Contents

1 Basic Test Results

```
ex2/
ex2/README.md
ex2/answer_q1.txt
ex2/answer_q2.txt
ex2/answer_q3.txt
ex2/sol2.py
Ex2 Presubmission Script

    Disclaimer
    ----------
    The purpose of this script is to make sure that your code is compliant
    with the exercise API and some of the requirements
    The script does not test the quality of your results.
    Don't assume that passing this script will guarantee that you will get
    a high grade in the exercise



=== Check Submission ===

README file:

tal.porezky
sol2.py
answer_q1.txt
answer_q2.txt
answer_q3.txt
Answer to q1:
change_rate function changes the rate of the audio file - but keeping the same
 number of samples. This causes the new audio file, when modified with ratio =
  2 to have an higher pitch.
change_samples changes the new audio file so it will have the same rate, but
with smaller amount of samples on the audio. In practice we cut the higher
frequencis so we still have the "main" frequncies within the audio file.
Answer to q2:
Sample created using resize_spectrogram is "gibrish" and it is not possible to
recognize any words in the audio file after it is fast forwarded.
Sample created using resize_vocoder is much better and words can be recognized
 in the fast forwarded audio file.
The difference between the two file is, of course, due to the phase fixing we
implemented in the resize_vocoder function.
Since resize_spectrogram does not handle the phases, there is an interference
between the waves of the over laping windows and the audio changes.

Answer to q3:
Actually, there should be no difference between the magnitude images.
I guess the difference is caused due to using my own a-bit less accurate DFT
and IDFT functions (and not numpy's fft and ifft).
Or maybe the difference is caused because there is no one definision for
derrevitae in the discrete case - we can use [-1, 1] or [-1, 0, 1] and get
different results although we try to do the same thing.

=== Load Student Library ===

Loading...
=== Section 1.1 ===

DFT and IDFT
=== Section 1.2 ===
```

```

2D DFT and IDFT
=== Section 2.1 ===

Fast foward by rate change
=== Section 2.2 ===

Fast forward using Fourier
=== Section 2.3 ===

Fast forward using Spectrogram
=== Section 2.4 ===

Fast forward using Spectrogram and phase vocoder
=== Section 3.1 ===

derivative using convolution
=== Section 3.2 ===

derivative using fourier


=== Presubmission Completed Successfully ===


    Please go over the output and verify that there were no failures / warnings.
    Remember that this script tested only some basic technical aspects of your implementation.
    It is your responsibility to make sure your results are actually correct and not only
    technically valid.
```

2 ex2/README.md

```
tal.porezky
sol2.py
answer_q1.txt
answer_q2.txt
answer_q3.txt
```

3 ex2/answer q1.txt

```
change_rate function changes the rate of the audio file - but keeping the same
 number of samples. This causes the new audio file, when modified with ratio =
  2 to have an higher pitch.
change_samples changes the new audio file so it will have the same rate, but
with smaller amount of samples on the audio. In practice we cut the higher
frequencis so we still have the "main" frequncies within the audio file.
```

4 ex2/answer q2.txt

```
Sample created using resize_spectrogram is "gibrish" and it is not possible to
recognize any words in the audio file after it is fast forwarded.
Sample created using resize_vocoder is much better and words can be recognized
 in the fast forwarded audio file.
The difference between the two file is, of course, due to the phase fixing we
implemented in the resize_vocoder function.
Since resize_spectrogram does not handle the phases, there is an interference
between the waves of the over laping windows and the audio changes.
```

5 ex2/answer q3.txt

```
Actually, there should be no difference between the magnitude images.
I guess the difference is caused due to using my own a-bit less accurate DFT
and IDFT functions (and not numpy's fft and ifft).
Or maybe the difference is caused because there is no one definision for
derrevitae in the discrete case - we can use [-1, 1] or [-1, 0, 1] and get
different results although we try to do the same thing.
```

6 ex2/sol2.py

```python
import numpy as np
import scipy.io.wavfile as wavfile
from scipy import signal
from scipy.ndimage.interpolation import map_coordinates
import scipy.signal
from imageio import imread
from skimage.color import rgb2gray
import matplotlib.pyplot as plt


# ------------------ helper functions ------------------


def stft(y, win_length=640, hop_length=160):
    fft_window = signal.windows.hann(win_length, False)

    # Window the time series.
    n_frames = 1 + (len(y) - win_length) // hop_length
    frames = [y[s:s + win_length] for s in np.arange(n_frames) * hop_length]

    stft_matrix = np.fft.fft(fft_window * frames, axis=1)
    return stft_matrix.T


def istft(stft_matrix, win_length=640, hop_length=160):
    n_frames = stft_matrix.shape[1]
    y_rec = np.zeros(win_length + hop_length * (n_frames - 1), dtype=np.float)
    ifft_window_sum = np.zeros_like(y_rec)

    ifft_window = signal.windows.hann(win_length, False)[:, np.newaxis]
    win_sq = ifft_window.squeeze() ** 2

    # invert the block and apply the window function
    ytmp = ifft_window * np.fft.ifft(stft_matrix, axis=0).real

    for frame in range(n_frames):
        frame_start = frame * hop_length
        frame_end = frame_start + win_length
        y_rec[frame_start: frame_end] += ytmp[:, frame]
        ifft_window_sum[frame_start: frame_end] += win_sq

    # Normalize by sum of squared window
    y_rec[ifft_window_sum > 0] /= ifft_window_sum[ifft_window_sum > 0]
    return y_rec


def phase_vocoder(spec, ratio):
    time_steps = np.arange(spec.shape[1]) * ratio
    time_steps = time_steps[time_steps < spec.shape[1]]

    # interpolate magnitude
    yy = np.meshgrid(np.arange(time_steps.size), np.arange(spec.shape[0]))[1]
    xx = np.zeros_like(yy)
    coordiantes = [yy, time_steps + xx]
    warped_spec = map_coordinates(np.abs(spec), coordiantes, mode='reflect', order=1).astype(np.complex)

    # phase vocoder
    # Phase accumulator; initialize to the first sample
    spec_angle = np.pad(np.angle(spec), [(0, 0), (0, 1)], mode='constant')
```

```
    phase_acc = spec_angle[:, 0]

    for (t, step) in enumerate(np.floor(time_steps).astype(np.int)):
        # Store to output array
        warped_spec[:, t] *= np.exp(1j * phase_acc)

        # Compute phase advance
        dphase = (spec_angle[:, step + 1] - spec_angle[:, step])

        # Wrap to -pi:pi range
        dphase = np.mod(dphase - np.pi, 2 * np.pi) - np.pi

        # Accumulate phase
        phase_acc += dphase

    return warped_spec


def read_image(file_name, representation):
    """
    this function read an image from a given path and represent it in rgb
    or grayscale, according to the user request
    :param file_name: the path of the image
    :param representation: 1 - grayscale 2 - rgb
    :return: an rgb or grayscale image
    """
    image = imread(file_name)
    if representation == 1:
        new_image = rgb2gray(image)
    else:
        new_image = image.astype(np.float64)
        new_image = new_image / 255
    return new_image


# ----------------- part 1 -----------------


def DFT(signal):
    """
    Transforms 1D discrete signal to fourier representation.
    :param signal: is an array of dtype float64 with shape (N, 1)
    :return: fourier representation
    """
    N = signal.shape[0]
    assert(N > 0)
    u = np.arange(N)
    x = u[:, np.newaxis]
    exp = complex(np.cos(2 * np.pi / N), - np.sin(2 * np.pi / N))
    new_basis = np.power(exp, x * u)
    dft = new_basis @ signal
    return dft


def IDFT(fourier_signal):
    """
    Transforms inverse 1D discrete signal to fourier representation.
    :param fourier_signal: is an array of dtype complex128 with shape (N, 1)
    :return: signal representation.
    """
    N = fourier_signal.shape[0]
    assert(N > 0)
    x = np.arange(N)
    u = x[:, np.newaxis]
    exp = complex(np.cos(2 * np.pi / N), np.sin(2 * np.pi / N))
    new_basis = np.power(exp, u * x)
    idft = (new_basis @ fourier_signal) / N
    return idft
```

```python


def DFT2(image):
    """
    Transforms 2D image to its fourier representation.
    :param image: grayscale of dtype float64
    :return: fourier representation.
    """
    return DFT(DFT(image).T).T


def IDFT2(fourier_image):
    """
    Transforms inverse  fourier representation to its 2D image
    :param fourier_image:  2D array of dtype complex128 with shape
    :return: image representation
    """
    return IDFT(IDFT(fourier_image).T).T


# ------------------ part 2 ------------------


def change_rate(filename, ratio):
    """
    changes the duration of an audio file by keeping the same samples, but
    changing the sample rate written in the file header.
    :param filename: is a string representing the path to a WAV file
    :param ratio: positive float64 representing the duration change s.t
    0.25 < ratio < 4.
    :return: nothing.
    """
    original_sample_rate, original_data = wavfile.read(filename)
    new_sample_rate = int(original_sample_rate * ratio)
    wavfile.write('change_rate.wav', new_sample_rate, original_data)
    return


def change_samples(filename, ratio):
    """
    changes the duration of an audio file by reducing the number of samples
    using Fourier
    :param filename: string representing the path to a WAV file.
    :param ratio: positive float64
    :return: a 1D ndarray of dtype float64 representing the new sample points
    """
    original_sample_rate, original_data = wavfile.read(filename)
    new_signal = resize(original_data, ratio)
    wavfile.write('change_samples.wav', original_sample_rate,
                  np.real(new_signal) / np.max(np.real(new_signal)))
    return


def resize(data, ratio):
    """
    change the number of samples by the given ratio.
    :param data: 1D ndarray of dtype float64 representing the original
    sample points.
    :param ratio: positive float64
    :return: 1D ndarray of dtype of float64 representing the new sample
    points.
    """
    assert(ratio > 0)
    assert(data.shape[0] > 0)
    fft_data = DFT(data)
    if ratio < 1:  # I should pad it
        new_num_of_samples = int(data.shape[0] / ratio)
        total_num_of_pads = new_num_of_samples - data.shape[0]
```

```python
        return IDFT(np.fft.ifftshift(np.pad(np.fft.fftshift(fft_data),
                                            (int(np.floor(total_num_of_pads /
                                                          2)),
                                             int(np.ceil(total_num_of_pads /
                                                         2))),
                                            'constant',
                                            constant_values=0)))
    elif ratio > 1:  # I should clip it
        return IDFT(clip_data(fft_data, ratio))
    else:  # ratio = 1, do nothing
        assert(ratio == 1)
        return data


def clip_data(data, ratio):
    """
    clip the data with the new ratio
    :param data:  1D ndarray of dtype float64 representing the original
    sample points.
    :param ratio: positive float64
    :return: clipped data
    """
    new_num_of_samples = int(data.shape[0] / ratio)
    cut_index = int((data.shape[0] - new_num_of_samples) / 2)
    shift_data = np.fft.fftshift(data)
    cut_fft = shift_data[cut_index  : cut_index + new_num_of_samples]
    return np.fft.ifftshift(cut_fft)


def resize_spectrogram(data, ratio):
    """
    change the number of samples by the given ratio.
    :param data: 1D ndarray of dtype float64 representing the original
    sample points.
    :param ratio: positive float64
    :return: 1D ndarray of dtype of float64 representing the new sample
    points.
    """
    stft_matrix = stft(data)
    new_stft_matrix = list()
    for index, stft_vec in enumerate(stft_matrix):
        new_stft_matrix.append(resize(stft_vec, ratio))
    new_stft_matrix = np.array(new_stft_matrix)
    new_data = istft(new_stft_matrix)
    return np.real(new_data)


def resize_vocoder(data, ratio):
    """
    change the number of samples by the given ratio.
    :param data: 1D ndarray of dtype float64 representing the original
    sample points.
    :param ratio: positive float64
    :return: 1D ndarray of dtype of float64 representing the new sample
    points.
    """
    spec = stft(data)
    resized_spec = phase_vocoder(spec, ratio)
    resized_data = istft(resized_spec)
    return np.real(resized_data)


# ------------------ part 3 ------------------


def conv_der(im):
    """
    computes the magnitude of image derivatives.
```

```python
    :param im: grayscale image of type float64
    :return: magnitude of the derivative, with the same dtype and shape.
    """
    dx = np.array([0.5, 0, -0.5]).reshape(1, 3)
    dy = dx.T
    im_by_dx = scipy.signal.convolve2d(im, dx, mode='same')
    im_by_dy = scipy.signal.convolve2d(im, dy, mode='same')
    magnitude = np.sqrt(np.abs(im_by_dx) ** 2 +
                        np.abs(im_by_dy) ** 2)
    return magnitude


def fourier_der(im):
    """
    computes the magnitude of image derivatives using Fourier transform.
    :param im: float64 grayscale image.
    :return: magnitude of the derivative calculated using fourier.
    """
    dft = np.fft.fftshift(DFT2(im))
    u = np.arange(-im.shape[1] / 2, im.shape[1] / 2)[np.newaxis, :]
    v = np.arange(-im.shape[0] / 2, im.shape[0] / 2)[:, np.newaxis]
    derivative_x_factor = complex(0, 2 * np.pi / im.shape[1]) * u
    derivative_y_factor = complex(0, 2 * np.pi / im.shape[0]) * v
    dft_x_derivative = derivative_x_factor * dft
    dft_y_derivative = derivative_y_factor * dft
    idft_x_derivative = IDFT2(np.fft.ifftshift(dft_x_derivative))
    idft_y_derivative = IDFT2(np.fft.ifftshift(dft_y_derivative))
    magnitude = np.sqrt(np.abs(idft_x_derivative) ** 2 +
                        np.abs(idft_y_derivative) ** 2)
    return magnitude
```