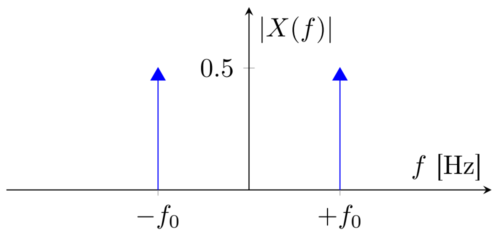
7.7 Frequency Analysis using Python

Here, we provide an example of using Python to perform frequency analysis on a sampled sine wave and using spectral peak detection to find the frequency of the sine wave. We know that the continuous-time Fourier transform (CTFT) of an analog sine wave with frequency f_0 is

$$X(f) = \mathcal{F}[\sin(2\pi f_0 t)] = \frac{1}{2j}[\delta(f - f_0) + \delta(f + f_0)].$$



Of course, once the sine wave is sampled and windowed, what is retrieved will not quite be the same as the CTFT plot above. Before starting, three Python modules will need to be loaded in, as seen below:

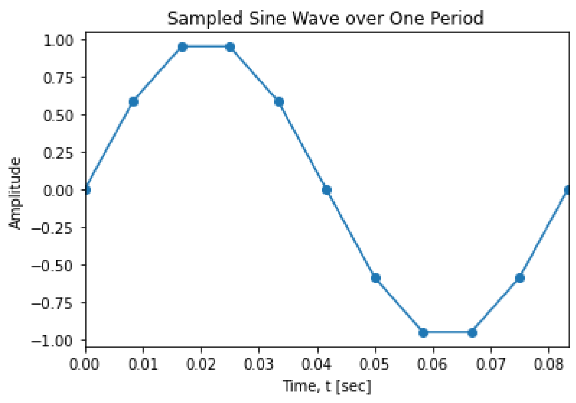
```python
import numpy as np
import matplotlib.pyplot as plt
import time
```

7.7.1 Single-Frequency Tone (Sampled at Integer Multiples of f_0)

Suppose an analog sine wave with frequency $f_0 = 12$ [Hz] is being sampled at sampling rate f_s. If f_s is an integer multiple of f_0 such that letting frame size $N = f_s/f_0$ represents one cycle being sampled, then f_0 will be detected.

First, we simulate a sine wave being sampled at $f_s = 120$ [samp/sec], i.e. ten times the wave frequency $f_0 = 12$ [Hz].

```python
# SIMULATE SAMPLING A SINE WAVE
f0 = 12                     # analog signal frequency = 12 Hz
fs = 120                    # sampling rate = 120 samp/sec
Ts = 1/fs                   # sampling period
n = np.arange(0,1000)       # array of (DT) indices, n
t = n*Ts                    # array of (CT) time points corresponding to n
x_t = np.sin(2*np.pi*f0*t) # analog signal sampled at fs

# PLOT SAMPLED WAVE OVER ONE PERIOD
plt.plot(t,x_t, marker='o')
plt.xlim([0,1/f0])
plt.title('Sampled Sine Wave over One Period')
plt.xlabel('Time, t [sec]')
plt.ylabel('Amplitude')
plt.show()
```



7.7.1.1 Window One Cycle of Sampled Wave

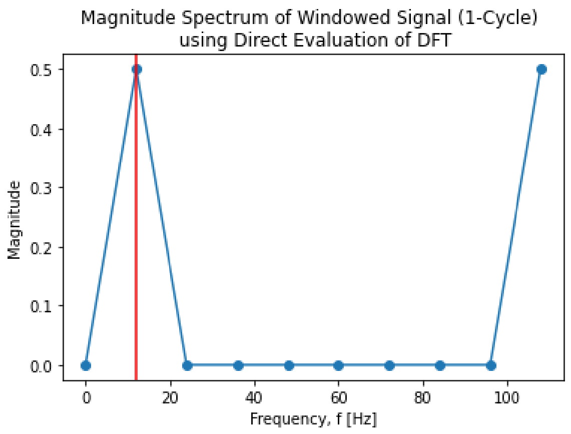
Computing the DFT via direct evaluation of the DFT matrix equation $X = Wx$ gives time complexity $O(N^2)$.

In the plot generated below, the blue curve depicts the magnitude spectrum of the windowed sine wave that was sampled, whereas the red vertical line represents the frequency f_0 of the sine wave.

The code block also outputs the time it took to compute the DFT for this particular instance of execution, as well as the peak frequency.

```
# WINDOW ONE PERIOD OF SAMPLED WAVE
t_cycle = t[t < 1/f0]                         # time points corresponding to one period of
                                                 sampled wave
x_t_cycle = x_t[:len(t_cycle)]                # one period of sampled wave
w = np.ones_like(t_cycle)                     # rectangular window, w
N = len(t_cycle)                              # frame size N

# COMPUTE N-POINT DFT
f = np.arange(0, N) * fs / N                  # array of CT frequencies, f

t0 = time.time()
kn = np.array([[(k*n) \
        for k in np.arange(0,N)] \
            for n in np.arange(0,N)])
W_twiddle = np.exp(-1j * 2*np.pi * kn / N)    # matrix of twiddle factors
X_f = W_twiddle @ (x_t_cycle * w)             # N-point DFT of windowed signal

dft_elapsed = time.time()-t0
print(f"DFT elapsed time: {dft_elapsed:.6f} seconds")

rect_normz = 1 / len(t_cycle)         # normalization factor for spectrum of rect-
                                                 windowed signal

plt.plot(f, abs(X_f) * rect_normz, marker='o')
plt.title('Magnitude Spectrum of Windowed Signal (1-Cycle) \n using Direct Evaluation
                                                  of DFT')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.axvline(x=f0, color='r')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f[np.argmax(abs(X_f))]):.3f} [Hz]")
```

```
DFT elapsed time:  0.000470 seconds
```

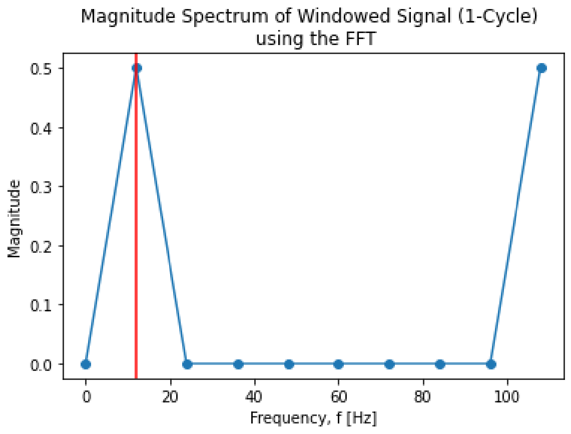


```
Peak frequency:  12.000 [Hz]
```

Notice that since f_s is a multiple of f_0 and one cycle is sampled, we are able to pick up f_0 precisely. Before moving on, we briefly introduce the Fast Fourier Transform (FFT), which is designed to compute the DFT in less time. Generally, the time complexity is $O(N \log_2 N)$.

```
# WINDOW ONE PERIOD OF SAMPLED WAVE
t_cycle = t[t < 1/f0]                    # time points corresponding to one period of
                                           sampled wave
x_t_cycle = x_t[:len(t_cycle)]           # one period of sampled wave
w = np.ones_like(t_cycle)                # rectangular window, w
N = len(t_cycle)                         # frame size N

# COMPUTE N-POINT DFT
f = np.arange(0, N) * fs / N             # array of CT frequencies, f
t0 = time.time()
X_f = np.fft.fft(x_t_cycle * w, N)       # N-point DFT of windowed signal using FFT

fft_elapsed = time.time()-t0
print(f"FFT elapsed time: {fft_elapsed:.6f} seconds")
rect_normz = 1 / len(t_cycle)            # normalization factor for spectrum of rect-
                                           windowed signal

# PLOT MAGNITUDE SPECTRUM
plt.plot(f, abs(X_f) * rect_normz, marker='o')
plt.title('Magnitude Spectrum of Windowed Signal (1-Cycle) \n using the FFT')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.axvline(x=f0, color='r')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f[np.argmax(abs(X_f))]):.3f} [Hz]")
```

```
FFT elapsed time:  0.000221 seconds
```

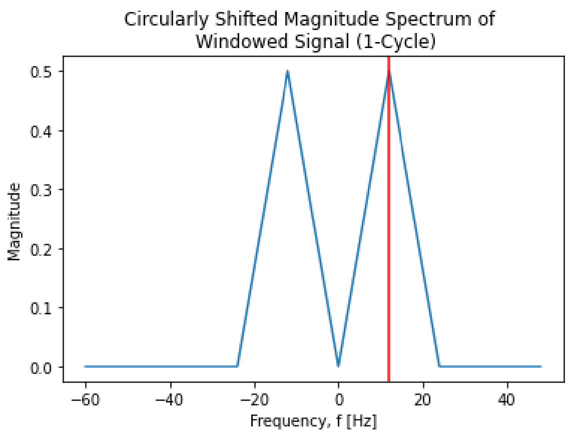


```
Peak frequency:  12.000 [Hz]
```

Here we can see that the FFT took less time to compute the DFT than directly evaluating the DFT matrix equation. While the difference in compute time is small in this example, the FFT becomes more advantageous as N increases to some large frame size. This particularly matters in the realm of big data, when thousands and thousands of samples are collected and analyzed at once. For the rest of this section, we will use **fft** to compute the DFT.

Now we want to circularly shift the spectrum such that $f = 0$ is at the center of the shifted spectrum.

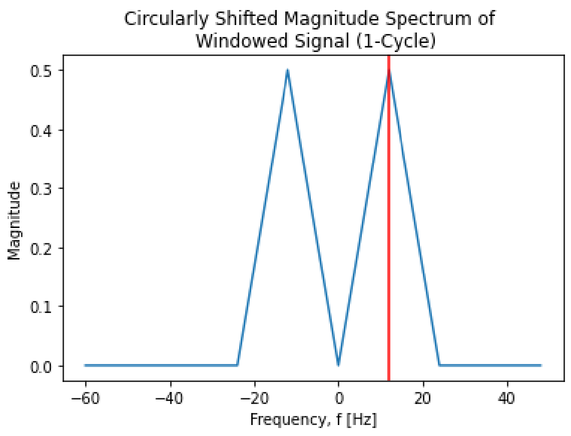
```
# CIRCULARLY SHIFT TO GET SYMMETRIC MAGNITUDE SPECTRUM
if N % 2 == 0:
    f_shift = np.arange(-N/2, N/2) * fs / N
    X_f_shift = np.hstack((X_f[int(N/2):], X_f[:int(N/2)]))
else:
    f_shift = np.linspace(-(N-1)/2, (N-1)/2, N) * fs / N
    X_f_shift = np.hstack((X_f[int((N+1)/2):], X_f[:int((N+1)/2)]))

plt.plot(f_shift, abs(X_f_shift) * rect_normz)
plt.title('Circularly Shifted Magnitude Spectrum of \n Windowed Signal (1-Cycle)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.axvline(x=f0, color='r')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f_shift[np.argmax(abs(X_f_shift))]):.3f} [Hz]")
```



```
Peak frequency:  12.000 [Hz]
```

The numpy package in Python has built-in functions **fftshift** and **fftfreq** that will take care of the circular shifting.

```
# CIRCULARLY SHIFT TO GET SYMMETRIC MAGNITUDE SPECTRUM
f_shift = np.fft.fftshift(np.fft.fftfreq(N, d=Ts))
X_f_shift = np.fft.fftshift(np.fft.fft(x_t_cycle * w, N))

plt.plot(f_shift, abs(X_f_shift) * rect_normz)
plt.title('Circularly Shifted Magnitude Spectrum of \n Windowed Signal (1-Cycle)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.axvline(x=f0, color='r')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f_shift[np.argmax(abs(X_f_shift))]):.3f} [Hz]")
```

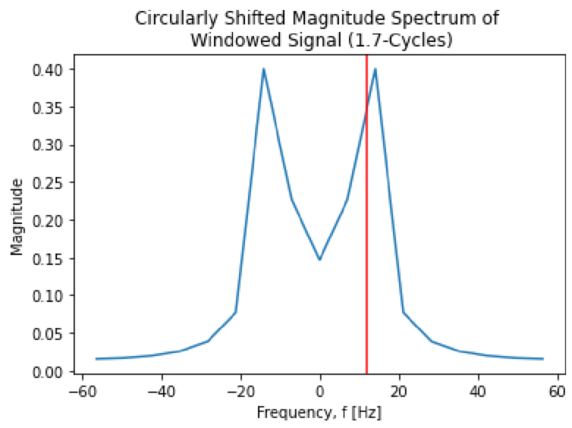


```
Peak frequency:  12.000 [Hz]
```

7.7.1.2 Window Non-Integer Cycles of Sampled Wave

Now suppose a non-integer number of cycles were windowed.

```
# WINDOW NON-INTEGER PERIOD OF SAMPLED WAVE
t_cycle = t[t < 1.7/f0]                    # time points corresponding to 1.7 cycles of
                                              sampled wave
x_t_cycle = x_t[:len(t_cycle)]             # 1.7 cycles of sampled wave
w = np.ones_like(t_cycle)                  # rectangular window, w
N = len(t_cycle)                           # frame size N

# CIRCULARLY SHIFT TO GET SYMMETRIC MAGNITUDE SPECTRUM
f_shift = np.fft.fftshift(np.fft.fftfreq(N, d=Ts))
X_f_shift = np.fft.fftshift(np.fft.fft(x_t_cycle * w, N))
rect_normz = 1 / len(t_cycle)

plt.plot(f_shift, abs(X_f_shift) * rect_normz)
plt.title('Circularly Shifted Magnitude Spectrum of \n Windowed Signal (1.7-Cycles)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.axvline(x=f0, color='r')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f_shift[np.argmax(abs(X_f_shift))]):.3f} [Hz]")
```

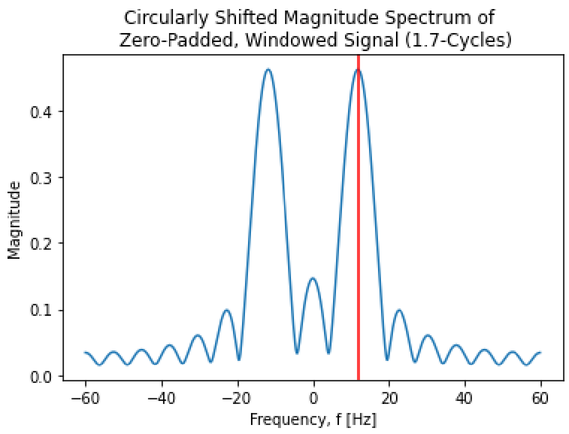


```
Peak frequency:  14.118 [Hz]
```

As you can see, the peak frequency detected is far from 12 [Hz]. To remedy that as best as we can, increase the frame size N either by extending the number of samples or by zero-padding. Assuming that storage is too expensive to allow hundreds of samples store at once, we opt to zero-pad.

In doing so, we also begin to smooth out the spectrum. Here, we can see the effects of convolving a sine wave with the Dirichlet kernel from windowing the signal.

```
# APPLY ZERO PADDING
N = 2048

f_pad = np.fft.fftshift(np.fft.fftfreq(N, d=Ts))
X_f_pad = np.fft.fftshift(np.fft.fft(x_t_cycle * w, N))

plt.plot(f_pad, abs(X_f_pad) * rect_normz)
plt.title('Circularly Shifted Magnitude Spectrum of \n Zero-Padded, Windowed Signal (
                                              1.7-Cycles)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.axvline(x=f0, color='r')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f_pad[np.argmax(abs(X_f_pad))]):.3f} [Hz]")
```

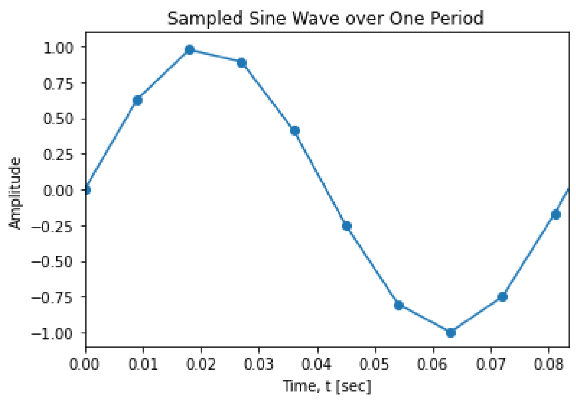


```
Peak frequency:  11.836 [Hz]
```

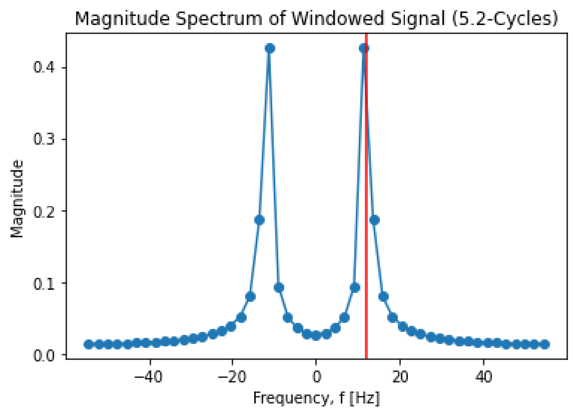
7.7.2 Single-Frequency Tone (Sampled at Non-Integer Multiples of f_0)

Now suppose the same analog sine wave (with $f_0 = 12$ [Hz]) is being sampled at a sampling rate that is not an integer multiple of f_0 – for example, at $f_s = 111$ [samp/sec]. Then f_0 will not be easily picked up with precision, regardless of how many cycles are windowed (integer or not).

```
# SIMULATE SAMPLING A SINE WAVE
f0 = 12                         # analog signal frequency = 12 Hz
fs = 111                        # sampling rate = 111 samp/sec
Ts = 1/fs                       # sampling period
n = np.arange(0,1000)           # array of (DT) indices, n
t = n*Ts                        # array of (CT) time points corresponding to n
x_t = np.sin(2*np.pi*f0*t)      # analog signal sampled at fs

# PLOT SAMPLED WAVE OVER ONE PERIOD
plt.plot(t,x_t, marker='o')
plt.xlim([0,1/f0])
plt.title('Sampled Sine Wave over One Period')
plt.xlabel('Time, t [sec]')
plt.ylabel('Amplitude')
plt.show()
```



```
# INCREASE NUMBER OF CYCLES
t_cycle = t[t < 5.2/f0]                # time points corresponding to 5.2 cycles of
                                         sampled wave
x_t_cycle = x_t[:len(t_cycle)]         # 5.2 cycles of sampled wave
w = np.ones_like(t_cycle)              # rectangular window, w
N = len(t_cycle)                       # frame size N

# COMPUTE N-POINT DFT
f = np.fft.fftshift(np.fft.fftfreq(N, d=Ts))
X_f = np.fft.fftshift(np.fft.fft(x_t_cycle * w, N))
rect_normz = 1 / len(t_cycle)

# PLOT MAGNITUDE SPECTRUM
plt.plot(f, abs(X_f) * rect_normz, marker='o')
plt.title('Magnitude Spectrum of Windowed Signal (5.2-Cycles)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.axvline(x=f0, color='r')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f[np.argmax(abs(X_f))]):.3f} [Hz]")
```

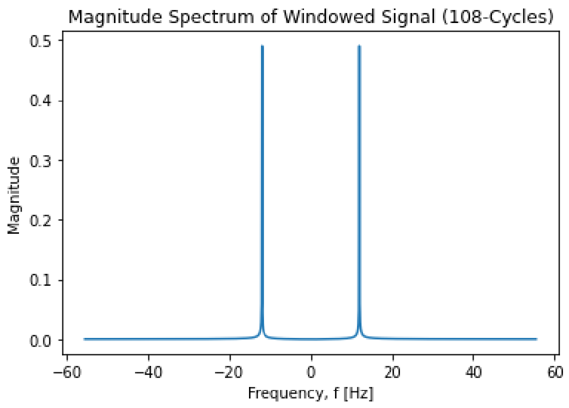


```
Peak frequency: 11.327 [Hz]
```

As discussed before, there are two ways to resolve this. The first method is to sample more of the signal to get a larger frame size N. Below, we use all 1000 samples from our simulation as our N, which corresponds to 108 cycles here.

```
# INCREASE NUMBER OF CYCLES
```

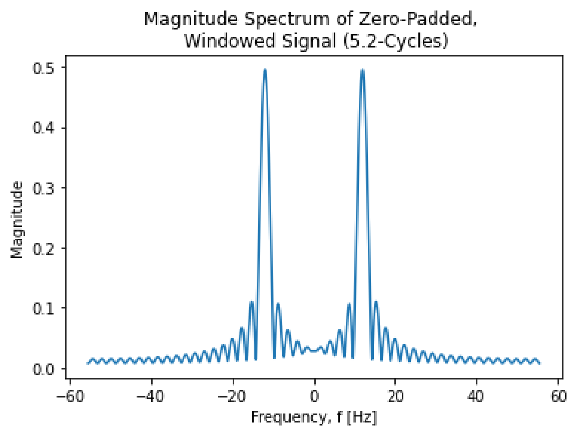
```
t_cycle = t[:]                            # 1000 time points corresponding to sampled
                                            wave
x_t_cycle = x_t[:]                        # 1000 samples of sampled wave
w = np.ones_like(t_cycle)                 # rectangular window, w
N = len(t_cycle)                          # frame size N

# COMPUTE N-POINT DFT
f = np.fft.fftshift(np.fft.fftfreq(N, d=Ts))
X_f = np.fft.fftshift(np.fft.fft(x_t_cycle * w, N))
rect_normz = 1 / len(t_cycle)

# PLOT MAGNITUDE SPECTRUM
plt.plot(f, abs(X_f) * rect_normz)
plt.title('Magnitude Spectrum of Windowed Signal (108 Cycles)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f[np.argmax(abs(X_f))]):.3f} [Hz]")
```



```
Peak frequency:  11.988 [Hz]
```

However, as mentioned before, storage can be costly. In the event that 1000 samples is too high of a number to store all at once, one can choose to zero-pad instead so that the frame size N is larger.

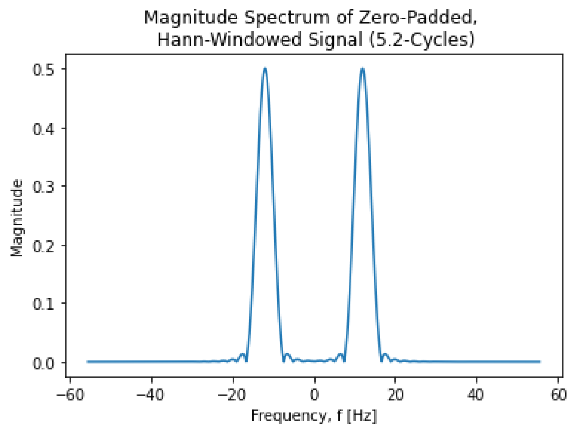
```
t_cycle = t[t < 5.2/f0]                   # time points corresponding to 5.2 cycles of
                                            sampled wave
x_t_cycle = x_t[:len(t_cycle)]            # 5.2 cycles of sampled wave
w = np.ones_like(t_cycle)                 # rectangular window, w

# APPLY ZERO PADDING
N = 2048

f_pad = np.fft.fftshift(np.fft.fftfreq(N, d=Ts))
X_f_pad = np.fft.fftshift(np.fft.fft(x_t_cycle * w, N))
rect_normz = 1 / len(t_cycle)

plt.plot(f_pad, abs(X_f_pad) * rect_normz)
plt.title('Magnitude Spectrum of Zero-Padded, \n Windowed Signal (5.2-Cycles)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f_pad[np.argmax(abs(X_f_pad))]):.3f} [Hz]")
```



```
Peak frequency:  12.032 [Hz]
```

Of course, before zero-padding, we can also choose a window that gives us the mainlobe width and sidelobe behavior we would want from the spectrum. Here, we use the most common non-rectangular window: the Hann window.

```
# APPLY HANNING, then ZERO PAD
w = np.hanning(len(t_cycle))
N = 2048

f_pad = np.fft.fftshift(np.fft.fftfreq(N, d=Ts))
X_f_pad = np.fft.fftshift(np.fft.fft(x_t_cycle * w, N))
hann_normz = 1 / ((len(t_cycle) - 1) / 2) # normalization factor for spectrum of Hann
                                           -windowed signal

plt.plot(f_pad, abs(X_f_pad) * hann_normz)
plt.title('Magnitude Spectrum of Zero-Padded, \n Hann-Windowed Signal (5.2-Cycles)')
plt.xlabel('Frequency, f [Hz]')
plt.ylabel('Magnitude')
plt.show()

# PRINT FREQ WHERE PEAK
print(f"Peak frequency: {abs(f_pad[np.argmax(abs(X_f_pad))]):.3f} [Hz]")
```



```
Peak frequency:  11.978 [Hz]
```