FFT: Exercises

DSP Lab (ECE 4163 / ECE 6183)

Fall 2018

Demo files

`FFT_demo_01.py`
`FFT_demo_02.py`
`FFT_demo_03.py`
`FFT_demo_04.py`
`FFT_demo_05.py`
`plot_microphone_input.py`
`plot_microphone_input_spectrum.py`

Exercises

1. **Amplitude Modulation.** Modify the Python program `plot_microphone_input_spectrum` so it applies amplitude modulation (AM) to the microphone input audio signal. The program should plot the live frequency spectra (Fourier transform) of both the input and output signals (use two different colors). The Fourier transform should be computed using the FFT. The program should also play the output (result of AM) to the speaker/headphones. What is the relation between the spectra of the output and input signals? SUBMIT

2. **Filtering.** Modify the Python program `plot_microphone_input_spectrum` so it applies a bandpass filter to the microphone input audio signal. The bandpass filter should have a passband from 500 Hz to 1000 Hz. The program should plot the live frequency spectra (Fourier transform) of both the input and output signals (use two different colors). The Fourier transform should be computed using the FFT. The program should also play the output (result of AM) to the speaker/headphones. What is the relation between the spectra of the output and input signals? What is the relation between the two spectra?

Blocking and AM: Exercises

DSP Lab (ECE 4163 / ECE 6183)

Fall 2018

Demo files

`AM_noblocking.py`
`AM_blocking.py`
`AM_blocking_savewave.py`
`AM_blocking_corrected.py`
`AM_blocking_corrected_savewave.py`
`AM_from_microphone.py`

The demo program `AM_noblocking.py` applies amplitude modulation (AM) to a signal obtained from a wave file. This moves the signal to higher frequencies and changes the way the signal sounds.

The usual practice is to read and write of samples to and from audio devices in *blocks* rather than one sample at a time. The demo programs include a program that reads the input signal from the microphone. When reading the input signal from the microphone it is recommended that headphones be used to avoid feedback problems (sound passing from the speaker back into the microphone).

Exercises

1. Write a Python program to implement audio AM with a stereo output, where different modulation frequencies are used for the left and right channels. The output stereo signal should be saved to a wave file. Listen to the output using headphones. Submit your wave file as part of your work.

   The following exercises refer to the vibrato effect and the corresponding demo programs.

2. The demo program `play_vibrato_interpolation.py` does not use blocking (it reads and writes one sample at a time). Write a version of this program that reads, processes, and writes the audio signal in blocks of 64 frames.

3. Modify the demo program `play_vibrato_interpolation.py` so that it reads and writes the audio signal in blocks, and takes the input signal from the microphone instead of a wave file. SUBMIT

4. Modify the demo program `play_vibrato_interpolation.py` so that it reads and writes the audio signal in blocks, takes the input signal from the microphone, and produces a stereo output audio signal. The left and right channels of the output signal should have different vibrato parameters (frequency and amplitude).

Blocking and filtering: Exercises

DSP Lab (ECE 4163 / ECE 6183)

Fall 2018

Demo files

`demo_filter.py`
`demo_filter_blocking.py`
`demo_filter_blocking_corrected.py`
`myfunctions.py`
`author.wav`

In previous demos we used the Matlab function `filter` to implement a difference equation. In Python, a similar function called `lfilter` is available in the SciPy library for scientific computing. (Here `lfilter` means *linear* filter.)

`http://docs.scipy.org/doc/scipy/reference/signal.html`

To avoid transient artifacts at the start of each block, we specify the initial states `zi` in the `lfilter` function as the final states `zf` from the previous block.

Exercises

1. The demo programs take the input audio signal from a wave file and apply a bandpass filter. In this exercise, modify the demo program `demo_filter_blocking_corrected.py` to take the input audio signal from the microphone.

2. Like the previous exercise. Also plot the input and output signals in real-time in a figure window (use different colors for the input and output signals). SUBMIT

3. The Matlab function `butter` gives the coefficients of a digital Butterworth filter. For example, a band-pass filter with a pass-band from 500 Hz to 1000 Hz can be obtained in Matlab using:

   ```
   [b, a] = butter(2, [500 1000]*2/Fs)
   ```

   What is the order of this filter?

   In Python, there is also a function `butter` in the SciPy library `scipy.signal`. Verify that the Python function gives the same coefficients as the Matlab function.

Keypress: Exercises

DSP Lab (ECE 4163 / ECE 6183)

Fall 2018

Demo files

To detect when a key on the keyboard is pressed, we can use open-cv (called cv2). This is a software library for computer vision. It has a command called `waitKey` which we use in these demos.

```
waitkey_demo_01.py
waitkey_demo_02.py
waitkey_demo_03.py
key_play.py
```

Exercises

1. Make a version of `key_play.py` with two different notes (frequencies) instead of just one. SUBMIT

   The notes should play overlappingly (if a note is played before the previous notes has become silent, then both notes should be heard at the same time). There should be a difference equation (filter) to implement each note. Each difference equation should have its own input and output signals. The output signals of the separate filters should be added together to give the total output signal which should be played on the speaker/headphones.

2. Make a version of `key_play.py` with twelve notes (a full musical octave).

   Adjacent notes on a piano keyboard are related via

   $$f_k = \alpha^k f_0, \quad k = 0,\ 1,\ 2,$$

   where $\alpha = 2^{1/12}$. You can set $f_0 = 440$ Hz which is 'middle A'.

Demo 17: Tkinter (Part 1)

DSP Lab (ECE 4163 / ECE 6183)
Fall 2018

Tkinter is a Python library for making graphical user interfaces (GUIs). Tkinter stands for *ToolKit Interface*.

For documentation and resources about Tkinter, see *notes_TKinter.pdf* in the folder of demo programs.

Exercise

Make your own original GUI using Tkinter.

• Your GUI should use all the widgets shown in the demo programs (label, button, entry, scale).

• Your GUI should also use at least one additional widget: checkbutton, radiobutton, or listbox.

• Optional: use *bind* illustrated in the demo programs *TKdemo_06_click.m* and *TKdemo_07_clickreturn.py* to read information from the keyboard and mouse.

Your GUI does not need to include any audio functionality. We will combine TKinter and audio later.