Combined PDF Example

Your Name

September 20, 2023

1 Python Outputs

1a

September 20, 2023

0.0.1 Library Imports

[ ]:
```
import numpy as np
import plotly.graph_objects as go
```

Lets imagine we have an LED in some sort of optical detection aparatus. We need multiple color LED's to detect different materials. If we want to make simultaneous detections with the LED's we might need a way to distinguish their signals. We could do this by "blinking" them really fast. The photo-diodes that detects their signals could send all of the information through a common mixer while the MCU performs an FFT to seperate the detection channels.

Lets simulate one of these detection channels

[ ]:
```
def main():
    # Constants
    TIMESTEP = 0.01
    NUMSAMPLES = 1000
    MOD_FREQ_HZ = 40
    CHANNEL_SEPARATION_HZ = 40

    # Main script execution
    signal = generate_signal(TIMESTEP, NUMSAMPLES)
    visualize_signal_and_fft_simpsons(signal, TIMESTEP, MOD_FREQ_HZ,␣
 ↪CHANNEL_SEPARATION_HZ)
```

- $y(t) = A \cdot \sin(2\pi f t + \phi)$

  Where:

  - $y(t)$ is the value of the wave at time t.
  - A is the amplitude of the wave, determining its maximum and minimum values.
  - f is the frequency of the wave, which specifies how many cycles occur in one second (measured in Hertz, Hz).
  - ϕ is the phase angle, which determines the horizontal shift of the wave along the time axis.

- $y(t) = \sin(40.0 \cdot 2\pi t)$

  In this equation, the frequency (f) is set to 40.0 Hz.

```
[ ]: def generate_signal(timestep, numsamples):
         t = np.linspace(0, numsamples*timestep, numsamples)
         windowed_signal = np.sin(40.0 * 2.0 * np.pi * t) * np.hamming(numsamples)
         return windowed_signal

     def fft_calculate(data, timestep):
         yf = np.abs(np.fft.fft(data))
         numsamples = len(data)
         freq = np.fft.fftfreq(numsamples, d=timestep)
         xf = freq[:numsamples//2]
         yf = yf[:numsamples//2] * 2.0 / numsamples
         return xf, yf

     def find_nearest(array, value):
         idx = np.argmin(np.abs(array - value))
         nearestValue = array[idx]
         return idx, nearestValue
```

The Simpson's rule for integration is given by:

$$\int_{x_{2i}}^{x_{2i+2}} f(x)\,dx \approx \frac{h}{3}\left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right]$$

```
[ ]: def simpsons_integration(xf, yf, idx_start, idx_stop):
         # Ensure that the number of intervals is even
         n = idx_stop - idx_start
         if (idx_stop - idx_start) % 2 != 0:
             idx_stop -= 1  # or idx_start += 1, depending on your requirements

         h = (xf[idx_stop] - xf[idx_start]) / n
         result = 0

         # Loop in steps of 2 since Simpson's rule integrates over two intervals at␣
      ↪once
         for i in range(0, n - 1, 2):
             result += (h/3) * (yf[idx_start + i] + 4*yf[idx_start + i + 1] +␣
      ↪yf[idx_start + i + 2])

         return result
```

```
[ ]: def visualize_signal_and_fft_simpsons(signal, timestep, mod_freq_hz,␣
      ↪channel_separation_hz):
         xf, yf = fft_calculate(signal, timestep)

         freq_start = mod_freq_hz - channel_separation_hz / 2
         freq_stop = mod_freq_hz + channel_separation_hz / 2
```

```
    idx_start, _ = find_nearest(xf, freq_start)
    idx_stop, _ = find_nearest(xf, freq_stop)

    # Ensure even number of intervals
    if (idx_stop - idx_start) % 2 == 0:
        idx_stop += 1

    integrated_area = simpsons_integration(xf, yf, idx_start, idx_stop)

    # Time-domain Signal plot
    fig1 = go.Figure()
    fig1.add_trace(go.Scatter(y=signal, mode='lines', name='Signal'))
    fig1.update_layout(title='Time-domain Signal')
    fig1.show()

    # FFT Magnitude plot
    fig2 = go.Figure()
    fig2.add_trace(go.Scatter(x=xf, y=yf, mode='lines', name='FFT'))
    fig2.add_trace(go.Scatter(x=xf[idx_start:idx_stop+1], y=yf[idx_start:
↪idx_stop+1], fill='tozeroy'))
    fig2.update_layout(title=f'FFT Magnitude - Integrated Simpsons Area:␣
↪{integrated_area:.3f}')
    fig2.show()
```

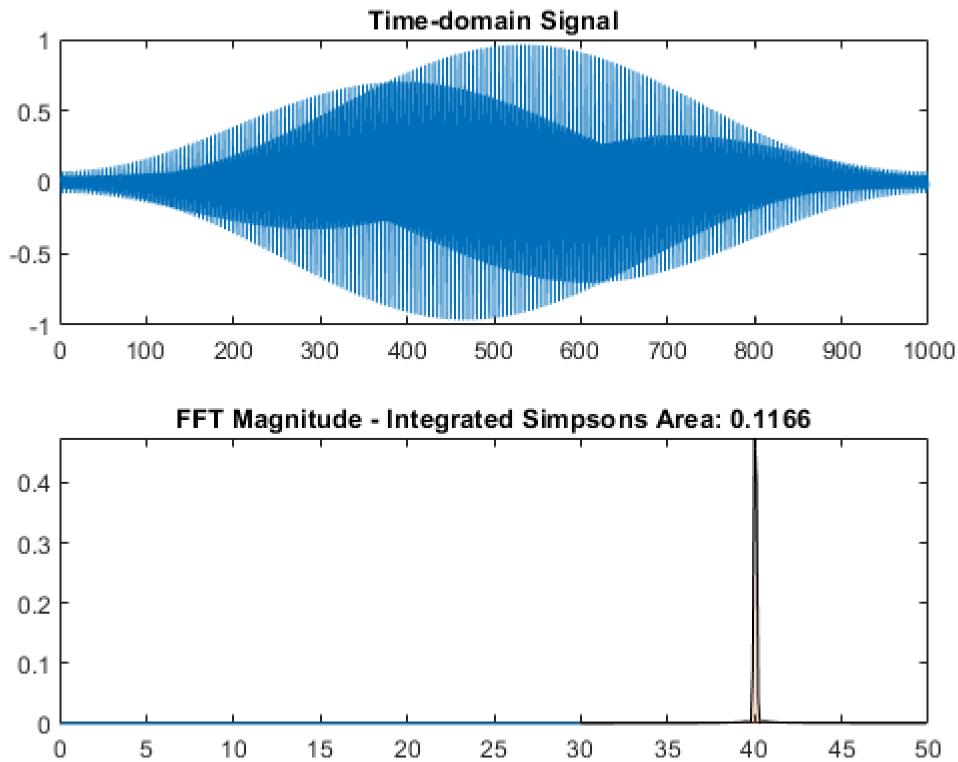
[ ]:
```
main()
```

Now I have a tight regions where to take my area under the curve for a given frequency. I could add another channel with a different modulation frequency now!

2 MATLAB Outputs

```
function main_function()

    % Constants
    TIMESTEP = 0.01;
    NUMSAMPLES = 1000;
    MOD_FREQ_HZ = 40;
    CHANNEL_SEPARATION_HZ = 40;

    % Main script execution
    signal = generate_signal(TIMESTEP, NUMSAMPLES);
    visualize_signal_and_fft_simpsons(signal, TIMESTEP, MOD_FREQ_HZ,
 CHANNEL_SEPARATION_HZ);

end

function signal = generate_signal(timestep, numsamples)
    t = linspace(0, numsamples*timestep, numsamples);
    windowed_signal = sin(40.0 * 2.0 * pi * t) .* hamming(numsamples).';
    signal = windowed_signal;
end

function [xf, yf] = fft_calculate(data, timestep)
    yf = abs(fft(data));
    numsamples = length(data);
    freq = 0:1/timestep/numsamples:1/timestep - 1/timestep/numsamples;
    xf = freq(1:numsamples/2);
    yf = yf(1:numsamples/2) * 2.0 / numsamples;
end

function [idx, nearestValue] = find_nearest(array, value)
    [~, idx] = min(abs(array - value));
    nearestValue = array(idx);
end

function result = simpsons_integration(xf, yf, idx_start, idx_stop)
    n = idx_stop - idx_start;
    if mod(n, 2) ~= 0
        error('Number of intervals should be even for composite Simpson''s
 rule.');
    end

    h = (xf(idx_stop) - xf(idx_start)) / n;
    result = 0;

    for i = 0:2:n-2
        result = result + (h/3) * (yf(idx_start + i) + 4*yf(idx_start + i + 1)
 + yf(idx_start + i + 2));
    end
end

function visualize_signal_and_fft_simpsons(signal, timestep, mod_freq_hz,
 channel_separation_hz)
```

```
    [xf, yf] = fft_calculate(signal, timestep);

    freq_start = mod_freq_hz - channel_separation_hz / 2;
    freq_stop = mod_freq_hz + channel_separation_hz / 2;

    [idx_start, ~] = find_nearest(xf, freq_start);
    [idx_stop, ~] = find_nearest(xf, freq_stop);

    % Ensure an odd number of indices (even number of intervals) for Simpson's
 rule
    if mod(idx_stop - idx_start, 2) == 1
        idx_stop = idx_stop - 1;
    end

    integrated_area = simpsons_integration(xf, yf, idx_start, idx_stop);

    % Time-domain Signal plot
    subplot(2, 1, 1);
    plot(signal);
    title('Time-domain Signal');

    % FFT Magnitude plot
    subplot(2, 1, 2);
    plot(xf, yf);
    hold on;

    % Highlight Area of Interest
    freq_start = mod_freq_hz - (channel_separation_hz * 0.25);
    freq_stop = mod_freq_hz + (channel_separation_hz * 0.25);
    [idx_start, ~] = find_nearest(xf, freq_start);
    [idx_stop, ~] = find_nearest(xf, freq_stop);

    area(xf(idx_start:idx_stop), yf(idx_start:idx_stop), 'FaceAlpha', 0.2);
    title(['FFT Magnitude - Integrated Simpsons Area: ',
 num2str(integrated_area, '%.4f')]);
    hold off;
end
```



Published with MATLAB® R2023a