Name: Abijith J. Kamath
Student Id: 17788

E1 244: Detection and Estimation

February-May 2021

Solution – Homework 3

Analysis and Algorithms for Spectrum Sensing in Cognitive Radio

Part A: Derivation and Modelling

Consider the OFDM transmission of the sequence s of length N_d, where the N_d-point IFFT of the sequence along with an N_c length cyclic prefix is transmitted over an AWGN channel. The transmitted sequence is defined using:

$$x[n] = \frac{1}{\sqrt{N_d}} \sum_{k=0}^{N_d-1} s[k] e^{\mathrm{j}2\pi nk/N_d},\ n = 0, 1, \cdots, N_d - 1, \tag{1}$$

The transmitted vector has entries defined by $x[n]$ with the last N_c points are prefixed to itself to form an $N_d + N_c$ length transmission vector $\mathbf{x}_i = [x[0]\ x[1]\ \cdots\ x[N_d + N_c - 1]]^\mathsf{T}$. This forms one OFDM symbol block. $K + 1$ such OFDM symbol blocks $\mathbf{x} = [\mathbf{x}_0^\mathsf{T}\ \mathbf{x}_1^\mathsf{T}\ \cdots\ \mathbf{x}_K^\mathsf{T}]^\mathsf{T}$ are transmitted over an AWGN channel to give measurements $\mathbf{y} = \mathbf{x} + \mathbf{w}$, where $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \sigma_w^2\mathbf{I})$.

Energy Detector

Consider the data symbols $\mathbf{s} = [s[0]\ s[1]\ \cdots\ s[N_d - 1]]^\mathsf{T}$ where the entries are QPSK with variance $\sigma_s^2 = 1$, i.e., $s[k] \in \{\pm 1/\sqrt{2} \pm \mathrm{j}1/\sqrt{2}\}$. Suppose the number of data points N_d is large, using the central limit theorem, the OFDM symbol blocks $\mathbf{x}_i$ can be assumed to be zero-mean Gaussians with identity covariance of size $(K + 1)(N_d + N_c)$ as $\mathbb{E}[\mathbf{x}_i\mathbf{x}_i^\mathsf{H}] = \sigma_s^2\mathbf{I}$.

Given measurements $\mathbf{y} \in \mathbb{C}^{(K+1)(N_d+N_c)}$, the signal detection problem is to select between one of the two following hypothesis:

$$\begin{aligned} \mathcal{H}_0 &: \mathbf{y} = \mathbf{w}, \\ \mathcal{H}_1 &: \mathbf{y} = \mathbf{x} + \mathbf{w}. \end{aligned} \tag{2}$$

The Neymann-Pearson detector uses the likelihood ratio test (LRT). Let the measurements have the density function $p_Y(\mathbf{y}; \mathcal{H}_0)$ and $p_Y(\mathbf{y}; \mathcal{H}_1)$ under the hypothesis $\mathcal{H}_0$ and $\mathcal{H}_1$ respectively. The LRT compares the ratio of the likelihoods to a threshold, and decides on the hypothesis $\mathcal{H}_1$ if:

$$\begin{aligned} L(\mathbf{y}) &= \frac{p_Y(\mathbf{y}; \mathcal{H}_1)}{p_Y(\mathbf{y}; \mathcal{H}_0)} > \gamma, \\ &= \frac{\frac{1}{(2\pi(\sigma_s^2+\sigma_w^2))^{N/2}} \exp\left(-\frac{1}{2(\sigma_s^2+\sigma_w^2)} \sum_{n=0}^{N} |y[n]|^2\right)}{\frac{1}{(2\pi\sigma_w^2)^{N/2}} \exp\left(-\frac{1}{2\sigma_w^2} \sum_{n=0}^{N} |y[n]|^2\right)}, \\ &= \left(\frac{\sigma_w^2}{\sigma_s^2 + \sigma_w^2}\right)^{N/2} \exp\left(\frac{1}{2} \frac{\sigma_s^2}{\sigma_w^2(\sigma_s^2 + \sigma_w^2)} \sum_{n=0}^{N-1} |y[n]|^2\right), \\ \implies T(\mathbf{y}) &= \sum_{n=0}^{N-1} |y[n]|^2 > \gamma'. \end{aligned} \tag{3}$$

where $N = N_d + N_c$ and γ' is a threshold that is set by P_{FA}. The test statistic $T(\mathbf{y})$ is the square of sum of Gaussian random variables and hence, has the chi-squared distribution. The hypotheses with this test statistic are:

$$
\begin{aligned}
\mathcal{H}_0 &: \frac{T(\mathbf{y})}{\sigma_w^2} \sim \chi^2 \\
\mathcal{H}_1 &: \frac{T(\mathbf{y})}{\sigma_w^2 + \sigma_s^2} \sim \chi^2.
\end{aligned} \tag{4}
$$

The probability of false alarm, $P_{FA} = p_Y(T(\mathbf{y}) > \gamma'; \mathcal{H}_0) = Q(\frac{\gamma'}{\sigma_w^2})$, where $Q(\cdot)$ is the CDF of the chi-squared distribution with N degrees of freedom. Hence, given a choice for P_{FA}, the threshold can be computed as $\gamma' = \sigma_w^2 Q^{-1}(P_{FA})$. Using this, the probability of detection:

$$
\begin{aligned}
P_D &= p_Y(T(\mathbf{y}) > \gamma'; \mathcal{H}_1), \\
&= Q\left(\frac{\gamma'}{\sigma_w^2 + \sigma_s^2}\right).
\end{aligned} \tag{5}
$$

Cyclostationarity Detector

Under the same transmission settings, consider test function for LRT defined as:

$$
\begin{aligned}
T(\mathbf{y}) &= \sum_{n=0}^{N_c-1} \hat{R}[n], \\
&= \sum_{n=0}^{N_c-1} \frac{1}{K} \sum_{k=0}^{K-1} \hat{r}[n + k(N_c + N_d), N_d], \\
&= \frac{1}{K} \sum_{n=0}^{N_c-1} \sum_{k=0}^{K-1} y[n + k(N_c + N_d)] y^*[n + k(N_c + N_d) + N_d].
\end{aligned} \tag{6}
$$

Suppose the number of data samples are high, using the central limit theorem, the test statistic $T(\mathbf{y})$ can be taken to be a complex Gaussian random variable.

a) Distribution of the test statistic: Let $T(\mathbf{y}) = \bar{T}(\mathbf{y}) + \mathrm{j}\tilde{T}(\mathbf{y})$ be defined as in (6) be approximated as a complex Gaussian random variable.

Under $\mathcal{H}_0$, $\mathbf{y} = \mathbf{w}$, i.e., $y[m] = w[m]$, $\forall m$. Therefore, $\mathbb{E}\left[y[n + kN] y^*[n + kN + N_d]\right] = \mathbb{E}\left[w[n + kN] w^*[n + kN + N_d]\right] = 0$.

$$
\begin{aligned}
\mathbb{E}[T(\mathbf{y})] &= \frac{1}{K} \sum_{n=0}^{N_c-1} \sum_{k=0}^{K-1} \mathbb{E}\left[y[n + kN] y^*[n + kN + N_d]\right], \\
&= 0, \\
\implies \mathbb{E}[\bar{T}(\mathbf{y})] &= \mathbb{E}[\tilde{T}(\mathbf{y})] = 0.
\end{aligned} \tag{7}
$$

To find the variance, consider the equations for $T^2(\mathbf{y}) = \bar{T}^2(\mathbf{y}) + \mathrm{j}2\bar{T}(\mathbf{y})\tilde{T}(\mathbf{y}) - \tilde{T}^2(\mathbf{y})$ and $|T(\mathbf{y})|^2 = \bar{T}^2(\mathbf{y}) + \tilde{T}^2(\mathbf{y})$.

$$
\begin{aligned}
T^2(\mathbf{y}) = \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} & y[n_1 + k_1 N] y^*[n_1 + k_1 N + N_d] \\
& y[n_2 + k_2 N] y^*[n_2 + k_2 N + N_d].
\end{aligned} \tag{8}
$$

We have, $\mathbb{E}\left[y[n_1+k_1N]y^*[n_1+k_1N+N_d]y[n_2+k_2N]y^*[n_2+k_2N+N_d]\right]$ $=\mathbb{E}\left[w[n_1+k_1N]w^*[n_1+k_1N+N_d]w[n_2+k_2N]w^*[n_2+k_2N+N_d]\right]=\sigma_w^4\delta(n_1-n_2,k_1-k_2)$, and hence:

$$\mathbb{E}\left[T^2(\mathbf{y})\right]=\frac{1}{K^2}N_cK\sigma_w^4=\frac{1}{K}N_c\sigma_w^4. \tag{9}$$

Since the quantity is real, $\mathbb{E}\left[\bar{T}(\mathbf{y})\tilde{T}(\mathbf{y})\right]=\operatorname{cov}\left(\bar{T}(\mathbf{y}),\tilde{T}(\mathbf{y})\right)=0$. Since the measurements are noise-only and real, $\mathbb{E}\left[|T(\mathbf{y})|^2\right]=\mathbb{E}\left[T^2(\mathbf{y})\right]=\frac{1}{K}N_c\sigma_w^4$. Using this and (9):

$$\operatorname{var}\left(\bar{T}(\mathbf{y})\right)=\operatorname{var}\left(\tilde{T}(\mathbf{y})\right)=\frac{1}{2K}N_c\sigma_w^4. \tag{10}$$

Under $\mathcal{H}_1$, $\mathbf{y}=\mathbf{x}+\mathbf{w}$, i.e., $y[m]=x[m]+w[m],\ \forall m$. Therefore, $\mathbb{E}\left[y[n+kN]y^*[n+kN+N_d]\right]=\mathbb{E}\left[\left(x[n+kN]+w[n+kN]\right)\left(x[n+kN+N_d]+w[n+kN+N_d]\right)^*\right]=\sigma_s^2=1$.

$$\begin{aligned}
\mathbb{E}[T(\mathbf{y})]&=\frac{1}{K}\sum_{n=0}^{N_c-1}\sum_{k=0}^{K-1}\mathbb{E}\left[y[n+kN]y^*[n+kN+N_d]\right],\\
&=\frac{1}{K}N_cK=N_c,\\
\implies \mathbb{E}[\bar{T}(\mathbf{y})]&=N_c,\\
\mathbb{E}[\tilde{T}(\mathbf{y})]&=0,
\end{aligned} \tag{11}$$

since $\mathbb{E}[T(\mathbf{y})]$ is real. To find the variance, similar to the case in $\mathcal{H}_0$, using (13), we have:

$$\begin{aligned}
&\mathbb{E}\left[y[n_1+k_1N]y^*[n_1+k_1N+N_d]y[n_2+k_2N]y^*[n_2+k_2N+N_d]\right]=\\
&\mathbb{E}[\left(x[n_1+k_1N]+w[n_1+k_1N]\right)\left(x[n_1+k_1N+N_d]+w[n_1+k_1N+N_d]\right)^*\\
&\left(x[n_2+k_2N]+w[n_2+k_2N]\right)\left(x[n_2+k_2N+N_d]+w[n_2+k_2N+N_d]\right)^*]=\begin{cases}\sigma_s^4, & n_1\neq n_2,k_1\neq k_2,\\ \sigma_s^4+\sigma_s^4, & n_1=n_2,k_1=k_2,\end{cases}
\end{aligned}$$

and hence:

$$\mathbb{E}\left[T^2(\mathbf{y})\right]=\frac{1}{K^2}\left(2N_cK+N_c^2K^2-N_cK\right)=N_c^2+\frac{N_c}{K}. \tag{12}$$

Since the quantity is real, $\mathbb{E}\left[\bar{T}(\mathbf{y})\tilde{T}(\mathbf{y})\right]=\operatorname{cov}\left(\bar{T}(\mathbf{y}),\tilde{T}(\mathbf{y})\right)=0$. In this case, the measurements are not all real. Hence:

$$\begin{aligned}
|T(\mathbf{y})|^2=\frac{1}{K^2}\sum_{n_1=0}^{N_c-1}\sum_{k_1=0}^{K-1}\sum_{n_2=0}^{N_c-1}\sum_{k_2=0}^{K-1}&y[n_1+k_1N]y^*[n_1+k_1N+N_d]\\
&y^*[n_2+k_2N]y[n_2+k_2N+N_d],
\end{aligned} \tag{13}$$

with the mean of each term in the sum:

$$\begin{aligned}
&\mathbb{E}\left[y[n_1+k_1N]y^*[n_1+k_1N+N_d]y^*[n_2+k_2N]y[n_2+k_2N+N_d]\right]=\\
&\mathbb{E}[\left(x[n_1+k_1N]+w[n_1+k_1N]\right)\left(x[n_1+k_1N+N_d]+w[n_1+k_1N+N_d]\right)^*\\
&\left(x[n_2+k_2N]+w[n_2+k_2N]\right)^*\left(x[n_2+k_2N+N_d]+w[n_2+k_2N+N_d]\right)]\\
&=\begin{cases}\sigma_s^4, & n_1\neq n_2,k_1\neq k_2,\\ 2\sigma_s^4+2\sigma_s^2\sigma_w^2+\sigma_w^4, & n_1=n_2,k_1=k_2,\end{cases}
\end{aligned}$$

and hence,

$$\mathbb{E}\left[|T(\mathbf{y})|^2\right]=N_c^2+\frac{N_c}{K}\left(1+2\sigma_w^2+\sigma_w^4\right). \tag{14}$$

Using (12) and (14):

$$
\begin{aligned}
\operatorname{var}\left(\bar{T}(\mathbf{y})\right) &= N_c^2 + \frac{N_c}{K}\left(1 + \sigma_w^2 + \frac{\sigma_w^4}{2}\right), \\
\operatorname{var}\left(\tilde{T}(\mathbf{y})\right) &= \frac{N_c}{K}\left(\sigma_w^2 + \frac{\sigma_w^4}{2}\right).
\end{aligned} \tag{15}
$$

b) Neymann-Pearson detector: Using the test statistic defined in (6), the detector chooses hypotheses $\mathcal{H}_1$ if $|T(\mathbf{y})| > \gamma$ or equivalently, $|T(\mathbf{y})|^2 > \gamma^2$. The probability of false alarm:

$$
\begin{aligned}
P_{FA} &= p_Y(|T(\mathbf{y})|^2 > \gamma^2); \mathcal{H}_0), \\
&= p_Y(\bar{T}^2(\mathbf{y}) + \tilde{T}^2(\mathbf{y}) > \gamma^2); \mathcal{H}_0), \\
&= Q\left(\frac{\gamma^2}{\frac{N_c}{2K}\sigma_w^4}\right),
\end{aligned} \tag{16}
$$

where $Q(\cdot)$ is the CDF of the chi-squared distribution with 2 degrees of freedom. Therefore, given a choice for the probability of false alarm, the threshold is chosen as:

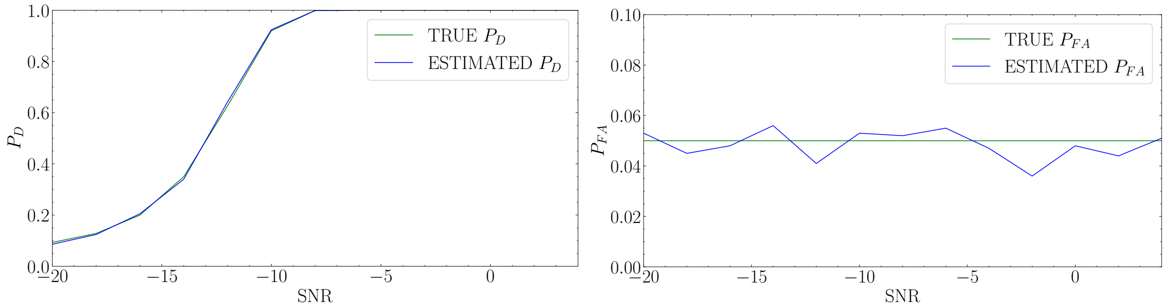
$$
\gamma = \left(\frac{N_c}{2K}\sigma_w^4 Q^{-1}(P_{FA})\right)^{1/2}. \tag{17}
$$

Part B: Implementation

Energy Detector

a) Monte-Carlo simulations with exact parameters:

Figures 1(a) and 1(b) shows the variation of P_D and P_{FA} with the signal-to-noise ratio (SNR). The estimated results are averaged over 1000 realisations of the test statistic. The estimated probability and the theoretical probability match up to numerical precision. It can observed that the probability of detection increases monotonically with increase in SNR. High probabilities of detection > 0.9 are achieved with SNR > -10dB. The estimated probability of false alarm is consistent to be around the true value of 0.05.



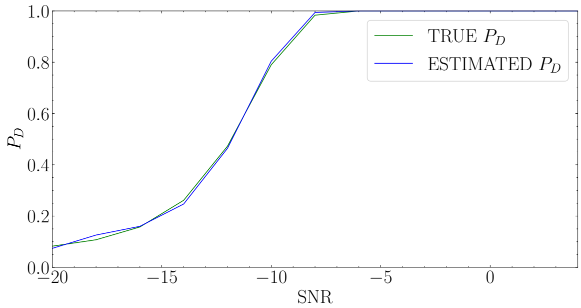
(a) Probability of detection.

(b) Probability of false alarm.

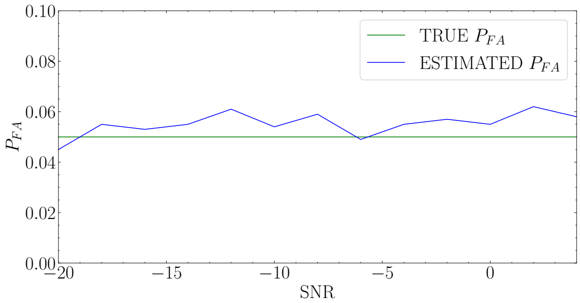
Figure 1: Monte-Carlo probabilities of the energy detector varying with SNR when the parameters are exact.

b) Monte-Carlo simulations with inexact parameters:

Figures 2(a) and 2(b) shows the variation of P_D and P_{FA} with the signal-to-noise ratio (SNR). The estimated results are averaged over 1000 realisations of the test statistic. As compared to Figures 1(a) and 1(b), the probability of detection is only as high as 0.8 at the SNR of -10dB. The probability of false alarm is also consistently higher than the true value.



(a) Probability of detection.



(b) Probability of false alarm.

Figure 2: Monte-Carlo probabilities of the energy detector varying with SNR when the parameters are inexact.

c) Comparison with the Bayes' detector: The Bayes' detector, similar to the LRT, compares the ratio of the likelihoods to the ratio of the prior probabilities. The Bayes' detector decides on $\mathcal{H}_1$ if:

$$\begin{aligned} L(\mathbf{y}) &= \frac{p_Y(\mathbf{y};\mathcal{H}_1)}{p_Y(\mathbf{y};\mathcal{H}_0)} > \frac{P[\mathcal{H}_0]}{P[\mathcal{H}_1]}, \\ &= \frac{\frac{1}{(2\pi(\sigma_s^2+\sigma_w^2))^{N/2}}\exp\left(-\frac{1}{2(\sigma_s^2+\sigma_w^2)}\sum_{n=0}^{N}|y[n]|^2\right)}{\frac{1}{(2\pi\sigma_w^2)^{N/2}}\exp\left(-\frac{1}{2\sigma_w^2}\sum_{n=0}^{N}|y[n]|^2\right)} > \frac{P[\mathcal{H}_0]}{P[\mathcal{H}_1]} \\ &= \left(\frac{\sigma_w^2}{\sigma_s^2+\sigma_w^2}\right)^{N/2}\exp\left(\frac{1}{2}\frac{\sigma_s^2}{\sigma_w^2(\sigma_s^2+\sigma_w^2)}\sum_{n=0}^{N-1}|y[n]|^2\right) > \frac{P[\mathcal{H}_0]}{P[\mathcal{H}_1]} \\ \implies T(\mathbf{y}) &= \sum_{n=0}^{N-1}|y[n]|^2 > 2\left(\frac{(\sigma_s^2+\sigma_w^2)\sigma_w^2}{\sigma_s^2}\right)\left(\frac{N}{2}\ln\left(\frac{\sigma_s^2+\sigma_w^2}{\sigma_w^2}\right)+\ln\left(\frac{P[\mathcal{H}_0]}{P[\mathcal{H}_1]}\right)\right). \end{aligned} \tag{18}$$

Figure 3 shows the variation of the thresholds of the Neymann-Pearson detector and the Bayes' detector with the SNR. It can be observed that the thresholds decrease monotonically with SNR as the energy in the noise decreases with increasing SNR, and a small threshold reliably detects the signal. Between the detectors, it can be observed that, for a probability of false alarm $P_{FA} = 0.05$ in the Neymann-Pearson detector and a prior $P[\mathcal{H}_0] = 0.2$ in the Bayes' detector, the thresholds are similar. The threshold of the Neymann-Pearson detector is marginally smaller than the threshold of the Bayes' detector at lower SNR, and gradually meet as the SNR increases.

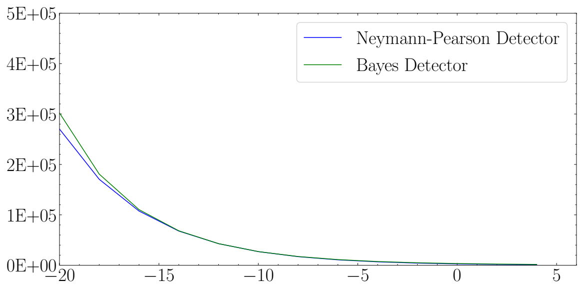
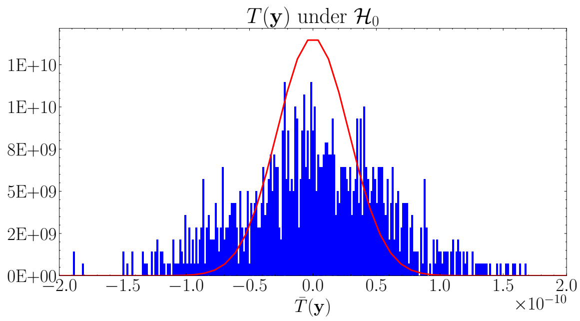


Figure 3: Variation of Neymann-Pearson threshold vs. Bayes' threshold with SNR.

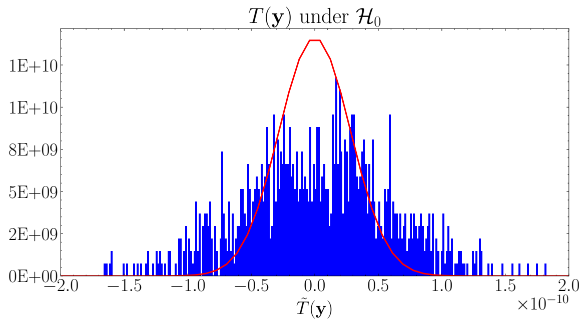
Cyclostationarity Detector

a) Distribution of the test statistics:

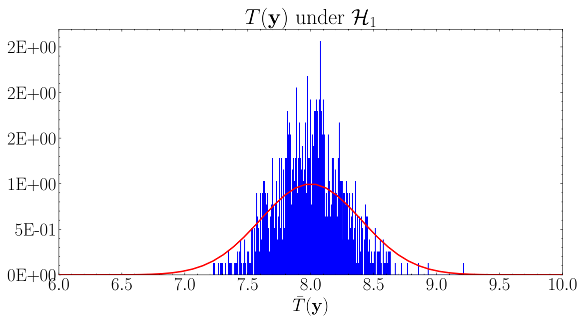
Figures 4(a) and 4(b) shows the real and imaginary parts of the density function of the test statistic under $\mathcal{H}_0$, and Figures 4(c) and 4(d) shows the real and imaginary parts of the density function of the test statistic under $\mathcal{H}_1$. The histogram is plotted using 1000 realisation of each test statistic, and is compared with the Gaussian PDF with parameters derived. The approximation is known to improve with longer data samples N_d. With $N_d = 32$ and $K = 50$, the distributions are fairly well approximated by a Gaussian random variable.



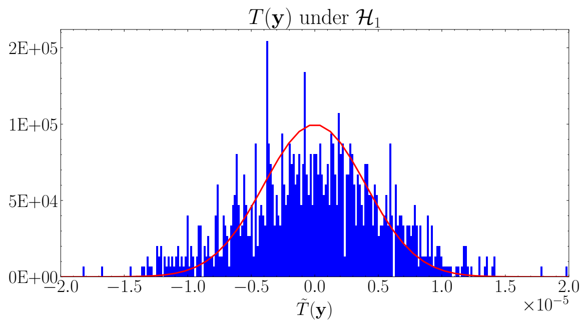
(a) Real component in $\mathcal{H}_0$.



(b) Imaginary component in $\mathcal{H}_0$.



(c) Real component in $\mathcal{H}_1$.



(d) Imaginary component in $\mathcal{H}_1$.

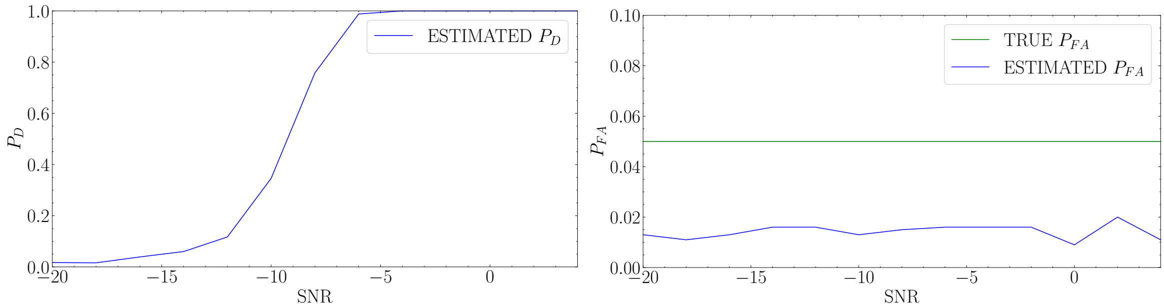
Figure 4: Monte-Carlo distributions of the test statistics of the cyclostationary detector.

b) Monte-Carlo simulations with exact parameters:

Figures 5(a) and 5(b) shows the variation of P_D and P_{FA} with the signal-to-noise ratio (SNR). The estimated results are averaged over 1000 realisations of the test statistic. The estimated probability and the theoretical probability match up to numerical precision. It can observed that the probability of detection increases monotonically with increase in SNR. As compared to the energy detector, the cyclostationary detector gives probability of detection > 0.9 for SNR only > -7dB, which is 3dB higher than the energy detector. However, the probability of false alarm is consistently, much lower than the probability of false alarm in the energy detector.

c) Monte-Carlo simulations with inexact parameters:

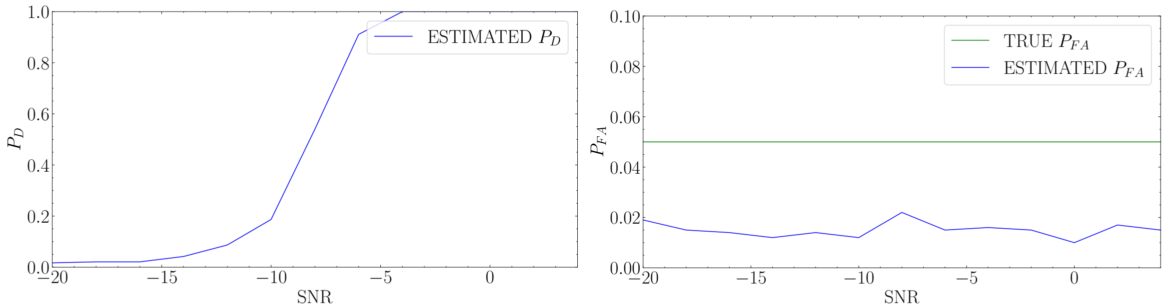
Figures 6(a) and 6(b) shows the variation of P_D and P_{FA} with the signal-to-noise ratio (SNR). The estimated results are averaged over 1000 realisations of the test statistic. As compared to Figures 5(a) and 5(b), the probability of detection is only as high as 0.8 at the SNR of -7dB. The probability of false alarm is marginally higher than in Figure 5(b).



(a) Probability of detection.

(b) Probability of false alarm.

Figure 5: Monte-Carlo probabilities of the cyclostationary detector varying with SNR when the parameters are exact.



(a) Probability of detection.

(b) Probability of false alarm.

Figure 6: Monte-Carlo probabilities of the cyclostationary detector varying with SNR when the parameters are inexact.

Scripts

The Python3 scripts to generate all figures can be downloaded from the GitHub repository `https://github.com/kamath-abhijith/Spectrum_Sensing`. Use `requirements.txt` to install all dependencies. Also, see the following code snippets for reference.

Implementation of Energy Detector

The relevant functions are in `utils.py`.

```python
'''

NEYMANN-PEARSON SIGNAL DETECTOR FOR
SPECTRUM SAMPLING IN COGNITIVE RADIO
BASED ON SIGNAL ENERGY

AUTHOR: ABIJITH J. KAMATH
abijithj@iisc.ac.in, kamath-abhijith.github.io

'''

# %% LOAD LIBRARIES

import os
import numpy as np

```

```python
from tqdm import tqdm

from scipy.stats import chi2

from matplotlib import style
from matplotlib import rcParams
from matplotlib import pyplot as plt

import utils

# %% PLOT SETTINGS

plt.style.use(['science','ieee'])

plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["cm"],
    "mathtext.fontset": "cm",
    "font.size": 24})

# %% PARAMETERS

Nd = 32
Nc = 8
K = 50
N = (K+1)*(Nc+Nd)
NUM_STATS = 1000

PFA = 0.05

SNR_MIN = -20
SNR_MAX = 6
SNR_STEP = 2

# %% MONTE CARLO SIMULATIONS // CLEAN PARAMETERS

SNRS = np.arange(SNR_MIN, SNR_MAX, SNR_STEP)

true_PFA = np.zeros(len(SNRS))
true_PD = np.zeros(len(SNRS))
est_PFA = np.zeros(len(SNRS))
est_PD = np.zeros(len(SNRS))

for itr, SNR in tqdm(enumerate(SNRS)):
    noise_var = 1 / 10**(SNR/10)
    threshold = chi2.isf(q=PFA, df=N) * noise_var

    stats_H0 = utils.energy_stat_H0(NUM_STATS, Nd, Nc, K, noise_var)
    stats_H1 = utils.energy_stat_H1(NUM_STATS, Nd, Nc, K, noise_var)

    false_alarms = sum(stats_H0 > threshold)
    detections = sum(stats_H1 > threshold)

    est_PFA[itr] = false_alarms / NUM_STATS
    est_PD[itr] = detections / NUM_STATS

    true_PFA[itr] = PFA
    true_PD[itr] = chi2.sf(x=threshold / (1 + noise_var), df=N)

# %% PLOTS // CLEAN PARAMETERS

```

```python
os.makedirs('./results/', exist_ok=True)
path = './results/'

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, true_PD, ax=ax, plot_colour='green',
    legend_label=r'TRUE $P_D$', show=False)
utils.plot_signal(SNRS, est_PD, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{D}$', yaxis_label=r'$P_{D}$',
    xaxis_label=r'$\mathrm{SNR}$', show=True,
    save=path+'eneProb_PD')

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, true_PFA, ax=ax, plot_colour='green',
    legend_label=r'TRUE $P_{FA}$', show=False)
utils.plot_signal(SNRS, est_PFA, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{FA}$', yaxis_label=r'$P_{FA}$',
    xaxis_label=r'$\mathrm{SNR}$', ylimits=[0,2*PFA], show=True,
    save=path+'eneProb_PFA')

# %% MONTE CARLO SIMULATIONS // NOISY PARAMETERS

SNRS = np.arange(SNR_MIN, SNR_MAX, SNR_STEP)

true_PFA = np.zeros(len(SNRS))
true_PD = np.zeros(len(SNRS))
est_PFA = np.zeros(len(SNRS))
est_PD = np.zeros(len(SNRS))

for itr, SNR in tqdm(enumerate(SNRS)):
    noise_var = 1 / 10**(SNR/10)
    noise_var = noise_var * 10**(1/10)
    threshold = chi2.isf(q=PFA, df=N) * noise_var

    stats_H0 = utils.energy_stat_H0(NUM_STATS, Nd, Nc, K, noise_var)
    stats_H1 = utils.energy_stat_H1(NUM_STATS, Nd, Nc, K, noise_var)

    false_alarms = sum(stats_H0 > threshold)
    detections = sum(stats_H1 > threshold)

    est_PFA[itr] = false_alarms / NUM_STATS
    est_PD[itr] = detections / NUM_STATS

    true_PFA[itr] = PFA
    true_PD[itr] = chi2.sf(x=threshold / (1 + noise_var), df=N)

# %% PLOTS // NOISY PARAMETERS

os.makedirs('./results/', exist_ok=True)
path = './results/'

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, true_PD, ax=ax, plot_colour='green',
    legend_label=r'TRUE $P_D$', show=False)
utils.plot_signal(SNRS, est_PD, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{D}$', yaxis_label=r'$P_{D}$',
    xaxis_label=r'$\mathrm{SNR}$', show=True,
    save=path+'eneProb_PD_Noisy')

```

```python
plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, true_PFA, ax=ax, plot_colour='green',
    legend_label=r'TRUE $P_{FA}$', show=False)
utils.plot_signal(SNRS, est_PFA, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{FA}$', yaxis_label=r'$P_{FA}$',
    xaxis_label=r'$\mathrm{SNR}$', ylimits=[0,2*PFA], show=True,
    save=path+'eneProb_PFA_Noisy')

# %% THRESHOLD COMPARISONS

prior1 = 0.2
prior0 = 1-prior1
SNRS = np.arange(SNR_MIN, SNR_MAX, SNR_STEP)

threshold_NP = np.zeros(len(SNRS))
threshold_BD = np.zeros(len(SNRS))
for itr, SNR in tqdm(enumerate(SNRS)):
    noise_var = 1 / 10**(SNR/10)
    noise_var = noise_var * 10**(1/10)

    threshold_NP[itr] = chi2.isf(q=PFA, df=N) * noise_var
    threshold_BD[itr] = 2 * (1+noise_var) * noise_var * \
        (N/2 * np.log((1+noise_var)/noise_var) + np.log(prior0/prior1))

# %% PLOTS :: THRESHOLD COMPARISON

os.makedirs('./results/', exist_ok=True)
path = './results/'

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, threshold_NP, ax=ax,
    legend_label=r'Neymann-Pearson Detector', show=False)
utils.plot_signal(SNRS, threshold_BD, ax=ax, plot_colour='green',
    ylimits=[0,0.5*1e6], legend_label=r'Bayes Detector', show=True,
    save=path+'thresholds')
# %%
```

Implementation of Cyclostationary Detector

The relevant functions are in `utils.py`.

```python
'''

NEYMANN-PEARSON SIGNAL DETECTOR FOR
SPECTRUM SAMPLING IN COGNITIVE RADIO
BASED ON CYCLOSTATIONARITY

AUTHOR: ABIJITH J. KAMATH
abijithj@iisc.ac.in, kamath-abhijith.github.io

'''

# %% LOAD LIBRARIES

import os
import numpy as np

from tqdm import tqdm
```

```python

from scipy.stats import chi2

from matplotlib import style
from matplotlib import rcParams
from matplotlib import pyplot as plt

import utils

# %% PLOT SETTINGS

plt.style.use(['science','ieee'])

plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["cm"],
    "mathtext.fontset": "cm",
    "font.size": 24})

# %% PARAMETERS

Nd = 32
Nc = 8
K = 50
N = (K+1)*(Nc+Nd)
NUM_STATS = 1000

PFA = 0.05

SNR_MIN = -20
SNR_MAX = 6
SNR_STEP = 2

# %% DISTRIBUTION OF TEST STATISTIC

SNR = 100
noise_var = 1 / 10**(SNR/10)

stats_H0 = utils.cyclo_stat_H0(NUM_STATS, Nd, Nc, K, noise_var)
stats_H1 = utils.cyclo_stat_H1(NUM_STATS, Nd, Nc, K, noise_var)

# %% PLOTS :: DISTRIBUTION OF TEST STATISTIC

os.makedirs('./results/', exist_ok=True)
path = './results/'

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_gaussian(np.linspace(-2*1e-10,2*1e-10), 0, Nc/(2*K)*noise_var**2,
    ax=ax, show=False)
utils.plot_histogram(np.real(stats_H0), 256, ax=ax,
    xaxis_label=r'$\bar{T}(\mathbf{y})$',
    title_text=r'$T(\mathbf{y})$ under $\mathcal{H}_0$', show=True, save=path+'
    cycDist_H0R')

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_gaussian(np.linspace(-2*1e-10,2*1e-10), 0, Nc/(2*K)*noise_var**2,
    ax=ax, show=False)
utils.plot_histogram(np.imag(stats_H0), 256, ax=ax,
    xaxis_label=r'$\tilde{T}(\mathbf{y})$',
```

```
    title_text=r'$T(\mathbf{y})$ under $\mathcal{H}_0$', show=True, save=path+'
    cycDist_H0I')

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_gaussian(np.linspace(6,10), Nc, Nc/K*(1+noise_var+(noise_var**2)/2),
    ax=ax, show=False)
utils.plot_histogram(np.real(stats_H1), 256, ax=ax,
    xaxis_label=r'$\bar{T}(\mathbf{y})$', xlimits=[6,10],
    title_text=r'$T(\mathbf{y})$ under $\mathcal{H}_1$', show=True, save=path+'
    cycDist_H1R')

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_gaussian(np.linspace(-2*1e-5,2*1e-5), 0, Nc/K*(noise_var+(noise_var**2)/2),
    ax=ax, show=False)
utils.plot_histogram(np.imag(stats_H1), 256, ax=ax,
    xaxis_label=r'$\tilde{T}(\mathbf{y})$', xlimits=[-2*1e-5,2*1e-5],
    title_text=r'$T(\mathbf{y})$ under $\mathcal{H}_1$', show=True, save=path+'
    cycDist_H1I')

# %% MONTE CARLO SIMULATIONS // CLEAN PARAMETERS

SNRS = np.arange(SNR_MIN, SNR_MAX, SNR_STEP)

true_PFA = np.zeros(len(SNRS))
est_PFA = np.zeros(len(SNRS))
est_PD = np.zeros(len(SNRS))

for itr, SNR in tqdm(enumerate(SNRS)):
    noise_var = 1 / 10**(SNR/10)
    threshold = chi2.isf(q=PFA, df=2) * Nc / K * (noise_var**2)

    stats_H0 = utils.cyclo_stat_H0(NUM_STATS, Nd, Nc, K, noise_var)
    stats_H1 = utils.cyclo_stat_H1(NUM_STATS, Nd, Nc, K, noise_var)

    false_alarms = sum(np.square(np.abs(stats_H0)) > threshold)
    detections = sum(np.square(np.abs(stats_H1)) > threshold)

    est_PFA[itr] = false_alarms / NUM_STATS
    est_PD[itr] = detections / NUM_STATS

    true_PFA[itr] = PFA

# %% PLOTS :: MONTE CARLO SIMULATIONS // CLEAN PARAMETERS

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, est_PD, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{D}$', yaxis_label=r'$P_{D}$',
    xaxis_label=r'$\mathrm{SNR}$', show=True, save=path+'cycProb_PD')

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, true_PFA, ax=ax, plot_colour='green',
    legend_label=r'TRUE $P_{FA}$', show=False)
utils.plot_signal(SNRS, est_PFA, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{FA}$', yaxis_label=r'$P_{FA}$',
    xaxis_label=r'$\mathrm{SNR}$', ylimits=[0,2*PFA], show=True,
    save=path+'cycProb_PFA')
```

```python
# %% MONTE CARLO SIMULATIONS // NOISY PARAMETERS

SNRS = np.arange(SNR_MIN, SNR_MAX, SNR_STEP)

true_PFA = np.zeros(len(SNRS))
est_PFA = np.zeros(len(SNRS))
est_PD = np.zeros(len(SNRS))

for itr, SNR in tqdm(enumerate(SNRS)):
    noise_var = 1 / 10**(SNR/10)
    noise_var = noise_var * 10**(1/10)
    threshold = chi2.isf(q=PFA, df=2) * Nc / K * (noise_var**2)

    stats_H0 = utils.cyclo_stat_H0(NUM_STATS, Nd, Nc, K, noise_var)
    stats_H1 = utils.cyclo_stat_H1(NUM_STATS, Nd, Nc, K, noise_var)

    false_alarms = sum(np.square(np.abs(stats_H0)) > threshold)
    detections = sum(np.square(np.abs(stats_H1)) > threshold)

    est_PFA[itr] = false_alarms / NUM_STATS
    est_PD[itr] = detections / NUM_STATS

    true_PFA[itr] = PFA

# %% PLOTS :: MONTE CARLO SIMULATIONS // NOISY PARAMETERS

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, est_PD, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{D}$', yaxis_label=r'$P_{D}$',
    xaxis_label=r'$\mathrm{SNR}$', show=True, save=path+'cycProb_PD_Noisy')

plt.figure(figsize=(12,6))
ax = plt.gca()
utils.plot_signal(SNRS, true_PFA, ax=ax, plot_colour='green',
    legend_label=r'TRUE $P_{FA}$', show=False)
utils.plot_signal(SNRS, est_PFA, ax=ax, plot_colour='blue',
    legend_label=r'ESTIMATED $P_{FA}$', yaxis_label=r'$P_{FA}$',
    xaxis_label=r'$\mathrm{SNR}$', ylimits=[0,2*PFA], show=True,
    save=path+'cycProb_PFA_Noisy')
```

utils.py

This script contains all the relevant functions and helpers.

```python
'''

TOOLS FOR SPECTRUM SENSING FOR COGNITIVE RADIO

AUTHOR: ABIJITH J. KAMATH
abijithj@iisc.ac.in, kamath-abhijith.github.io

'''

# %% LOAD LIBRARIES

import numpy as np

from matplotlib import pyplot as plt
from matplotlib.ticker import StrMethodFormatter
```

```
from scipy.stats import multivariate_normal

# %% PLOTTING FUNCTIONS

def plot_signal(x, y, ax=None, plot_colour='blue', xaxis_label=None,
    yaxis_label=None, title_text=None, legend_label=None, legend_show=True,
    legend_loc='upper right', line_style='-', line_width=None,
    show=False, xlimits=[-20,6], ylimits=[0,1], save=None):
    '''
    Plots signal with abscissa in x and ordinates in y

    '''
    if ax is None:
        fig = plt.figure(figsize=(12,6))
        ax = plt.gca()

    plt.plot(x, y, linestyle=line_style, linewidth=line_width, color=plot_colour,
        label=legend_label)
    if legend_label and legend_show:
        plt.legend(loc=legend_loc, frameon=True, framealpha=0.8, facecolor='white')
    plt.xlabel(xaxis_label)
    plt.ylabel(yaxis_label)

    plt.xlim(xlimits)
    plt.ylim(ylimits)
    plt.title(title_text)

    plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0E}'))

    if save:
        plt.savefig(save + '.pdf', format='pdf')

    if show:
        plt.show()

    return

def plot_histogram(x, num_bins, ax=None, plot_colour='blue', xaxis_label=None,
    yaxis_label=None, title_text=None, xlimits=[-2*1e-10,2*1e-10], show=False, save=None):
    '''
    Plots histogram of data in x

    '''

    if ax is None:
        fig = plt.figure(figsize=(12,6))
        ax = plt.gca()

    plt.hist(x, bins=num_bins, color=plot_colour, density=True)

    plt.xlabel(xaxis_label)
    plt.ylabel(yaxis_label)
    plt.title(title_text)

    plt.xlim(xlimits)
    # plt.ylim([0,30])

    plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0E}'))
```

```python
    if save:
        plt.savefig(save + '.pdf', format='pdf')

    if show:
        plt.show()

    return

def plot_gaussian(xvals, mean, var, ax=None, plot_colour='red',
    xaxis_label=None, yaxis_label=None, line_style='-', line_width=2,
    title_text=None, show=False, save=None):
    '''
    Plots 1D Gaussian with mean and variance

    '''

    if ax is None:
        fig = plt.figure(figsize=(12,6))
        ax = plt.gca()

    gaussian = multivariate_normal.pdf(xvals, mean, var)
    plt.plot(xvals, gaussian, linestyle=line_style, linewidth=line_width,
        color=plot_colour)

    plt.xlabel(xaxis_label)
    plt.ylabel(yaxis_label)
    plt.title(title_text)

    plt.gca().yaxis.set_major_formatter(StrMethodFormatter('{x:,.0E}'))

    if save:
        plt.savefig(save + '.pdf', format='pdf')

    if show:
        plt.show()

    return

# %% SIGNALS

def generate_QPSK_block(Nd, Nc):
    '''
    Generates a QPSK block with unit variance

    :param Nd: IFFT length
    :param Nc: length of cyclic repetition

    :return: OFDM signal block

    '''

    s = np.zeros(Nd)
    for k in range(Nd):
        s[k] = (2*np.random.binomial(1, 0.5)-1)/np.sqrt(2) + \
            1j * (2*np.random.binomial(1, 0.5)-1)/np.sqrt(2)

    x = np.fft.ifft(s) * np.sqrt(2*Nd)

    return np.hstack([x[:Nc], x])

def generate_QPSK_signal(Nd, Nc, K):
```

```
    '''
    Generate a QPSK signal with unit variance

    :param Nd: IFFT length of each block
    :param Nc: length of cyclic repetition
    :param K: number of OFDM blocks

    :return: K+1 OFDM signal blocks

    '''

    signal = generate_QPSK_block(Nd, Nc)
    for _ in range(K):
        signal = np.hstack([signal, generate_QPSK_block(Nd, Nc)])

    return signal

# %% DETECTORS

def energy_stat_H0(num_stats, Nd, Nc, K, noise_var):
    '''
    Generates signal energy statsitic for noise only hypothesis

    :param num_stats: number of realisations
    :param Nd: length of IFFT
    :param Nc: length of cyclic repetitions
    :param K: number of OFDM blocks
    :param noise_var: variance of AWGN channel

    :return: energy statistics

    '''

    stats = np.zeros(num_stats)
    for itr in range(num_stats):
        noise = np.sqrt(noise_var)*np.random.randn((K+1)*(Nd+Nc))

        stats[itr] = np.sum(np.square(np.abs(noise)))

    return stats

def energy_stat_H1(num_stats, Nd, Nc, K, noise_var):
    '''
    Generates signal energy statsitic for signal present hypothesis

    :param num_stats: number of realisations
    :param Nd: length of IFFT
    :param Nc: length of cyclic repetitions
    :param K: number of OFDM blocks
    :param noise_var: variance of AWGN channel

    :return: energy statistics

    '''

    stats = np.zeros(num_stats)
    for itr in range(num_stats):
        signal = generate_QPSK_signal(Nd, Nc, K)
        noise = np.sqrt(noise_var)*np.random.randn((K+1)*(Nd+Nc))

        stats[itr] = np.sum(np.square(np.abs(signal + noise)))
```

```
    return stats

def cyclo_stat_H0(num_stats, Nd, Nc, K, noise_var):
    '''
    Generates cyclostationary ACF statistic for noise only hypothesis

    :param num_stats: number of realisations
    :param Nd: length of IFFT
    :param Nc: length of cyclic repetitions
    :param K: number of OFDM blocks
    :param noise_var: variance of AWGN channel

    :return: cyclostationary ACF

    '''

    stats = np.zeros(num_stats, dtype=np.complex)
    for itr in range(num_stats):
        noise = np.sqrt(noise_var)*np.random.randn((K+1)*(Nd+Nc),2).view(np.complex)
        y = noise

        stat = np.complex(0)
        for n in range(Nc):
            for k in range(K):
                stat += y[n+k*(Nc+Nd)] * np.conjugate(y[n+k*(Nc+Nd)+Nd])

        stats[itr] = stat/K

    return stats

def cyclo_stat_H1(num_stats, Nd, Nc, K, noise_var):
    '''
    Generates cyclostationary ACF statistic for signal present hypothesis

    :param num_stats: number of realisations
    :param Nd: length of IFFT
    :param Nc: length of cyclic repetitions
    :param K: number of OFDM blocks
    :param noise_var: variance of AWGN channel

    :return: cyclostationary ACF

    '''

    stats = np.zeros(num_stats, dtype=np.complex)
    for itr in range(num_stats):
        signal = generate_QPSK_signal(Nd, Nc, K)[:,None]
        for k in range(K):
            signal[k*(Nc+Nd):k*(Nc+Nd)+Nc] = signal[k*(Nc+Nd)+Nd:(k+1)*(Nc+Nd)]
        noise = np.sqrt(noise_var)*np.random.randn((K+1)*(Nd+Nc),2).view(np.complex)
        y = signal + noise

        stat = np.complex(0)
        for n in range(Nc):
            for k in range(K):
                stat += y[n+k*(Nc+Nd)] * np.conjugate(y[n+k*(Nc+Nd)+Nd])

        stats[itr] = stat/K

    return stats
```