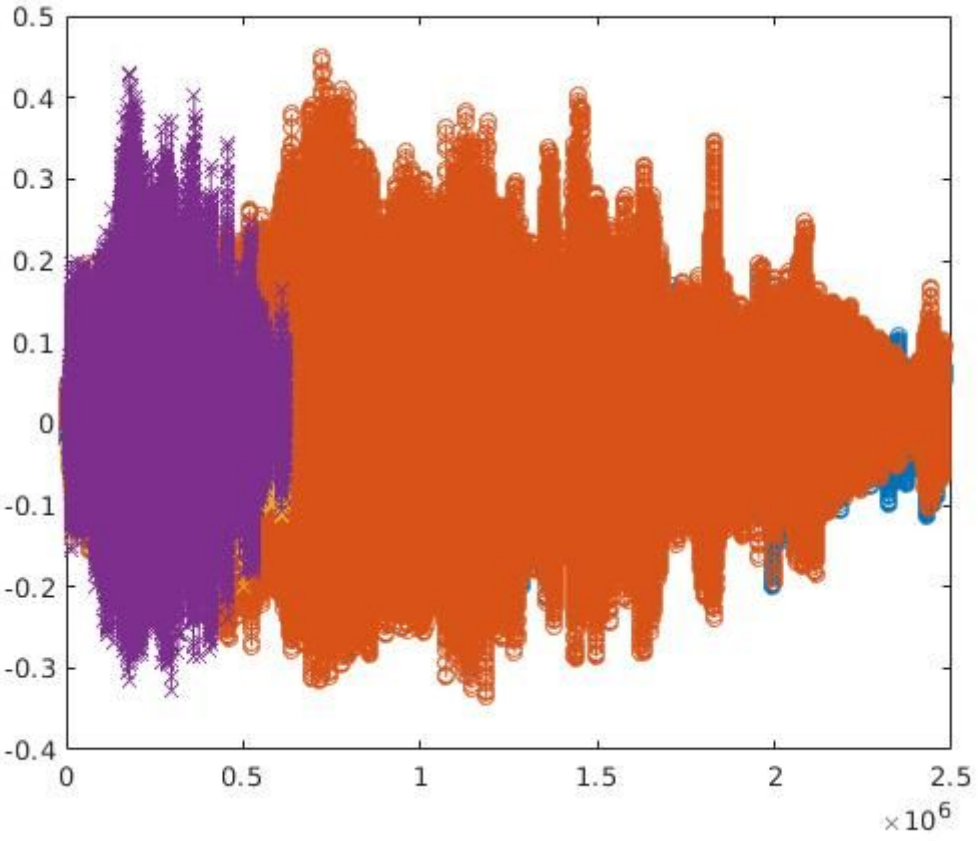
Roll number: 20171159
Name: Kripa Anne

DSAA Assignment 1

1. We use the inbuilt function resample to change the speed of the audio signal. Resample changes the sampling rate for a sequence to any rate that is a ratio of two integers. The basic syntax for resample is *y = resample(x, p, q)* where the function resamples the sequence x at p/q times the original sampling rate. The length of the result y is p/q times the length of x.
   To make the audio twice as fast: p = 1, q = 2
   To make the audio twice as slow: p = 2, q = 1
   The frequency of the signal visibly changes during resampling.



Purple graph (x) - faster audio, higher frequency. As a result, pitch increased.
Orange graph(o) - slower audio, lower frequency. As a result, pitch decreased.

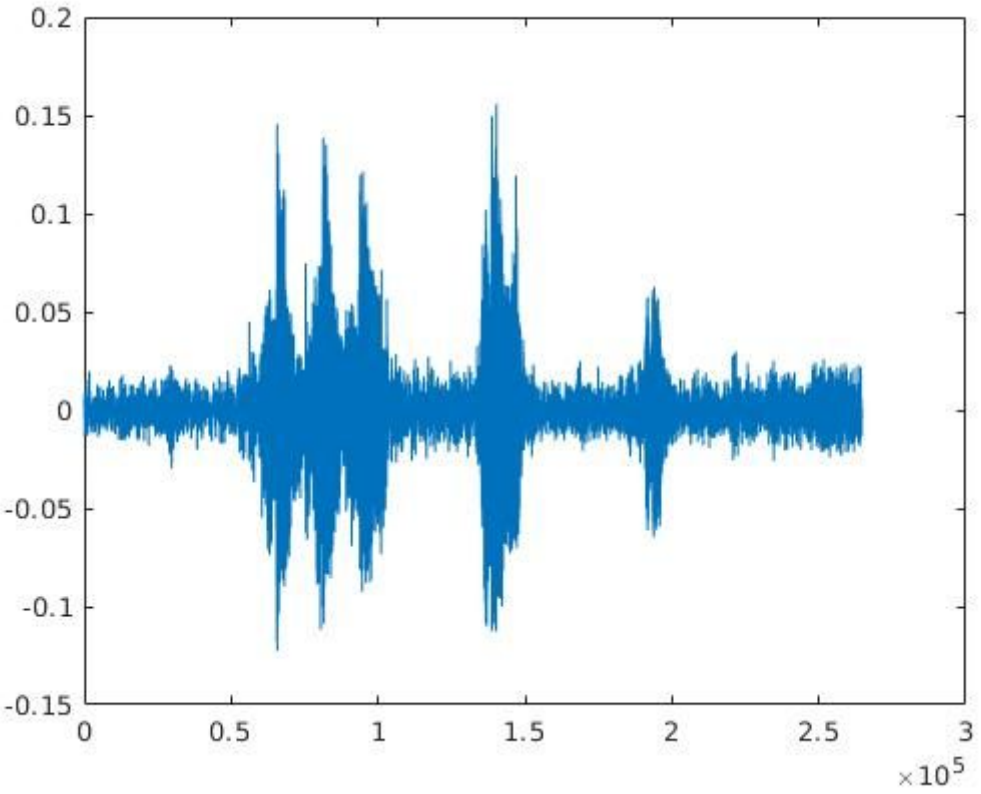
Audio read with:
filename = "sa_re_ga_ma.mp3";
[y,fs] = audioread(filename);

2. Own audio recording made and stored as MyRecording.wav at 44.1KHz and 24 bits. Commands:

```
 fs = 44.1e3;
 bits = 24;
 input = audiorecorder(fs,bits,1);
recordblocking(input,6);
```

Stored in a wav file:

```
data = getaudiodata(input);
audiowrite("MyRecording.wav", data, fs);
[x,y] = audioread("MyRecording.wav");
```



To subsample it to a particular sample rate, we again use resample function.

```
[p,q] = rat(array(1,i)/fs);
newsound = resample(x,p,q);
sound(newsound, fs);
```

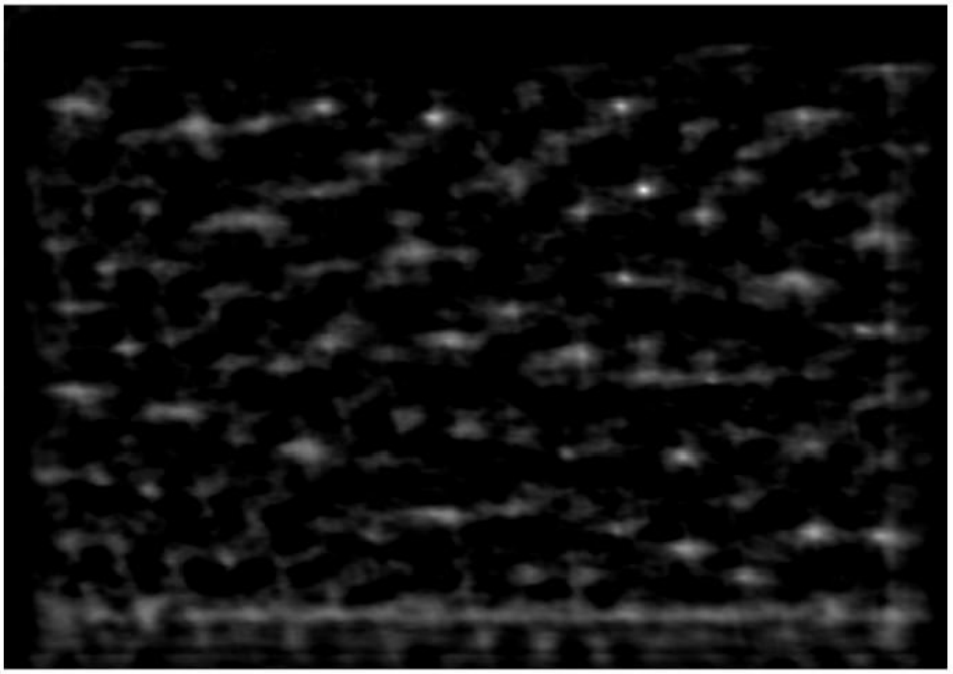
To simulate it in different environments, we acquire the impulse response audio files and read it, storing the corresponding data and fs values. We then apply N-dimensional convolution on the audio we recorded and the environment signal.

```
env_arr = ["MINI CAVES E001 M2S.wav","MEDIUM METAL ROOM E001
M2S.wav","SMALL CHURCH E001 M2S.wav"];

    [p,q] = audioread(file);
    output = convn(x,p,full);
    sound(output,fs);
```

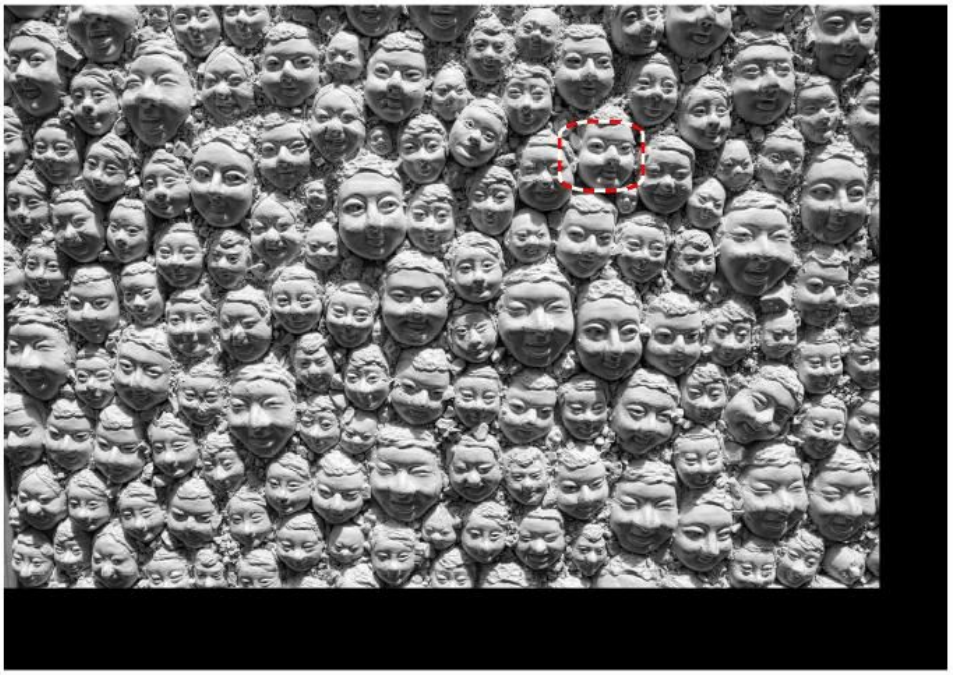
3. We can spot a part of an image within another one using 2D image cross correlation. We initially try the function xcorr2 for finding the correlation between both images but due to low accuracy, we shift to normalized 2D cross correlation with the function normxcorr2. Normxcorr2 divides by the product of the local standard deviations and subtracts the mean of the template before performing the multiplications.

```
img = imread("Faces.jpg");
sub_img = imread("F1.jpg");

img = rgb2gray(img);
sub_img = rgb2gray(sub_img);

new_img = normxcorr2( sub_img(:,:,1), img(:,:,1));
```

In this correlation image, the various white spots represent the highly similar sections in the main image with respect to the sub image. We detect the sub image at all these locations in the main image.

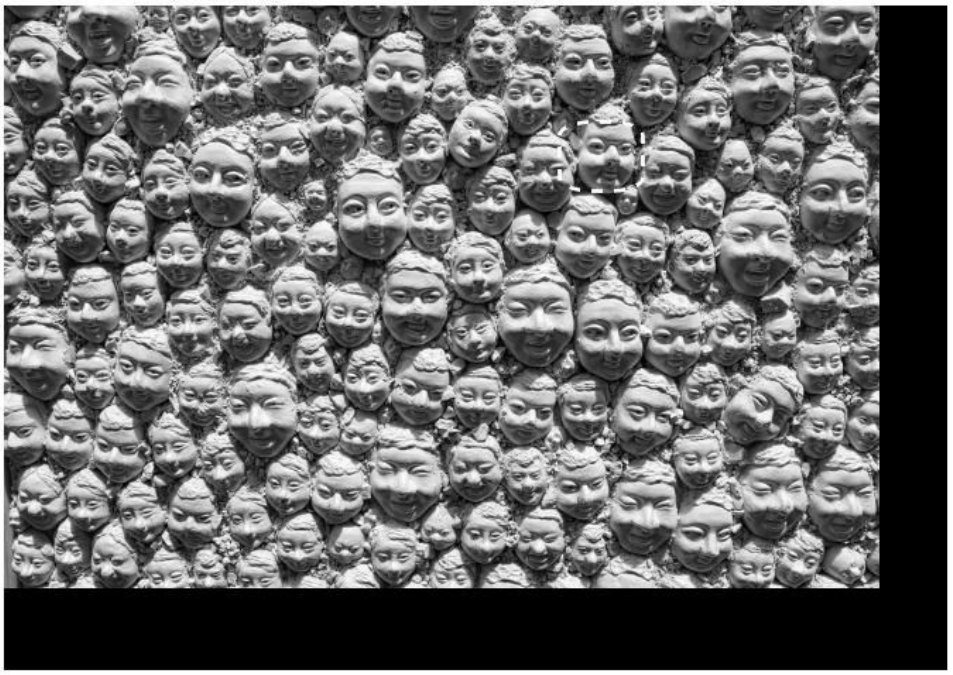


The final image contains a highlighted portion outlined by red where the highest correlation value was found in the main image.



The same method works for noisy images too, as the exact position was again identified and highlighted. A change in approach doesn't seem necessary at this

point due to the similar outputs.



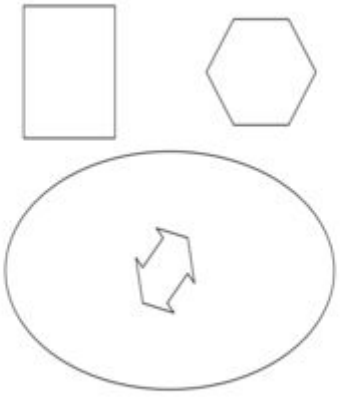
4. For resizing an image without using an inbuilt function, we can use either Nearest Neighbour Interpolation or Bilinear Interpolation.

In Nearest Neighbour Interpolation, the empty value of pixels after resizing the final image to the appropriate scale will be occupied with the value of the nearest pixel. This method results in clunkier images when we make the image larger.

We initially find the scaled new image's dimensions and give appropriate values to every element in the new image.

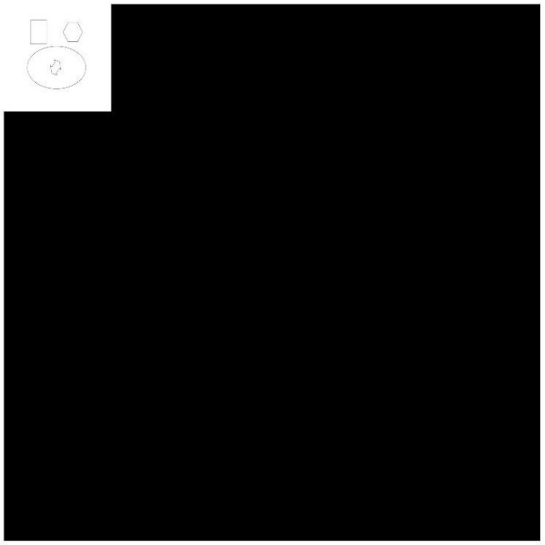
```
row_indices = ceil((1:new_ht)/sf);
col_indices = ceil((1:new_wd)/sf);

B = A(row_indices, col_indices, B_ch);
```

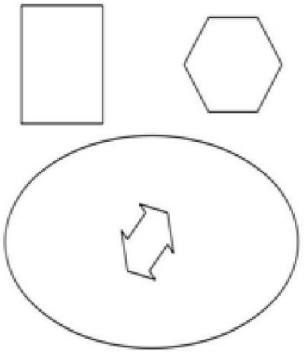
a) B&W image with ellipse
   Original: (300x300)



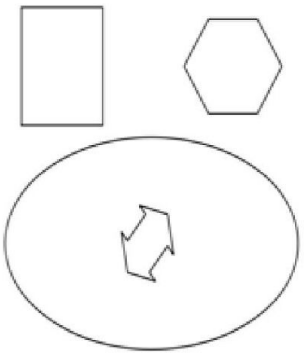
Scale = 5

Warning: Image is too big to fit on screen; displaying at 50%
> In images.internal.initSize (line 71)
In imshow (line 337)
In q4 (line 21)





Scale = 10



Warning: Image is too big to fit on screen; displaying at 25%
> In images.internal.initSize (line 71)
In imshow (line 337)
In q4 (line 21)

b) B&W image
Original:

Scale = 5
Warning: Image is too big to fit on screen; displaying at 50%
> In images.internal.initSize (line 71)
In imshow (line 337)
In q4 (line 21)

Scale = 10

Warning: Image is too big to fit on screen; displaying at 25%

> In images.internal.initSize (line 71)

In imshow (line 337)

In q4 (line 21)

3) Coloured image

Original:

Scale = 5

Scale = 10

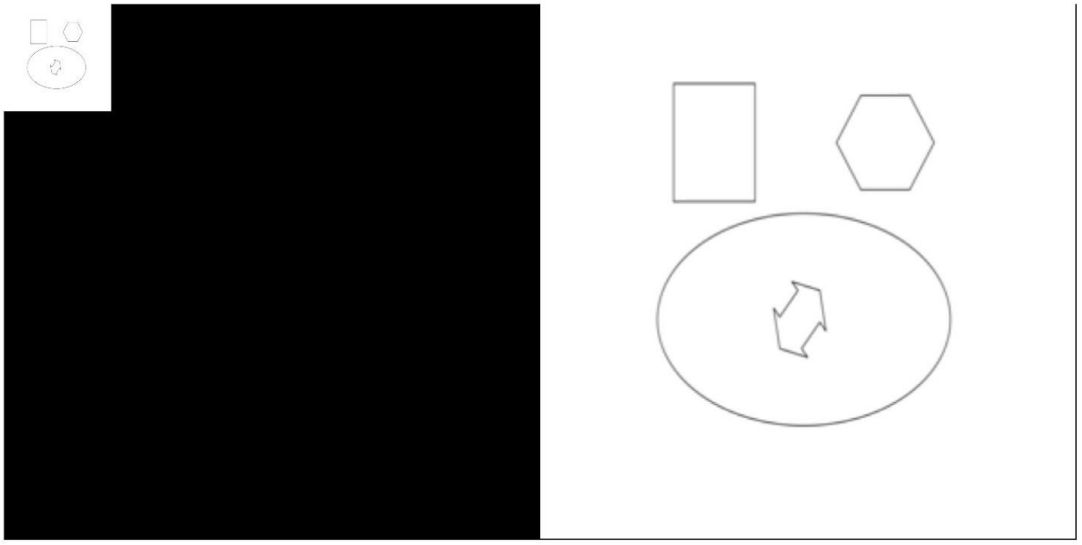
In bilinear interpolation, we first linearly interpolate along 2 points (say, along the horizontal axes) and then linearly interpolate the results from the newly calculated values above and below the required point to get the bilinearly interpolated result.

B(k,l,ch) = (B(start_y,start_x,ch)(end_x -l) + (l - start_x)*(B(end_y,end_x,ch)))/sf;*

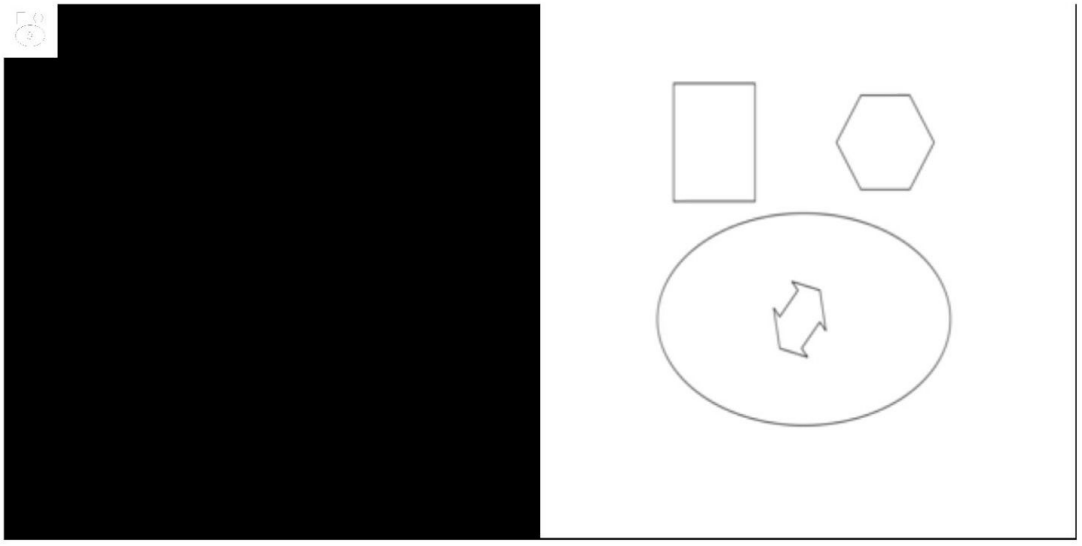
Bilinear interpolation uses values of only the 4 nearest pixels, located in diagonal directions from a given pixel, in order to find the appropriate color intensity values of that pixel. Bilinear interpolation considers the closest 2 × 2 neighborhood of known pixel values surrounding the unknown pixel's computed location. It then takes a weighted average of these 4 pixels to arrive at its final, interpolated value.

a) B&W image with ellipse (300x300)

Scale = 5



Scale = 10



b) B&W image

Scale = 5

Scale = 10

c) Colour image

Scale = 5

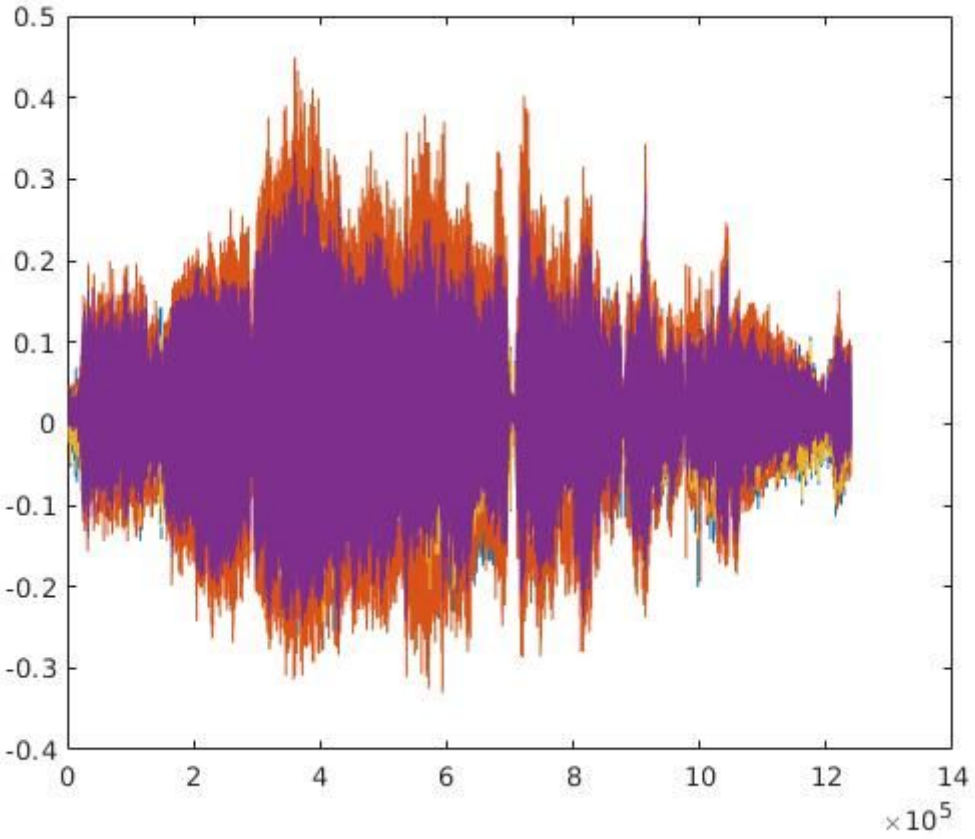
Scale = 10

We use bilinear interpolation to get smoother edges and less clunkier image resizing. It is less pixelated for images with a lot of curved edges. NNI fares equally good on images with a lot of straight edges.

5. We smooth the given audio signal using smoothnoise function which denoises the audio using different methods, depending on the parameters. The default has

B = smoothdata(A)

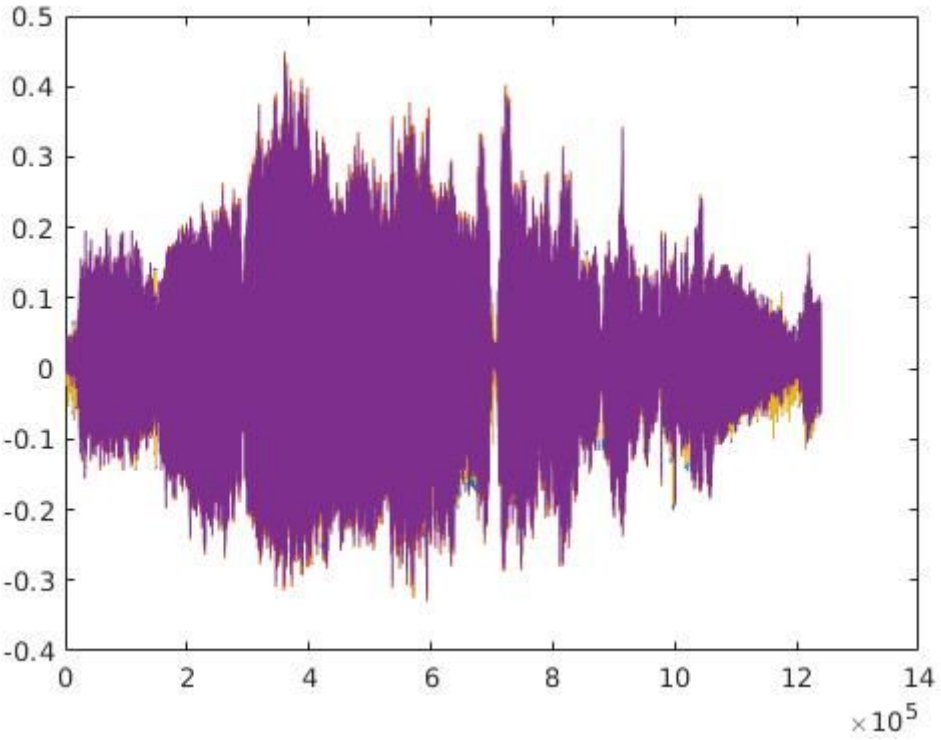
returning a moving average of the elements of a vector using a fixed window length that is determined heuristically. The window slides down the length of the vector, computing an average over the elements within each window.



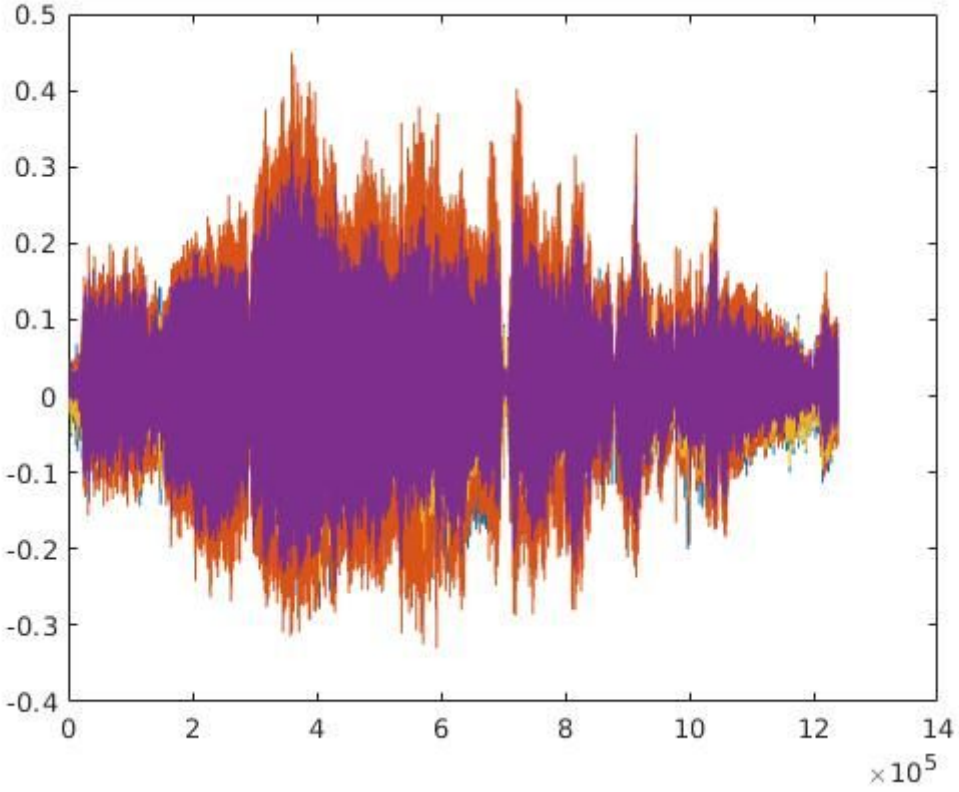
The output has a signal with a lot of noise shorn off.

Other parameters:

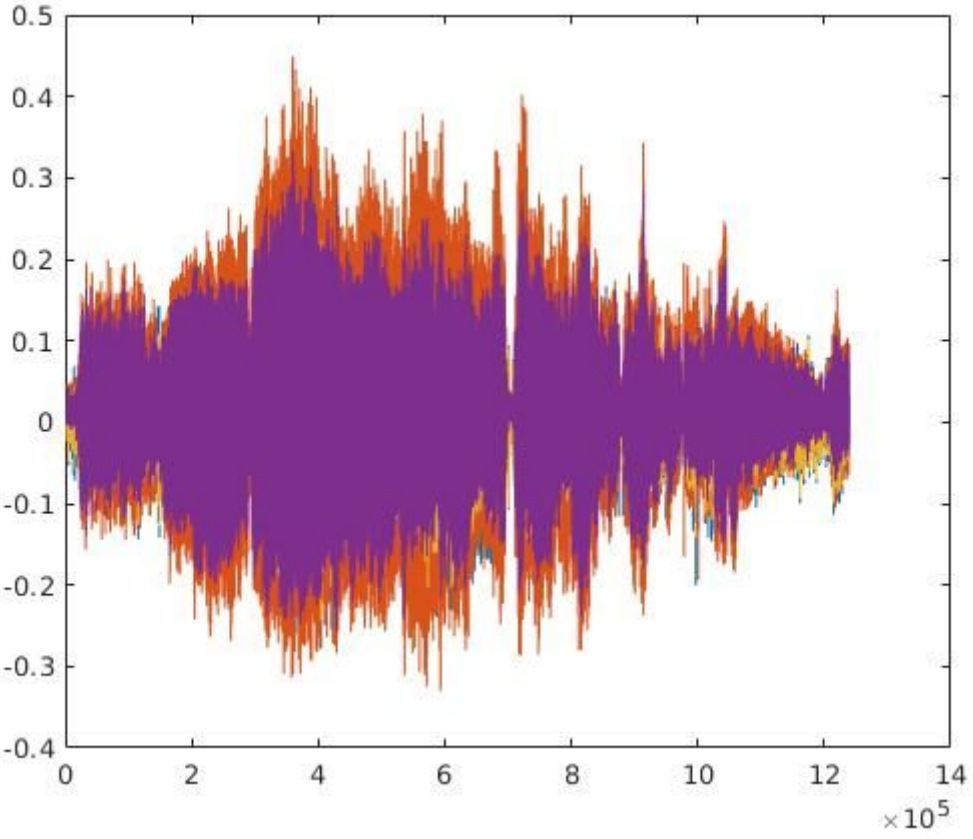
- Gaussian filter - Smooth a vector of noisy data with a Gaussian-weighted moving average filter. Differs based on window size.
  Window = 4


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
2
4
6
8
10
12
14
×10^5

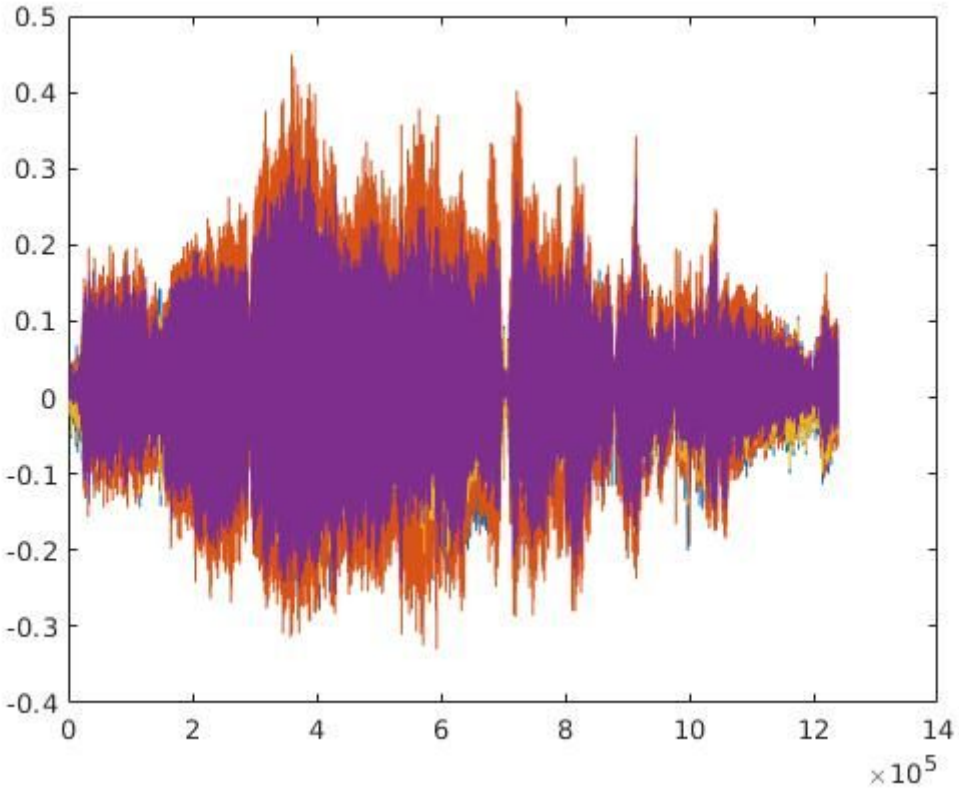
Window = 20


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
2
4
6
8
10
12
14
×10^5

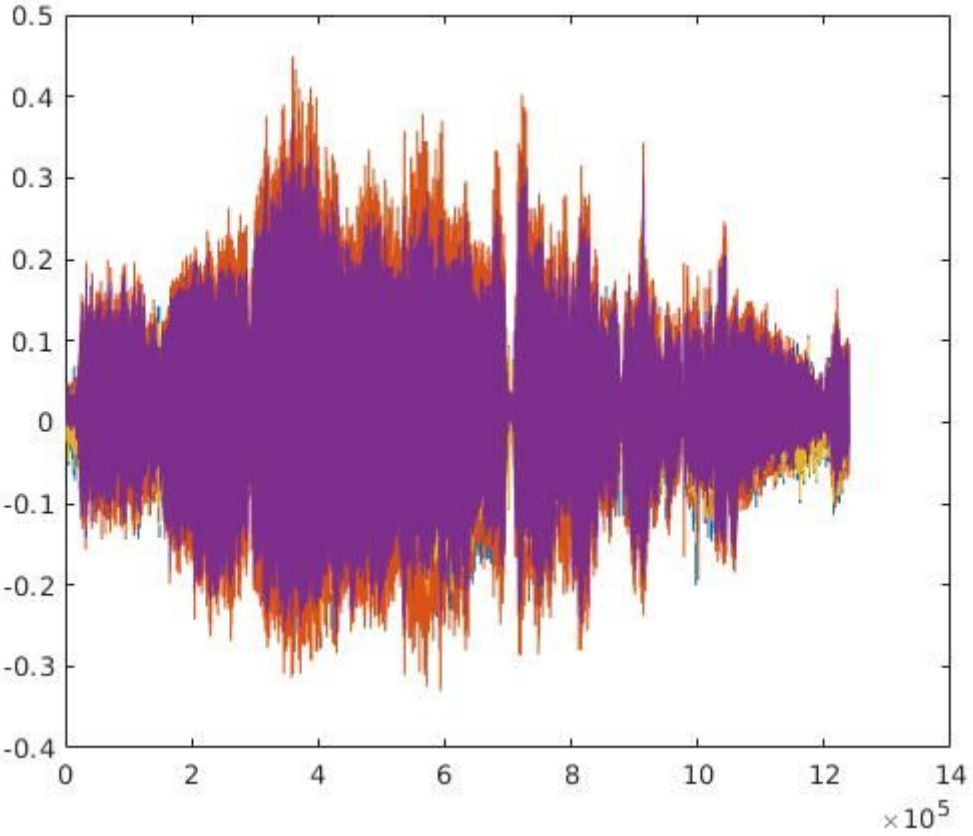
- Vector with NaN - Create a noisy vector containing NaN values, and smooth the data ignoring NaN, which is the default.



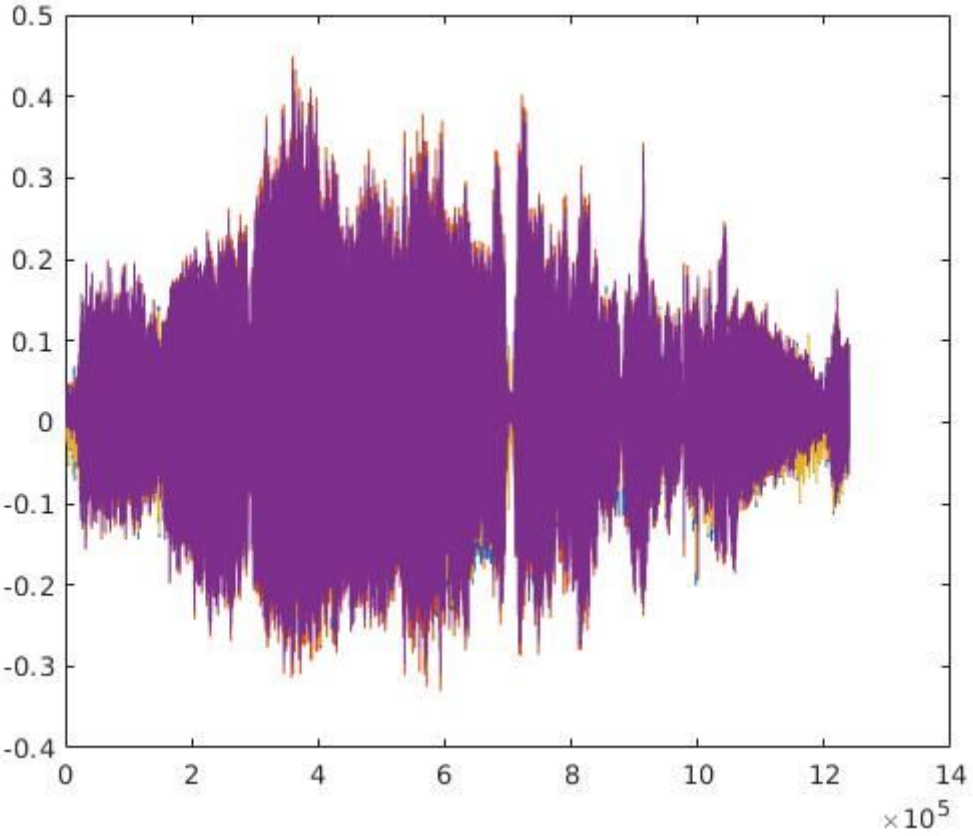
- movmean — Moving average over each window of A. This method is useful for reducing periodic trends in data.



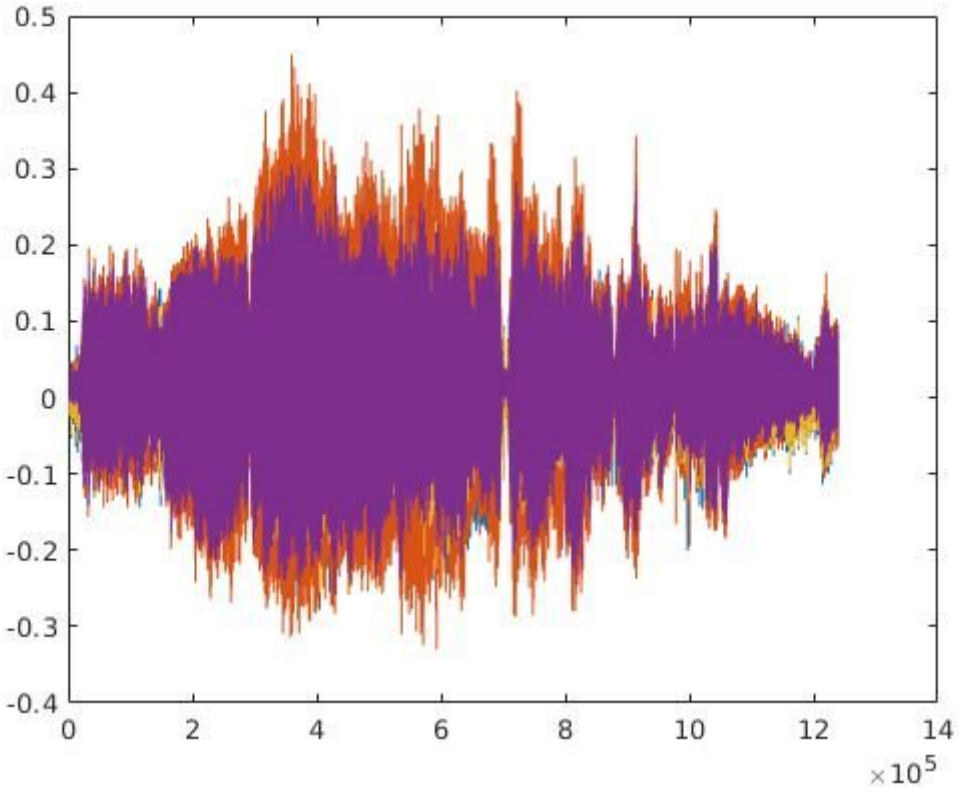
- movmedian — Moving median over each window of A. This method is useful for reducing periodic trends in data when outliers are present.



- lowess — Linear regression over each window of A. This method can be computationally expensive, but results in fewer discontinuities.

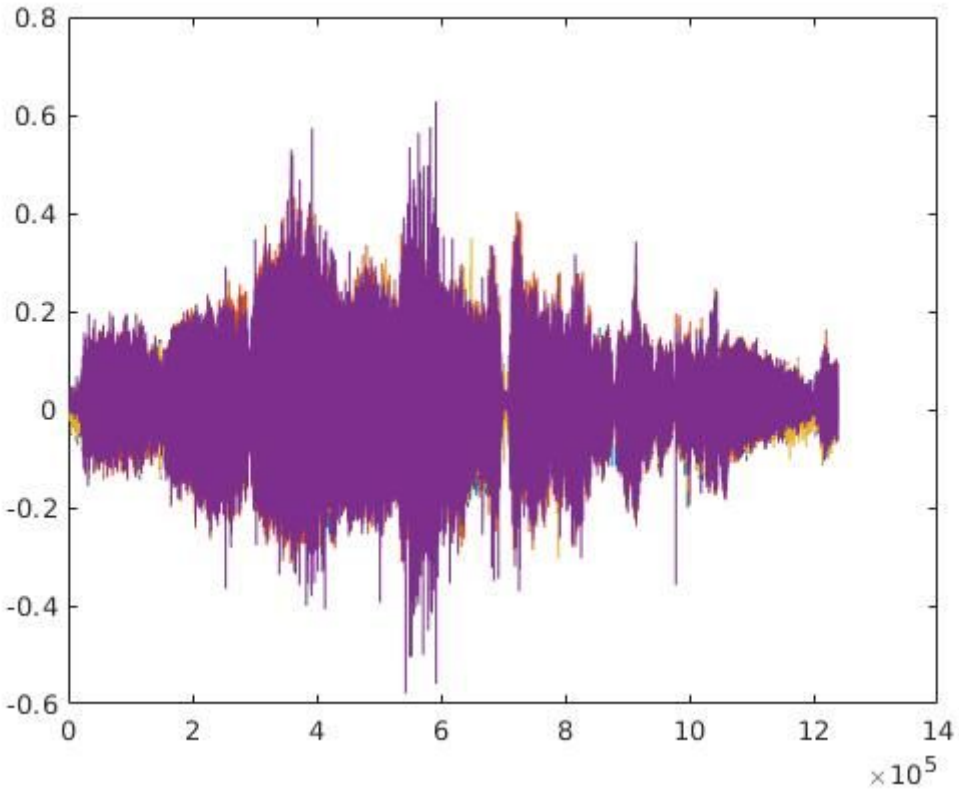


- loess — Quadratic regression over each window of A. This method is slightly more computationally expensive than lowess.



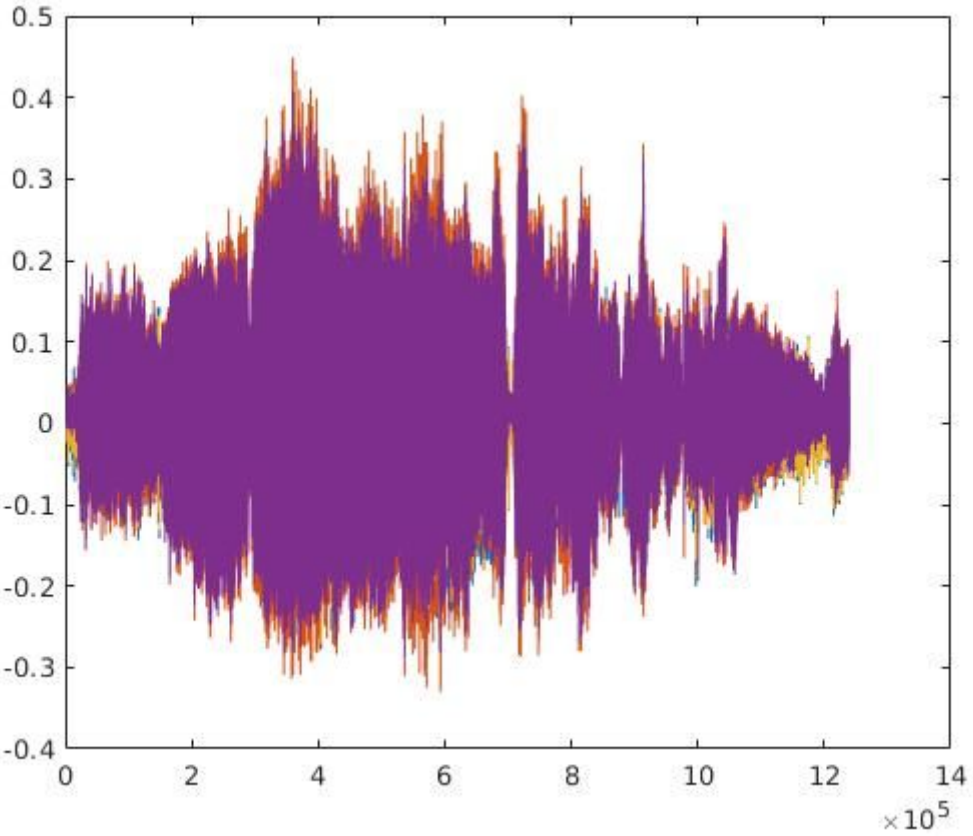
- rlowess — Robust linear regression over each window of A. This method is a more computationally expensive version of the method lowess, but it is more

robust to outliers.



- sgolay — Savitzky-Golay filter, which smooths according to a quadratic polynomial that is fitted over each window of A. This method can be more

effective than other methods when the data varies rapidly.



The best technique can be loess due to better smoothening of output signal.