

## Supplement to Chapter 4 of *The Science of Digital Media* – Digital Audio Representation

### Mathematical Modeling and Visualization Exercise – Digital Audio Representation > Windowing Functions for the Fourier Transform<sup>1</sup>

---

#### Introduction:

The Fourier transform is used to convert data from the time domain to the frequency domain. Once in data is in the frequency domain, a frequency spectrum can be generated which shows the magnitude of each of the frequency components. A spike in the frequency spectrum corresponds to a dominant presence of that frequency in the waveform, while a low value indicates that there is little or none of that frequency in the signal.

However, there are some problems with the frequency spectrum. In most cases, the spike on the spectrum for each component frequency is *not* infinitesimally thin; there are surrounding frequencies that also have positive values on the frequency spectrum even though they are *not* actually present in the original signal. This phenomenon is called *spectral leakage*, and it occurs because of the way the Fourier transform separates the signal into windows. A window is a set of samples upon which the transform operates. When performing its calculations, the transform assumes that the window upon which it is currently operating repeats indefinitely; that is, the window is assumed to constitute one complete cycle of a periodic wave. However, this usually does not accurately depict the signal, and a problem arises when there is a discontinuity between the wave's position at the end of one cycle and its position at the beginning of the next. If there is a discrepancy here, spectral leakage will result. This will cause the frequency spectrum to show spurious frequencies that are not actually present in the signal.

In order to reduce spectral leakage, windowing functions are used. These functions reduce the amplitude of the waveform at the beginning and end of each window, thereby decreasing the amount of spectral leakage by reducing the discrepancy between the end of the cycle and the beginning. There are four popular windowing functions, shown below.

$$u(t) = \begin{cases} \frac{2t}{T} & \text{for } 0 \leq t < \frac{T}{2} \\ 2 - \frac{2t}{T} & \text{for } \frac{T}{2} \leq t \leq T \end{cases}$$

**Triangular Windowing Function**

$$u(t) = \frac{1}{2} \left[ 1 - \cos\left(\frac{2\pi t}{T}\right) \right] \text{ for } 0 \leq t \leq T$$

**Hanning Windowing Function**

---

<sup>1</sup>This material is based on work supported by the National Science Foundation under Grant No. DUE-0340969. This worksheet was written by Todd Martin and Jennifer Burg (burg@wfu.edu).

$u(t) = 0.54 - 0.46 \cos\left(\frac{2\pi t}{T}\right) \text{ for } 0 \leq t \leq T$ <p style="text-align: center;"><b>Hamming Windowing Function</b></p>	$u(t) = 0.42 - 0.5 \cos\left(\frac{2\pi t}{T}\right) + 0.08 \cos\left(\frac{4\pi t}{T}\right)$ <p style="text-align: center;"><i>for</i> <math>0 \leq t \leq T</math></p> <p style="text-align: center;"><b>Blackman Windowing Function</b></p>
--	---

In this worksheet, we will use MATLAB first to create the frequency spectrum of a simple sinusoidal wave and illustrate the existence of spectral leakage. A windowing function will then be used to reduce the spectral leakage, and the resulting frequency spectrum (without leakage) will be calculated.

**Tips for Using MATLAB:**

In MATLAB, the Fourier transform can be implemented through the *fft* function. This function operates on a vector (i.e., array) of data and actually performs the *fast* Fourier transform (FFT) on the data. The results will be the same as the DFT, but the operations can be performed faster for the MATLAB calculations. Because we are studying digital audio, we will be using the one-dimensional Fourier transform, but MATLAB also provides the *fft2* function to implement the two-dimensional Fourier transform for image processing.

In order to implement the Fourier transform, first define a function in the time domain upon which you wish to perform the transform. *Do not* define the function using the *inline* command; the *fft* function will not operate on *inline* functions. Before we do this, we must consider the time values over which our function will vary. For our example, we will use 1024 samples, sampled at 1000 samples/sec. This means that the signal will have a length, in time, of 1.024 seconds. This will be our domain, and MATLAB requires that it be specified before defining the function itself. This is shown below:

```
>> t=0:0.001:1.023;
```

Note that the above command is of the form  
 $t=0:sampleInterval:seconds;$   
 where  $seconds/sampleInterval =$  the number of samples per second

We will thus have 1024 samples in this period, each one separated by 0.001 seconds – or 1000 samples/second. Once we have done this, we can define the function itself. An example of a function that produces a simple cosine wave with a frequency of 450 Hz is shown below.

```
>> f = cos(450*2*pi*t);
```

The next step in the process is to apply the *fft* function to *f*. In MATLAB, the *fft* function requires the specification of the number of points upon which it will be applied. The number of points indicates frequency resolution, or number of frequency components, that the Fourier transform will calculate. This should match the number of samples that you established when defining

the time domain above. For our example, there are 1024 samples in our function, so the *fft* function should have 1024 points. Once we know this, we can apply the *fft* function to our cosine wave to transform it into the frequency domain. We will assign it to the variable *y* so that we can reference it later.

```
>> y = fft(f, 1024);
```

Now *y* defines a vector with length 1024 (because this was the number of points in our transform). However, because of the Nyquist theorem, only the first one-half (or 512) of these values are *valid* frequencies. Thus, when we graph the output of the Fourier transform, we will restrict the domain to only the first 512 frequency components. Also, the frequency spectrum that we are trying to generate requires the *magnitude* of all of the complex numbers. The magnitude of a complex number  $z = a + bi$  is given below:

$$|z| = \sqrt{a^2 + b^2}$$

In MATLAB, the magnitude can be calculated with the *abs* function when applied to a complex number. Both the magnitude and the Nyquist adjustments can be accomplished as follows:

```
>> y = abs(y);  
>> y = y(1:512);
```

The second statement above cuts *y* down to its first 512 elements.

Now we must define the scale upon which this function will be plotted in MATLAB. This can be done with the following command of the form  $x = (0:numSamples/2 - 1) * (sampling\ rate / numSamples)$ .

Let's break down this command. With  $(0:numSamples/2)$  we create a vector of integers values that go from 0 to  $numSamples/2 - 1$ . We then scale each value in the vector by multiplying it by  $sampling\ rate / numSamples$ . These are the *x* values for which *y* values will be plotted to give us a graphical representation of the results of the Fourier transform. We have to scale the *x* values in this way so that they correctly represent the frequency components determined by the Fourier transform, with the *y* values representing their corresponding amplitudes.

```
>> x = (0:511) * (1000/1024);
```

We can now create graph with the *plot* command.

```
>> plot(x, y);
```

This command plots the magnitude of 512 frequency components, the maximum of which is 500 Hz, calculated with the *fft* function and stored in the variable *y*. The graph that results should have a spike at 450 Hz – corresponding to the 450 Hz cosine wave that we used for *f*.

Now that you have an idea of how to implement and use the Fourier transform in MATLAB, let's do some exercises to examine the effect of windowing functions on the results of the transform.

### Exercise 1

In order to see the effect of windowing functions, you will first perform the Fourier transform (using the MATLAB *fft* function) on a simple sine wave with a frequency of 300 Hz. In later exercises, we will apply a windowing function and then apply the Fourier transform to the resulting windowed waveform.

The first step is to define the time domain for the variable *t*. For this exercise, use a sampling rate of 1000 samples/second and a time variable that varies between 0 and 2.047 seconds. (This will result in 2048 samples.)

Then, define a sine wave with a frequency of 300 Hz and assign it to the variable *p*.

Perform the Fourier transform on *p* using the MATLAB *fft* function with the appropriate number of points. Assign the result to the variable *r*. Perform the transformations discussed above in order to adjust *r* for the frequency spectrum and define the *scale* variable.

Once *r* is calculated, plot *only the valid* frequency components of the transformed wave and verify the spike at the correct frequency.

### Exercise 2

Now that you have plotted the frequency spectrum of the sine wave stored in the variable *p*, analyze what it means. While there should be a well-defined spike at 300 Hz, the spike is *not* infinitesimally thin; even though the *only* frequency present in the sine wave is 300 Hz, the frequencies around 300 Hz are also part of the spike. Why does this occur?

From the earlier discussion of windowing functions, you should recognize this as spectral leakage. This occurs when an integer number of cycles of the 300 Hz sine wave are not captured in the 2048 samples of the wave. Calculate the exact number of cycles of the sine wave that occur in the 2048 samples. (It should not be an integer.)

To see the effect of the windowing function, first plot the 300 Hz sine wave stored as variable *p* using the *plot* command. You may want to plot only the first 100 samples in order to get a more detailed view of the wave.

We will use the Hanning windowing function for this exercise. Plot the windowing function by first storing the equation given in the Introduction as  $h$ . Use 2.048 for the  $T$  value – it is the total time of the wave that we are analyzing.

Now, implement the Hanning windowing function to reduce the amount of spectral leakage when the sine wave is transformed. In order to implement the windowing function, the 300 Hz sine wave must be multiplied by the equation for the Hanning windowing function. In MATLAB, the equation below can be used. Note that `.*` is used for multiplication rather than `*`. This is necessary because both  $p$  and  $h$  are vectors, and we do not want to perform matrix multiplication.

```
>> w = p.*h;
```

Now graph  $w$  using the MATLAB `plot` command. What difference can you see between this graph and that of the original function? Based on the discussion of windowing functions, how will this affect spectral leakage?

### Exercise 3

Now that the windowing function has been defined and calculated for our 300 Hz sine wave, the next step is to compute the Fourier transform for this new, windowed function and see how the frequency spectrum compares to the previous one.

Using the format from Exercise 1, calculate the Fourier transform of  $w$  using the `fft` command and store this result in a variable called  $r2$ . Remember to calculate the `scale` variable and modify  $r2$  according to the example.

Plot the frequency spectrum of  $w$  using the `plot` command as demonstrated in Exercise 1 to plot the first spectrum.

How does this frequency spectrum compare to the first? Focus on the area around the spike in the graph. Is the “slope” up to the spike more or less gradual than before? What does this tell you about the effect of the windowing function on spectral leakage?