ConstellationPhaseNoise-hints

November 3, 2017

1 Noise and constellation diagrams

This assignment looks at the manifestation of an AWGN channel in a constellation diagram. We will use QPSK modulation to show the effects here.

```
In [10]: # common setup code

         # Imports for numerical library and plotting library:

         import numpy as np

         # the following line is necessary to use plotting in Jupyter notebook
         # remove if you use this code outside a notebook
         %matplotlib notebook
         import matplotlib.pyplot as plt
```

1.1 Basic variables to control visualization

```
In [11]: # base frequency of the carrier signal
         # assumption: symbol duration is unit time
         f = 5

         snrs = [0.1, 1, 10]
         samples = 50 # samples per symbol duration
         repetitions = 1000 # how many samples to use?

         # In case we want to visualize some of the signals directly:
         show_time_plots = True
```

Setup the carrier, modulation scheme, etc. Watch out for correct alignment of samples; else, FFT functions will get confused

```
In [12]: # Create a carrier per symbol period
         time, dT = np.linspace(0, 1, num=samples,
                                endpoint=False, retstep=True
                                )

         # For easier plotting of sequence of bits, let's get the
```

```
    # time samples for that as well:
    longtime, _ = np.linspace(0, repetitions, num=repetitions*samples,
                              endpoint=False, retstep=True
                              )


    # four constelation points for QPSK:
    constellation_points = np.array([
        complex(np.cos(2*np.pi/4*phase + np.pi/4),
                np.sin(2*np.pi/4*phase + np.pi/4))
        for phase in range(0,4)])

    # and for each constellation point, the corresponding carrier signal
    # for one unit time
    carriers = [
        np.cos(2*np.pi*f*time +  2*np.pi/4*phase + np.pi/4)
        for phase in range(0,4)   ]
```

1.2 Compute results for one particular SNR value

A function to make it easier to iterate over SNRs

```
In [13]: def one_run(snr,
                     repetitions=repetitions,
                     carriers=carriers, f=f):
             """
             For given SNR, generate number of repetitions many symbols,
             add noise corresponding to the given SNR (watch out: Power!),
             and for each noisy symbol, compute its constellation point at
             the given frequency f via an FFT.
             Return lists of (1) symbols, (2) received constellations points
             and (3) True/False for each symbol, whether received correctly or not.

             # Create a sequence of QPSK constellation points.
             # Effectively, that means choose phases out of 45, 135, -45, -135 degr

             ### BEGIN SOLUTION
             symbols = np.random.randint(0, len(carriers), repetitions)
             ### END SOLUTION

             # modulate the signals: create a long signal for all samples

             ### BEGIN SOLUTION
             signal = np.concatenate([carriers[s] for s in symbols])
             ### END SOLUTION


             # add noise
```

```
    ### BEGIN SOLUTION
    noise = np.random.normal(loc=0, scale=1/snr**0.5,
                             size=len(signal))
    ### END SOLUTION


    noisy_signal = signal + noise

    # demodulate; look at each symbol separate, do FFT to compute phase
    received_signals = np.split(noisy_signal, repetitions)

    ### BEGIN SOLUTION
    # FFT for each symbol to determine its constellation point
    # at our operating frequency:
    complex_symbol = lambda signal, dT, f: 2*(np.fft.rfft(signal)*dT)[f]
    # fft is parameterized and normalized properly, so we can use f as inc
    # 2* ? mirror frequency would be lost!

    received_cps = np.array([complex_symbol(rs, dT, f)
                    for rs in received_signals])
    ### END SOLUTION

    # check which symbols where correctly transmitted
    ### BEGIN SOLUTION
    correct_bits = np.array([ np.argmin(np.abs(constellation_points-d)) ==
        for s, d in zip(symbols, received_cps) ])
    ### END SOLUTION

    return symbols, received_cps, correct_bits
```

2 Visualize Signal

2.1 Plot one constellation diagram

Corresponds to one particular SNR value. To be called with a matplotlib axis!

```
In [14]: def show_one(axis, symbols, received_cps, correct_bits, snr,
                     constellation_points=constellation_points):
             axis.scatter(constellation_points.real,
                          constellation_points.imag,
                          c=range(0, 4),
                          marker="*", s=500)

             # plot the "correct" bits:
             axis.scatter(received_cps[correct_bits].real,
                          received_cps[correct_bits].imag,
                          c=symbols[correct_bits],
                          marker='o'
```

```
                )
        # and the incorect ones:
        axis.scatter(received_cps[np.logical_not(correct_bits)].real,
                     received_cps[np.logical_not(correct_bits)].imag,
                     c=symbols[np.logical_not(correct_bits)],
                     marker='x',
                     s=100,
                     )

        axis.plot((-1.5, 1.5), (0, 0), 'r--')
        axis.plot((0, 0), (-1.5, 1.5), 'r--')
        axis.set(aspect=1, adjustable="box-forced")
        axis.set_title("SNR={}".format(snr))
```

2.2 Show all constellation diagrams, for all SNRs

Also, call the actual computation function (TODO: split in two functions?)

```
In [15]: def visualize(snrs,
                       repetitions=repetitions,
                       constellation_points=constellation_points):

             fig, axes = plt.subplots(len(snrs), 1, sharex=True)
             for s, a in zip(snrs, axes):
                 symbols, received, correct = one_run(s)
                 show_one(a, symbols, received, correct, s)

             plt.show()
```

3 Run it!

(TODO: check axis scaling? again, works fine outside the notebook :-( )

```
In [16]: visualize(snrs)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```