Report on FFT-Based Signal Analysis Functions for Oscilloscope Waveforms

Overview

The FFT analysis functions provide a comprehensive framework for frequency-domain characterization of signals acquired from an oscilloscope. By combining precise waveform acquisition with advanced spectral processing, these functions enable extraction of fundamental frequencies, harmonic content, and key signal quality metrics such as SNR and THD.

---

1. Function: `set_scope_timebase_and_fetch`

Purpose

Configures the oscilloscope's horizontal timebase and record length to match desired sampling frequency (`fs`) and number of samples (`num_samples`), then fetches waveform data for analysis.

Key Implementation Snippet

```
def set_scope_timebase_and_fetch(self, fs=None, num_samples=None, channel=1):
    if num_samples is None:
        num_samples = int(float(self.inst.query("HORizontal:RECOrdlength?")))
    if fs is None:
        dt = float(self.inst.query("WFMPRE:XINCR?"))
        fs = 1.0 / dt

    if fs is not None and num_samples is not None:
        time_per_div = num_samples / (fs * 10)
        self.inst.write(f"HORizontal:MAIn:SCAle {time_per_div}")
        self.inst.write(f"HORizontal:RECOrdlength {num_samples}")
        time.sleep(0.5)  # Allow scope to settle

    times, volts, timebase = self.fetch_channel_waveform(channel=channel,
num_samples=num_samples)
    dt = timebase["x_increment"]
    actual_fs = 1.0 / dt
    print(f"Requested fs: {fs}, Actual scope fs: {actual_fs:.2f} Hz, dt: {dt:.2e}
s, N: {len(volts)}")
    return times, volts, timebase
```

---

2. Function: `analyze_fft_with_inputs`

Purpose

Performs windowed FFT on the acquired waveform, identifies fundamental frequency and harmonics, and computes signal quality metrics.

Key Steps and Code Snippets

a. Acquire Waveform Data

```
times, volts, timebase = self.set_scope_timebase_and_fetch(fs, num_samples,
channel)
N = len(volts)
dt = timebase["x_increment"]
fs_actual = 1.0 / dt
```

b. Apply Window Function

```
from scipy.signal import windows

window = getattr(windows, window_type)(N)
volts_win = volts * window
```

c. Compute FFT and Frequencies

```
fft_vals = np.fft.fft(volts_win)
fft_freqs = np.fft.fftfreq(N, dt)
pos_mask = fft_freqs > 0
fft_freqs_pos = fft_freqs[pos_mask]
fft_vals_pos = np.abs(fft_vals[pos_mask])
```

d. Detect Peaks (Fundamental and Harmonics)

```
from scipy.signal import find_peaks

peaks, _ = find_peaks(fft_vals_pos, height=np.max(fft_vals_pos)*0.05)
if len(peaks) == 0:
    # Handle no peaks found case
    ...
fundamental_idx = peaks[np.argmax(fft_vals_pos[peaks])]
Fc = fft_freqs_pos[fundamental_idx]
fundamental_amp = fft_vals_pos[fundamental_idx]
```

e. Extract Harmonics Amplitudes

```
harmonics_freqs = Fc * np.arange(1, 11)
harmonics_amps = []
for h_freq in harmonics_freqs:
    idx = np.argmin(np.abs(fft_freqs_pos - h_freq))
    harmonics_amps.append(fft_vals_pos[idx])
```

f. Calculate Signal Metrics

```
signal_rms = harmonics_amps[0] / np.sqrt(2)
total_rms = np.sqrt(np.mean(volts_win**2))

harmonics_bins = [np.argmin(np.abs(fft_freqs_pos - h)) for h in harmonics_freqs]
noise_bins = np.ones_like(fft_vals_pos, dtype=bool)
for b in harmonics_bins:
    noise_bins[max(0, b-1):b+2] = False
noise_rms = np.sqrt(np.mean(fft_vals_pos[noise_bins]**2))

SNR = 20 * np.log10(signal_rms / noise_rms) if noise_rms > 0 else np.nan
thd_numerator = np.sqrt(np.sum(np.array(harmonics_amps[1:])**2))
THD = 20 * np.log10(thd_numerator / harmonics_amps[0]) if harmonics_amps[0] > 0
else np.nan

try:
    DNR = 20 * np.log10(np.max(fft_vals_pos) / np.min(fft_vals_pos[fft_vals_pos >
0]))
except:
    DNR = np.nan
```

g. Return Metrics

```
metrics = {
    "Fc": Fc,
    "Fundamental Amplitude": fundamental_amp,
    "Harmonics Frequencies": harmonics_freqs,
    "Harmonics Amplitudes": harmonics_amps,
    "SNR_dB": SNR,
    "THD_dB": THD,
    "DNR_dB": DNR,
    "Signal RMS": signal_rms,
    "Total RMS": total_rms,
    "Noise RMS": noise_rms,
    "fft_freqs": fft_freqs_pos,
    "fft_magnitude": fft_vals_pos,
    "window_type": window_type,
    "N_samples": N,
    "dt": dt,
    "fs": fs_actual,
    "channel": channel,
    "timebase": timebase,
}
return metrics
```

---

3. Usage Example

The following example demonstrates how to use these functions in a typical measurement and analysis workflow:

```
import matplotlib.pyplot as plt

# Assume 'scope' is an instance of the oscilloscope control class with the FFT
functions.

def example_usage():
    desired_fs = 1e6          # 1 MHz sampling frequency
    desired_samples = 10000   # Number of samples
    channel = 1               # Channel number

    # Configure oscilloscope and fetch waveform
    times, volts, timebase = scope.set_scope_timebase_and_fetch(
        fs=desired_fs,
        num_samples=desired_samples,
        channel=channel
    )

    # Plot time-domain waveform
    plt.figure()
    plt.plot(times, volts)
    plt.title(f"Channel {channel} Waveform")
    plt.xlabel("Time (s)")
    plt.ylabel("Voltage (V)")
    plt.grid(True)
    plt.show()

    # Perform FFT analysis
    metrics = scope.analyze_fft_with_inputs(
        fs=desired_fs,
        num_samples=desired_samples,
        window_type='hann',
        channel=channel
    )

    # Print key metrics
    print(f"Fundamental Frequency: {metrics['Fc']:.2f} Hz")
    print(f"Fundamental Amplitude: {metrics['Fundamental Amplitude']:.4f} V")
    print(f"SNR (dB): {metrics['SNR_dB']:.2f}")
    print(f"THD (dB): {metrics['THD_dB']:.2f}")
    print(f"DNR (dB): {metrics['DNR_dB']:.2f}")

    # Plot FFT magnitude spectrum
    plt.figure()
    plt.plot(metrics['fft_freqs'], metrics['fft_magnitude'])
    plt.title("FFT Magnitude Spectrum")
    plt.xlabel("Frequency (Hz)")
    plt.ylabel("Magnitude")
```

```
    plt.grid(True)
    plt.show()

if __name__ == "__main__":
    example_usage()
```

---

4. Summary

- The `set_scope_timebase_and_fetch` function ensures waveform acquisition with user-defined or instrument-default sampling parameters.
- The `analyze_fft_with_inputs` function applies windowing, computes FFT, identifies harmonics, and calculates detailed signal quality metrics.
- The usage example illustrates how to integrate these functions for a complete measurement and analysis cycle, including visualization of results.
- This approach supports automated, repeatable, and insightful frequency-domain analysis directly from the oscilloscope.
