Communications Project

Coherent and Incoherent Integration

Homework Results

Authors:

Saqib Faraz 458317
Ankita Kannan Iyer 457620
Harsh Godhani 461986
Richard Grünert 289427
Kanaiyalal Thummar 435534

12.5.2023

Homework Tasks

```
HW4)

due May 11th 23:59 LT

1)
- create a long complex valued time series signal, perhaps a 2 or 4 state QAM
- the states should be persistent for 400 samples
- add complex noise to the time series

1a)
- derive a "signal to noise ratio"
- proposal: derive the deviation for each sample to the nominal QAM state position
(e.g. 43+33j to 50+50j)
-> sum of those distances in respect to the magnitute of the I/Q state

1b)
- apply coherent integrations: 1, 5, 10, 25, 50, 100
- plot the time series for 1, 25 and 100 CI (in one plot)

1c)
- investigate the effect of the diffent number of applied CI
-> plot "SNR" over CI, any mathematical approximation visible?

2) basically repeat the equivalent for incoherent integrations
- create a gaussian-like signal in the spectral domain and add noise

2a)
- apply 1, 5, 10, 25, 50, 100 incoherent integration
-> create different realisations of such spectra
-> do not split just one time series

2b)
- derive a "signal to noise ratio"
- proposal: sum the magnitudes of the spectra, first without noise, then for the
1..100 integrations
- plot the integrated spectrum for 1, 25 and 100 NCI

2c)
- investigate and plot the dependence of "SNR" to the number of integrated spectra
(NCI)
-> any mathematical approximation visible?
```

1 Introduction

The goal of coherent and incoherent integration is to improve the signal-to-noise-ratio (SNR) of a measured signal by means of averaging. [1][3]

With the assumption that the underlying noise is white and gaussian distributed (uncorrelated) with a mean of zero, we can conclude that averaging this form of noise will move it towards its mean value of zero. For averaging, one has to define a window size (number of samples) which should match the duration that the complex symbols are stable. This defines the number of samples that will contribute to a new averaged sample and therefore determine the sample rate of the result.

The (non-)coherent integrations will act as a low pass averaging filter and reduce the number of samples and therefore the sampling rate which also affects the frequency resolution. So ideally, the signal of interest is oversampled, i.e. there are more samples than needed so that the downsampled version is still sufficient for spectral analysis and provides the benefits of the integrations.

An example application is pulsed radar, where the pulse duration is generally known so that echos can be averaged with these methods.

2 Test Signals

2.1 Coherent Integration

Using python and some inspiration from [4], a QAM16 constellation was generated (`QAM_constellation`). Each constellation point was copied 50 times to simulate sampled data symbols which were then superimposed by normally distributed complex-valued noise. The noisy constellation diagram can be seen in figure 1. The code can be found in appendix A.1.

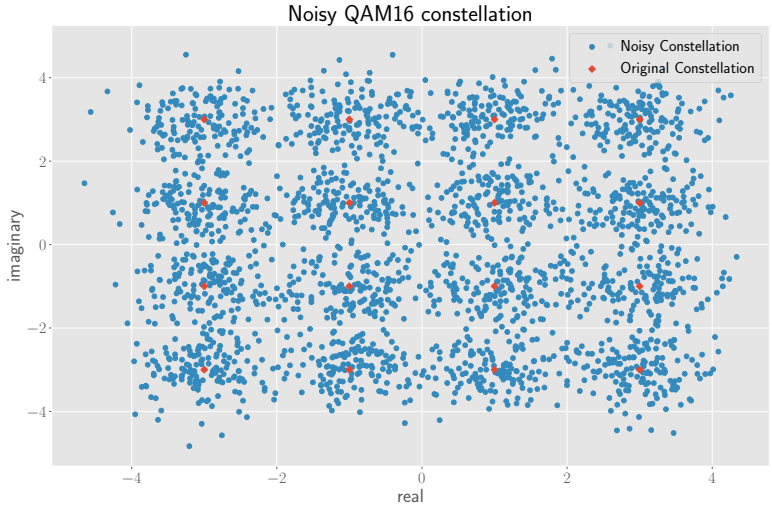

Figure 1: Noisy constellation used as test data.

2.2 Incoherent Integration

For the incoherent integration, a sinusoidal signal with a frequency of 100 Hz was multiplied with a gaussian with standard deviation of 0.01 s to create a gaussian-shape spectrum centered around 100 Hz.

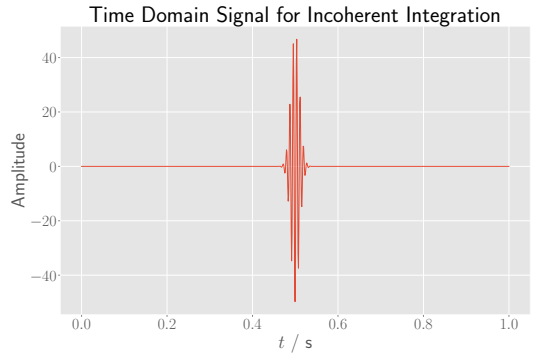

Figure 2: Time domain of NCI test signal (1 realization).

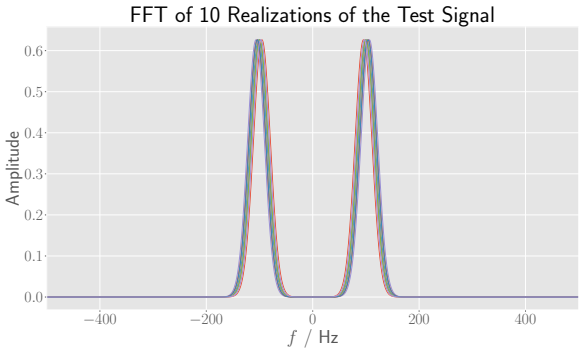

Figure 3: FFTs of NCI test signals.

The center frequency of each realization was additionally superimposed by random jitter. The signal can be seen in figures 2. Figure 3 shows the realization spectra.

2.3 Signal-To-Noise-Ratio

For the coherent integration, the mean of the squared differences of each noisy sample towards their actual constellation points (the signal without the added noise) was calculated to obtain a measure of noise variance (power, σ^2). The square of the symbol distance ($1\,\mathrm{V}^2$) is then divided by this variance to get the SNR.

For incoherent integration, the definition of an SNR is more complicated. [2] Here, the SNR was approximated by taking the peak value of the averaged spectrum and dividing it by the noise power (standard deviation squared) which was taken from the first tenth of the spectrum.

3 Coherent Integration

With coherent integration, both the real and imaginary parts of a received signal are taken into account (averaged), thereby retaining the phase information. [1][2]

The integration period (window / sample size) should not be greater than the time that the signal is stable. Otherwise, samples from a different state (amplitude, phase) will be included in the averaging process, which results in a detrimental effect. This can be seen for the $CI = 100$ case in both the constellation diagram (figure 4 and the plot of the real part (figure 5). Since there are 50 samples per symbol in this case, $CI = 100$ leads to two independent symbols being averaged together.

As for the SNR, the noise power rises with N but the signal power rises with N^2 resulting in an SNR increase which is proportional to the number of integrations. [2] Figure 7 confirms this linear relationship. In figure 7, the value at $CI = 100$ is misleading, though. The difference used for the SNR calculation might be high but the actual samples are erroneous which further confirms the fact about the length of the integration period.

4 Incoherent Integration

For incoherent integration, multiple realizations of the noisy data were translated into the frequency domain. An average of these spectra yields a spectrum (magnitude) which discards the phase information. Like in the coherent case, there is a tradeoff between the SNR gain and the reduced frequency resolution (as one could also take the FFT of the combined realizations). As can be seen in figure 6, the noise level

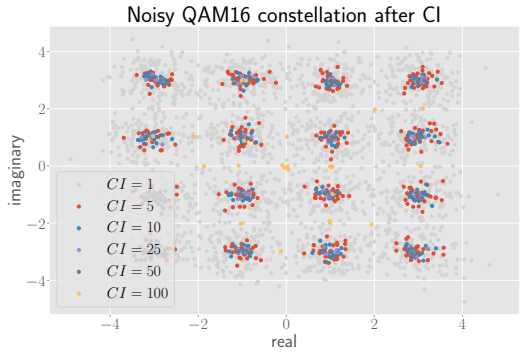

Figure 4: Constellation after coherent integration

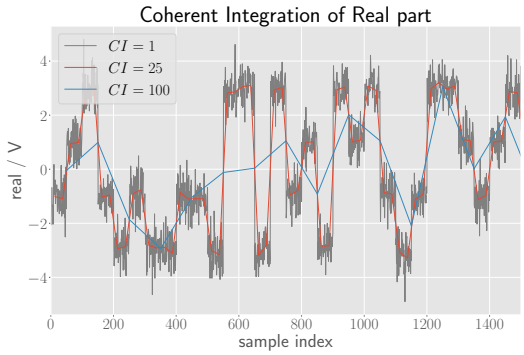

Figure 5: Real part after coherent integration.

is reduced with an increasing number of integrations.

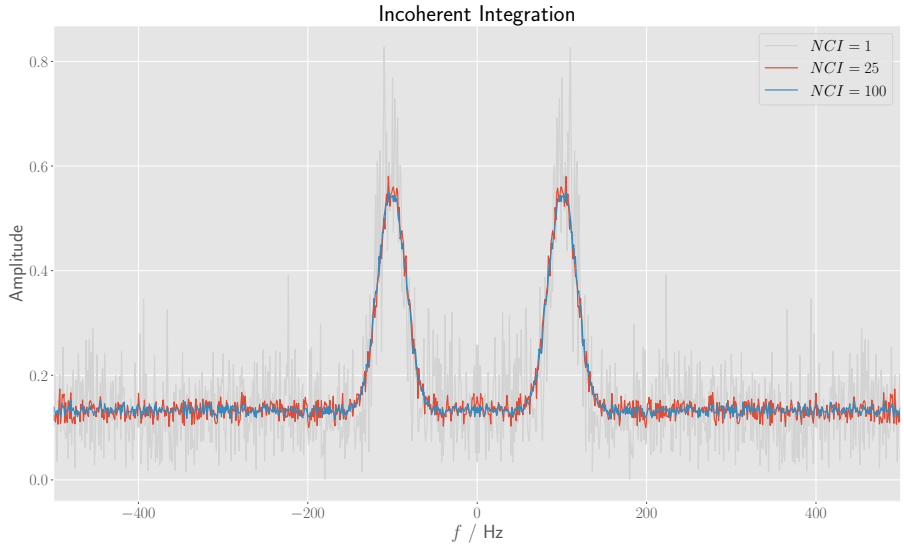

Figure 6: Resulting spectra after different numbers of incoherent integrations.

The result for the SNR can be seen in figure 8. Just like in the case of coherent integration, it rises linearly with an increased number of integrations which is as expected [3].

5 Conclusions

Coherent and incoherent integrations have been shortly investigated. Coherent integration acts on the complex valued signals and provides a linear SNR gain with increasing numbers of integrations which might be useful in situations where low signal powers are measured or noise is great. The incoherent integration method is simpler and also scales the SNR linearly with the number of integrations. It may be used in cases when the phase information is not needed.

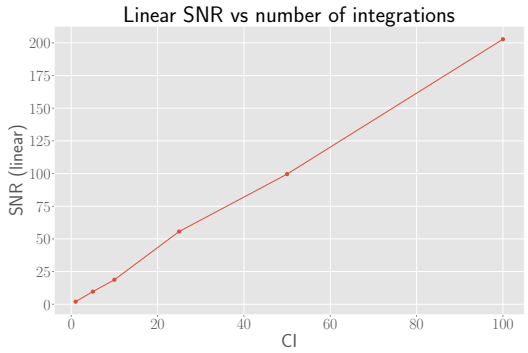

Figure 7: SNR estimate vs number of CIs.

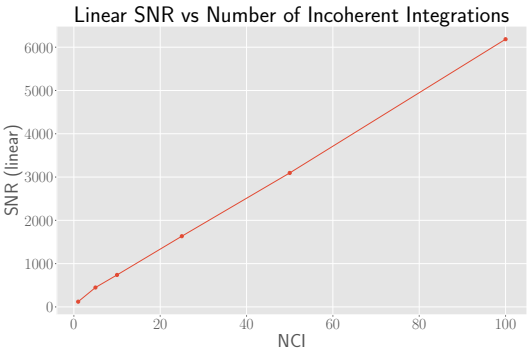

Figure 8: SNR estimate vs number of NCIs.

References

Literature

[1] David Hysell. *Antennas and Radar for Environmental Scientists and Engineers*. ISBN 9781108164122. Cambridge University Press, Feb. 2018. URL: https://doi.org/10.1017/9781108164122.

[2] Mark A. Richards. *Noncoherent Integration Gain, and its Approximation*. Last Access: 05/2023. June 2010. URL: https://cpb-us-w2.wpmucdn.com/sites.gatech.edu/dist/5/462/files/2016/12/Noncoherent-Integration-Gain-Approximations.pdf.

[3] Standford University. *5.9 - I,Q channels and non coherent integration*. https://www.youtube.com/watch?v=eJa7Vb6QfXE. Last Access: 05/2023. Oct. 2014.

[4] B. Trotobas V. Taranalli and contributors. *CommPy: Digital Communication with Python*. github.com/veeresht/CommPy.

Software Used

[1] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[2] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[3] Python Core Team. *Python: A dynamic, open source programming language*. Python version 3.7. Python Software Foundation. 2019. URL: https://www.python.org/.

[4] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

A Python Code

A.1 Constellation

```python
# constellation.py

import numpy as np
from matplotlib import pyplot as plt


def QAM_constellation(bits_per_symb):
    # only for even bits_per_symb
    pam = np.arange(-bits_per_symb + 1, bits_per_symb, 2)
    real = pam.repeat(bits_per_symb)
    imag = np.tile(np.hstack((pam, pam[::-1])), int(bits_per_symb) // 2)
    return real + 1j * imag


def complex_gaussian_noise(size, std=1):
    real = np.random.normal(size=size)
    imag = np.random.normal(size=size)
    return std * (real + 1j*imag)


def my_complex_signal(bits_per_symbol, samples_per_symbol, reps=1):
    constellation = QAM_constellation(bits_per_symbol).repeat(reps)
    np.random.shuffle(constellation)
    signal = constellation.repeat(samples_per_symbol)
    return signal


def main():
    bits_per_symbol = 4 # should be even
    sig = my_complex_signal(bits_per_symbol, 50, 3)
    noise = complex_gaussian_noise(sig.size, 0.5)

    data = sig + noise

    constellation = QAM_constellation(bits_per_symbol)

    plt.figure()
    plt.scatter(data.real, data.imag,
                label="Noisy Constellation",
                color="C1")
    plt.scatter(constellation.real, constellation.imag,
                label="Original Constellation",
                color="C0", marker="D")
    plt.xlabel("real")
    plt.ylabel("imaginary")
    plt.title("Noisy QAM{} constellation".format(str(bits_per_symbol**2)))
    plt.legend()
    plt.savefig("outputs/constellation.pdf")
    plt.show()


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

A.2 Coherent Integration

```python
# coherent_integration.py

import numpy as np
from matplotlib import pyplot as plt


def QAM_constellation(bits_per_symb):
    # only for even bits_per_symb
    pam = np.arange(-bits_per_symb + 1, bits_per_symb, 2)
    real = pam.repeat(bits_per_symb)
    imag = np.tile(np.hstack((pam, pam[::-1])), int(bits_per_symb) // 2)
    constellation = real + 1j * imag
    return constellation


def complex_gaussian_noise(size, std=1):
    real = np.random.normal(size=size)
    imag = np.random.normal(size=size)
    return std * (real + 1j*imag)


def my_complex_signal(bits_per_symbol, samples_per_symbol, reps=1):
    constellation = QAM_constellation(bits_per_symbol).repeat(reps)
    np.random.shuffle(constellation)
    signal = constellation.repeat(samples_per_symbol)
    return signal


def coherent_integration(time, data, ci):
    # split the data into chunks of data.size / ci
    split_data = np.array_split(data, int(np.floor(data.size / ci)))
    split_time = np.array_split(time, int(np.floor(time.size / ci)))

    # calc mean of all chunks
    data_mean = np.mean(split_data, axis=1)
    time_mean = np.mean(split_time, axis=1)

    return time_mean, data_mean


def estimate_snr_time_domain(data, actual):
    diffs = np.abs(data- actual)
    return 1 / np.mean(diffs ** 2)


def main():
    bits_per_symbol = 4 # should be even
    sig = my_complex_signal(bits_per_symbol, 50, 3)
    noise = complex_gaussian_noise(sig.size, 0.5)

    data = sig + noise

    x = np.arange(1, data.size + 1)

    ci_values = [1, 5, 10, 25, 50, 100]
    snrs_ci = []

    const_fig, const_ax = plt.subplots(1,1)
    const_ax.set_title("Noisy QAM{} constellation after CI"
                       .format(str(bits_per_symbol**2)))
    const_ax.set_xlabel("real")
    const_ax.set_ylabel("imaginary")

    time_fig, time_axs = plt.subplots(1, 1)
```

```
    time_axs.set_xlim([0, 1500])
    time_axs.set_title("Coherent Integration of Real part")
    time_axs.set_xlabel("sample index")
    time_axs.set_ylabel("real / V")

    snr_fig, snr_ax = plt.subplots(1, 1)
    snr_ax.set_title("Linear SNR vs number of integrations")
    snr_ax.set_xlabel("CI")
    snr_ax.set_ylabel("SNR (linear)")


    for ci in ci_values:

        coh_time, coh_data = coherent_integration(x, data, ci)

        # calculate CI of signal without noise
        # just to downsample it for snr calculation
        coh_time_sig, coh_data_sig = coherent_integration(x, sig, ci)

        snr = estimate_snr_time_domain(coh_data, coh_data_sig)
        snrs_ci.append(snr)

        # constellation plot
        if ci == 1:
            const_ax.scatter(np.real(coh_data), np.imag(coh_data),
                             label="$CI={}$"
                             .format(str(ci)), color="lightgray")
        else:
            const_ax.scatter(np.real(coh_data), np.imag(coh_data),
                             label="$CI={}$"
                             .format(str(ci)))

        # time plot
        if ci in [1, 25, 100]:
            if ci == 1:
                time_axs.plot(coh_time, np.real(coh_data),
                              label="$CI={}$"
                              .format(str(ci)), color="gray")
            else:
                time_axs.plot(coh_time, np.real(coh_data),
                              label="$CI={}$"
                              .format(str(ci)))

    # SNR plot
    snr_ax.plot(ci_values, snrs_ci, 'o-')

    const_ax.legend()
    time_axs.legend()

    const_fig.savefig("outputs/ci_constellation.pdf")
    time_fig.savefig("outputs/ci_real.pdf")
    snr_fig.savefig("outputs/ci_snr.pdf")

    plt.tight_layout()
    plt.show()


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

A.3 Incoherent Integration

```python
# incoherent_integration.py

import scipy
import numpy as np
from numpy.fft import fft, fftshift
from matplotlib import pyplot as plt


def get_spectrum(signal, ts, ax=-1):
    # assuming signal is along columns
    if ax == -1:
        N = signal.size
    else:
        N = signal.shape[ax]
    spectrum = fft(signal, axis=ax) / N
    freq = np.fft.fftfreq(N, ts)
    freq = fftshift(freq)
    spectrum = fftshift(spectrum)
    return (freq, spectrum)


def my_signal(t, f0, fstd):
    # TODO: shape check of t
    jitter = np.random.normal(size=t.shape[0]) * fstd
    f = f0 + np.repeat(jitter[:, np.newaxis], t.shape[1], axis=1)
    Ts = t[1][1] - t[1][0]
    return 50 * np.sin(2 * np.pi * np.multiply(t, f)) * \
        scipy.signal.windows.gaussian(t.shape[1], 0.01/(Ts))


def incoherent_integration(spectra, nci):
    specs_to_avg = np.abs(spectra[0:nci])
    avg_spec = np.mean(specs_to_avg, axis=0)
    return avg_spec


def cheap_snr(spec):
    mmax = np.max(np.abs(spec))
    nmax = np.std(np.abs(spec[0:1000]))
    return (mmax / nmax) ** 2


def main():

    N = 10000  # number of samples of each realization
    R = 100  # total number of realizations
    fc = 100  # Hz
    t = np.linspace(0, 1, N)
    Ts = t[1] - t[0]

    # generate the realizations
    t = np.broadcast_to(t, (R, t.size))
    sig = my_signal(t, fc, 10)
    noise = np.random.normal(scale=15, size=(R, t.shape[1]))
    data = sig + noise

    # apply the incoherent integrations
    NCI_values = [1, 5, 10, 25, 50, 100]

    # array with all realization spectra
    freq, spectra = get_spectrum(data, Ts, ax=1)

    plt.figure()
    snr_values = []
```

```
    for nci in NCI_values:
        spec = incoherent_integration(spectra, nci)
        snr = cheap_snr(spec)
        snr_values.append(snr)
        if nci in [1, 25, 100]:
            if nci == 1:
                plt.plot(freq, np.abs(spec), label="$NCI={}$".format(nci),
                         color="lightgray")
            else:
                plt.plot(freq, np.abs(spec), label="$NCI={}$".format(nci))

    # spectrum plot
    plt.title("Incoherent Integration")
    plt.xlabel("$f$ / Hz")
    plt.ylabel("Amplitude")
    plt.xlim([-500, 500])
    plt.legend()
    plt.tight_layout()
    plt.savefig("outputs/nci_spec.pdf")

    # snr plot
    plt.figure()
    plt.plot(NCI_values, snr_values, 'o-')
    plt.title("SNR vs Number of Incoherent Integrations")
    plt.xlabel("NCI")
    plt.ylabel("SNR")
    plt.tight_layout()
    plt.savefig("outputs/nci_snr.pdf")

    plt.show()


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