How to Use the Python Pattern Playback System

Hahn Koo (`hahn.koo@sjsu.edu`)

May 28, 2022

1 Introduction

The software is designed to convert a grayscale image of the magnitude spectrogram to a waveform. This document explains how to use it.

2 Installation

2.1 Setup

The software runs on Python 3. It uses the following libraries:

(1) `sys`

(2) `argparse`

(3) `tkinter`

(4) `numpy`

(5) `scipy`

(6) `matplotlib`

(7) `pillow` (`PIL`)

(8) `librosa`

(1-3) are part of the Python standard library. So you don't need to install them separately. (4-6) are not part of the standard library but are very common libraries people use these days. If you've installed Python 3 through Anaconda you should already have them. If not, see here to install them. (7-8) are not that common. You probably should install them. See here or here for (7) and here for (8).

2.2 Download

Once you have all of the above installed, download the software from here or here. The crucial file is `pattern_playback.py`. The rest are supplementary files to illustrate how to use it.

3 Command-line interface

The software has a command-line interface. So enter `python pattern_playback.py` followed by named arguments and options. To see a comprehensive list of arguments and options, enter

```
python pattern_playback.py --help
```

Let me explain them further below.

3.1 `--sampling_rate`

This is used to specify the sampling rate to synthesize the waveform with. For example, to set the sampling rate to 16,000 Hz,

```
--sampling_rate 16000
```

This argument must be specified.

3.2 `--duration`

This is used to specify the duration of the waveform to synthesize in seconds. For example, to set the duration to 1.234 seconds,

```
--duration 1.234
```

This argument must be specified.

3.3 `--load` vs. `--draw`

You can either load a grayscale image file (e.g. `PNG`, `JPG`) containing a spectrogram or draw a spectrogram in black and white on a blank canvas from scratch. Use `--load` for the former and `--draw` for the latter.

`--load` must be followed by a path to the image file you want to load. For example,

```
--load ./examples/example1.jpg
```

`--draw` is not followed by any argument. Just "turn it on" as

```
--draw
```

You must specify either of the two but not both of course.

There are some optional arguments you can specify in relation to these two.

3.3.1 `--crop`

This can be used in tandem with `--load` if you want to crop image in the file via click-and-drag because, for example, the file not only contains a spectrogram but also a figure caption. For example, to crop a portion of an image in `./examples/example2.png`,

```
--load ./examples/example2.png --crop
```

3.3.2 `--canvas_width`, `--canvas_height`, `--canvas_margin`

These can be used in tandem with `--draw` if you want to configure the canvas to draw the spectrogram on. By default, the canvas has a width of 1200 pixels, a height of 600 pixels, and a margin of 50 pixels. The dimensions can be modified by specifying the desired size in pixels after each named argument. For example,

```
--draw --canvas_width 600 --canvas_height 800
```

or

```
--draw --canvas_width 1600 --canvas_height 1200 --canvas_margin 100
```

3.3.3 `--save_drawing`

By default, the spectrogram you drew is saved as `spectrogram.png`. But you can use `save_drawing` in tandem with `--draw` followed by a file name to save your drawing as such. For example, to save your drawing as `chirp.png` instead,

```
--draw --save_drawing chirp.png
```

3.4 `--griffinlim`

By default, waveform is synthesized from the spectrogram via inverse short-time Fourier transform assuming a zero-phase spectrum. To improve the output quality, one can use the Griffin-Lim algorithm instead. Just turn the option on as follows:

```
--griffinlim
```

3.5 `--show_graphs`

This is to display the magnitude reconstructed from the image as well as the synthesized waveform. Just turn it on as follows:

```
--show_graphs
```

3.6 `--save_wav`

By default, the synthesized waveform is saved as `out.wav`. The name can be changed with `--save_wav` followed by the file name. For example, to save the output as `chirp.wav` instead,

```
--save_wav chirp.wav
```

4 Illustrations

Let me illustrate how to enter commands and what happens after you do.

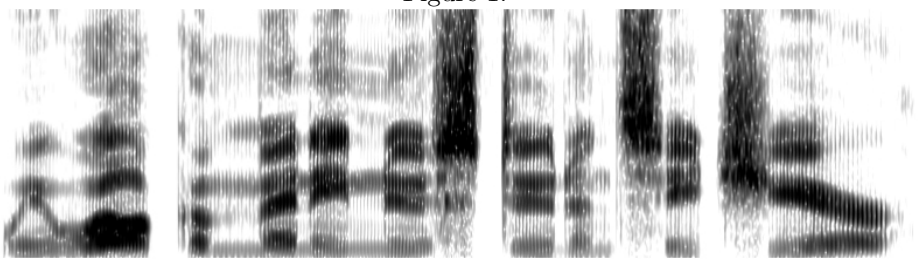
4.1 Loading from an image file

Suppose you wanted to do the following:

- load image from `./examples/example1.jpg`
- synthesize waveform at sampling rate = 16000 Hz with duration = 3 seconds using the Griffin-Lim algorithm
- show the reconstructed spectrogram and waveform
- save the result as `example1.wav`

FYI, Figure 1 illustrates what the image file (downloaded from here) looks like. It contains nothing but a spectrogram:

Figure 1:

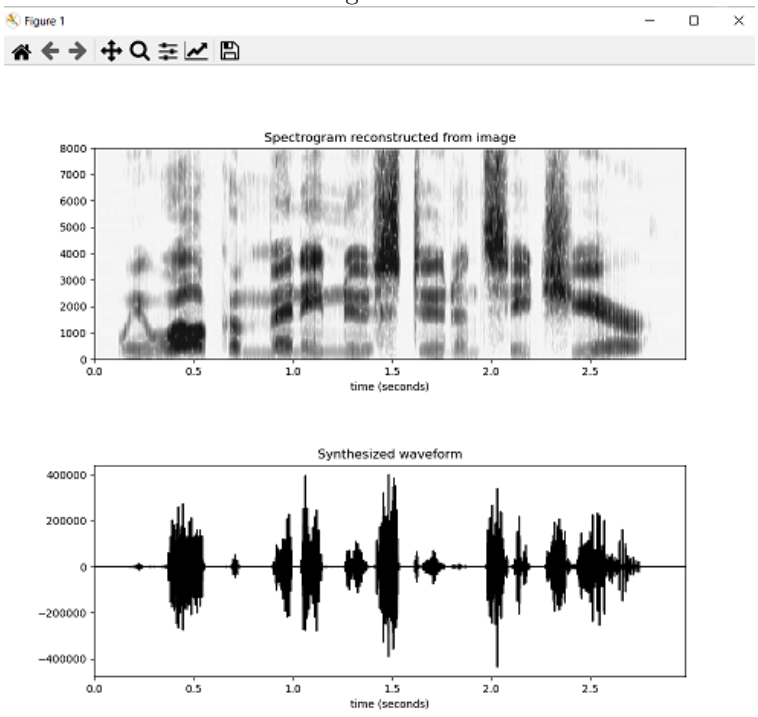


You would enter the following command (all in one line):

```
python pattern_playback.py --sampling_rate 16000 --duration 3
--load ./examples/example1.jpg --griffinlim --show_graphs --save_wav example1.wav
```

This will bring up a window that looks like the one in Figure 2. Click the save icon if you want to save the figure. You must click the `x` button at the top right corner for the software to move on, save the result, and finish.

Figure 2:



4.2 Cropping a portion from an image file

Suppose you wanted to do the following:

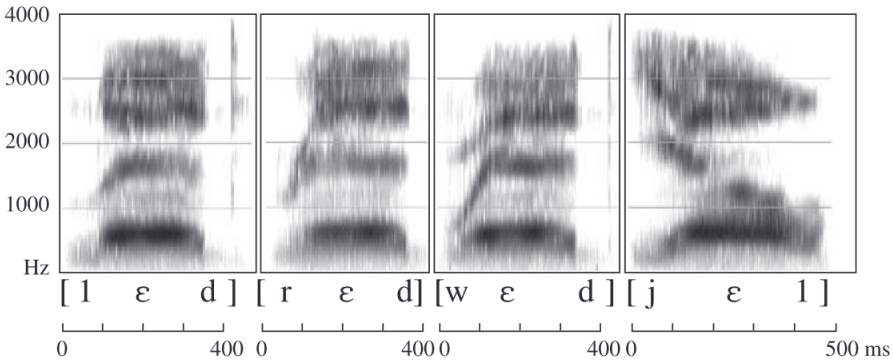
- load image from `./examples/example2.png` and crop a portion of it

- synthesize waveform at sampling rate = 8000 Hz with duration = 0.4 seconds using the Griffin-Lim algorithm
- show the reconstructed spectrogram and waveform
- save the result as `example2.wav`

FYI, Figure 3 illustrates what the image file looks like. It's a figure from Ladefoged and Johnson (2014) that contains four spectrograms. Let's say we're interested in the third one from the left.

Figure 3:

Figure 8.11 A spectrogram of *led*, *red*, *wed*, *yell*.

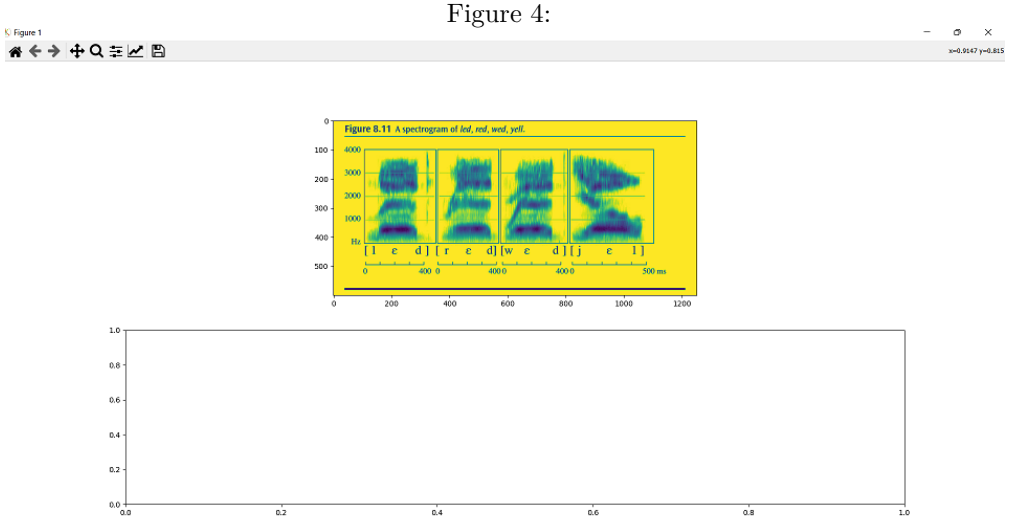


You would enter the following command (all in one line):

```
python pattern_playback.py --sampling_rate 8000 --duration 0.4 --crop
--load ./examples/example2.png --griffinlim --show_graphs --save_wav example2.wav
```

This will first bring up a window that looks like the one in Figure 4.

Figure 4:

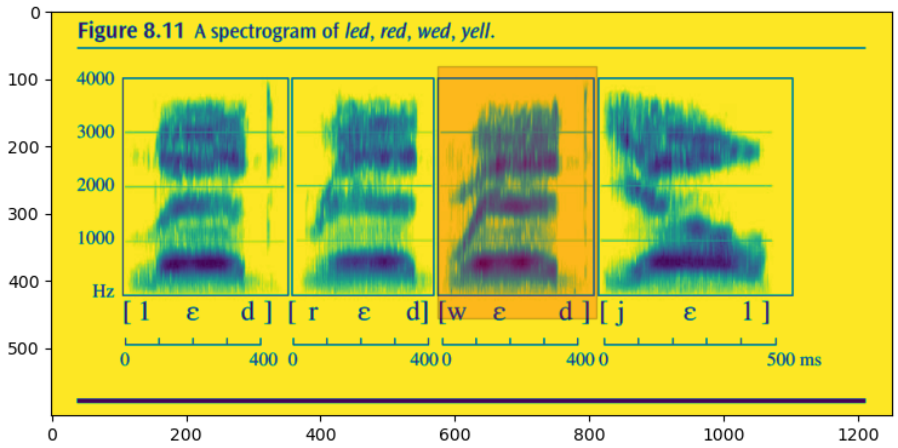


There are two panes in the window. The top one contains the image to crop. The bottom one is blank for

now but will be filled with the cropped image later.

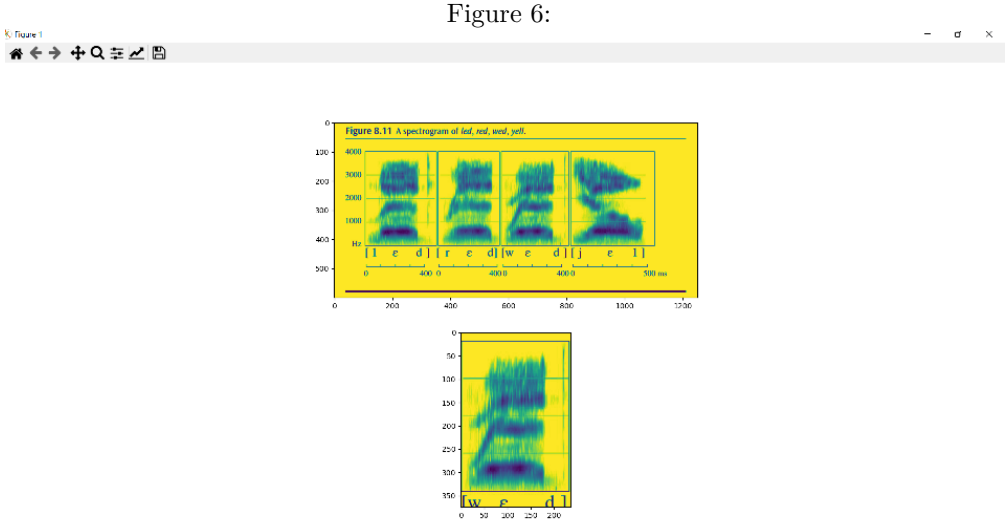
Click and drag your mouse cursor to select a portion you want to crop. The portion will be highlighted in pink as in Figure 5.

Figure 5:



As soon as you let go, the portion you cropped will fill in the bottom pane as in Figure 6.

Figure 6:



You can click-and-drag as often as you want to make sure you cropped it properly. When you're satisfied, click the `x` button at the top-right corner to exit the window.

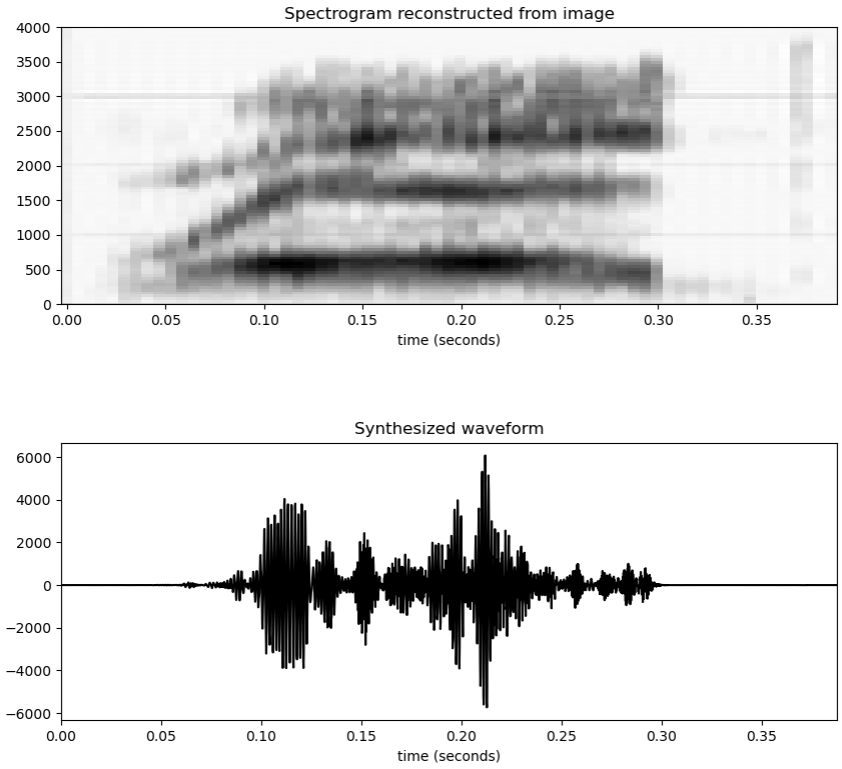
The software will then open another window to display the spectrogram and the waveform for your information as in Figure 7. If you don't want to see this, you can simply omit `--show_graphs` when you enter the command. At any rate, just like in the previous example, close the window to finish.

4.3 Drawing on a canvas

Suppose you wanted to do the following:

- draw a spectrogram from scratch and save the drawing as `smiley.png`

Figure 7:



- synthesize waveform at sampling rate = 10000 Hz with duration = 0.5 seconds assuming a zero phase spectrum
- show the reconstructed spectrogram and waveform
- save the result as `smiley.wav`

You would enter the following command (all in one line):

```
python pattern_playback.py --sampling_rate 10000 --duration 0.5 --draw
--save_drawing smiley.png --show_graphs --save_wav smiley.wav
```

This will bring up a blank canvas like the one in Figure 8.

Use a pointing device (e.g. mouse, stylus, fingertip) to draw some spectrogram. Left click and drag to draw. Right click and drag to erase. One embarrassing feature about erasing: the gridlines will be erased too. Something to be revised in the future.

Here let me just draw a smiley face as in Figure 9.

Click the `x` button at the top right corner to close the canvas when you're done drawing.

The software will then open another window to display the spectrogram and the waveform for your information as in Figure 10. Close the window to finish off.

Figure 8:

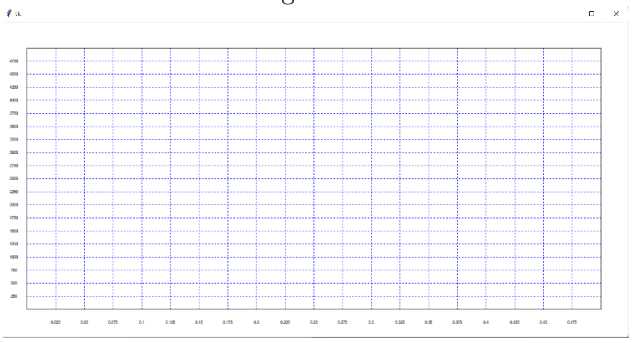


Figure 9:

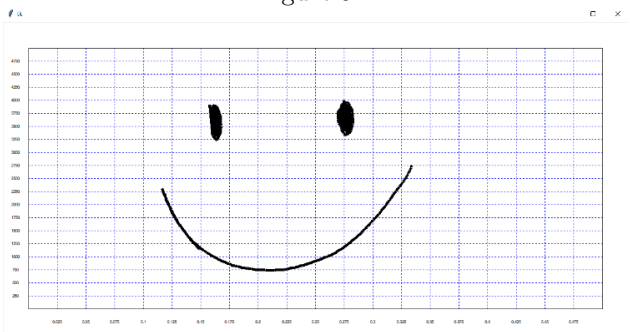
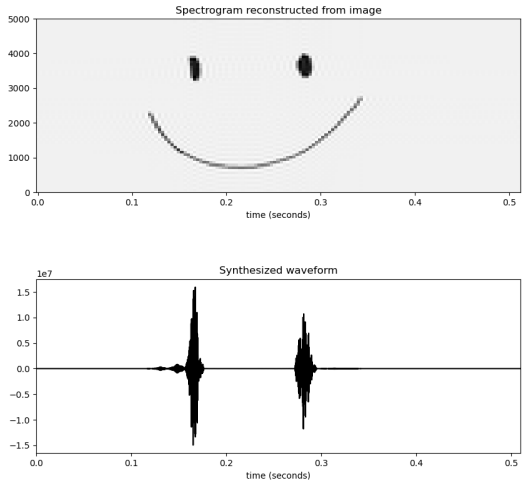


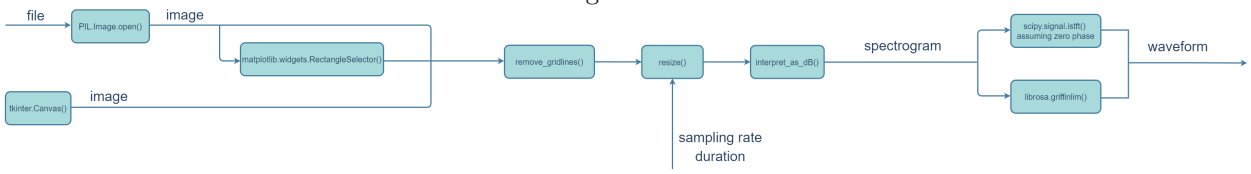
Figure 10:



5 Under the hood

Understanding how the software works may help you use it more effectively. Figure 11 illustrates the software architecture. Novel components are `remove_gridlines()`, `resize()`, and `interpret_as_dB()`. Let me explain them below.

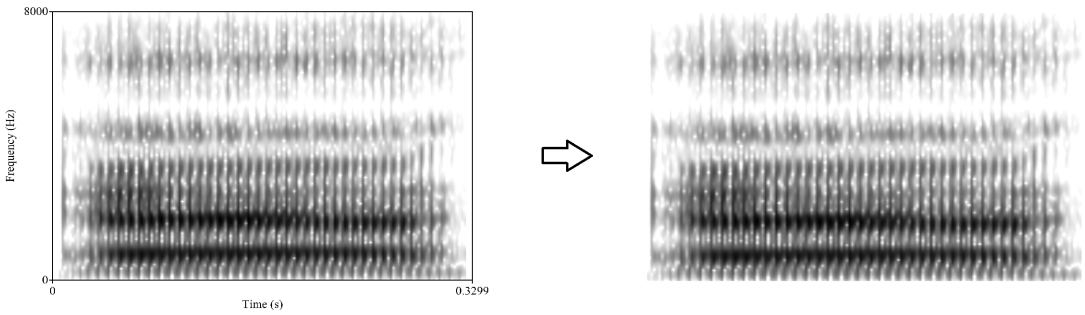
Figure 11:



5.1 `remove_gridlines()`

This looks for any borders – including the time and frequency axes – that surround the spectrogram in the image and removes the boundaries as well as anything beyond them. In hindsight, I should have named it `remove_borders()`. At any rate, Figure 12 illustrates how it removes borders and beyond. It tends to work well when the borders are drawn in thin straight lines and there's no other straight lines of similar length beyond the borders. But it doesn't always work unfortunately. Something I'll try to improve in the future. In the mean time, if the image file contains extraneous stuff, turn on `--crop` and take care to select just the spectrogram as closely as possible.

Figure 12:



5.2 `resize()`

An image is an array of pixel values. We can't directly apply inverse transform to it and convert it to a waveform. Two things before we do so:

(1) resize the array

(2) convert the pixel values to magnitude spectral coefficients

`resize()`, explained here, is for (1). `interpret_as_dB()`, explained in the next section, is for (2).

Resizing is necessary because we don't know how the image of spectrogram came to be. Somebody applied the short-time Fourier transform (STFT) to a waveform and got an array of spectral coefficients. The shape of the resulting array depends on what the STFT parameters were: e.g. frame size (number of samples per analysis frame), frame shift (number of samples by which the analysis frame is shfted). They then plotted the array and saved it as image. The shape of the array will change further in the process depending on things like if and how intermediate values were added to make the plot look smoother, what resolution was used in saving the plot as image, etc. We could retrace the steps if we knew all the parameters involved. But we don't.

So here's a workaround: We ask the user how long the waveform is and calculate what the shape of array of spectral coefficients would be if (a) we applied STFT with typical parameters to a waveform of such length

(b) while keeping the aspect ratio the same between the array of spectral coefficients and the array of pixel values. We then resize the array of pixel values to the target shape by resampling its values. The arguments that the user enters via `--sampling_rate` and `--duration` tell us how long the waveform is. The input image tells us what the aspect ratio is. We use default parameters of `librosa.stft()` as *typical* parameters.

So how to calculate the target shape? Let

- (r_1, c_1) = shape of the original array of pixel values, i.e. r_1 rows and c_1 columns
- (r_2, c_2) = target shape, i.e. r_2 rows and c_2 columns
- N = number of samples in the waveform
- s = number of samples by which the analysis frame is shifted (frame shift)

We want the following to hold:

(a) $r_1 : c_1 \approx r_2 : c_2$

(b) r_2 is an odd number

(c) $s = (r_2 - 1) \times 2/4$

(d) $N = s \times (c_2 - 1)$

(a) because we want to keep the aspect ratio. (b-d) because they reflect default parameters in `librosa.stft()`.

From (c) and (d),

$$N = \frac{(r_2 - 1)(c_2 - 1)}{2} \tag{1}$$

From (a), we can assume

$$c_2 = \frac{r_2 c_1}{r_1} \tag{2}$$

So equation (1) becomes

$$N = \frac{(r_2 - 1)(r_2 c_1 - r_1)}{2r_1} \tag{3}$$

Shuffling the terms around,

$$c_1 r_2^2 - (r_1 + c_1) r_2 + r_1 - 2N r_1 = 0 \tag{4}$$

We know N, r_1, c_1 from the user input. So we can solve equation (4) for r_2. We take the larger of the two roots and round it to the nearest odd number because (b). We then plug r_2 in to equation (2) and round it down to the nearest integer to get c_2.

`resize()` uses `numpy.roots()` to find the roots for r_2 and `scipy.signal.resample()` to resample the pixel values in resizing the array to the target shape.

5.3 `interpret_as_dB()`

The pixel values in the reshaped array are interpreted as dB values after

(1) subtracting from 255

(2) dividing by two

Pixel values range from 0 to 255: 0 for black and 255 for white. This is the opposite of what's expected of spectral energy. (1) flips the values so that white means 0 dB and black means 255 dB. It turns out that this range is too high. The difference in shade of gray gets amplified so much that less dark parts of the spectrogram get virtually ignored in reconstructing the waveform. (2) counters this effect by reducing the range to $0 - 127.5$ dB. I felt cutting the range by half was appropriate for a number of examples I have tried. But it may be too much or may not be enough for other examples. I may add a dB discount factor as another command-line argument to give users more flexibility in the future. In the meantime, you can directly modify the following line in the function definition if you're not satisfied: e.g. change `self.X / 2` to `self.X / 3` to reduce the range further to $0 - 85$ dB.

```
self.X = self.X / 2
```

6 How to cite

If you want to cite the software,

Koo, H. (2022). A digital pattern playback system implemented in Python. *Journal of the Acoustical Society of America, 151*(4), A132.

References

Griffin, D., & Lim, J. (1984). Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 32*(2), 236-243.

Ladefoged, P., & Johnson, K. (2014). *A Course in Phonetics*. Cengage Learning.