Assignment 7: Sounds, Frequency, and the DFT/FFT

EC602 Design by Software

Fall 2017

Contents

1 Introduction

Frequency analysis is an important part of engineering design and analysis, and is at the heart of the digital world.

1.1 Assignment Goals

The assignment goals are to

- introduce the numpy / scipy suite of computational tools for python
- introduce the discrete Fourier transform (DFT)
- review how the DFT is related to various frequency representations of signals
- explore processing of audio signals in time and frequency

1.2 Group Size

For this assignment, the maximum group size is 3.

1.3 Due Date

This assignment is due 2017-11-7 at midnight.

1.4 Submission Link

You can submit here: `dft_sound` submit link

1.5 Points

This assignment is worth 7 points: 2 for `dft`, 2.5 for `dialer`, and 2.5 for `loudest_band`

2 Background: Continuous and Discrete Time Signals and Sequences

2.1 Getting started: sequence notation

Mathematicians use the notation x_n for a sequence $x_0, x_1, x_2, x_3, \ldots$.

We will use the alternate notation $x[n]$ for a sequence $x[0], x[1], x[2], \ldots$ for two reasons:

1. It is closer to the notation for indexing in C++ and Python.
2. It is the notation most commonly employed by engineering treatments of signals and systems theory.

2.2 Discrete-time and Continuous-time Signals

Continuous-time signals (CT signals) are a representation of real-world signals such as audio or voice signals. We use the notation $s(t)$ to identify continuous-time signals, where $t \in R$. The letter s represents a particular signal, and could be replaced with other letters to represent other signals.

CT signals must be digitized or sampled in order to be stored, processed, transmitted, and analyzed by electronic or computer devices.

Sampling is done by storing the signal $s(t)$ at regular times spaced by τ seconds, where τ is called the sampling interval. The samples are stored in a sequence $s[n]$, such that

$$s[n] = s(n\tau)$$

and we assume the interesting portion of $s(t)$ exists in the range

$$0 \leq t < N\tau$$

Sampling is also called *analog to digital* conversion (A/D) or *continuous to discrete* conversion (C/D). The sampling rate or sampling frequency $f_s = 1/\tau$ determines the range of frequencies that can be recorded from the original based on the Nyquist Theorem: frequencies f up to $f_s/2$ are correctly recorded.

2.3 The Discrete Fourier Transform

The discrete Fourier transform (DFT) of a sequence of N complex numbers $x[0], x[1], x[2], \ldots x[N-1]$ is defined as follows:

$$X[k] = DFT(x) = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}$$

Notice that by convenction, the sequence in time is lower-case x and its corresponding transform is upper-case X. The letter x is not special: it could be h, y, w, etc.

The result of the DFT is another sequence of complex numbers which is N-periodic. The DFT is N-periodic (which means that `X[k]=X[k+N]`) because

$$X[k+N] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi n(k+N)/N} = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N}e^{-j2\pi n} = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N} = X[k]$$

because $e^{-j2\pi n} = 1$ for any integer n.

So, in practice, we always think of the DFT values k for $k = 0$ up to $N - 1$.

We can re-create the original sequence $x[n]$ from the DFT sequence using the inverse DFT (IDFT) as follows:

$$x[n] = IDFT(X) = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j2\pi nk/N}$$

2.4 Applications of the DFT

2.4.1 Discrete Time Fourier Transform

The discrete-time Fourier transform (DTFT) is defined as

$$X(e^{j\omega}) = \textstyle\sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}$$

which can also be viewed as the z-transform

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$$

evaluated on the unit circle $z = e^{j\omega}$.

Notice that both $X(z)$ and $X(e^{j\omega})$ are polynomials where the exponent is n.

There are two important cases for the signal $x[n]$: it is time-limited or it is periodic.

2.4.1.1 Time-limited Signals

In the case of a time-limited signal, we can compute a sampled version of $X(e^{j\omega})$ directly from the DFT equation.

Since the DTFT is now given by (we eliminate the terms $x[n]$ which are zero)

$$X(e^{j\omega}) = \sum\nolimits_{n=0}^{N-1} x[n]e^{-j\omega n}$$

we see that the DFT gives us

$$X[k] = DFT(x) = X(e^{j2\pi k/N})$$

which means that the DFT provides N samples of the DTFT equally spaced around the unit circle, at the angular values $2\pi k/N$.

If a more precise representation of the DTFT is desired, the signal $x[n]$ can be *zero-padded* by adding extra zero terms to the end of the sequence. This causes the sampled values of $X(e^{j\omega})$ to be more closely spaced around the unit circle.

2.4.1.2 Periodic Signals

If $x[n]$ is N-periodic, then

$$X(e^{j\omega}) = \sum_{k=-\infty}^{\infty} 2\pi X[k]\delta(\omega - k\omega_0)$$

where we have defined $\omega_0 = 2\pi/N$, the fundamental frequency of the signal. The $X[k]$ in the equation are the $DFT(x)$ sequence.

Notice in this case, the result of the DFT is a perfect representation of the DTFT, since there are only N unique values of $X[k]$ ($X[k]$ is N-periodic, which means $X[k] = X[k+N]$).

The signal $x[n]$ can be reconstructed using

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{jk\omega_0 n} = IDFT(X)$$

2.4.2 Discrete Time Fourier Series

If $x[n]$ is N-periodic, then it can be represented as the sum of N-periodic complex exponentials, as follows.

Define $\omega_0 = 2\pi/N$, then we consider the harmonics of ω_0 with multiples $0, 1, \ldots, N-1$. These are the signals:

$$1, e^{j\omega_0 n}, e^{j2\omega_0 n}, e^{j3\omega_0 n}, \ldots, e^{j(N-1)\omega_0 N}$$

So, suppose that the periodic signal is represented as

$$x[n] = \sum_{k=0}^{N-1} a_k e^{j\omega_0 n}$$

This is called the discrete-time Fourier series (DTFS). If can be shown that the *Fourier series coefficients* a_k in the DTFS are given by

$$a_k = \frac{1}{N} DFT(x) = \frac{1}{N} X[k]$$

2.4.3 Continuous Time Fourier Transform

The DFT can be used to approximate the continuous-time Fourier transform.

The Fourier transform of a CT signal $x(t)$ is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt = \lim_{\tau \to 0} \sum_{n=-\infty}^{\infty} x(n\tau) e^{-j2\pi f(n\tau)} \tau$$

and assume that $x(t)$ is time-limited to between 0 and T, so that

$$\int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt = \int_{0}^{T} x(t) e^{-j2\pi f t} dt \approx \sum_{n=0}^{N-1} x(n\tau) e^{-j2\pi f(n\tau)} \tau$$

where $T = N\tau$ for some integer N.

Here is the method:

1. Decide on a value of τ, the sample spacing, and a time length N. This means we can represent $x(t)$ in the interval $[0, T)$ where $T = N\tau$.

2. Store the N samples $x(n\tau)$ in an array `x`.

3. Calculate `X=tau*DFT(x)`, which gives the result `X[k]`$=X(f_k)$ where the f_k are given by

$$f_k = \begin{cases} \frac{k}{N\tau}, & 0 \le k < N/2 \\ \frac{k}{N\tau} - \frac{1}{\tau} & N/2 \le k < N \end{cases}$$

   This means that the first half of the vector `X` represents the positive frequencies of $X(f)$ and the second half the negative frequencies of $X(f)$.

4. Re-order the vector `X` by using `fftshift(X)` which results in `X[k]` being the CTFT evaluated at $-\frac{1}{2\tau} + \frac{k}{N\tau}$.

2.4.4 Continuous Time Fourier Series

The DFT can be used to approximate the continuous time Fourier Series.

If a continuous-time signal is periodic with period T, it can be represented as

$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t}$$

(this is called the continuous-time Fourier series) where $\omega_0 = 2\pi/T$ and

$$a_k = \int_{<T>} x(t) e^{-jk\omega_0 t}\, dt$$

These values can be approximated by choosing $N\tau = T$ and calculating

$$a_k \approx \sum_{n=0}^{N-1} x(n\tau) e^{-j2\pi nk/N} \cdot \tau = \tau \cdot DFT(x) = \tau X[k]$$

which is valid for $k = 0, \ldots, N/2 - 1$.

The other values $X[k]$ of the DFT which are generated, $k = N/2, \ldots, N-1$ actually correspond to a_k by the relationship

$$X[k] = \tau a_{k-N}$$

This means that the second half of the $X[k]$ sequence corresponds to the negative frequencies of $x(t)$.

2.5 Implementation of the DFT

The DFT is implemented in python in the `numpy.fft` module, as well as in the `scipy` module by the function `fft`.

The IDFT is implemented by the function `ifft` in `scipy` and `numpy.fft`

The Fast Fourier Transform (FFT) is an algorithm for computing the N-point DFT in $O(N \log N)$ time. The direct implementation of the DFT requires $O(N^2)$ time.

We will explore the significance of this in more detail in the next assignment.

2.6 Dual Tone Multi Frequency

Touch-tone phones, or as they are now called, phones, use tones to indicate dialed numbers.

The system started to replace *pulse* or *rotary* phones in North America in the 1960s.

The details are described here: about DTMF

2.7 Definition of Loudest band.

Given a continuous-time signal $s(t)$, the loudest band of bandwidth B is defined as the range $[L, H)$ such that $B = H - L$ and the energy of the signal in the frequency domain in that band is the largest possible value for this signal.

This means that

$$\int_L^H |S(f)|^2 df$$

is maximized over $L > 0$ and $B = H - L$

2.8 Filtering

A signal $x(t)$ can be filtered by a linear system represented by its impulse response $h(t)$ or the Fourier transform of $h(t)$ which is indicated by $H(f)$.

The filtered signal $y(t) = h(t) * x(t)$ which in the frequency domain is equivalent to

$$Y(f) = H(f)X(f)$$

Hence, by designing the shape of $H(f)$, the system can choose frequencies in the input signal $x(t)$ that will *survive* or pass through the system and remain in $y(t)$. Thus, signals can be filtered to extract features. For example, if $H(f) = 1$ whenever $L < |f| < H$, then this represents a bandpass filter that passed all frequencies between L and H and eliminates all other frequency content.

3 Numpy and Scipy

Here is the main introduction to SciPy from scipy.org

> SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

- NumPy: Base N-dimensional array package scipy
- SciPy: library Fundamental library for scientific computing
- Matplotlib: Comprehensive 2D Plotting
- IPython: Enhanced Interactive Console
- Sympy: Symbolic mathematics
- pandas: Data structures & analysis

Please visit scipy-lectures for an introduction.

Here is a good tutorial on numpy and arrays: numpy quickstart tutorial

3.1 Other Python tools

- cython C extensions to python
- theano multi-dimensional arrays
- pypy JIT compiler
- bokeh browser-based visualization

- numba high-performance directly in python

Some of these tools are part of the anaconda distribution: `bokeh` and `numba`

4 Sounds in the devbox with python

The devbox includes a command-line utility `aplay` to play sound files.

4.1 Setting up

We need a new module to get sound working in the devbox from within python. Run the following commands from the terminal in the devbox:

```
sudo apt-get install libasound2-dev
easy_install pyalsaaudio
```

There are two examples

- sounds_example.py
- play_bach_example.py

4.2 Jupyter Notebook / IPython

Jupyter notebook has a facility for playing audio. Run

```
jupyter notebook
```

and then open the following example notebook: audio_example.ipynb

4.3 Example Audio Files

For your convenience, here are some audio files to play with.

Some of them are Halloween themed.

- bach10sec.wav
- scary.wav
- bachish.wav

5 The assignment

5.1 Part A: python DFT function

Write a python function `DFT(x)` which returns the DFT of a sequence of complex numbers. Your function should be contained in an importable python file called `dft.py`.

Your program must satisfy the following specifications:

- the function `DFT` must return a numpy.ndarray with shape `(N,)` and this returned value should match the definition of the DFT provided in this assignment
- the function `DFT` must raise a `ValueError` exception if the input value `x` is not a sequence of numerical values
- the program `dft.py` can (and must) use one and only one import statement:

  from numpy import zeros, exp, array, pi

5.2 Part B: telephone dialer.

Write a python function `dialer(file_name,frame_rate,phone,tone_time)` which creates a WAV file of the sound of a telephone number being dialed.

The parameters are

- `file_name`, a string representing the name of the WAV file to be created. Do not append ".wav" to this string.
- `frame_rate`, a number representing the number of samples per second to use in the sound
- `phone`, a string of digits representing a phone number to dial.
- `tone_time`, a number representing the time in seconds of each tone to generate.

The function `dialer` should be defined in an importable python file `dialer.py`

5.3 Part C: loudest band.

Write a python function`loudest_band(music,frame_rate,bandwidth)` which returns a tuple `(low,high,loudest)` containing information about the loudest part of `music`.

The input parameters are:

- `music`: an ndarray of shape `(N,)` representing a continuous time signal which has been digitized. You may assume that `music` is real, and so the Fourier Transform will be conjugate symmetric.
- `frame_rate`: the sampling rate that was used to sample `music`
- `bandwidth`: the width of the frequency band to be selected (the loudest band)

The output parameters `(low,high,loudest)` are:

- `low`: the low end of the loudest frequency band in `music`
- `high`: the high end of the loudest frequency band in `music`
- `loudest`: a time signal extracted (filtered) from `music` which contains only the frequencies of `music` that are in the loudest band.

The function `loudest_band` should be defined in an importable python file `loudest.py`.

5.4 Checkers

- dft_checker.py

The other checkers are in progress.