DSP Core Functions and Logic Flow

In [342]:
```
import numpy as np
from numpy import fft as fft
import matplotlib.pyplot as plt
import scipy.signal as signal
```

Remove Baseline and Check for Anomalies:

Goal is to quickly find anomalies and filter out any baseline signals in the phase angle and Marine Resistivity data components from the signals. The most reduced, compressed output format possible is best.

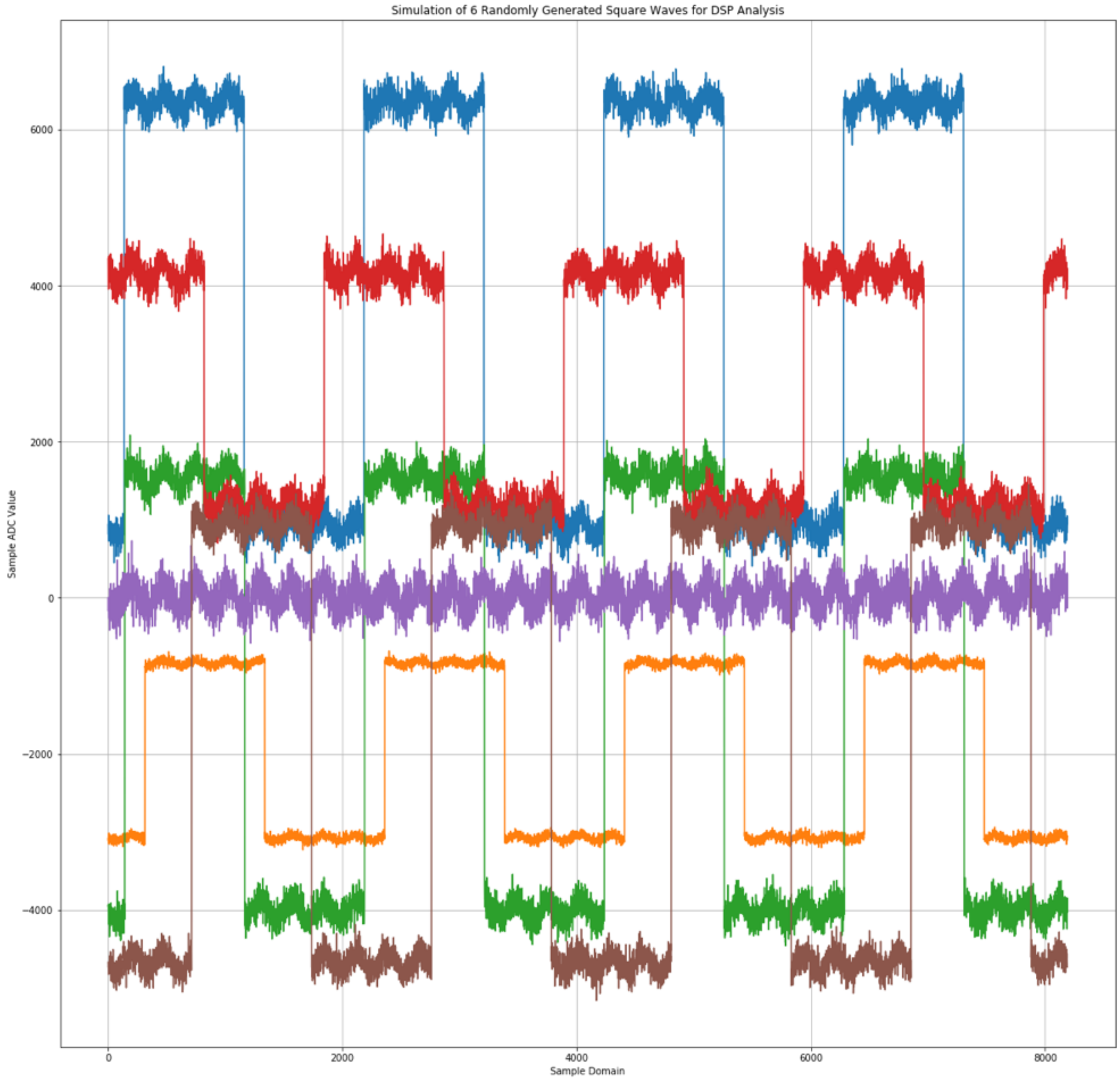
In [343]:
```
# Make simulated "baseline" signals and process them with dsp()
def simulate_squares(number_of_channels, dc_offset_max, dc_offset_min,
                               amplitude_max, amplitude_min):
    N = 8192  # number of samples
    T = 1 # Sampling period (seconds)
    F_s = 8192   # [Samples/second]
    tau = np.float(1./F_s)
    t = np.linspace(0, N-1, N, endpoint=False)
    offsets = np.random.randint(60, size=number_of_channels)
    # define the matrix of signals.

    all_channel_wavepacket = np.empty((N, number_of_channels))

    for j in range(number_of_channels):
        TIME_OFFSET = N / offsets[j]  # time offset provided for simul
ation
        AMP = np.random.randint(amplitude_min, amplitude_max)  # ampli
tude (in ADC values)
        NOISE_LVL = AMP / np.random.randint(10, 40)  # Naive way to se
t noise on the signal (not true SNR)
        DC_OFFSET = np.random.randint(dc_offset_min, dc_offset_max)  #
DC offset in ADC values
        F_tx = 4.00  # Fundamental frequency of the waveform (Hz)
        noise = NOISE_LVL * np.random.normal(0, 1, t.shape)
        pure_sig = AMP * signal.square(2 * np.pi * F_tx * t - TIME_OFF
SET) + NOISE_LVL * np.sin(np.pi* 60 *t) + DC_OFFSET
        sig = pure_sig + noise
        all_channel_wavepacket[:, j] = sig
    return t, T, tau, all_channel_wavepacket
```

In [344]:
```
# Plotting the waveforms as a sanity check:
def plot_all_waveforms(t, signals):
    domain_size = t.shape[0]
    plt.figure(figsize=(20,20))
    number_of_channels = signals.shape[-1]
    for p in range(number_of_channels):
        plt.plot(t, signals[:, p])
    plt.title(f"Simulation of {number_of_channels} Randomly Generated
Square Waves for DSP Analysis")
    plt.xlabel('Sample Domain')
    plt.ylabel('Sample ADC Value')
    plt.grid(True)
    plt.show()
```

In [345]:
```
# Simulate 6 channels of baseline noisy data with variable offsets wit
hin certain ranges.
n_channels = 6
# Random parameter ranges
base_offset_max = 4000
base_offset_min = -4000
base_amp_max = 4000
base_amp_min = 500

t_domain, period, tau, baseline_packets = simulate_squares(n_channels,
base_offset_max, base_offset_min,
                                base_amp_max, base_amp_min)
```

```
/Volumes/Storage/Users-moved/josephj.radler/opt/anaconda3/lib/python
3.7/site-packages/ipykernel_launcher.py:15: RuntimeWarning: divide b
y zero encountered in long_scalars
  from ipykernel import kernelapp as app
```

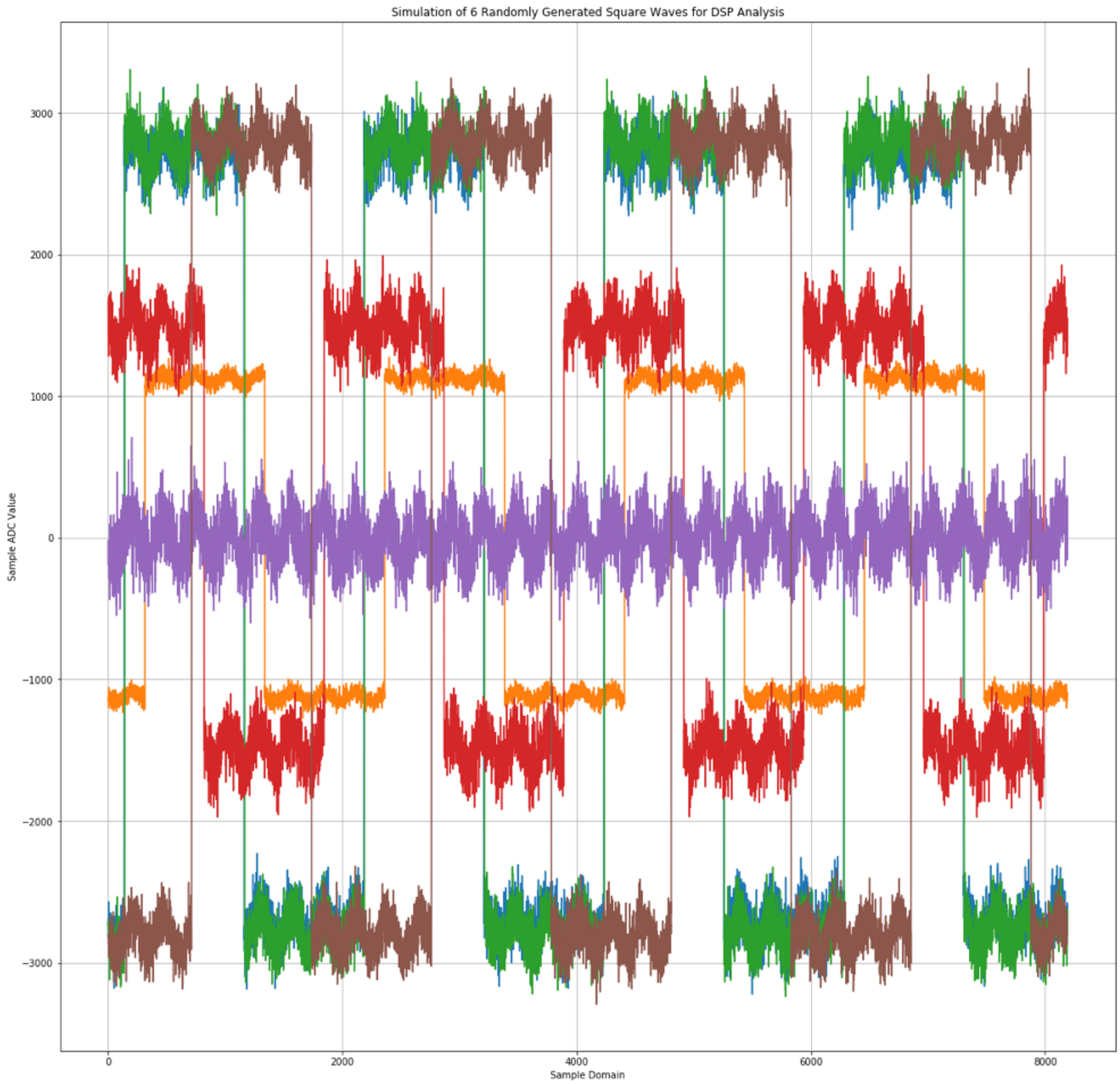
In [346]:
```
# Sanity Check plot
plot_all_waveforms(t_domain, baseline_packets)
```



In [347]:
```
def remove_dc_offset(sig):
    for q in range(sig.shape[-1]):
        offset_average = np.average(sig[:, q])
        print(f"Waveform Average Value for Channel [q] = {offset_avera
ge}")
        sig[:,q] = sig[:,q] - offset_average
    return sig
```

In [348]:
```
# Sanity  check that offset removal still works...
ac_baseline_packets = remove_dc_offset(baseline_packets)
plot_all_waveforms(t_domain, ac_baseline_packets)
```

```
Waveform Average Value for Channel [q] = 3628.2973897227994
Waveform Average Value for Channel [q] = -1954.7210424879404
Waveform Average Value for Channel [q] = -1220.6424296233813
Waveform Average Value for Channel [q] = 2671.26101313633
Waveform Average Value for Channel [q] = 22.04944557637867
Waveform Average Value for Channel [q] = -1863.804973941021
```



Filtering

In [349]:
```
def butterworth_digital_lpf(sig, order, f_cut, f_sample):
    """
    .. function:: butterworth_analog_lpf
    .. description::
    :param sig:
    :param order:
    :param f_cut:
    :return w:
    :return h:
    :return filt_sig:
    """
    # Compute the numerator and denominator polynomials of the IIR fil
ter.
    b, a = signal.butter(order, f_cut, 'lp', fs=f_sample, analog=False
)
    # Compute the frequency response of an analog filter.
    w, h = signal.freqs(b, a)
    # Define second-order sections representation of the IIR filter.
    sos = signal.butter(order, f_cut, 'lp', fs=f_sample, analog=False,
output='sos')
    # Apply the filter to our signal.
    filt_sig = signal.sosfilt(sos, sig)
    return w, h, filt_sig
```

Triggering

In [350]:
```
def rising_edge_trigger(filt_zeroed_sig, filt_sig_gradient, gradient_m
ax, N):
    """
    """
    positive_trigger_indices = (j for j in range(N - 1) if
                               ((filt_sig_gradient[j] >= (2/3) * gradient_
max) and
                                filt_zeroed_sig[j - 1] < filt_zeroed_sig[j
]))
    t0 = next(positive_trigger_indices)
    print(f"First rising trigger found at index {t0}!")
    return t0
```

In [351]:
```
def falling_edge_trigger(filt_zeroed_sig, filt_sig_gradient, gradient_
max, N):
    """
    """
    negative_trigger_indices = (j for j in range(N-1, 0, -1)  if
                               ((filt_sig_gradient[j] <= (2/3) * -gradien
t_max) and
                                filt_zeroed_sig[j + 1] < filt_zeroed_sig[
j]))
    tf = next(negative_trigger_indices)
    print(f"Last falling trigger found at index {tf}!")
    return tf
```

In [352]:
```
def edge_triggers(s_0, filt_sig_gradient, gradient_max, N):
    """
    """
    t0 = rising_edge_trigger(s_0, filt_sig_gradient, gradient_max, N)
    tf = falling_edge_trigger(s_0, filt_sig_gradient, gradient_max, N)
    t0, tf, Nw = adjust_trigger_window(t0, tf)

    return t0, tf, Nw
```

In [353]:
```
def adjust_trigger_window(t0, tf):
    """
    .. description:: Define the window (change to even number of value
s if window is odd)
    """
    Nw = np.abs(t0 - tf)
    if Nw % 2 != 0:
        tf = tf - 1
        Nw = np.abs(t0 - tf)
        print("Window altered for even number of samples...")

    print(f"(t_0, t_f) = ({t0}, {tf}) with sample size {Nw}")
    # Shift the original signal
    return t0, tf, Nw
```

In [354]:
```
def define_windowed_signal(s_0, N, forder, fcut):
    """
    .. description:: Filters the original signal, locates rising and f
alling edge triggers,
    and then windows the signal and adjusts the timing offsets automat
ically.
    :param s_0: Original unfiltered signal to apply triggers to
    :param N:
    :param forder:
    :param fcut:
    :return windowed_signal: Windowed original signal
    :return Nw:
    :return tw:
    """
    # Filter the signal
    ang_freq, resp, sig = butterworth_digital_lpf(s_0, forder, fcut, N
)
    # Compute the 1D gradient of the filtered signal
    sig_gradient = np.gradient(sig)
    gradient_max = np.nanmax(sig_gradient)
    # Set trigger indices and adjust the window
    t0, tf, Nw = edge_triggers(s_0, sig_gradient, gradient_max, N)
    # Window the signal
    windowed_sig = s_0[t0:tf]
    # Define the new time domain given the window size
    tw = np.linspace(0, Nw-1, Nw, endpoint=False)

    return Nw, tw, windowed_sig
```

In [369]:
```
def window_and_match_signals(zeroed_signal, f_order, f_cut):
    N_windowed_min = -1
    N_samples = zeroed_signal.shape[0]
    number_of_channels = zeroed_signal.shape[-1]
    print(f"Zeroed signal matrix shape is {zeroed_signal.shape}")
    print(f"Triggering and windowing signals of length {N_samples}
samples across {number_of_channels} channels...")
    triggered_signal = None

    # TODO:  There are bugs lurking in here that need to be addressed.
..They're giving ridiculous output for
    # perfectly reasonable input and I'm not sure why.
    for q in range(number_of_channels):
        N_windowed_q, t_windowed_q, windowed_signal_q = define_windowe
d_signal(zeroed_signal[:, q], N_samples, f_order, f_cut)
        if N_windowed_min == -1:
            # should only occur on the first loop through with initial
ized min value
            N_windowed_min = N_windowed_q
            print(f"Window minimum size now {N_windowed_min}")
```

```
            triggered_signal = windowed_signal_q
            print(f"triggered_signal matrix shape is now {triggered_signal.shape}")
        elif N_windowed_q == N_windowed_min:
            print("Window sizes match! Appending to output signal matrix...")
            triggered_signal = np.append(triggered_signal, windowed_signal_q, axis=0)
            print(f"Window minimum size now {N_windowed_min}")
            print(f"triggered_signal matrix shape is now {triggered_signal.shape}")
        elif N_windowed_q < N_windowed_min:
            print("Window size too small! Rolling and trimming triggered_signal matrix…")
            N_difference = np.abs(N_windowed_q - N_windowed_min)
            N_windowed_min = N_windowed_q
            print(f"Window minimum size now {N_windowed_min}")
            rolled_triggered_signal = np.roll(triggered_signal, -N_difference, axis=0)[:,:N_windowed_min]
            print(f"Triggered signal matrix shape now {triggered_signal.shape}")
            triggered_signal = np.append(rolled_triggered_signal, windowed_signal_q, axis=0)
        else:
            print(f"Window size too large! Rolling and trimming column {q}…")
            N_difference = np.abs(N_windowed_q - N_windowed_min)
            rolled_signal_q = np.roll(windowed_signal_q, -N_difference)[0:N_windowed_min]
            triggered_signal = np.append(triggered_signal, rolled_signal_q, axis=0)
            print(f"triggered_signal matrix shape is now {triggered_signal.shape}")
    triggered_signal = np.reshape(triggered_signal, (N_windowed_min, number_of_channels))
    print(f"The triggered signal matrix dimensions are {triggered_signal.shape}")

    return triggered_signal
```

In [370]:
```
def signal_conditioning(t_domain, time_series_matrix, f_order=4, f_cut
=200):
    """
    :param t_domain:
    :param time_series_matrix:
    :return win_sig_0: Trigger-windowed ORIGINAL noisy signal (no filt
ers) array with offset removed.
    """
    number_of_channels = baseline_packets.shape[-1]
    N_samples = t_domain.shape[0]
    zeroed_time_series_matrix = remove_dc_offset(time_series_matrix)
    triggered_signal = window_and_match_signals(zeroed_time_series_mat
rix, f_order, f_cut)
    return triggered_signal
```

In [371]:
```
def pairwise_phase_difference_spectrum(sa_k, sb_k):
    """
    Estimate the phase angle of each waveform and the associated shift
between them, holding
    S_a(k) as a reference wavform and taking the difference of the der
ived phase arrays.

    The phase shift corresponds to modulation of the complex part of t
he transmitted/
    received waveform, which is an indication of either inductive or c
apacitive frequency-
    dependent responses of the signal due to the electrical network be
tweewn the electrodes
    formed by the water and target.

    Calculate the phase shift from a reference spectrum (sa_k) and a s
hifted spectrum (sb_k)
    :param sa_k:
    :param sb_k:
    :return phase_shift_spectrum:
    """
    phase_a_k = np.angle(sa_k)
    phase_b_k = np.angle(sb_k)
    return phase_b_k - phase_a_k
```

In [372]:
```
def apparent_impedance_spectrum(s_k):
    """
    """
    z_k = np.absolute(s_k)
    return z_k
```

In [373]:
```
# define a DSP signal processing function that calls the other relevan
t functions.
def packet_dsp(t, s_t, tau, filter_order=4, filter_cut=200):
    triggered_signal = signal_conditioning(t, s_t, filter_order, filte
r_cut)
    number_of_channels = triggered_signal.shape[-1]
    print("Now computing FFT, Marine Resistivity, and Marine IP Respon
ses for the packet...")
    N_samples = triggered_signal.shape[0]
    k_domain = fft.rfftfreq(N_samples, tau)
    s_k = np.empty((k_domain.shape[0], number_of_channels), dtype=comp
lex)
    marine_ip_k = np.empty((k_domain.shape[0], number_of_channels), dt
ype=complex)
    marine_r_k = np.empty((k_domain.shape[0], number_of_channels), dty
pe=complex)

    for q in range(0, number_of_channels):
        # calculate frequency spectra
        s_k[:, q] = fft.rfft(triggered_signal[:, q])
        marine_ip_k[:, q]= pairwise_phase_difference_spectrum(s_k[:,0]
, s_k[:,q])
        marine_r_k[:, q] = apparent_impedance_spectrum(s_k[:,q])
    print(f"Yielded two matrices: marine_ip_k ({marine_ip_k.shape}) an
d marine_r_k ({marine_r_k.shape}))")
    return k_domain, triggered_signal, marine_ip_k, marine_r_k
```

In [374]:
```
def plot_phase_shift_and_magnitude(raw_signal, triggered_signal, k, ma
rine_ip_k, marine_r_k):

    number_of_channels = raw_signal.shape[-1]
    t_raw = np.linspace(0, raw_signal.shape[0] - 1, raw_signal.shape[0
], endpoint=False)
    t_triggered = np.linspace(0, triggered_signal.shape[0] - 1, trigge
red_signal.shape[0], endpoint=False)

    plt.figure(figsize=(15, 15))

    # Raw waveforms replotted for comparison.
    plt.subplot(221)
    for p in range(number_of_channels):
        plt.plot(t_raw, raw_signal[:, p])
    plt.title(f"Simulation of {number_of_channels} Randomly Generated
Square Waves for DSP Analysis")
    plt.xlabel('Sample Domain')
    plt.ylabel('Sample ADC Value')
    plt.grid(True)

    # Triggered, AC-only waveforms plotted for comparison.
```

```python
    plt.subplot(222)
    for p in range(number_of_channels):
        plt.plot(t_triggered, triggered_signal[:, p])
    plt.title(f"Simulation of {number_of_channels} Randomly Generated
Square Waves for DSP Analysis")
    plt.xlabel('Sample Domain')
    plt.ylabel('Sample ADC Value')
    plt.grid(True)

    # Triggered Phase Shift across all channels
    plt.subplot(223)
    for q in range(number_of_channels):
        plt.plot(fft.fftshift(k), fft.fftshift(marine_ip_k))
    plt.title('Simulated Baseline Packet $p$ Marine IP Response $\Delt
a\theta_{p}(k) by Frequency')
    plt.xlabel('Angular Frequency $\omega$ [rad/s]')
    plt.ylabel('Phase Angle Deflection ${S_a}\angle{S_b}$ [rad]')
    plt.xlim(0,200)
    plt.grid(True)

    # Triggered ||Z||^2 across all channels
    plt.subplot(224)
    for q in range(number_of_channels):
        plt.semilogy(k, marine_r_k[:,q])
    plt.title('Simulated Baseline Packet $p$ Marine Resistivity Respon
se $||Z_{p}(k)||^{2}$ by Frequency')
    plt.xlabel('Frequency $\omega$ Domain [Hz]')
    plt.ylabel('Spectral Intensity')
    plt.grid(which='both', axis='both')
    plt.xlim(0,200)
    plt.tight_layout()
    plt.show()
```

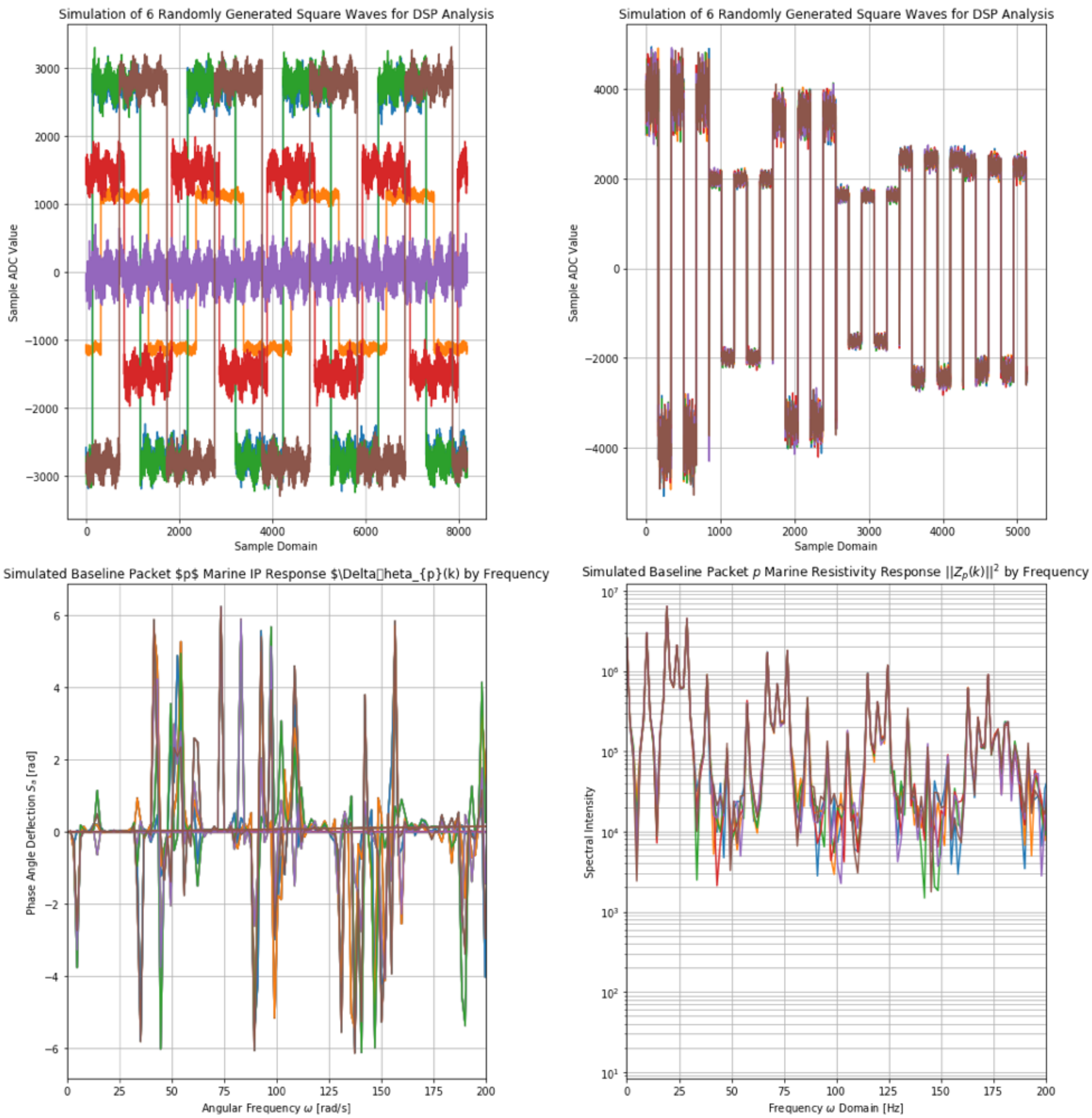
In [375]:
```python
k, triggered_signals, marine_ip_k, marine_r_k = packet_dsp(t_domain, b
aseline_packets, tau, filter_order=4, filter_cut=200)
```

```
Waveform Average Value for Channel [q] = -2.842170943040401e-14
Waveform Average Value for Channel [q] = -1.2079226507921703e-13
Waveform Average Value for Channel [q] = 0.0
Waveform Average Value for Channel [q] = -5.684341886080802e-14
Waveform Average Value for Channel [q] = -2.220446049250313e-15
Waveform Average Value for Channel [q] = -1.7053025658242404e-13
Zeroed signal matrix shape is (8192, 6)
Triggering and windowing signals of length 8192 samples across 6 cha
nnels...
First rising trigger found at index 149!
Last falling trigger found at index 7330!
Window altered for even number of samples...
(t_0, t_f) = (149, 7329) with sample size 7180
Window minimum size now 7180
triggered_signal matrix shape is now (7180,)
First rising trigger found at index 326!
Last falling trigger found at index 7508!
(t_0, t_f) = (326, 7508) with sample size 7182
Window size too large! Rolling and trimming column 1…
triggered_signal matrix shape is now (14360,)
First rising trigger found at index 154!
Last falling trigger found at index 7335!
Window altered for even number of samples...
(t_0, t_f) = (154, 7334) with sample size 7180
Window sizes match! Appending to output signal matrix...
Window minimum size now 7180
triggered_signal matrix shape is now (21540,)
First rising trigger found at index 1856!
Last falling trigger found at index 6989!
Window altered for even number of samples...
(t_0, t_f) = (1856, 6988) with sample size 5132
Window size too small! Rolling and trimming triggered_signal matrix…
Window minimum size now 5132
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-375-1c7fac9be6f1> in <module>
----> 1 k, triggered_signals, marine_ip_k, marine_r_k = packet_dsp(t_domain, baseline_packets, tau, filter_order=4, filter_cut=200)

<ipython-input-373-6a44b067c6a2> in packet_dsp(t, s_t, tau, filter_order, filter_cut)
      1 # define a DSP signal processing function that calls the other relevant functions.
      2 def packet_dsp(t, s_t, tau, filter_order=4, filter_cut=200):
----> 3     triggered_signal = signal_conditioning(t, s_t, filter_order, filter_cut)
      4     number_of_channels = triggered_signal.shape[-1]
      5     print("Now computing FFT, Marine Resistivity, and Marine IP Responses for the packet...")

<ipython-input-370-78faf8b67da8> in signal_conditioning(t_domain, time_series_matrix, f_order, f_cut)
      8     N_samples = t_domain.shape[0]
      9     zeroed_time_series_matrix = remove_dc_offset(time_series_matrix)
---> 10     triggered_signal = window_and_match_signals(zeroed_time_series_matrix, f_order, f_cut)
     11     return triggered_signal

<ipython-input-369-b1ba85ca46c4> in window_and_match_signals(zeroed_signal, f_order, f_cut)
     25             N_windowed_min = N_windowed_q
     26             print(f"Window minimum size now {N_windowed_min}")
---> 27             rolled_triggered_signal = np.roll(triggered_signal, -N_difference, axis=0)[:,:N_windowed_min]
     28             print(f"Triggered signal matrix shape now {triggered_signal.shape}")
     29             triggered_signal = np.append(rolled_triggered_signal, windowed_signal_q, axis=0)

IndexError: too many indices for array
```

In [376]:

```
plot_phase_shift_and_magnitude(baseline_packets, triggered_signals, k,
marine_ip_k, marine_r_k)
```



In [ ]:

In [ ]: