Fourier Analysis of Audio Signals 7

Electrical Engineering 20N
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley

Hsin-I Liu, Jonathan Kotker, Howard Lei, and Babak Ayazifar

1 Introduction

In this lab session, we will use the theory of, and the concepts behind, the Discrete-Time Fourier Series (DTFS or DFS) in different applications involving audio signals. In the in-lab sections, we will separate the voices of the different frog species present in a given sound sample; in a related exercise, we will also alter the frequency of a voice sample. Along the way, we will study and determine how altering the frequency content of an audio signal *directly* modifies the signal itself.

1.1 Lab Goals

- Understand the distinctions between the time domain and the frequency domain.
- Get acquainted with the Discrete-Time Fourier Series through a practical application.
- Explore the effects of manipulating the frequency content of a signal.
- Determine relationships between the frequency contents of different versions of the same signal.

1.2 Checkoff Points

1. Pre-Lab Section . . . .
   (a) Discrete-Time Fourier Series . . . . (15 minutes)
   (b) Another Way To Convolve . . . . (15 minutes)
   (c) Up, Down, Uncertain . . . . (15%, 30 minutes)
2. In-Lab Section . . . .
   (a) Transforms: Signals in Disguise . . . . (15 minutes)
   (b) Hear a Frog, There a Babak . . . . (15%, 15 minutes)
   (c) May the Filter Be With You . . . . (15 minutes)
   (d) Frogs Will Be Frogs . . . . (25%, 45 minutes)
   (e) The Voices Are Telling Me About A Quiz . . . . (45%, 45 minutes)

2 Pre-Lab Section

2.1 Discrete-Time Fourier Series

The theory of **Discrete-Time Fourier Series** is born from the powerful idea that, given the set of signals

DTFS OR DFS

$$\mathcal{S} = \{1, e^{i\omega_0 n}, e^{2i\omega_0 n}, \dots, e^{(p-1)i\omega_0 n}\},$$

where

$$\omega_0 = \frac{2\pi}{p},$$

then *every* (yes, every) discrete-time signal with **fundamental period** p can be represented as a linear combination of elements in the set $\mathcal{S}$:

$$x(n) = \sum_{k=0}^{p-1} X_k e^{ik\omega_0 n}. \tag{1}$$

This is known as the **DFS expansion** of the periodic signal x. The complex scalars X_k are known as the **DFS coefficients** of the periodic signal x.

DFS EXPANSION

DFS COEFFICIENTS

We shall now attempt to gain an intuition about the DFS coefficients X_k. Notice that each X_k gives the amplitude of the complex exponential component of the signal with discrete-time frequency $k\omega_0 = k(2\pi/p)$, which has units of **radians per sample**. Once more, we look to the conversion equation that we used in lab 07 to convert this frequency into a form we are more familiar with. If we had a signal with a continuous-time frequency of f_d cycles per second, and the signal was sampled with a sampling frequency of f_s samples per second, then the discrete-time frequency of the sampled signal, Ω radians per sample, is given by

$$\Omega = \frac{2\pi f_d}{f_s}.$$

As per the conversion equation, we deduce that the discrete-time frequency of $k\omega_0$ radians per sample corresponds to a continuous-time frequency of $k(f_s/p)$ Hz, f_s being the sampling frequency. As a result, each X_k gives the amplitude of the complex exponential component at frequency $k(f_s/p)$ Hz.

We make another important observation: why does the DFS expansion of a signal only utilize the signals in the set $\mathcal{S}$? For instance, we do not need to utilize the complex exponential signal $e^{i(p+1)\omega_0 n}$; why? In general, if we restrict ourselves to the signals in the set $\mathcal{S}$, we do not need to utilize the complex exponential signals $e^{i(k+lp)\omega_0 n}$, $k, l \in \mathbb{Z}$, $0 \le k < p$, because, for all $n \in \mathbb{Z}$,

$$\begin{aligned}
e^{i(k+lp)\omega_0 n} &= e^{ik\,+\,lp\omega_0 n} \\
&= e^{ik\omega_0 n} \cdot e^{ilp\omega_0 n} \\
&= e^{ik\omega_0 n} \cdot e^{i(2\pi)n} \\
&= e^{ik\omega_0 n} \cdot 1 \\
&= e^{ik\omega_0 n}.
\end{aligned}$$

We conclude that $e^{i(k+lp)\omega_0 n}$ and $e^{ik\omega_0 n}$ are *the same signal*. This allows us to clamp our frequencies in the range $[-\pi, \pi)$. In fact, all we need is any set of p complex exponential signals with frequencies that are consecutive multiples of ω_0, allowing us to also write

$$x(n) = \sum_{k=\langle p \rangle} X_k e^{ik\omega_0 n},$$

where $\langle p \rangle$ represents *any* interval of integers of length p, such as $[0, \ldots, p-1]$, $[-1, \ldots, p-2]$, and $[5, \ldots, p+4]$, among many others. Context and convenience determine which range to use.

Thus, if we can break a signal down into its constituent complex exponential signals, as the DFS expansion implies, we can determine the contributions of different frequencies in the original signal, which will prove very useful in manipulating the signal, and in determining the effects of different filters on the signal. However, this expansion only works for periodic signals, and unfortunately, we will not always be working with periodic signals. For instance, later in the lab, we will deal with audio signals that are finite in length.

We can, however, do the next best thing: a finite signal $\tilde{x}$ with p samples, *not necessarily periodic*, can be considered as one cycle of a periodic signal x with fundamental period p and fundamental frequency ω_0, with units of radians per sample. To be rigorous, assume that $\tilde{x}$ is a signal that is nonzero for $0 \leq n < p$. Then, we define a signal x as

$$x(n) = \sum_{l=-\infty}^{\infty} \tilde{x}(n - lp),$$

which essentially means, "stick infinite copies of the signal $\tilde{x}$ together". (Don't take our word for it: test it!) Now, the DFS expansion of x is

$$x(n) = \sum_{k=\langle p \rangle} X_k e^{ik\omega_0 n}.$$

Since $\tilde{x}(n) = x(n)$ for $0 \leq n < p$, we deduce that

$$\tilde{x}(n) = \sum_{k=\langle p \rangle} X_k e^{ik\omega_0 n},$$

as well, but only in the range $0 \leq n < p$. Elsewhere, of course, $\tilde{x}(n)$ is zero. With this trick, we are thus able to determine the contributions of different frequencies in the original signal $\tilde{x}(n)$ using the DFS expansion of the new signal $x(n)$[1]. In this lab session, we will utilize the DFS expansion of x to find the frequency spectrum of the corresponding finite, though not necessarily periodic, signal $\tilde{x}$.

Unfortunately, LabVIEW does not have any built-in functions that directly compute the DFS coefficients. Fortunately, it does have a realization of the **Fast Fourier Transform**, which is an algorithm that can be used to construct the Fourier series coefficients. This realization is a block called `FFT`; we will use this block to find the frequency spectra of our signals. (We will not be concerned with the algorithm itself, but we will be concerned with its output.)

2.2 Another Way To Convolve

Recall that if an LTI filter H has impulse response $h(n)$ and frequency response $H(\omega)$, then the output signal $y(n)$ can be obtained from the input signal by directly convolving the input signal with the impulse response:

$$y(n) = (h * x)(n) \qquad \forall n.$$

We will now use DFS expansions to our advantage. Suppose that $x(n)$ were a discrete-time periodic signal with fundamental period p, and thus fundamental frequency ω_0. Then, it has the DFS expansion

$$x(n) = \sum_{k=0}^{p-1} X_k e^{ik\omega_0 n}.$$

[1]Notice that we are using the phrases 'contribution of a frequency ω_0' and 'contribution of a complex exponential with frequency ω_0' interchangeably. This is because the complex exponential signal $e^{i\omega_0 n}$ is the only signal that has a frequency of ω_0, and ω_0 alone.

Since the system H is LTI, it cannot generate new frequencies in the output signal (why?). As a result, the output signal $y(n)$ has a DFS expansion of the form

$$y(n) = \sum_{k=0}^{p-1} Y_k e^{ik\omega_0 n}. \tag{2}$$

But, we use the definition of the frequency response of an LTI system to conclude that

$$y(n) = \sum_{k=0}^{p-1} X_k \cdot H(k\omega_0) e^{ik\omega_0 n}. \tag{3}$$

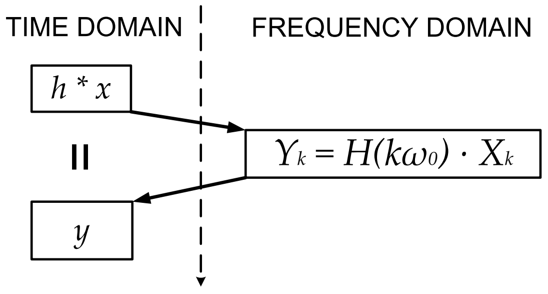
Comparing Equation 2 with Equation 3, we conclude that

$$Y_k = H(\omega)\,|_{\omega=k\omega_0} \cdot X_k \qquad \forall k.$$

This gives us another way to determine the output signal $y(n)$ from the input signal $x(n)$. We use the DFS coefficients X_k of the input signal to generate the DFS coefficients Y_k of the output signal. Once we have determined the DFS coefficients Y_k, we can construct the actual output signal $y(n)$.

This method is intuitively easier to understand, since multiplication is involved instead of convolution, but it involves first moving from the time domain to the frequency domain, and then moving back to the time domain, as visualized in Figure 1. This is the route that we will be taking in the in-lab sections[2].

Figure 1 Another way to convolve two discrete-time signals h and x.



In the transfomation above, we move a signal from its time-domain representation to its frequency-domain representation, to make the process of finding the output signal easier. Examples abound where transforming a problem into another domain makes the problem easier to solve: for instance, how would you multiply the Roman numerals `MCXI` and `LV`? You would probably first convert them to their decimal equivalents, find the answer in decimal, and then convert the decimal answer to its Roman numeral equivalent. (By the way, `(LXI)CV`.)

[2]We will explore the terms *time domain* and *frequency domain* more rigorously in the next section.

2.3 Up, Down, Uncertain

Upsampling an input signal $x(n)$ by a factor of $N \in \mathbb{Z}_{>0}$ produces the output signal

$$\forall n \in \mathbb{Z}, \qquad y(n) = \begin{cases} x\left(\frac{n}{N}\right) & \text{if } n \bmod N = 0, \\ 0 & \text{otherwise.} \end{cases}$$

1. With this definition in mind, does the input signal contract or dilate in time to produce the output signal?

2. We define the **frequency content** or **frequency spectrum** of a signal $x(n)$ to be a complex function $X(\omega)$, which takes in a frequency ω and returns a complex scalar that denotes the contribution of the signal $e^{i\omega n}$ in the signal $x(n)$. This is what we meant by the terms *time domain* and *frequency domain* in section 2.2: every signal has a representation $x(n)$ in the time domain, where the domain is either discrete-time or continuous-time, and a corresponding representation $X(\omega)$ in the frequency domain, where the domain is a continuum of frequencies. Both representations refer to the same signal: to draw on a previous analogy, `MCXI` and `1111` refer to the same number.

   FREQUENCY CONTENT

   If $x(n)$ were a periodic signal having fundamental frequency ω_0 radians per sample, for what values of ω would $X(\omega)$ be potentially nonzero?

3. The **time-frequency uncertainty principle** is an important concept in signal processing. In short, it dictates that the more localized a signal is in the time domain, the more spread out it is in the frequency domain, and vice-versa. When you dilate a signal in the time domain, it will contract in the frequency domain; when you contract a signal in the time domain, it will dilate in the frequency domain.

   TIME-FREQUENCY UNCERTAINTY PRINCIPLE

   Wait, why? Let $x(n)$ be a periodic signal with fundamental period p and fundamental frequency ω_0. Let us *dilate* $x(n)$ in the time domain by a factor of N to produce a new signal $\widehat{x}(n)$, whose frequency content is given by $\widehat{X}(\omega)$. What is the fundamental frequency of $\widehat{x}(n)$? Using this fact, how is $\widehat{X}(\omega)$ related to $X(\omega)$: is $X(\omega)$ a contracted or a dilated version of $\widehat{X}(\omega)$, and by what factor? Similar logic will apply when $x(n)$ is *contracted* in the time domain.

4. Recall that the frequency content of a discrete-time signal is 2π-periodic; in other words, $X(\omega + 2\pi k) = X(\omega)$, if k is an integer. Use this fact and the time-frequency uncertainty principle to answer the following question: Suppose you were given a plot of the frequency content of the input signal, $X(\omega)$, and a plot of the frequency content of the output signal, $Y(\omega)$. How would you determine, using the two plots alone, what factor N was used for upsampling? Answers to previous questions will prove helpful here.

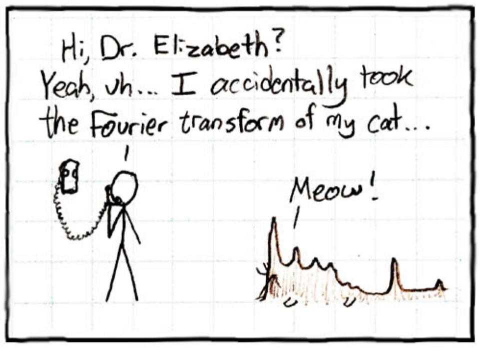
In contrast, **downsampling** an input signal $x(n)$ by a factor of $N \in \mathbb{Z}_{>0}$ produces the output signal

$$\forall n \in \mathbb{Z}, \qquad y(n) = x(nN).$$

5. With this definition in mind, does the input signal contract or dilate in the time domain?

6. Using logic similar to that you used to answer question 4, answer the following question: Suppose you were to be given a plot of the frequency content of the input signal, $X(\omega)$, and a plot of the frequency content of the output signal, $Y(\omega)$. How would you determine, using the two plots alone, what factor N was used for downsampling?

Figure 2 Don't try this at home, kids! [1]


Hi, Dr. Elizabeth?
Yeah, uh... I accidentally took the Fourier transform of my cat...
Meow!

2.4 Submission Rules

1. Submit your files *no later than* 10 minutes after the beginning of your next lab session the week of **April 11, 2011**.
2. Late submissions will *not* be accepted, except under unusual circumstances.
3. If the pre-lab exercises are not performed, you will get an immediate zero for the entire lab.
4. These exercises should be done *individually.*
5. Keep your work safe for further usage in the in-lab sections.

2.5 Submission Instructions

1. Log on to bSpace and click on the `Assignments` tab.
2. Locate the assignment for `Lab 7 Pre-Lab` corresponding to your section.
3. Attach your responses to the questions in section 2.3. Templates for this assignment are available, in DOC and TEX formats, but you need not use them.

3 In-Lab Section

3.1 Transforms: Signals in Disguise

We know that the Discrete Fourier Series expansion of a p-periodic signal $x(n)$ is given by

$$x(n) = \sum_{k=\langle p \rangle} X_k e^{ik\omega_0 n}.$$

This is also known as the **synthesis equation**, and allows us to synthesize a signal $x(n)$ from its DFS coefficients X_k.

SYNTHESIS EQUATION

For a few well-known signals, we can determine the DFS coefficients directly using, for example, the inverse Euler relations. In general, however, we determine these coefficients using the **analysis equation**:

ANALYSIS EQUATION

$$X_k = \frac{1}{p} \sum_{n=\langle p \rangle} x(n) e^{-ik\omega_0 n}.$$

This pair of equations—the analysis and the synthesis equations—helps us hop between the time and frequency domain representations of a signal. For this lab, however, we will use the **Discrete Fourier Transform**, which is very similar to the Discrete Fourier Series, except a factor of $1/p$ is attached to the synthesis equation instead of the analysis equation. Since we will only be concerned with the relative magnitudes of different coefficients, this minor change will not affect us, and we will not adjust for it.

DFT

We elect to use the DFT because we will be using the **Fast Fourier Transform**, a popular family of algorithms that implement the DFT (and thus, with a small change, the DFS). Computing the DFT directly from its definition is slow; with clever optimizations, the FFT allows us to compute the DFT significantly faster[3].

FFT

Before we proceed, a note of caution: **The DFT is *not* the same as the DTFT (the Discrete-Time Fourier Transform)**: the Discrete-Time Fourier Transform (DTFT), as you will see later in the course, applies to discrete-time signals that are not necessarily periodic. The DFT and the DTFT are significantly different. It is an unfortunate product of conventional nomenclature that two important concepts in signal processing are similarly and confusingly named.

Assume a periodic signal $x(n)$ with period p. For this signal, the FFT algorithm generates coefficients associated with equally spaced frequencies in the range from $-\pi$ radians per sample to π radians per sample, or from $-f_s/2$ Hz to $f_s/2$ Hz. It also generates only p coefficients, since, as we have shown in the pre-lab sections, we only really need p coefficients to completely characterize a periodic signal $x(n)$. With this in mind, **how far apart are two adjacent coefficients, in terms of the frequencies they are associated with?** Give your answer in the units of Hz; you will need this answer later.

There is, of course, the inverse FFT algorithm, which takes coefficients and constructs a signal from those coefficients. In rigorous terms,

$$x = \mathsf{ifft}(\mathsf{fft}(x)),$$

where fft is a function that represents the FFT algorithm, ifft is a function that represents the inverse FFT algorithm, and x is a periodic signal. We will also use the inverse FFT algorithm in the in-lab sections.

3.2 Hear a Frog, There a Babak

The audio files that will be used for this lab session are available on the lab section of the course website, as part of the lab 7 resources:

1. `Frogs`: A WAV file containing a sample of the calls of three Coqui frog species.
2. `Babak`: A WAV file containing a sample of the voice of Babak Ayazifar, specifically when he informs his EE20N lecture of an impending quiz, in the Fall 2008 semester.

In this lab session, since we will need to analyze sound files, the `Sound File Read Simple` block, located under `Programming` → `Graphics & Sound` → `Sound` → `Files`, will prove useful.

This block takes in a WAV file (short for **waveform audio format**) and returns an array of waveforms, one for each **channel** that the file occupies. Independent channels allow for the existence of **stereophonic sound**; in other words, independent channels can be played from different directions, to produce the phenomenon of *surround sound*. For our purposes, however, we will only need the first channel, and in doing so, we are instead using a **monaural sound**.

WAV
CHANNEL
STEREOPHONIC SOUND
MONAURAL SOUND

[3]For you computer science geeks out there, the direct implementation of the DFT has a running time of $O(n^2)$, n being the size of the input. This is a polynomial running time, but in real-life applications, inputs can be large, making this a slow algorithm. The FFT can compute the DFT with a running time of $O(n \log n)$, an exponential improvement. You will encounter the barebones of this algorithm if you take either EE123, Mathematics 116, or CS170.

1. Create a new VI called `Sound Frequency Content`. Drag a `Sound File Read Simple` block into the block diagram for the VI, and use an `Index Array` block to obtain the first channel (at array location `0`). Feed the result as input to a `Play Waveform` block.
2. Create a control as input on the `path` terminal to the `Sound File Read Simple` block. (Right-click on the terminal and select `Create` $\rightarrow$ `Control`.) Download the `Frogs` WAV sound file from the lab site, and access the file using this control, to make the file the input to your VI. Alternatively, you can replace the control with a constant if you do not want to select the file every time.
3. On the front panel, create a `Waveform Graph` to plot the waveform obtained in step 1.
4. Run the resultant VI: if done correctly, you will have successfully imported a channel of a WAV file into LabVIEW, as well as seen its visual representation.

We will now use the `FFT` block to determine the frequency content of our sound files.

5. Use a `Get Waveform Components` block to obtain the data values of the actual sound signal (`Y`) and the associated sampling period (`dt`).
6. Feed the `Y` component to an `FFT` block, located under `Signal Processing` $\rightarrow$ `Transforms`.
7. Add a boolean control or a constant to the `shift?` port of the `FFT` block, and set it to `true`.

   > Before running this VI, *always* set this boolean control or constant to `true`. This is important because, by default, the `FFT` block generates the DFS coefficients for frequencies in the range $[0, 2\pi)$, instead of $[-\pi, \pi)$. Setting the `shift?` boolean to true ensures that the `FFT` block centers the DFS coefficients around the **DC component** (zero frequency); in other words, setting the `shift?` boolean to `true` causes the block to generate the DFS coefficients for frequencies in the range $[-\pi, \pi)$.

8. On the front panel, plot the **magnitude** of the DFS components on the front panel against the correct frequencies on the x-axis.

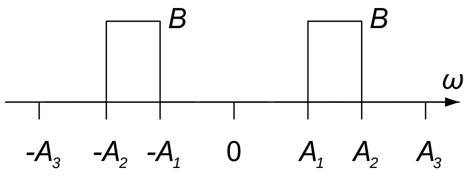
   > - Plot the **magnitude** of the DFS components, and not the DFS components themselves. Plotting the DFS components themselves requires an extra dimension: why?
   > - The frequencies used for plotting should span from $-f_s/2$ Hz to $f_s/2$ Hz, where f_s denotes the sampling frequency. How can you determine f_s with the information you have so far (specifically `dt`)?
   > - Based on your response to the question in section 3.1, what should the step size of your frequency array be?
   > - Relabel the x-axis as `Frequencies (Hz)` and the y-axis as `Coefficients (Magnitude)`.

9. Run the VI again with the `Frogs` WAV sound file. You should now see the frequency content of the `Frogs` file. The frequency content should be symmetric about the DC frequency; we will discover why in a later section. In particular, you should see three distinct frequency bands—one close to 2000 Hz (2 kHz), another close to 3000 Hz (3 kHz), and yet another at 5000 Hz (5 kHz). Each of these frequency bands are from one species of Coqui frog, and part of the in-lab section will be concerned with separating these frequency bands. For reference, the three frequency bands are how the three frogs manage to communicate with each other.
10. Run the VI with the `Babak` WAV sound file, and thus plot the frequency content of this sound file. Check that the frequency range of Babak's voice makes sense; as reference, remember that conversational human voice is generally between 300 Hz to 3000 Hz.

3.3 May the Filter Be With You

The filters that we will implement have the general frequency response $F(\omega)$ as shown in Figure 3. The assumptions made in Figure 3 are that A_1, A_2, and A_3 are integers, that $A_1 < A_2 < A_3$, and that the frequency content of the signal to be filtered lies in the range $[-A_3, A_3)$.

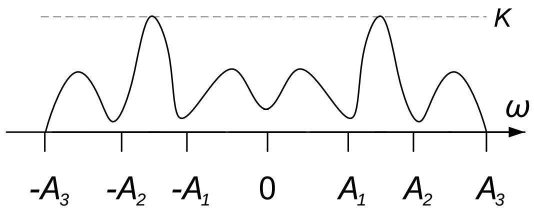
Figure 3 General frequency response of the filters to be implemented.



Consider a filter F that has the frequency response $F(\omega)$ presented in Figure 3 and the corresponding impulse response $f(t)$. Our task is to obtain a filtered signal $y(t)$ with frequency content $Y(\omega)$, from the input signal $x(t)$ with frequency content $X(\omega)$. As explained in section 2.2, we can do this either by convolving $x(t)$ with $f(t)$ (hard!), or we can multiply $X(\omega)$ by $F(\omega)$ to obtain $Y(\omega)$ and then generate $y(t)$ from $Y(\omega)$ (relatively easier!). To this end, we can multiply $F(\omega)$ with the frequency content of the input signal to obtain a filtered signal, with the frequencies in the ranges $[-A_2, -A_1)$ and $[A_1, A_2)$ **preserved and scaled** by B, with the other frequencies **suppressed**.

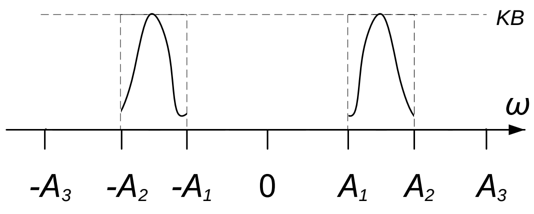
Thus, for example, an input signal with the frequency content shown in Figure 4:

Figure 4 Frequency content of the input signal.



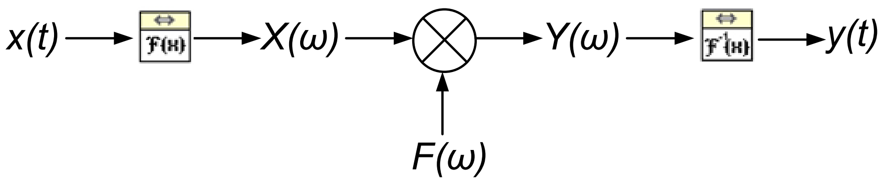
will produce an output signal with the frequency content shown in Figure 5.

Figure 5 Frequency content of the output signal.



With these observations in mind, we come up with the following important idea: Since we already have the frequency content of the input signal in a one-dimensional array, generated by the `FFT` block, all we need is a one-dimensional array representation of the **frequency response of the filter**, as shown in Figure 3, and multiply the two arrays pointwise. The result is the frequency content of the output signal, also in a one-dimensional array. We then use the `Inverse FFT` block to generate the actual output signal. This process is summarized in Figure 6.

Figure 6 Process flow to generate the filtered output signal.



To generate the one-dimensional array representation of the frequency content of the filter, we notice that, because we will be zeroing out the frequencies in the range $[-A_3, -A_2)$, we need to start out with an array of as many zeros as can span the range $A_3 - A_2$. Since we will want to preserve the frequencies in the range $[-A_2, -A_1)$, we need to append an array of ones that span the range $A_2 - A_1$. Eventually, we achieve the following line of `MathScript` code to obtain the array representation of the frequency response of the filter (all in one line):

```
filter = B .* [zeros(1, (A3 - A2))   ones(1, (A2 - A1))
    zeros(1, (2 * A1))    ones(1, (A2 - A1))   zeros(1, (A3 - A2))];
```

If we have n one-dimensional arrays `V1`, `V2`, ..., `Vn`, we can concatenate these arrays into one one-dimensional array `V` by simply using the construct

```
V = [ V1   V2   V3   ...   Vn ];
```

as we have done above.

Also, the `MathScript` command `zeros(m, n)` generates an $m \times n$ matrix of zeros, while the command `ones(m, n)` generates an $m \times n$ matrix of ones. The second argument to both commands is important: without it, `zeros(m)` returns an $m \times m$ matrix.

Now, we will only need to multiply the above array with the array containing the DFS coefficients of the input signal, to suppress the DFS coefficients corresponding to the frequencies to be filtered out, and to scale the DFS coefficients corresponding to the frequencies to be allowed through.

OK, enough theory. Let's get cracking!

3.4 Frogs Will Be Frogs

In this section, we will work on creating an equalizer-like VI that will allow us to control the contributions of the sounds of the different frog species, in the `Frogs` WAV file, to the final output sound signal. In particular, after this section, we will be able to block the sound of one frog species and amplify the sound of another, merely by manipulating sliders.

1. Create a copy of your `Sound Frequency Content` VI, from section 3.2, and save it as the `Frog Filter` VI.

2. Now, we should determine the ranges of frequencies in the input array that we will need to filter or amplify. To this end, *temporarily* change how the x-axis of the plot of the DFS coefficients is generated, so that the **position** of the various DFS coefficients in the array generated by the `FFT` block is displayed, instead of the frequency corresponding to each of the DFS coefficients as shown by the

earlier plot. In other words, you should now be plotting the DFS coefficients against **array indices**, not frequencies.

3. Determine the approximate portions in the array `FFT` corresponding to the frequency content of each frog sound; for example, the frequency content for the voices of one of the frogs fills array locations (or sample indices) from about 17000 to 26000, and from about 61000 to 70000. Do the same for the voices of the other two frogs.

   Do not simply copy the above numbers alone: they are not the entire solution. In fact, you can make the numbers given above more precise.

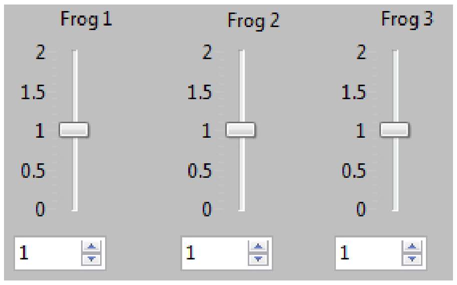
4. Once done, *reinstate* the old x-axis, so that the DFS coefficients are now plotted against their corresponding frequencies again.

5. Create a new `MathScript Node` with inputs `B1`, `B2`, `B3`, and `FFT`. To each of the inputs `B1`, `B2`, and `B3`, attach a `System Vertical Slide` control, available under the `System` palette on the front panel.

6. Change the bounds of each slider to go from `0` to `2`. Also, enable the `Digital Display` for each slider, to enable easier entry of, and reading of, values on the slider: right-click on the slider and select `Visible Items` $\rightarrow$ `Digital Display`.

7. Rename the labels of the slider corresponding to `B1` as `Frog 1`, corresponding to `B2` as `Frog 2`, and corresponding to `B3` as `Frog 3`. The resultant front panel should look similar to the one shown in Figure 7.

Figure 7 Front panel of the `Frog Filter` VI.



8. Move back to the new `MathScript Node` in the block diagram. For each frog voice, create an array that represents a filter to extract that voice.

   Use the values you found in step 3 to create the following example array for each frog voice (all in one line):

```
filter = B .* [zeros(1, A3 - A2)   ones(1, (A2 - A1))
zeros(1, ___)   ones(1, ___)   zeros(1, ___)];
```

   Note that here, A_1, A_2, and A_3 are *indices* and not frequencies. Also, the above definition will be different for each frog.

   Ensure symmetry: make the arrays of zeroes and the arrays of ones on either end equal in length, and place the length of the remaining values in the middle `zeros` array. Again, you will find the `MathScript` command `length` useful. You may also need to tweak the values that you obtained in step 3 appropriately.

   If you find that you are running out of memory, you are probably forgetting the second parameter to the `zeros` and `ones` `MathScript` commands.

9. Connect the input FFT of the MathScript Node to the output of the FFT block.

10. Generate the DFS coefficients of the output signals Out1, Out2, and Out3, for each of the frog voices respectively, by multiplying the filters with the array FFT. Generate the output array Out that is obtained by adding together these output signals.

11. Bring Out as an output from the MathScript Node. Plot the output DFS components of the signal Out, similar to how you plotted the DFS components in section 3.2.

12. Run your VI with the sliders set to different values. If done correctly, you should be able to selectively amplify or block different frog voices; in particular, setting the slider for Frog 1 to zero should block the highest frequency frog voice, while setting the slider for Frog 3 to 2 should amplify the DFS coefficients for the lowest frequency frog voice by a factor of 2. However, we also have to *hear* the results of our handiwork.

13. We had used the FFT block to move from the time domain to the frequency domain; we should thus use the Inverse FFT block, also located under Signal Processing → Transforms, to move from the frequency domain back to the time domain. Feed a wire from the Out output of the MathScript Node into the FFT{X} terminal of the Inverse FFT block. As you did with the FFT block, add a boolean constant or a control to the shift? port of the Inverse FFT block and set it to true. The Inverse FFT block will then produce a signal in the time domain that has the provided DFS coefficients. This is the output signal that we need.

14. Build a new waveform from the output of the Inverse FFT block and then play the new waveform.

    Feed the output into the Y port of a Build Waveform block. The sampling period (dt) to be used is the same as that produced by the Get Waveform Components block in step 5 of section 3.2. Feed the resultant waveform into a Play Waveform block.

    You should have only one Play Waveform block in your block diagram. Delete unused blocks.

15. Congratulations! You have made a VI that can filter out different frog voices from the sound sample provided. Play with your toy for a bit; observe the frequency domain and time domain representations of the output signals, as well as hear the output signals themselves, for various inputs on the sliders.

3.5 The Voices Are Telling Me About A Quiz

We will now manipulate Babak's voice in various (mysterious) ways, simply by altering the frequency content of the sound signal representing his voice using a blackbox. Your task in this part of the in-lab section is to figure out what this blackbox does by performing a frequency content analysis of the output signals.

Download the Babak LLB from course lab site and explore its block diagram. The input audio signal is fed into a blackbox, which has four outputs. Each of these outputs are fed to an FFT block, which generates the DFS coefficients for the corresponding output; the magnitudes of these coefficients are then plotted using a Waveform Graph. There is also a Play Waveform block that plays the waveform fed into it; by default, it is set to play the input audio signal.

The blackbox performs the following four transformations to its input signal, **not necessarily in the same order as the order of the output signals**:

T1. Nothing at all.

T2. Upsampling by a factor of N.

T3. Downsampling by a factor of N.

T4. Moving frequencies where the frequency content was rich (or where the magnitudes of the corresponding DFS coefficients were large) *away* from the DC (zero) frequency, but without dilation or contraction.

Run the VI with the `Babak` WAV file. Based on the frequency content graphs of the input signal and the output signals that are produced, answer the following questions:

1. Determine the output signals that correspond to each of the four transformations.
2. By what factor N was the input signal upsampled during the upsampling transformation?
3. By what factor N was the input signal downsampled during the downsampling transformation?

We will now listen to each of the output audio signals of the blackbox. By connecting each output audio signal to the `Get Waveform Components` block in turn, and based on your observations, answer the following questions:

1. Why does the upsampling transformation produce an output signal that sounds like the input signal slowed down?
2. Why does the downsampling transformation produce a chipmunk-like output signal?

4 Acknowledgments

Special thanks go out to the teaching assistants (TAs) of the Spring 2009 semester (Vinay Raj Hampapur, Miklos Christine, Sarah Wodin-Schwartz), of the Fall 2009 semester (David Carlton, Judy Hoffman, Mark Landry, Feng Pan, Changho Suh), and of the Spring 2010 semester (Xuan Fan, Brian Lambson, Kelvin So) for providing suggestions, ideas, and fixes to this lab guide.

References

[1] Courtesy of XKCD