Matlab Project #73 report

Long Nguyen

April 19, 2025

1 Project requirements

Build a function `fun` that takes as input two vectors `u` and `v` and as output provides the vector `w` which contains in each entry the max between the same entry position of the two vectors.

Download 2 audio signals and load them into Matlab. Use the function `fun` applied to the first 0.3 seconds of the **first audio channel** of the 2 signals. Plot the original 2 audio signals and the one obtained using the function in a single figure using subplots.

Generate a report in pdf containing the results obtained, including all the codes created.

2 Results

Main function

```
% Add submodule paths
addpath('utils');
addpath('static');

audio1 = read_audio("static/clair-de-lune.mp3", 33, 33.3, false);
audio2 = read_audio("static/moonlight-sonata.mp3", 33, 33.3, false);

blended_audio = fun(audio1, audio2, "truncate");

sound_list = {audio1, audio2, blended_audio};

audio_plot(sound_list);
```

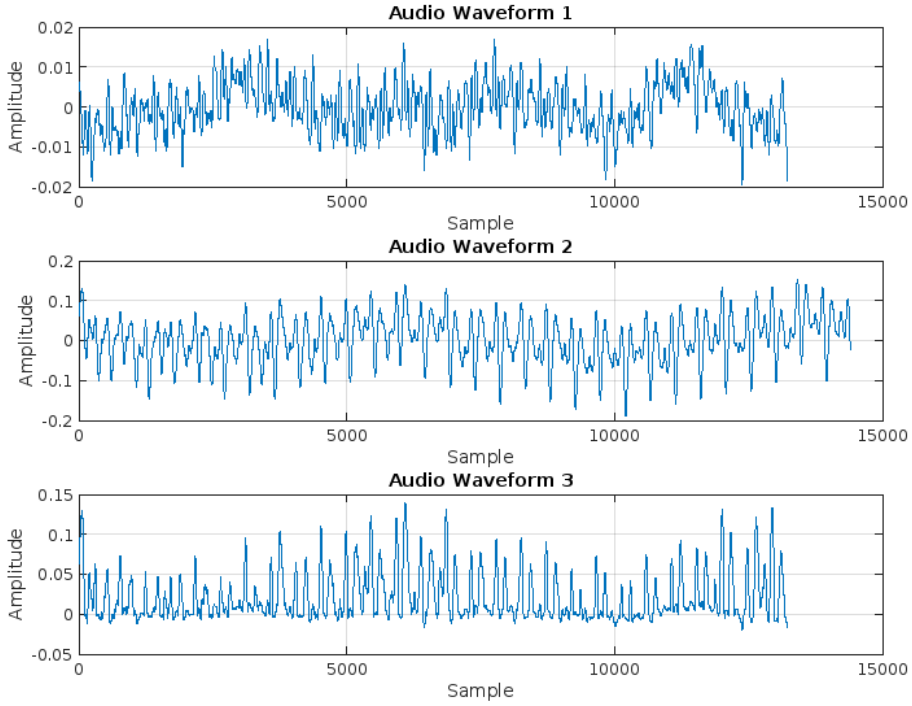
Function output

```
"File Name: "    "clair-de-lune"    ".mp3"

Number of Channels: 2
Sample Rate: 44100
Duration: 0.30002 seconds
    "File Name: "    "moonlight-sonata"    ".mp3"

Number of Channels: 2
Sample Rate: 48000
Duration: 0.30002 seconds
```

Output graphs



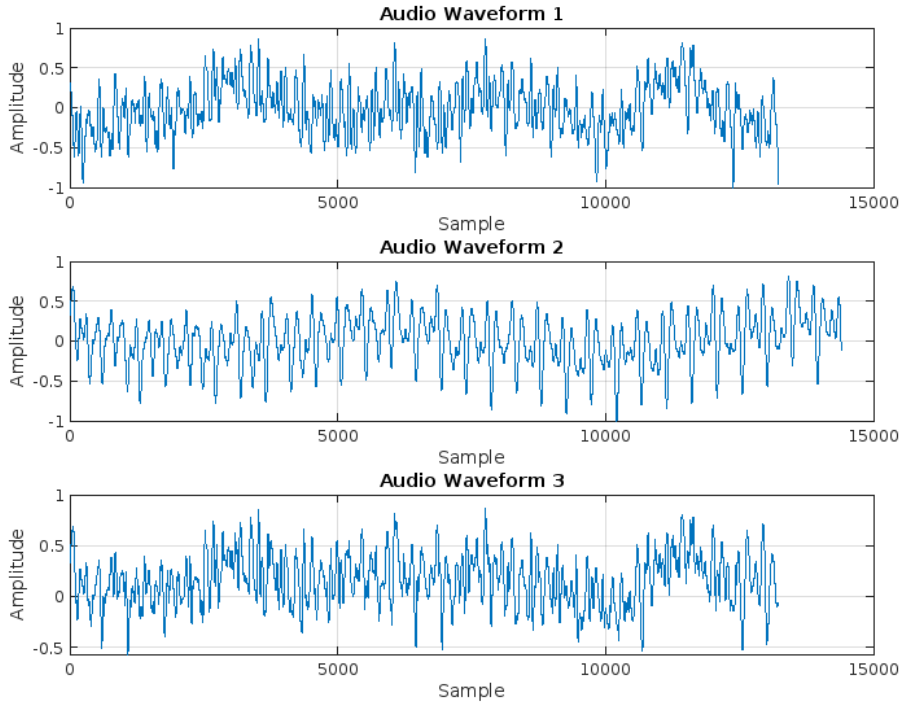
3 Extra funcitonalities

3.1 Function read_audio

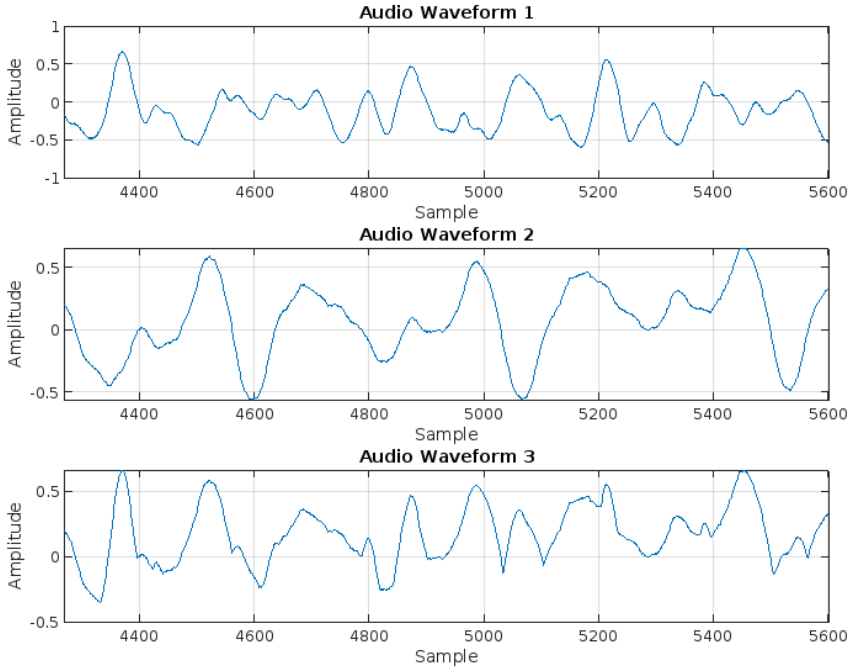
This function has the following extra functionalities:

- The start and stop time to process the audio can be adjusted. In the main file, I am getting data from second 33 till 33.3 of both audio files.
- The last parameter is for audio normalization if specified as true.

Ouput of the function if normalization is set to true



And when zoomed in:



3.2 Source control

Soucre control is enabled for this project and it is located at
https://github.com/longieee/univaq-intro-to-matlab-final-project

3.3 Unit tests

Unit tests are also written for the functions of this project. The unit test code is located inside the `tests` folder.

3.4 Github workflow

A simple Github workflow is also created for this project. The workflow is located at `.github/workflows/matlab_test.yml` and it is triggered on every push to the main branch to automatically run the unit tests.

A possible real-life application: The workflow runs the unit tests and if they pass, it will create a new release with the version number incremented by 1.

4 Listing of all source code

Folder structure

```
.
|____LICENSE
|____tests
| |____test_read_audio.m
| |____test_audio_plot.m
| |____test_extendVectors.m
| |____test_fun.m
|____utils
| |____fun.m
| |____read_audio.m
| |____extendVectors.m
| |____audio_plot.m
|____report.synctex.gz
|____README.md
|____report.tex
|____rendered
| |____audio_result_no_normalize.png
| |____audio_result_normalized.png
| |____project-description.pdf
| |____audio_result_normalized_zoomed.png
|____.gitignore
|____main.m
|____static
| |____moonlight-sonata.mp3
| |____clair-de-lune.mp3
|____.github
| |____workflows
| | |____matlab_tests.yml
|____report.pdf
```

Main file (main.m)

```
% Add submodule paths
addpath('utils');
addpath('static');

audio1 = read_audio("static/clair-de-lune.mp3", 33, 33.3, false);
```

```
audio2 = read_audio("static/moonlight-sonata.mp3", 33, 33.3, false);
blended_audio = fun(audio1, audio2, "truncate");
sound_list = {audio1, audio2, blended_audio};
audio_plot(sound_list);
```

Utilitiy functions

Function fun

```
function w = fun(u,v,method)
    %FUN takes as input two vectors 'u' and 'v' and as output provides
    % the vector 'w' which contains in each entry the max between
    % the same entry position of the two vectors.
    %   In the case the length of the 2 input vectors are different, t
    %   argument 'method' will decide how to deal with the situation
    %
    %   - If method="truncate", then the output's length is the length
    %   the shorter vector
    %   - If method="fill", then the output's length is the length of
    %   longer vector, and the missing part of the shorter vector is
    %   considered 0
    %
    %   Inputs:
    %       v1 - First input vector (row or column)
    %       v2 - Second input vector (row or column)
    %   Output:
    %       vector 'w' as described above

    % Input validation
    if nargin < 2
        error('fun:NotEnoughInputs', 'At least TWO input argument is r
    end

    % Case non-vectors
    if ~isvector(u)
        error('fun:InvalidInput', 'Input argument must be a vector, re
    end

    if ~isvector(v)
        error('fun:InvalidInput','Input argument must be a vector, rec
    end

    % Case empty vectors
    if isempty(u) || isempty(v)
        error('fun:InvalidInput', 'Input arguments must not be empty')
    end
```

```
    % Provide default value for method argument
    if nargin < 3
        method="truncate";
    end
    % and then check its validity
    if method~="truncate" && method~="fill"
        error('fun:InvalidMethod', 'Method must be one of {truncate, f
    end

    % Main logic
    % Easy case: Same length
    if length(u) == length(v)
        w = max(u,v);
    % 2 vectors are not of the same length
    else
        if method=="truncate"
            min_length = min(length(u),length(v));
            w = max( ...
                u(1:min_length), ...
                v(1:min_length) ...
                );
        else
            [u2,v2] = extendVectors(u,v);
            w = max(u2,v2);
        end
    end
end
```

Function read_audio

```
function channel1 = read_audio(file_path, start, stop, normalize)
    %READ_AUDIO Read an MP3 file and return the first channel
    %   channel1 = read_audio(file_path, start, stop, normalize) reads
    %   by file_path from the start sample to the stop sample and retu
    %   If normalize is true, the audio data is normalized to the rang
    %
    %   Inputs:
    %       file_path - Path to the MP3 file
    %       start - Optional start sample (default: 1)
    %       stop - Optional stop sample (default: end of file)
```

```
%       normalize - Optional boolean to indicate if normalization
%   Outputs:
%       channel1 - First channel of the audio data
%   Example usage:
%       channel1 = read_audio('path/to/audio.mp3', 1000, 5000, tru

% Handle optional arguments
if nargin < 2 || isempty(start)
    start = 1;
end

if nargin < 3 || isempty(stop)
    info = audioinfo(file_path);
    stop = info.TotalSamples;
end

if nargin < 4
    normalize = false;
end

% Read audio with specified range
% Convert time in seconds to sample indices if needed
info = audioinfo(file_path);
fs = info.SampleRate;

% Convert start form seconds to samples
start_sample = max(1, round(start * fs));
% Convert stop from seconds to samples
stop_sample = min(info.TotalSamples, round(stop * fs));

[audio_data, fs] = audioread(file_path, [start_sample, stop_sample
channel1 = audio_data(:, 1);

if normalize
    % Normalize the audio data to the range [-1, 1]
    channel1 = channel1 / max(abs(channel1));
end

% Display the file name
[~, file_name, ext] = fileparts(file_path);
disp(['File Name: ', file_name, ext]);
```

```
    % Display the number of channels
    num_channels = size(audio_data, 2);
    disp(['Number of Channels: ', num2str(num_channels)]);
    % Display the sample rate
    disp(['Sample Rate: ', num2str(fs)]);
    % Display the duration of the audio
    duration = length(channel1) / fs;
    disp(['Duration: ', num2str(duration), ' seconds']);
end
```

Function extendVectors

```
function [v1, v2] = extendVectors(v1, v2)
    % EXTENDVECTORS Extends the shorter vector to match the longer one
    %
    %   [v1, v2] = EXTENDVECTORS(v1, v2) takes two vectors and extends
    %   shorter one by padding with zeros to match the length of the l
    %   The function preserves the original orientation (row or column
    %
    %   Inputs:
    %       v1 - First input vector (row or column)
    %       v2 - Second input vector (row or column)
    %
    %   Outputs:
    %       v1 - First vector, extended if necessary
    %       v2 - Second vector, extended if necessary
    %
    %   Example usage:
    %       v1 = [1, 2, 3];
    %       v2 = [4, 5, 6, 7, 8];
    %       [v1_ext, v2_ext] = extendVectors(v1, v2);
    %       % v1_ext: [1 2 3 0 0], v2_ext: [4 5 6 7 8]
    %
    %       v1 = [1; 2];
    %       v2 = [3; 4; 5; 6];
    %       [v1_ext, v2_ext] = extendVectors(v1, v2);
    %       % v1_ext: [1; 2; 0; 0], v2_ext: [3; 4; 5; 6]

    % Validate inputs - consider empty arrays as valid vectors
    if (~isempty(v1) && ~isvector(v1)) || (~isempty(v2) && ~isvector(v
        error('Both inputs must be vectors.');
```

```
end

% Determine the orientation (row or column) for empty vectors
if isempty(v1)
    % If v1 is empty, use the orientation of v2, or default to row
    if isrow(v2)
        v1 = zeros(1, 0);  % Empty row vector
    elseif iscolumn(v2)
        v1 = zeros(0, 1);  % Empty column vector
    else
        v1 = zeros(1, 0);  % Default to row vector
    end
end

if isempty(v2)
    % If v2 is empty, use the orientation of v1, or default to row
    if isrow(v1)
        v2 = zeros(1, 0);
    elseif iscolumn(v1)
        v2 = zeros(0, 1);
    else
        v2 = zeros(1, 0);
    end
end

% Get lengths
len1 = length(v1);
len2 = length(v2);
maxLen = max(len1, len2);

% Extend the shorter vector while maintaining shape
if len1 < maxLen
    if isrow(v1) || (isempty(v1) && isrow(v2))
        v1(1, maxLen) = 0;  % Extend row vector
    else
        v1(maxLen, 1) = 0;  % Extend column vector
    end
end

if len2 < maxLen
    if isrow(v2) || (isempty(v2) && isrow(v1))
```

```
            v2(1, maxLen) = 0;  % Extend row vector
        else
            v2(maxLen, 1) = 0;  % Extend column vector
        end
    end
end
```

Function audio_plot

```
function audio_plot(sound_list)
    %AUDIO_PLOT Takes in a list of audio files and plots their waveforn
    % This function is intended to plot the waveforms before and after
    % written inside the rest of the script.
    % The "audios" are essentially vectors of the first channel of the
    % data. The function takes in a list of audio files or just a sing
    % plots their waveforms.
    %   Inputs:
    %       sound_list - A cell array of strings, each string is a pat
    %   Outputs:
    %       None, but the function will display the waveforms of the a
    %   Example usage:
    %       audio_plot({vector1, vector2, vector3});

    % Check if the cell array is empty
    if isempty(sound_list)
        error('audio_plot:InvalidInput', 'Input cell array is empty.')
    end

    % If the input is just 1 vector, convert it to a cell array
    if ~iscell(sound_list) && isvector(sound_list)
        sound_list = {sound_list};
    end

    % Check if the input is a cell array
    if ~iscell(sound_list)
        error('audio_plot:InvalidInput', 'Input must be a cell array o
    end

    % Determine number of audio files
    cell_size = size(sound_list);
    num_sounds = cell_size(2);
```

```
    % Create a figure with subplots for each audio file
    figure('Name', 'Audio Waveforms', 'NumberTitle', 'off');

       % Create a figure with subplots for each audio file
    figure('Name', 'Audio Waveforms', 'NumberTitle', 'off');
    % Initialize an array to store axes handles
    ax = zeros(num_sounds, 1);

    % Loop through each audio file and plot its waveform
    for i = 1:num_sounds
        ax(i) = subplot(num_sounds, 1, i);

        % Get the current audio data
        audio_data = sound_list{1,i};

        % Plot the waveform
        plot(audio_data);

        % Add title and labels
        title(['Audio Waveform ' num2str(i)]);
        xlabel('Sample');
        ylabel('Amplitude');

        % Add grid for better visualization
        grid on;
    end

    % Link all axes to share control
    linkaxes(ax, 'x');

    % Adjust the spacing between subplots
    set(gcf, 'Position', [100, 100, 800, 200*num_sounds]);
end
```

Unit tests

Unit test for fun

```
classdef test_fun < matlab.unittest.TestCase
    % TEST_FUN Unit tests for the function 'fun.m'.
    %
    % This test suite verifies the correctness of the function 'fun' u
    % the MATLAB Unit Testing Framework.

    properties (TestParameter)
        % Define test parameters for vector inputs
        u = { [1, 3, 5], [1; 3; 5], [1, 3, 5], [1; 3; 5], [], [5] };
        v = { [2, 2, 6], [2; 2; 6], [2, 2], [2; 2], [1, 2, 3], [10] };
        method = { "truncate", "fill" };
    end

    methods (Test)
        function testEqualLengthVectors(testCase)
            % Test case where vectors have the same length
            u = [1, 3, 5];
            v = [2, 2, 6];
            expected = [2, 3, 6];
            actual = fun(u, v);
            testCase.verifyEqual(actual, expected);
        end

        function testEqualLengthColumnVectors(testCase)
            % Test case for equal-length column vectors
            u = [1; 3; 5];
            v = [2; 2; 6];
            expected = [2; 3; 6];
            actual = fun(u, v);
            testCase.verifyEqual(actual, expected);
        end

        function testTruncateMode(testCase)
            % Test truncate mode with different length vectors
            u = [1, 3, 5];
            v = [2, 2];
            expected = [2, 3];
```

```
    actual = fun(u, v, "truncate");
    testCase.verifyEqual(actual, expected);
end

function testFillMode(testCase)
    % Test fill mode with different length vectors
    u = [1, 3, 5];
    v = [2, 2];
    expected = [2, 3, 5]; % Missing parts filled with zero
    actual = fun(u, v, "fill");
    testCase.verifyEqual(actual, expected);
end

function testFillModeColumnVectors(testCase)
    % Test fill mode with column vectors
    u = [1; 3; 5];
    v = [2; 2];
    expected = [2; 3; 5];
    actual = fun(u, v, "fill");
    testCase.verifyEqual(actual, expected);
end

function testInvalidInputNonVector(testCase)
    % Test for non-vector input, should throw an error
    testCase.verifyError(@() fun([1 2; 3 4], [5 6]), 'fun:Inva
end

function testInvalidMethodArgument(testCase)
    % Test for invalid method argument, should throw an error
    testCase.verifyError(@() fun([1, 2, 3], [4, 5, 6], "wrong_
end

function testSingleElementVectors(testCase)
    % Test case with single-element vectors
    u = [5];
    v = [10];
    expected = [10];
    actual = fun(u, v);
    testCase.verifyEqual(actual, expected);
end
```

```
        function testEmptyVectorFillMode(testCase)
            % Test empty vector with fill mode
            testCase.verifyError(@() fun([], [5 6]), 'fun:InvalidInput
        end
    end
end
```

Unit test for read_audio

```
classdef test_read_audio < matlab.unittest.TestCase
    % TEST_READ_AUDIO Unit tests for the function `read_audio.m`.
    %
    % This test suite verifies the correctness of the function `read_a
    % using the MATLAB Unit Testing Framework.

    properties
        TestFile
        TempDir
    end

    methods(TestMethodSetup)
        function createTestFile(testCase)
            % Create a temporary directory and synthetic audio file fo
            testCase.TempDir = tempname;
            mkdir(testCase.TempDir);

            % Create a simple sine wave audio signal
            fs = 44100;  % Sample rate
            t = 0:1/fs:2;  % 2 seconds
            y = sin(2*pi*440*t)';  % 440 Hz sine wave
            stereo_y = [y, y*0.5];  % Create stereo by duplicating wit

            % Save as a temporary WAV file
            testCase.TestFile = fullfile(testCase.TempDir, 'sample.wav
            audiowrite(testCase.TestFile, stereo_y, fs);
        end
    end

    methods(TestMethodTeardown)
        function cleanupTestFile(testCase)
            % Clean up temporary files
```

```
            if exist(testCase.TestFile, 'file')
                delete(testCase.TestFile);
            end
            if exist(testCase.TempDir, 'dir')
                rmdir(testCase.TempDir, 's');
            end
        end
    end

    methods(Test)
        function testBasicReading(testCase)
            % Test basic reading functionality
            channel1 = read_audio(testCase.TestFile);
            testCase.verifyTrue(isvector(channel1), 'Output should be
            testCase.verifyGreaterThan(length(channel1), 0, 'Output sh
        end

        function testRangeReading(testCase)
            % Test reading with sample range
            start = 100;
            stop = 500;
            channel1 = read_audio(testCase.TestFile, start, stop);
            testCase.verifyEqual(length(channel1), stop-start+1, 'Outp
        end

        function testNormalization(testCase)
            % Test normalization option
            channel1 = read_audio(testCase.TestFile, 1, 1000, true);
            testCase.verifyLessThanOrEqual(max(abs(channel1)), 1.0, 'N

            % Check if it's actually normalized to max amplitude
            if ~isempty(channel1) && any(abs(channel1) > 0)
                testCase.verifyEqual(max(abs(channel1)), 1.0, 'Output
            end
        end

        function testDefaultArguments(testCase)
            % Test default arguments
            % Full file reading
            full_channel = read_audio(testCase.TestFile);
```

```
        % With explicit defaults
        info = audioinfo(testCase.TestFile);
        channel_with_defaults = read_audio(testCase.TestFile, 1, i

        testCase.verifyEqual(full_channel, channel_with_defaults,
    end

    function testTimeBasedInput(testCase)
        % Test time-based input (seconds)
        info = audioinfo(testCase.TestFile);
        fs = info.SampleRate;

        % Test with start time in seconds
        time_start = 0.5; % 0.5 seconds
        sample_start = round(time_start * fs);

        % Define a duration in samples and calculate equivalent ti
        sample_duration = 1000;
        time_duration = sample_duration / fs;

        % First call with time parameters
        channel1 = read_audio(testCase.TestFile, time_start, time_

        % Second call with equivalent sample parameters
        channel2 = read_audio(testCase.TestFile, sample_start, sam

        testCase.verifyEqual(channel1, channel2, 'Time-based start
    end

    function testChannelExtraction(testCase)
        % Test that only first channel is returned
        % Our test file is stereo - first channel is sin(2*pi*440*
        [audio_data, ~] = audioread(testCase.TestFile, [1, 1000]);
        expected_channel1 = audio_data(:, 1);

        % Compare with read_audio output
        actual_channel1 = read_audio(testCase.TestFile, 1, 1000);

        testCase.verifyEqual(actual_channel1, expected_channel1, '
    end
end
```

end

Unit test for extendVectors

```
classdef test_extendVectors < matlab.unittest.TestCase
    % TEST_EXTENDVECTORS Unit tests for the function 'extendVectors.m'
    %
    % This test suite verifies the correctness of the function 'extend
    % using the MATLAB Unit Testing Framework.

    methods (Test)
        function testRowVectors(testCase)
            % Test with row vectors where first is shorter
            v1 = [1, 2, 3];
            v2 = [4, 5, 6, 7, 8];
            [v1_ext, v2_ext] = extendVectors(v1, v2);

            testCase.verifyEqual(length(v1_ext), length(v2_ext), 'Vect
            testCase.verifyEqual(v1_ext, [1, 2, 3, 0, 0], 'First vecto
            testCase.verifyEqual(v2_ext, [4, 5, 6, 7, 8], 'Second vect
            testCase.verifyTrue(isrow(v1_ext), 'Extended vector should
        end

        function testColumnVectors(testCase)
            % Test with column vectors where second is shorter
            v1 = [1; 2; 3; 4];
            v2 = [5; 6];
            [v1_ext, v2_ext] = extendVectors(v1, v2);

            testCase.verifyEqual(length(v1_ext), length(v2_ext), 'Vect
            testCase.verifyEqual(v1_ext, [1; 2; 3; 4], 'First vector s
            testCase.verifyEqual(v2_ext, [5; 6; 0; 0], 'Second vector
            testCase.verifyTrue(iscolumn(v2_ext), 'Extended vector sho
        end

        function testMixedOrientations(testCase)
            % Test with mixed orientations (row and column)
            v1 = [1, 2, 3];
            v2 = [4; 5; 6; 7];
            [v1_ext, v2_ext] = extendVectors(v1, v2);
```

```
        testCase.verifyEqual(length(v1_ext), length(v2_ext), 'Vect
        testCase.verifyTrue(isrow(v1_ext), 'First vector should ma
        testCase.verifyTrue(iscolumn(v2_ext), 'Second vector shoul
    end

    function testEqualLengthVectors(testCase)
        % Test with vectors of equal length
        v1 = [1, 2, 3];
        v2 = [4, 5, 6];
        [v1_ext, v2_ext] = extendVectors(v1, v2);

        testCase.verifyEqual(v1_ext, v1, 'Equal length vectors sho
        testCase.verifyEqual(v2_ext, v2, 'Equal length vectors sho
    end

    function testEmptyVectors(testCase)
        % Test with one empty vector
        v1 = [];
        v2 = [1, 2, 3];
        [v1_ext, v2_ext] = extendVectors(v1, v2);

        testCase.verifyEqual(length(v1_ext), length(v2_ext), 'Vect
        testCase.verifyEqual(v1_ext, [0, 0, 0], 'Empty vector shou
        testCase.verifyEqual(v2_ext, [1, 2, 3], 'Non-empty vector
    end

    function testNonVectorInput(testCase)
        % Test with non-vector input, should throw an error
        testCase.verifyError(@() extendVectors([1, 2; 3, 4], [5, 6
    end

    function testSingleElementVectors(testCase)
        % Test with single-element vectors
        v1 = [5];
        v2 = [10];
        [v1_ext, v2_ext] = extendVectors(v1, v2);

        testCase.verifyEqual(v1_ext, v1, 'Single-element vectors s
        testCase.verifyEqual(v2_ext, v2, 'Single-element vectors s
    end
end
```

end

Unit test for audio_plot

```
classdef test_audio_plot < matlab.unittest.TestCase
    % TEST_AUDIO_PLOT Unit tests for the function 'audio_plot.m'.
    %
    % This test suite verifies the correctness of the function 'audio_
    % using the MATLAB Unit Testing Framework.

    properties
        % Sample test data
        TestVector1
        TestVector2
        TestVector3
    end

    methods(TestMethodSetup)
        function createTestData(testCase)
            % Create sample test vectors for plotting
            testCase.TestVector1 = sin(linspace(0, 2*pi, 100))';
            testCase.TestVector2 = cos(linspace(0, 4*pi, 200))';
            testCase.TestVector3 = 0.5 * sin(linspace(0, 8*pi, 300))';
        end
    end

    methods(Test)
        function testSingleVector(testCase)
            % Test with a single vector input
            try
                audio_plot(testCase.TestVector1);
                fig = findobj('Type', 'figure', 'Name', 'Audio Waveforn
                testCase.verifyEqual(length(fig), 1, 'Should create on
                testCase.verifyEqual(length(findobj(fig, 'Type', 'axes
                close(fig);
            catch e
                close all;
                testCase.verifyFail(['Function failed with error: ', e
            end
        end
```

```
function testMultipleVectors(testCase)
    % Test with multiple vectors in cell array
    try
        audio_plot({testCase.TestVector1, testCase.TestVector2
        fig = findobj('Type', 'figure', 'Name', 'Audio Wavefor
        testCase.verifyEqual(length(fig), 1, 'Should create on
        testCase.verifyEqual(length(findobj(fig, 'Type', 'axes
        close(fig);
    catch e
        close all;
        testCase.verifyFail(['Function failed with error: ', e
    end
end

function testEmptyInput(testCase)
    % Test with empty input
    testCase.verifyError(@() audio_plot({}), 'audio_plot:Inval
end

function testNonVectorNonCellInput(testCase)
    % Test with non-vector and non-cell input
    testCase.verifyError(@() audio_plot(ones(3,3)), 'audio_plo
end

function testCellWithDifferentSizes(testCase)
    % Test with cell array containing vectors of different siz
    try
        audio_plot({testCase.TestVector1, testCase.TestVector2
        fig = findobj('Type', 'figure', 'Name', 'Audio Wavefor
        testCase.verifyEqual(length(fig), 1, 'Should create on
        testCase.verifyEqual(length(findobj(fig, 'Type', 'axes
        close(fig);
    catch e
        close all;
        testCase.verifyFail(['Function failed with error: ', e
    end
end

function testLabelingAndGridding(testCase)
    % Test that labels and grid are applied correctly
    try
```

```
            audio_plot({testCase.TestVector1});
            fig = findobj('Type', 'figure', 'Name', 'Audio Wavefor
            ax = findobj(fig, 'Type', 'axes');

            % Check that title exists using the Title property
            testCase.verifyNotEmpty(get(ax, 'Title'), 'Should have
            testCase.verifyNotEmpty(get(ax(1).Title, 'String'), 'T

            % Check that x-label and y-label exist
            testCase.verifyEqual(length(get(ax, 'XLabel')), 1, 'Sh
            testCase.verifyEqual(length(get(ax, 'YLabel')), 1, 'Sh

            % Check that grid is on
            testCase.verifyTrue(ax.XGrid, 'Grid should be on');
            testCase.verifyTrue(ax.YGrid, 'Grid should be on');

            close(fig);
        catch e
            close all;
            testCase.verifyFail(['Function failed with error: ', e
        end
    end

    function testFigureProperties(testCase)
        % Test that figure properties are set correctly
        try
            audio_plot({testCase.TestVector1, testCase.TestVector2
            fig = findobj('Type', 'figure', 'Name', 'Audio Wavefor
            pos = get(fig, 'Position');

            % Verify figure position includes the expected width a
            % proportional to the number of subplots
            testCase.verifyEqual(pos(3), 800, 'Figure width should
            testCase.verifyEqual(pos(4), 400, 'Figure height shoul

            close(fig);
        catch e
            close all;
            testCase.verifyFail(['Function failed with error: ', e
        end
    end
```

```
    end

    methods(TestMethodTeardown)
        function closeFigures(testCase)
            % Close any open figures after each test
            close all;
        end
    end
end
```

Other files

Github workflow

```
name: Run MATLAB Tests with Coverage

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  run-matlab-unittest:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v4

      - name: Set up MATLAB
        uses: matlab-actions/setup-matlab@v2
        with:
          release: R2024b

      - name: Run MATLAB tests and collect coverage
        uses: matlab-actions/run-tests@v2
        with:
          source-folder: utils
          select-by-folder: tests
          use-parallel: true
          test-results-pdf: test-results/results.pdf
          code-coverage-cobertura: code-coverage/coverage.xml
          logging-level: detailed
          output-detail: verbose

      - name: Upload Test result Report
        uses: actions/upload-artifact@v4
        with:
          name: test-results
          path: test-results/results.pdf
```

```
- name: Upload Code Coverage Report
  uses: actions/upload-artifact@v4
  with:
    name: code-coverage
    path: code-coverage/coverage.xml
```