

NAG5

by Cnn Cnn5

Submission date: 02-May-2022 09:37AM (UTC+0530)

Submission ID: 1825870927

File name: Working_and_analysis.pdf (2.45M)

Word count: 4108

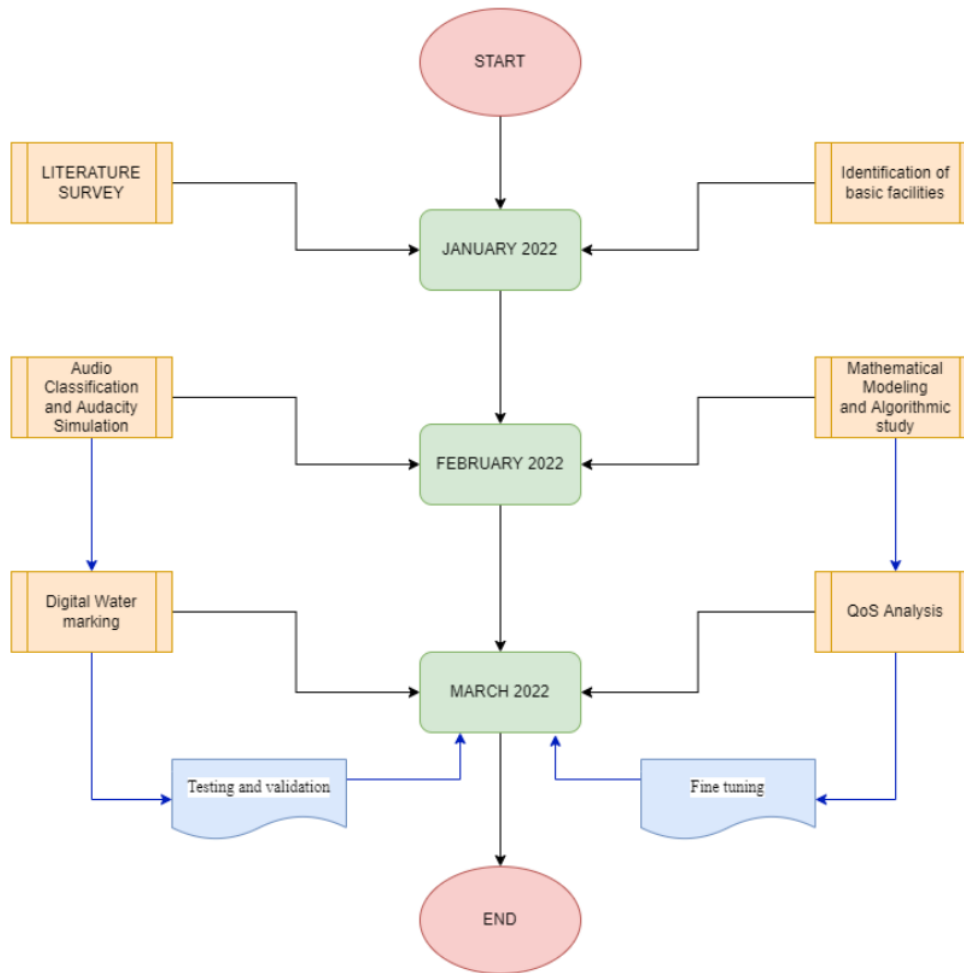
Character count: 20850

CHAPTER 4

WORKING & ANALYSIS

I. PLANNING

For the working of our project we came up with a methodology framework and divided it into 3 months worth of work. An “.io” diagram showing our framework is attached below:



Working Flowchart

We started with the classification of the audio signal and dividing it by parameters such as their bandwidth, nominal level, power level in decibels (dB), and voltage level. The relationship between power and voltage is determined by the impedance of the signal path. Signal paths may be single-ended or balanced.

After analyzing some research papers we found that audio signals have somewhat standardized levels depending on the application. Outputs of professional mixing consoles are most commonly at line level. Consumer audio equipment will also output at a lower line level. Microphones generally output at an even lower level, commonly referred to as mic level. For our project we will focus on the line level.

For our work we will be using MATLAB and Audacity which is an open source - multi platform software. This is an inhouse project for which the minimum specifications required to run the project are as follows:

Parameter	Minimum Specifications Needed
Operating System (OS)	Windows 7
Processing power	1.8 GHz
Random Access Memory (RAM)	2 GB

Table 1.0

We started with using Audacity to test the form of audio signal and the Waveform Audio File Format in the preliminary stages. We used a file that contained audio recordings with different sampling rates and bit rates but were saved in a 44.1 kHz, 16-bit format on Audacity. These files were divided into host and port WAV's and sampled onto Audacity and was mapped using the microsoft sound mapper at the following specifications:

Parameter Name	Value Set
Audio Channel	Mono
Frequency	44100 Hz
Bit point (floating)	32 bit
Mapping (range)	- 1.0 - 1.0
Time Sampled	Duration = 60 seconds

Table 2.0

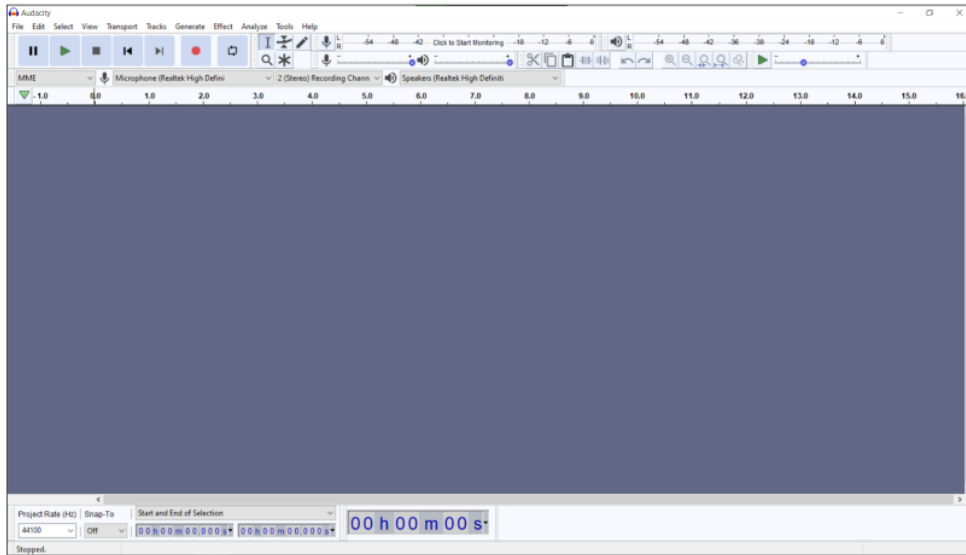
During this we found that any traditional form of watermarking on an audio file can be easily decrypted at 'line level' using software. To prove this we added a water mark at one junction and decrypted the signal using the "same software" and varying the parameters mentioned above. And found that the watermark (called as sis - gen) could easily be retrieved. Then we varied it with 4 other forms of waves (sine, square, sawtooth, triangle) and took the readings (comparison is yet to be done) with gain set at +5 dB and frequency at 440 Hz.

II. ENCRYPTION METHODS

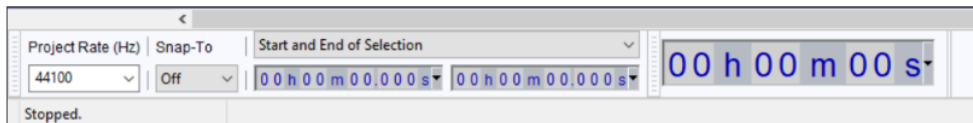
A. USING OVERLAPPING

During our literature survey we came across an existing process of signal encryption that uses a software or tool to overlap an audio signal with another audio signal before transmission and then proceeds to remove that same overlapped signal on the receiver's end.

We used a similar software called Audacity to replicate the same and to find out for ourselves whether the theoretical flaws we found were practical or not.

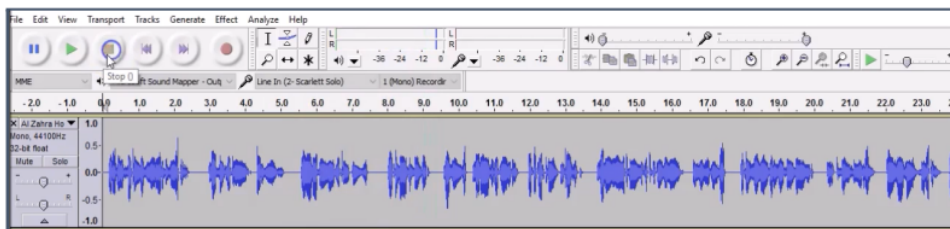


Audacity Homepage



Audacity Editables

For our audio we used a 1 minute clip from the audiobook called “The Adventures Of Sherlock Holmes”. This is an open source audio book available for sampling and free using. We added that audio file as a mono stereo file as tract one. Then we generated a secondary white noise and added it into track two (just below the previous track).



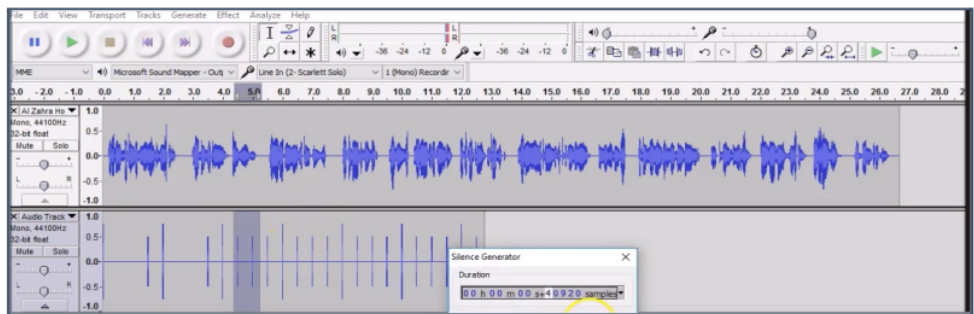
Audacity Audio Channel

Then we generate a tone (from the software) and superimpose it on the original audio file. In Audacity we can basically change every single parameter of an audio file. For our sampling test we changed it to the following settings:

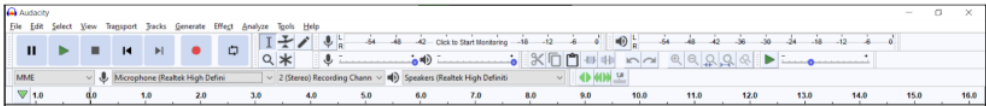
Parameter Name	Value Set
Audio Type	MP3 or WAV
Audio Channel	Mono
Frequency	44100 Hz
Bit point (floating)	32 bit
Mapping (range)	- 1.0 - 1.0
Time Sampled	Duration = 60 seconds

Table 3.0

The result screenshot is attached below:



Overlapped audio



Audacity toolbar

We then generated the traditionally encrypted audio signal and studied its pros and cons. We believe that encrypting an audio file like this has several advantages and disadvantages. But in the domain of encrypting an audio signal the disadvantages top the advantages.

Some of the disadvantages are which are:

1. The original audio signal will be susceptible to noise.
2. Anyone with the audio file can easily see the tune used to encrypt the file and remove it using a mediocre level software.
3. The sender has almost no control over the encryption.
4. The raw audio signal gets quite messy to discern the original content of the file

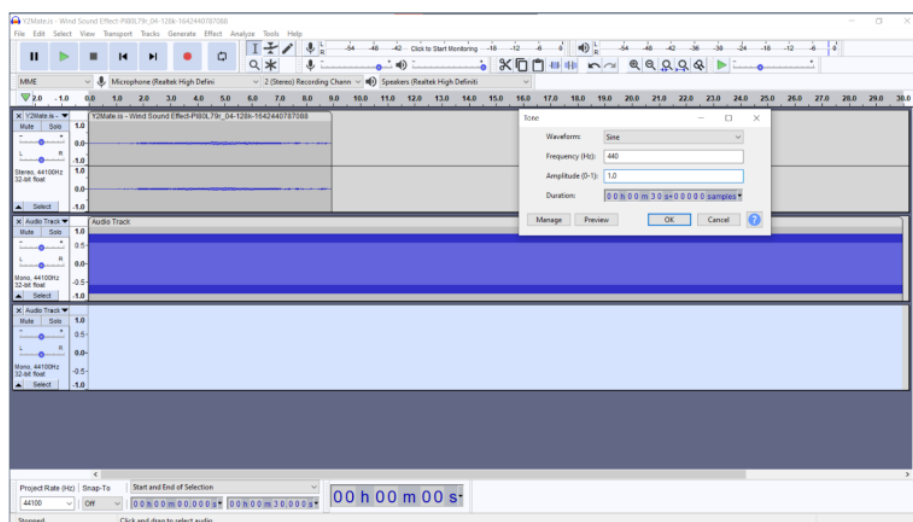
Therefore with these crucial disadvantages we proceeded to test it with another method to see if this method actually be improved upon and whether digital watermarking is actually needed.

B. USING WAVEFORM VARIATION

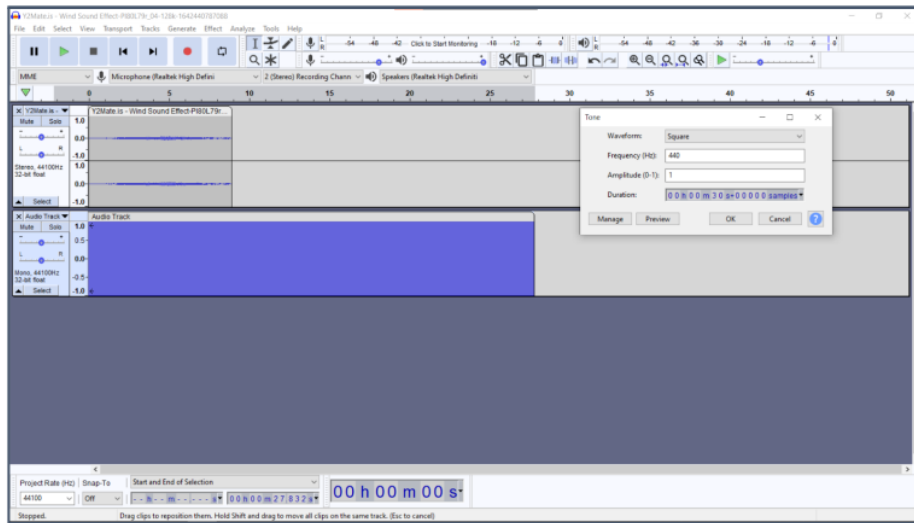
From the first method we saw that one of the major disadvantages was that “A TONE” was being used as the overlapping signal. This in turn generated noise that made the original audio quality bad. However what if we used a single waveform rather than a whole tone.

We used Audacity as before to do the same. We kept our original audio the same but changed the overlapping signal from a tone to a single waveform. We did this with 4 separated waveforms (sine, square, sawtooth and triangle).

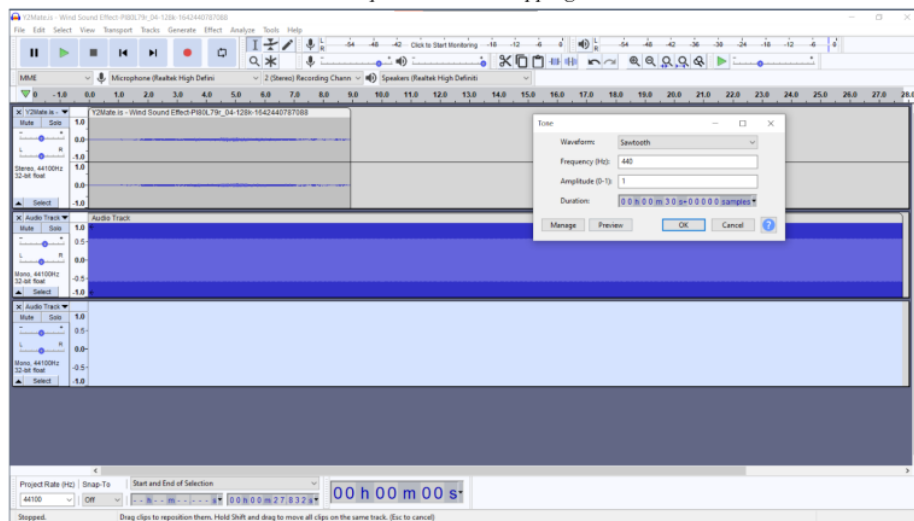
Firstly we used a sine wave (400Hz, 1.0 Amplitude). From this we find that even though the raw audio file ended up retaining its original audio quality it showed no improvement in terms of encrypting the audio. All the open loopholes that were present when using tone overlapping are still present when using a waveform overlapping. The same result was seen while using square waveform, sawtooth waveform, triangle.



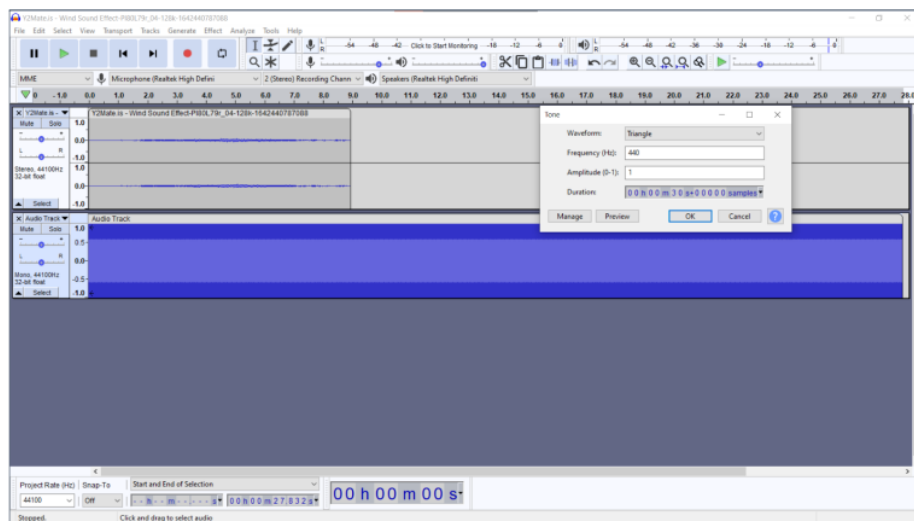
Sine wave overlapping



Square wave overlapping



Sawtooth wave overlapping



Triangle wave overlapping

Therefore after testing it with waveforms we are still left with 3 major disadvantages to this type of audio encryption that make it susceptible to 3rd party attacks and decryption:

1. Anyone with the audio file can easily see the tune used to encrypt the file and remove it using a mediocre level software.
2. The sender has almost no control over the encryption.
3. The raw audio signal gets quite messy to discern the original content of the file

C. USING LINEAR REGRESSION

Finally we use a testing method of regression based method to visualize the distribution of the model:

Plot Data

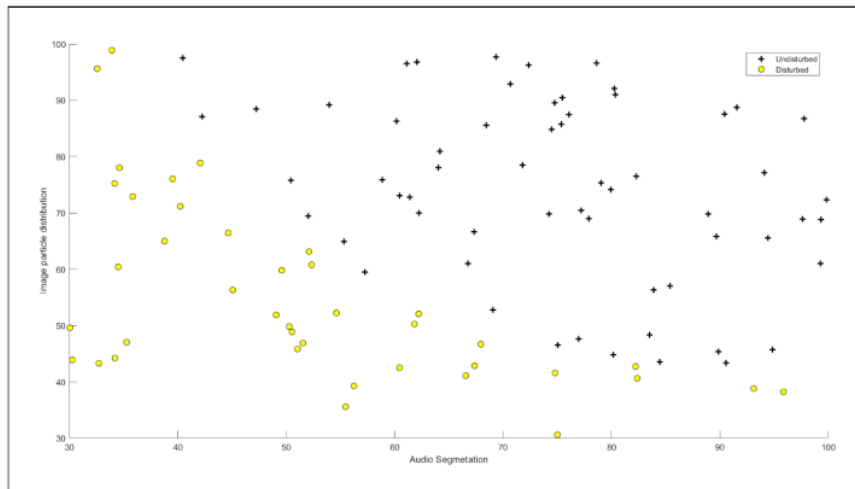
- 8 → Plots the data points X and y into a new figure.
- It will plot the data points with + for the positive values and o for the negative values. X is assumed to be a Mx2 matrix.
- Plotting the positive and negative values on a 2D plot, using the option 'k+' for the positive values and 'ko' for the negative values.

Regression Watermarking

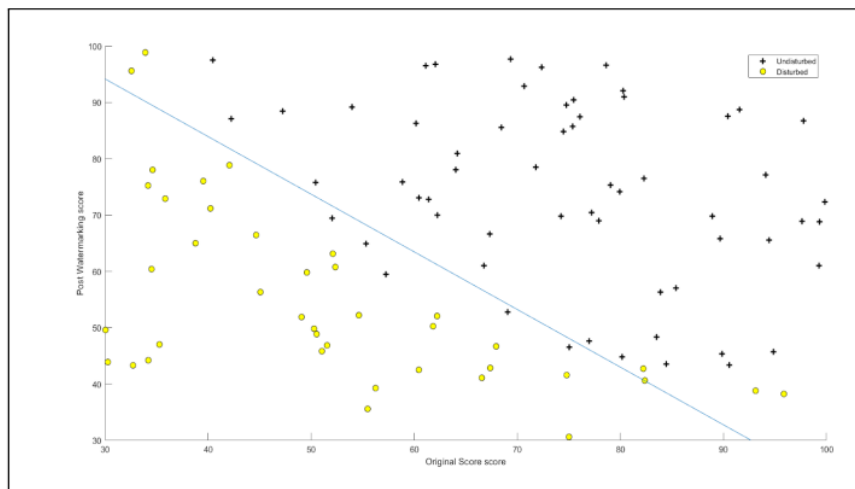
- Segmentation Plotting
- Logistic Regression
 - ◆ Setup the data matrix
 - ◆ Add intercept term to x and X_test
 - ◆ Initialize fitting parameters
 - ◆ Compute and display initial and final values of the gradient
 - 3 ◆ Compute and display gradient with non-zero theta
- Optimizing using fminunc
 - ◆ Set options for fminunc
 - ◆ Run fminunc to obtain the optimal theta
 - ◆ Print theta to screen
 - ◆ Plot Boundary
 - ◆ Put some labels
 - ◆ Specified in plot order
- Trainin2Set
 - ◆ Compute accuracy on our training set

Plot Decision Boundary

- Plots the data points X and y into a new figure with the decision boundary defined by η .
- Plots the data points with $+$ for the positive values and o for the negative values. X is assumed to be a either:
 - ◆ $M \times 3$ matrix, where the first column is an all-ones column for the intercept.
 - ◆ $M \times N$, $N > 3$ matrix, where the first column is all-ones.
- Plot Data
 - ◆ Only need 2 points to define a line, so choose two endpoints
 - ◆ Calculate the decision boundary line
 - ◆ Plot, and adjust axes for better viewing
 - ◆ Legend, specific for the distribution
 - ◆ Grid range

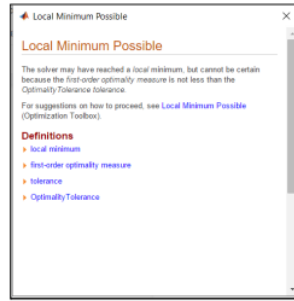


MATLAB Figure for Stage 1



MATLAB Figure for Stage 2

But we see that ¹⁰ even this method reached something called a Local Minimum possible. Because of this, the solver may have reached a local minimum, but cannot be certain because the first-order optimality measure is not less than the OptimalityTolerance tolerance.



Local Minimum

Also This method becomes extremely tedious when the data set is large. This adds to our claim that using digital watermarking for audio encryption is the best path possible.

Regression Based Results

⁸ Difference in value at initial theta (zeros)	0.693147
Expected difference (approx):	0.693
Gradient at initial theta (zeros)	-0.100000 -12.009217 -11.262842
Expected gradients (approx):	-0.1000 -12.0092 -11.2628
Value at test theta: Expected (approx):	0.218330 0.218
Gradient at test theta:	0.042903 2.566234 2.646797
Expected gradients (approx):	0.043 2.566 2.647

Regression based results

III. CODING

Therefore after analyzing the previous outputs and studying their disadvantages we see that using digital watermarking of audio signals for enhanced signal protection is indeed one of the most convenient ways to go about the problem statement of audio encryption.

Hence for our project we have used MATLAB to achieve the same. The entire process is divided into parts of matlab code namely “Embedded the watermark” and “Extracting the water marking”. It was important to use two separate code scripts for this process as mentioned in the QoS literature review, it protects the integrity of the code snippet in case some “pirate” affects one section of the code, the others will remain intact and free of interference from the 3rd party.

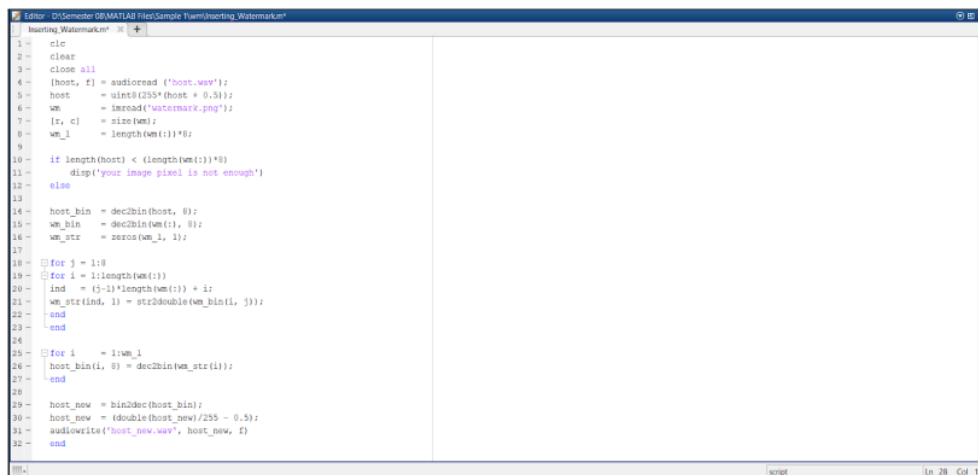
Our project includes 2 MATLAB function files (write) and 1 sample MP3 files (read). It works like the commands WAVWRITE and WAVREAD.

This version was made in MATLAB for WINDOWS only. Also note that we are using DOUBLE data type with double [-0.5 +0.5] to 'uint8' [0 255] as the parameter and this can be changed according to the receiver's need or the size of the watermark./host signal.

A. INSERTING WATERMARK

The insert watermark code script follows the following hierarchical approach:

- Clear memory and command window
- Load data
- Watermarking
- Prepare host
- Prepare watermark
- Insert watermark into host
- Watermarked host
- Save the watermarked host



```
1 = clear
2 = clear
3 = close all
4 [host, f] = audioread('host.wav');
5 host = uint8(255*(host + 0.5));
6 wm = imread('watermark.png');
7 [r, c] = size(wm);
8 wm_1 = length(wm(:))*8;
9
10 if length(host) < (length(wm(:))*8)
11     disp('your image pixel is not enough')
12 else
13
14     host_bin = dec2bin(host, 8);
15     wm_bin = dec2bin(wm(:), 8);
16     wm_str = zeros(wm_1, 1);
17
18     for j = 1:8
19         for i = 1:length(wm(:))
20             ind = (j-1)*length(wm(:)) + i;
21             wm_str(ind, 1) = str2double(wm_bin(i, j));
22         end
23     end
24
25     for i = 1:wm_1
26         host_bin(i, 8) = dec2bin(wm_str(i));
27     end
28
29     host_new = bin2dec(host_bin);
30     host_new = (double(host_new)/255 - 0.5);
31     audiowrite('host_new.wav', host_new, f)
32 end
```

Inserting watermark code

Working of the code:

- First of all we run a command to clear all the previous system clutter(memory) and get a clean command window.
- Then we use the “audioread” command to load the host audio and store it in a variable with its address stored into an index variable (which in our code is named as f).
- Then we load the watermark image (which is a png) into a using the “imread” command and store it in a variable named “wm”
- Then we use a 2D array to store the dimensions of the image we are using as a watermark.
- Then we use a conditional statement to check if the length of the host file is less than the length of the watermark * 8.
- If it is then we reject that image as a watermark and display a message saying to use a different watermark.

- Then we call the predefined MATLAB function called “dec2bin” to start the binary host.
- Then we write the code to prepare the watermark:
 - Firstly we call the dec2bin function used just above this function.
 - Then we declare a n X 8 zeroes matrix (where n is the proportional row size of the watermark).
 - Then we run a for loop to insert a watermark into the first plane of the host signal.
 - Then for every iteration of the for loop declare the LSB by using the command “host_bin(i, 8) = dec2bin(wm_str(i))”
- Finally host the watermark using the bin2dec() function and specify the data type as “double”.
- End the code block.
- Save the host code.
- RUN the code to insert the watermark.

After running the code the workspace values obtained are attached below.

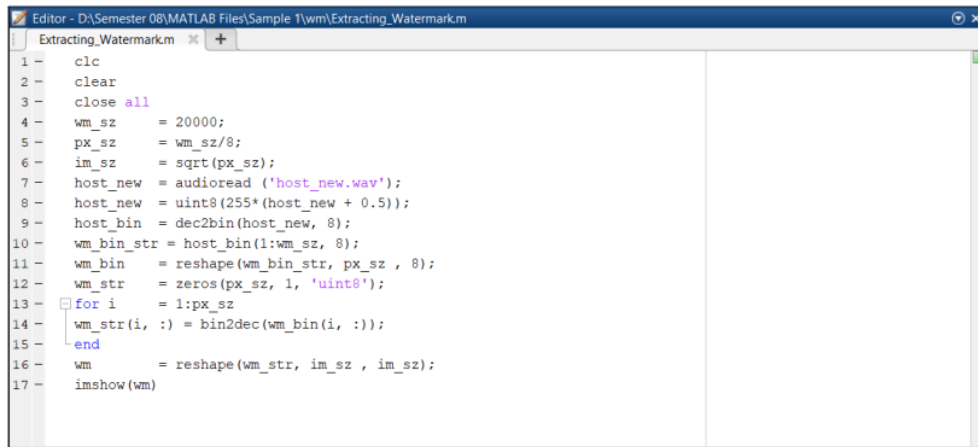


Watermark

B. EXTRACTING WATERMARK

The insert watermark code script follows the following hierarchical approach:

- Clear memory and command window
- Load data
- Prepare host
- Extract watermark
- Display image and supporting html file

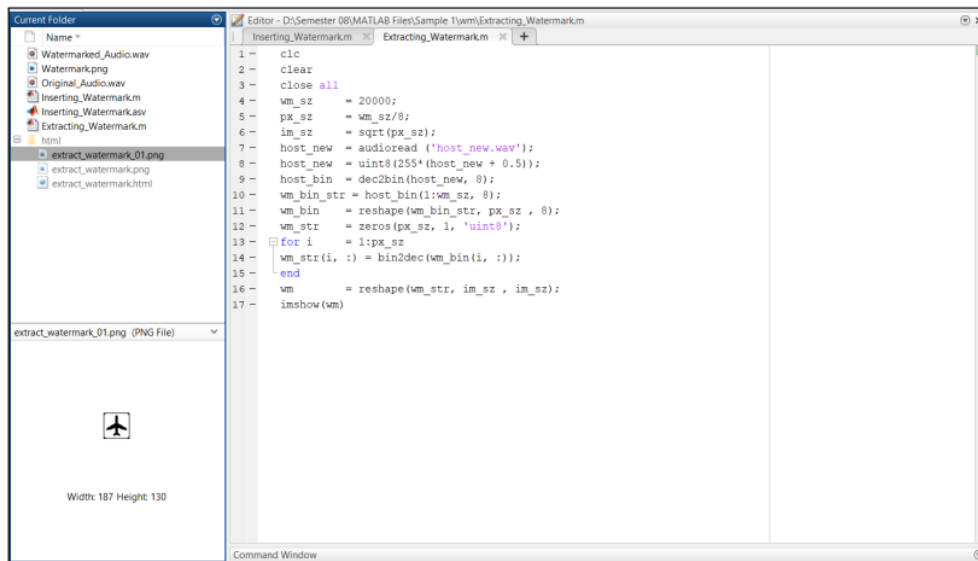


```
1- clc
2- clear
3- close all
4- wm_sz = 20000;
5- px_sz = wm_sz/8;
6- im_sz = sqrt(px_sz);
7- host_new = audioread('host_new.wav');
8- host_new = uint8(255*(host_new + 0.5));
9- host_bin = dec2bin(host_new, 8);
10- wm_bin_str = host_bin(1:wm_sz, 8);
11- wm_bin = reshape(wm_bin_str, px_sz, 8);
12- wm_str = zeros(px_sz, 1, 'uint8');
13- for i = 1:px_sz
14-     wm_str(i, :) = bin2dec(wm_bin(i, :));
15- end
16- wm = reshape(wm_str, im_sz, im_sz);
17- imshow(wm)
```

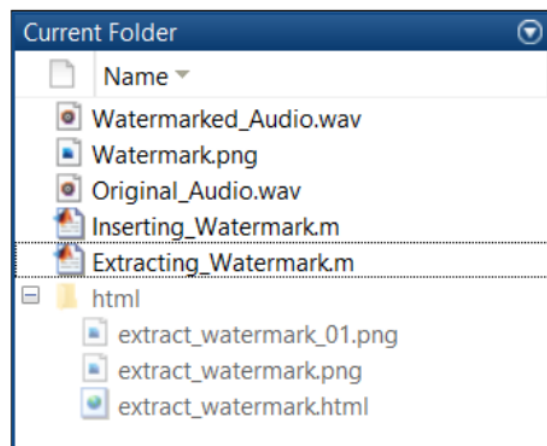
Extracting watermark

Working of the code:

- First of all we run a command to clear all the previous system clutter(memory) and get a clean command window.
- Specify an upper range for the watermark size and store it in a variable.
- Divide that with 8 to get the number of pixels used in generating the size.
- Store the image size in a variable as the square root of the pixel size.
- Call the “audioread” function and pass the path of the watermarked host signal’s path name into the function.
- Use the double [-0.5 +0.5] to 'uint8' [0 255] parameter to match the watermark dimensions matched to the double data type.
- Define the host bin size at (1:max size) and store it in a variable.
- Call the “reshape function” and pass the above variable along with the pixel size and the constant number 8(that we used to divide in step 2 into the function.
- Define a zeros matrix.
- Run a for loop on this matrix:
 - For every iteration of the for loop use the bin2dec to generate a LSB and store it in a variable.
- End the for loop.
- Call the reshape function used before and replace the constant 8 with the image size variable.
- Call in the imshow() function to finally extract and display the watermark.



Extracted watermark



Folder screenshot

IV. ANALYSIS

After the completion of the project we inferred that using digital watermarking for enhanced signal protection not only keeps the original quality of the sound unaltered but also encrypts the signal. On top of that the watermark stays hidden therefore it preserves the integrity of the encryption, making this approach a very fruitful form of audio encryption.

Also we drew the inference that using this form of audio encryption solves all the 4 disadvantages mentioned in the overlapping method of encryption. For starters not anyone who gets the encrypted audio can decode the audio as the sender has the numbers required to decrypt and extract the watermark. Secondly the sender has total control over the encryption which was not possible in the traditional encryption methods. And lastly the integrity of the original audio signal stays preserved and does not get out of sync or phase, so if we want to send the audio file to a 3rd party we can do so with the watermarked audio intact.

Contents

- clear memory & command window
- load data
- prepare host
- extract watermark
- show image

clear memory & command window

```

clc
clear
close all

```

load data

```

wm_sz = 20000; % watermark size
px_sz = wm_sz/8; % number of pixels
im_sz = sqrt(px_sz); % image size
host_new = audioread('host_new.wav'); % new (watermarked) host signal
host_new = uint8(255*(host_new + 0.5)); % double [-0.5 +0.5] to 'uint8' [0 255]

```

prepare host

```

host_bin = dec2bin(host_new, 8); % binary host [n 8]

```

extract watermark

```

wm_bin_str = host_bin(1:wm_sz, 8);
wm_bin = reshape(wm_bin_str, px_sz, 8);
wm_str = zeros(px_sz, 1, 'uint8');
for i = 1:px_sz % extract water mark from the first plane of host
    wm_str(i, :) = bin2dec(wm_bin(i, :)); % Least Significant Bit (LSB)
end
wm = reshape(wm_str, im_sz, im_sz);


```

show image

```

imshow(wm)

```



HTML output

The above HTML snippet shows the live script ONLY TO THE RECEIVER and can be accessed only if he/she wishes to see it and test the authenticity of the encryption.

V. CODE BLOCK

The code used to do the regression is attached below:

Audio Function

```
4
function [J, grad] = audioFunction(theta, X, y)
m = length(y);
J = 0;
grad = zeros(size(theta));
h_theta = sigmoid(X*theta);
J = (1 / m) * ((-y' * log(h_theta)) - (1 - y)' * log(1 - h_theta));
grad = (1 / m) * (h_theta - y)' * X;

End
```

Map Feature

```
4
function out = mapFeature(X1, X2)
degree = 6;
out = ones(size(X1(:,1)));
for i = 1:degree
    for j = 0:i
        out(:, end+1) = (X1.^(i-j)).*(X2.^j);
    end
end

End
```

Plot Data

```
5
function plotData(X, y)
% PLOTDATA Plots the data points X and y into a new figure.
% It will plots the data points with + for the positive values
% and o for the negative values. X is assumed to be a Mx2 matrix.

figure; hold on;

7
% Plotting the positive and negative values on a
% 2D plot, using the option 'k+' for the positive
% values and 'ko' for the negative values.
%

pos = find(y == 1); neg = find(y == 0);

plot(X(pos, 1), X(pos, 2), 'k+', 'LineWidth', 2, ...
     'MarkerSize', 7);
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', ...
     'MarkerSize', 7);

hold off;

End
```


Plot Decision Boundary

```
function plotDecisionBoundary(theta, X, y)
% PLOT DECISION BOUNDARY Plots the data points X and y into a new
figure with
% the decision boundary defined by theta
% plots the data points with + for the
% positive values and o for the negative values. X is assumed to be
% a either
% 1) Mx3 matrix, where the first column is an all-ones column for the
% intercept.
% 2) MxN, N>3 matrix, where the first column is all-ones

% Plot Data
plotData(X(:,2:3), y);
hold on

if size(X, 2) <= 3

    % Only need 2 points to define a line, so choose two endpoints
    plot_x = [min(X(:,2))-2, max(X(:,2))+2];

    % Calculate the decision boundary line
    plot_y = (-1./theta(3)).*(theta(2).*plot_x + theta(1));

    % Plot, and adjust axes for better viewing
    plot(plot_x, plot_y)

    % Legend, specific for the distribution
    legend('Disturbed', 'Undisturbed', 'Decision Boundary')
    axis([30, 100, 30, 100])
else
    % grid range
    u = linspace(-1, 1.5, 50);
    v = linspace(-1, 1.5, 50);

    z = zeros(length(u), length(v));
    % Evaluate z = theta*x over the grid
    for i = 1:length(u)
        for j = 1:length(v)
            z(i,j) = mapFeature(u(i), v(j))*theta;
        end
    end
    z = z'; % important to transpose z before calling contour

    9 Plot z = 0
    contour(u, v, z, [0, 0], 'LineWidth', 2)
end
hold off

End
```

Predict

```
function p = predict(theta, X)
m = size(X, 1);
p = zeros(m, 1);
p = sigmoid(X*theta) >= 0.5;
end
```

Regression Watermarking

```
clear ; close all; clc

data = load('ex2data1.txt');
X = data(:, [1, 2]); y = data(:, 3);

%% ===== Segmentation Plotting =====

fprintf(['Plotting Audio Embedding data\n']);

plotData(X, y);
hold on;

xlabel('Audio Segmentation')
ylabel('Image particle distribution')

legend('Undisturbed', 'Disturbed')
hold off;

fprintf('\nStage 1 completed. Press enter to continue.\n');
pause;
%% ===== Logistic Regression =====
%2 Setup the data matrix
[m, n] = size(X);

% Add intercept term to x and X_test
X = [ones(m, 1) X];

% Initialize fitting parameters
initial_theta = zeros(n + 1, 1);

% Compute and display initial and final values of the gradient
[3]
[cost, grad] = audioFunction(initial_theta, X, y);

fprintf('Difference in value at initial theta (zeros): %f\n', cost);
fprintf('Expected difference (approx): 0.693\n');
fprintf('Gradient at initial theta (zeros): \n');
fprintf(' %f \n', grad);
fprintf('Expected gradients (approx):\n -0.1000\n -12.0092\n -11.2628\n');

% Compute and display gradient with non-zero theta
test_theta = [-24; 0.2; 0.2];
[cost, grad] = audioFunction(test_theta, X, y);

fprintf('\nValue at test theta: %f\n', cost);
```

```

fprintf('Expected (approx): 0.218\n');
fprintf('Gradient at test theta: \n');
fprintf(' %f \n', grad);
fprintf('Expected gradients (approx):\n 0.043\n 2.566\n 2.647\n');

fprintf('\nStage 2 complete. Press enter to continue.\n');
pause;

%% ===== Part 3: Optimizing using fminunc =====
% Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);

% 3 Run fminunc to obtain the optimal theta
[theta, cost] = ...
    fminunc(@(t)(audioFunction(t, X, y)), initial_theta, options);

% Print theta to screen
fprintf('Value at theta found by fminunc: %f\n', cost);
fprintf('Expected (approx): 0.203\n');
fprintf('theta: \n');
fprintf(' %f \n', theta);

fprintf('Expected theta (approx):\n');

fprintf(' -25.161\n 0.206\n 0.201\n');

% Plot Boundary
plotDecisionBoundary(theta, X, y);

% Put some labels
hold on;
% Labels and Legend
xlabel('Original Score score')
ylabel('Post Watermarking score')

% Specified in plot order
legend('Undisturbed', 'Disturbed')
hold off;

fprintf('\nStage 3 complete. Press enter to continue.\n');
pause;

%% ===== Part 4: Training Set =====
prob = sigmoid([1 45 85] * theta);
fprintf(['For a disturbed value between 45 and 85, we predict the\n',
    'watermarking as successful ' ...
    'probability of %f\n'], prob);
fprintf('Expected value: 0.775 +/- 0.002\n\n');

% Compute accuracy on our training set
p = predict(theta, X);

fprintf('Train Accuracy: %f\n', mean(double(p == y)) * 100);

fprintf('Expected accuracy (approx): 89.0\n');

```

```
fprintf('\n');
```

Sigmoid

```
function g = sigmoid(z)
g = zeros(size(z));
g = 1 ./ (1+exp(-z));
end
```

Inserting Watermark

```
clc
clear
close all
[host, f] = audioread ('Original_Audio.mp3');
host      = uint8(255*(host + 0.5));
wm        = imread('Watermark.png');
[r, c]    = size(wm);
wm_l      = length(wm(:))*8;

if length(host) < (length(wm(:))*8)
    disp('your image pixel is not enough')
else

    host_bin = dec2bin(host, 8);
    wm_bin   = dec2bin(wm(:), 8);

    wm_str   = zeros(wm_l, 1);

    for j = 1:8
        for i = 1:length(wm(:))
            ind = (j-1)*length(wm(:)) + i;
            wm_str(ind, 1) = str2double(wm_bin(i, j));
        end
    end

    for i = 1:wm_l
        host_bin(i, 8) = dec2bin(wm_str(i));
    end

    host_new = bin2dec(host_bin);
    host_new = (double(host_new)/255 - 0.5);
    audiowrite('Watermarked_Audio.wav', host_new, f)
end
```

Extracting Watermark

```
5
clc
clear
close all
wm_sz = 20000;
px_sz = wm_sz/8;
im_sz = sqrt(px_sz);
host_new = audioread ('Watermarked_Audio.wav');
5
host_new = uint8(255*(host_new + 0.5));
```

```

5)st_bin = dec2bin(host_new, 8);
wm_bin_str = host_bin(1:wm_sz, 8);
wm_bin = reshape(wm_bin_str, px_sz, 8);
wm_str = zeros(px_sz, 1, 'uint8');
5)for i = 1:px_sz
wm_str(i, :) = bin2dec(wm_bin(i, :));
end
wm = reshape(wm_str, im_sz, im_sz);
imshow(wm)

```

Function Executable (READ)

```

1
function [Y,FS,NBITS,encoding_info,tag_info,out] = mp3read(FILE)
%MP3READ Read MP3 (".mp3") sound file.
% Y = MP3READ(FILE) reads a MP3 file specified by the string FILE,
% returning the sampled data in Y. Amplitude values are in the range
% [-1,+1].
%
% [Y,FS,NBITS,encoding_info,ID3v1_tag_info] = MP3READ(FILE) returns
% the sample rate (FS) in Hertz
% and the number of bits per sample (NBITS) used to encode the
% data in the file.
%
% 'encoding_info' is a string containing information about the mp3
% encoding used
%
% 'ID3v1_tag_info' is a string containing the tag information of the
% file
% (only ID3v1 tag supported in this version)
%
%
% Supports two channel or mono encoded data, with up to 16 bits per
% sample.
%
% See also MP3WRITE, WAVWRITE, AUREAD, AUWRITE.
a = length(FILE);
if a >= 4
    exten = FILE(a-3:a);
    if exten ~= '.mp3'
        FILE = strcat(FILE, '.mp3');
    end
end
if a <= 3
    FILE = strcat(FILE, '.mp3');
end
if exist(FILE) ~= 2
    error('File not Found')
end
%%%%%% Location of the ".exe" Files
s = which('mp3read.m');
ww = findstr('mp3read.m',s);
location = s(1:ww-2);
%%%%Temporary file%%%%%%%%
tmpfile = ['temp.wav'];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Data Decoding using "mpg123.exe"%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

[stat,raw_info] = dos([location,'\\mpg123', ' -w ', tmpfile, ' ',
'',FILE,'']);
data_init = findstr(raw_info,'MPEG');
blocks = findstr(raw_info,['0:']);
if raw_info(blocks+3) == '0'
    error('Error while decoding file. File may be corrupted')
end
[Y,FS,NBITS] = wavread(tmpfile); % Load the data and delete
temporary file
delete(tmpfile);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tag_info_start = strfind(raw_info,'Title');
tag_info_end = (strfind(raw_info,'Playing MPEG'))-1;
tag_info = raw_info(tag_info_start:tag_info_end);
encoding_info = raw_info(data_init(3):data_init(3)+53);

```

Function Executable (WRITE)

```

1
function mp3write(varargin)
%MP3WRITE Write MP3 (".mp3") sound file.
% MP3WRITE(Y,FS,NBITS,MP3FILE,ENCODING) writes data Y to a MP3
% file specified by the filename name MP3FILE, with a sample rate
% of FS Hz and with NBITS number of bits. Stereo data should
% be specified as a matrix with two columns.
% ENCODING must be specified as an integer number from 1 to 5
%
% 1 = Fixed bit rate 128kbs encoding.
% 2 = Fixed bit rate jstereo 128kbs encoding, high quality
(recommended).
% 3 = Average bitrate 112kbs encoding.
% 4 = Fast encode, low quality.
% 5 = Variable bitrate.
%
% Y,FS and NBITS are mandatory fields. If MP3 FILE is not defined the
file
% name will be 'Default_name.mp3'. If ENCODING is not defined
encoding
% type '2' will be used by default.
%
% See also MP3READ, WAVREAD, WAVWRITE.
if length(varargin) < 3 | length(varargin) > 5
    error('Unsupported number of argument inputs')
end
Y = varargin{1};
FS = varargin{2};
NBITS = varargin{3};
if NBITS~=8 & NBITS~=16 & NBITS~=24 & NBITS~=32
    error('Unsupported bit depth')
end
if length(varargin) >= 4
    MP3FILE = varargin{4};
    if ischar(MP3FILE) ~= 1
        error('File name is not a string')
    end
else
    MP3FILE = 'Default_name.mp3';

```

```

        disp('File name = Default_name.mp3')
    end

    if isempty(findstr(MP3FILE, '.mp3'))
        MP3FILE = strcat(MP3FILE, '.mp3');
    end

    if length(varargin) == 5
        ENCODING = varargin{5};
    else
        ENCODING = '2';
        disp('Fixed bit rate, joint-stereo, 128 kb/s encoding')
    end

    s = which('mp3write.m');
    ww = findstr('mp3write.m', s);
    lame = s(1:ww-2);
    wavwrite(Y, FS, NBITS, strcat(lame, '\temp.wav'));
    tmpfile = strcat(lame, '\temp.wav');
    MP3FILE = strcat(pwd, '\', MP3FILE);
    ENCODING = num2str(ENCODING);
    switch ENCODING
        case {'1'}
            cmd = [lame, '\lame', ' --quiet', ' ', tmpfile, ' ', MP3FILE];
        case {'2'}
            cmd = [lame, '\lame', ' --quiet', ' -b 128 ', tmpfile, ' ',
'MP3FILE];
        case {'3'}
            cmd = [lame, '\lame', ' --quiet', ' --abr 112 ', tmpfile, ' ',
'MP3FILE];
        case {'4'}
            cmd = [lame, '\lame', ' --quiet', ' -f ', tmpfile, ' ', MP3FILE];
        case {'5'}
            cmd = [lame, '\lame', ' --quiet', ' -h ', ' -V ', tmpfile, ' ',
'MP3FILE];
        otherwise
            error('Encoding parameters not supported')
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Data Encoding using "Lame.exe" %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    dos(cmd);
    % Delete temporary file
    delete(tmpfile);

```

NAG5

ORIGINALITY REPORT

39%

SIMILARITY INDEX

39%

INTERNET SOURCES

2%

PUBLICATIONS

33%

STUDENT PAPERS

PRIMARY SOURCES

1

www.mathworks.fr

Internet Source

14%

2

www.codenong.com

Internet Source

7%

3

www.cnblogs.com

Internet Source

6%

4

blog.csdn.net

Internet Source

3%

5

github.com

Internet Source

3%

6

en.wikipedia.org

Internet Source

2%

7

Submitted to University of Durham

Student Paper

2%

8

deepnote.com

Internet Source

2%

9

www.voidcn.com

Internet Source

1%

Exclude quotes On

Exclude matches < 20 words

Exclude bibliography On