

Assignment #3

Instructor: Sundeep Chepuri

Name: Vineeth S, SR No.: 16543

Consider a data symbol sequence $\mathbf{s} = [s[0], s[1], \dots, s[N_d - 1]]^T$ of length N_d , where the entries are QPSK with variance $\sigma_s = 1$, i.e., $s[k] \in \{\pm 1/2 \pm j/2\}$. According to the central limit theorem and assuming a sufficiently large IFFT, the OFDM symbol blocks \mathbf{x} can be assumed to be zero-mean Gaussian with covariance matrix $E[\mathbf{x}\mathbf{x}^H] = \sigma_s^2 \mathbf{I}_{(K+1)(N_c+N_d)}$. \mathbf{w} can be assumed to be a zero-mean complex Gaussian random process, where $E[\mathbf{w}\mathbf{w}^H] = \sigma_w^2 \mathbf{I}_{(K+1)(N_c+N_d)}$. We wish to decide between the following two hypotheses:

$$\mathcal{H}_0 : \mathbf{y} = \mathbf{w}$$

$$\mathcal{H}_1 : \mathbf{y} = \mathbf{x} + \mathbf{w}$$

where $\mathbf{w} \sim \mathcal{CN}(0, \sigma_w^2 \mathbf{I})$, and \mathbf{w} is independent of \mathbf{x} .

PART A: Problem 1 Energy Detector

Problem 1A Neyman-Pearson Detector

(Solution) Consider the Neyman-Pearson likelihood ratio test. We decide on hypothesis \mathcal{H}_1 if

$$L(y) = \frac{P(\mathbf{y}; \mathcal{H}_1)}{P(\mathbf{y}; \mathcal{H}_0)} > \gamma$$

Let $N = (K + 1)(N_d + N_c)$ for sake of simplicity.

$$\mathcal{H}_0 : y \sim \mathcal{CN}(0, \sigma_w^2 \mathbf{I}_N)$$

$$\mathcal{H}_1 : y \sim \mathcal{CN}(0, (\sigma_s^2 + \sigma_w^2) \mathbf{I}_N)$$

$$\begin{aligned} & \frac{\frac{1}{(2\pi)^{N/2} (\sigma_s^2 + \sigma_w^2)^{N/2}} \exp\left(-\frac{1}{2(\sigma_s^2 + \sigma_w^2)} \sum_{n=0}^{N-1} |y[n]|^2\right)}{\frac{1}{(2\pi)^{N/2} (\sigma_w^2)^{N/2}} \exp\left(-\frac{1}{2\sigma_w^2} \sum_{n=0}^{N-1} |y[n]|^2\right)} > \gamma \\ & \frac{(\sigma_w^2)^{N/2}}{(\sigma_s^2 + \sigma_w^2)^{N/2}} \exp\left(-\frac{1}{2} \sum_{n=0}^{N-1} |y[n]|^2 \left(\frac{1}{\sigma_s^2 + \sigma_w^2} - \frac{1}{\sigma_w^2}\right)\right) > \gamma \\ & \frac{1}{2} \sum_{n=0}^{N-1} |y[n]|^2 \left(\frac{\sigma_s^2}{(\sigma_s^2 + \sigma_w^2) \sigma_w^2}\right) > \gamma' \\ & \sum_{n=0}^{N-1} |y[n]|^2 > \gamma'' \end{aligned}$$

Hence, we have our test statistic $T(y)$ for NP detector as,

$$T(y) = \sum_{n=0}^{N-1} |y[n]|^2$$

We have our test statistic as sum of zero-mean Gaussian random variables. Hence the test statistic follows a χ_N^2 distribution.

$$\mathcal{H}_0 : \frac{T(y)}{\sigma_w^2} \sim \chi_N^2$$

$$\mathcal{H}_1 : \frac{T(y)}{\sigma_s^2 + \sigma_w^2} \sim \chi_N^2$$

For a specified P_{FA} we can find the NP threshold as,

$$P_{FA} = P(T(y) > \gamma''; \mathcal{H}_0)$$

$$P_{FA} = Q_{\chi_N^2} \left(\frac{\gamma''}{\sigma_w^2} \right)$$

$$\gamma'' = \sigma_w^2 * Q_{\chi_N^2}^{-1}(P_{FA})$$

And the theoretical detection probability P_D as,

$$P_D = P(T(y) > \gamma''; \mathcal{H}_1)$$

$$= Q_{\chi_N^2} \left(\frac{\gamma''}{\sigma_s^2 + \sigma_w^2} \right)$$

Problem 1B Bayes Detector

(Solution) This part is related to the Problem 1C of PART B: Implementation. We are given that the primary user is mostly inactive with $P(\mathcal{H}_1) = 0.2$. Prior probabilities cannot be incorporated into Neyman-Pearson detector and hence we would be using Bayes detector for this case.

We decide on hypothesis \mathcal{H}_1 if

$$L(y) = \frac{P(\mathbf{y}; \mathcal{H}_1)}{P(\mathbf{y}; \mathcal{H}_0)} > \frac{P(\mathcal{H}_0)}{P(\mathcal{H}_1)}$$

Let $N = (K + 1)(N_d + N_c)$ for sake of simplicity.

$$\mathcal{H}_0 : y \sim \mathcal{CN}(0, \sigma_w^2 \mathbf{I}_N)$$

$$\mathcal{H}_1 : y \sim \mathcal{CN}(0, (\sigma_s^2 + \sigma_w^2) \mathbf{I}_N)$$

$$\frac{\frac{1}{(2\pi)^{N/2}(\sigma_s^2 + \sigma_w^2)^{N/2}} \exp \left(-\frac{1}{2(\sigma_s^2 + \sigma_w^2)} \sum_{n=0}^{N-1} |y[n]|^2 \right)}{\frac{1}{(2\pi)^{N/2}(\sigma_w^2)^{N/2}} \exp \left(-\frac{1}{2\sigma_w^2} \sum_{n=0}^{N-1} |y[n]|^2 \right)} > \frac{P(\mathcal{H}_0)}{P(\mathcal{H}_1)}$$

$$\frac{(\sigma_w^2)^{N/2}}{(\sigma_s^2 + \sigma_w^2)^{N/2}} \exp \left(-\frac{1}{2} \sum_{n=0}^{N-1} |y[n]|^2 \left(\frac{1}{\sigma_s^2 + \sigma_w^2} - \frac{1}{\sigma_w^2} \right) \right) > \frac{P(\mathcal{H}_0)}{P(\mathcal{H}_1)}$$

$$\frac{1}{2} \sum_{n=0}^{N-1} |y[n]|^2 \left(\frac{\sigma_s^2}{(\sigma_s^2 + \sigma_w^2)\sigma_w^2} \right) > \ln \left(\frac{(\sigma_s^2 + \sigma_w^2)^{N/2}}{(\sigma_w^2)^{N/2}} \frac{P(\mathcal{H}_0)}{P(\mathcal{H}_1)} \right)$$

Hence, we have our test statistic $T(y)$ for Bayes detector as,

$$T(y) = \sum_{n=0}^{N-1} |y[n]|^2 > 2 \left(\frac{(\sigma_s^2 + \sigma_w^2)\sigma_w^2}{\sigma_s^2} \right) \left(\frac{N}{2} \ln \left(\frac{\sigma_s^2 + \sigma_w^2}{\sigma_w^2} \right) + \ln P(\mathcal{H}_0) - \ln P(\mathcal{H}_1) \right)$$

PART A: Problem 2 Cyclostationary Detector

Problem 2A Theory

(Solution) Following the theory given in the problem statement, we have our test statistic $T(y)$ as,

Under hypothesis \mathcal{H}_1

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

For $n = 0, \dots, N_c - 1$, let

$$y[n + kN] = x + w_1$$

$$y[n + kN + N_d] = x + w_2$$

Let $z = y[n + kN] y^*[n + kN + N_d]$.

We have,

$$\begin{aligned}
 E[z] &= E[(x + w_1)(x + w_2)^*] \\
 &= E[|x|^2 + xw_2^* + x^*w_1 + w_1w_2^*] \\
 &= E[|x|^2] \\
 &= \sigma_s^2 \\
 &= 1
 \end{aligned}$$

$$\begin{aligned}
 E[T(y)] &= \frac{1}{K} \sum_{n=0}^{N_c-1} \sum_{k=0}^{K-1} E[y[n + kN] y^*[n + kN + N_d]] \\
 &= \frac{1}{K} N_c K \\
 &= N_c
 \end{aligned}$$

$$T^2(y) = \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} y[n_1 + k_1N] y^*[n_1 + k_1N + N_d] y[n_2 + k_2N] y^*[n_2 + k_2N + N_d]$$

For $n = 0, \dots, N_c - 1$, let

$$y[n_1 + k_1N] = x_1 + w_{11}$$

$$y[n_2 + k_2N] = x_2 + w_{21}$$

$$y[n_1 + k_1N + N_d] = x_1 + w_{12}$$

$$y[n_2 + k_2N + N_d] = x_2 + w_{22}$$

Let $z = y[n_1 + k_1N] y^*[n_1 + k_1N + N_d] y[n_2 + k_2N] y^*[n_2 + k_2N + N_d]$

For $n_1 \neq n_2$ or $k_1 \neq k_2$, x_1 and x_2 are independent,

$$\begin{aligned}
 E[z] &= E[(x_1 + w_{11})(x_1 + w_{12})^*(x_2 + w_{21})(x_2 + w_{22})^*] \\
 &= E[(x_1 + w_{11})(x_1 + w_{12})^*]E[(x_2 + w_{21})(x_2 + w_{22})^*] \\
 &= \sigma_s^4 \\
 &= 1
 \end{aligned}$$

For $n_1 = n_2$ and $k_1 = k_2$,

$$\begin{aligned}
 E[z] &= E[(x_1 + w_{11})(x_1 + w_{12})^*(x_1 + w_{11})(x_1 + w_{12})^*] \\
 &= E[(|x_1|^2 + x_1 w_{12}^* + x_1^* w_{11} + w_{11} w_{12}^*)^2] \\
 &= E[|x_1|^4] \\
 &= 2\sigma_s^4 \\
 &= 2
 \end{aligned}$$

where $|x_1| = \sqrt{\text{Re}(x_1)^2 + \text{Im}(x_1)^2}$ and hence $|x_1|$ follows Rayleigh distribution. Using the equation for fourth moment of Rayleigh random variable, we have $E[|x_1|^4] = 2\sigma_s^4$. Also, from complex Gaussian distribution properties we have, $E[w_{ij}] = 0$ and $E[w_{ij}^2] = 0$. Hence we have,

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

Similarly,

$$|T(y)|^2 = \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} y[n_1 + k_1 N]y^*[n_1 + k_1 N + N_d]y^*[n_2 + k_2 N]y[n_2 + k_2 N + N_d]$$

Let $z = y[n_1 + k_1 N]y^*[n_1 + k_1 N + N_d]y^*[n_2 + k_2 N]y[n_2 + k_2 N + N_d]$

For $n_1 \neq n_2$ or $k_1 \neq k_2$, x_1 and x_2 are independent,

$$\begin{aligned}
 E[z] &= E[(x_1 + w_{11})(x_1 + w_{12})^*(x_2 + w_{21})^*(x_2 + w_{22})] \\
 &= E[(x_1 + w_{11})(x_1 + w_{12})^*]E[(x_2 + w_{21})^*(x_2 + w_{22})] \\
 &= \sigma_s^4 \\
 &= 1
 \end{aligned}$$

For $n_1 = n_2$ and $k_1 = k_2$,

$$\begin{aligned}
 E[z] &= E[(x_1 + w_{11})(x_1 + w_{12})^*(x_1 + w_{11})^*(x_1 + w_{12})] \\
 &= E[(|x_1|^2 + x_1 w_{12}^* + x_1^* w_{11} + w_{11} w_{12}^*)(|x_1|^2 + x_1^* w_{12} + x_1 w_{11}^* + w_{11}^* w_{12})] \\
 &= E[|x_1|^4 + |x_1|^2 |w_{12}|^2 + |x_1|^2 |w_{11}|^2 + |w_{11}|^2 |w_{12}|^2] \\
 &= 2\sigma_s^4 + 2\sigma_s^2 \sigma_w^2 + \sigma_w^4 \\
 &= 2 + 2\sigma_w^2 + \sigma_w^4
 \end{aligned}$$

Hence we have,

$$\begin{aligned}
E[|T(y)|^2] &= \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} E[(y[n_1 + k_1 N] y^*[n_1 + k_1 N + N_d] y^*[n_2 + k_2 N] y[n_2 + k_2 N + N_d])] \\
&= \frac{1}{K^2} [N_c K (2 + 2\sigma_w^2 + \sigma_w^4) + N_c^2 K^2 - N_c K] \\
&= N_c^2 + \frac{N_c}{K} (1 + 2\sigma_w^2 + \sigma_w^4)
\end{aligned}$$

Summarizing all data we have,

$$\begin{aligned}
E[T] &= N_c \\
E[T^2(y)] &= N_c^2 + \frac{N_c}{K} &= E[\bar{T}^2 + j2\bar{T}\tilde{T} - \tilde{T}^2] \\
E[|T(y)|^2] &= N_c^2 + \frac{N_c}{K} (1 + 2\sigma_w^2 + \sigma_w^4) &= E[\bar{T}^2 - \tilde{T}^2]
\end{aligned}$$

Solving we have,

$$\begin{aligned}
E[\bar{T}] &= N_c \\
E[\tilde{T}] &= 0 \\
E[\bar{T}\tilde{T}] &= 0 \\
E[\bar{T}^2] &= N_c^2 + \frac{N_c}{K} (1 + \sigma_w^2 + \frac{\sigma_w^4}{2}) \\
E[\tilde{T}^2] &= \frac{N_c}{K} (\sigma_w^2 + \frac{\sigma_w^4}{2})
\end{aligned}$$

Finally we have,

$$\begin{aligned}
E[\bar{T}] &= N_c \\
Var(\bar{T}) &= \frac{N_c}{K} (1 + \sigma_w^2 + \frac{\sigma_w^4}{2}) \\
E[\tilde{T}] &= 0 \\
Var(\tilde{T}) &= \frac{N_c}{K} (\sigma_w^2 + \frac{\sigma_w^4}{2}) \\
Cov(\bar{T}\tilde{T}) &= 0
\end{aligned}$$

Under hypothesis \mathcal{H}_0

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

For $n = 0, \dots, N_c - 1$, let

$$\begin{aligned}
y[n + kN] &= w_1 \\
y[n + kN + N_d] &= w_2
\end{aligned}$$

Let $z = y[n + kN]y^*[n + kN + N_d]$.

We have,

$$\begin{aligned} E[z] &= E[w_1 w_2^*] \\ &= 0 \end{aligned}$$

$$\begin{aligned} E[T(y)] &= \frac{1}{K} \sum_{n=0}^{N_c-1} \sum_{k=0}^{K-1} E[y[n + kN]y^*[n + kN + N_d]] \\ &= 0 \end{aligned}$$

$$T^2(y) = \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} y[n_1 + k_1N]y^*[n_1 + k_1N + N_d]y[n_2 + k_2N]y^*[n_2 + k_2N + N_d]$$

For $n = 0, \dots, N_c - 1$, let

$$\begin{aligned} y[n_1 + k_1N] &= w_{11} & y[n_2 + k_2N] &= w_{21} \\ y[n_1 + k_1N + N_d] &= w_{12} & y[n_2 + k_2N + N_d] &= w_{22} \end{aligned}$$

Let $z = y[n_1 + k_1N]y^*[n_1 + k_1N + N_d]y[n_2 + k_2N]y^*[n_2 + k_2N + N_d]$

For $n_1 \neq n_2$ or $k_1 \neq k_2$, x_1 and x_2 are independent,

$$\begin{aligned} E[z] &= E[w_{11}w_{12}^*w_{21}w_{22}^*] \\ &= E[w_{11}]E[w_{12}^*]E[w_{21}]E[w_{22}^*] \\ &= 0 \end{aligned}$$

For $n_1 = n_2$ and $k_1 = k_2$,

$$\begin{aligned} E[z] &= E[w_{11}w_{12}^*w_{11}w_{12}^*] \\ &= E[|w_{11}|^2|w_{12}|^2] \\ &= \sigma_w^4 \end{aligned}$$

$$\begin{aligned} E[T^2(y)] &= \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} E[(y[n_1 + k_1N]y^*[n_1 + k_1N + N_d]y[n_2 + k_2N]y^*[n_2 + k_2N + N_d])] \\ &= \frac{1}{K^2} [N_c K \sigma_w^4] \\ &= \frac{N_c}{K} \sigma_w^4 \end{aligned}$$

Similarly,

$$|T(y)|^2 = \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} y[n_1 + k_1N]y^*[n_1 + k_1N + N_d]y^*[n_2 + k_2N]y[n_2 + k_2N + N_d]$$

Let $z = y[n_1 + k_1N]y^*[n_1 + k_1N + N_d]y^*[n_2 + k_2N]y[n_2 + k_2N + N_d]$

For $n_1 \neq n_2$ or $k_1 \neq k_2$, x_1 and x_2 are independent,

$$\begin{aligned} E[z] &= E[w_{11}w_{12}^*w_{21}^*w_{22}] \\ &= E[w_{11}]E[w_{12}^*]E[w_{21}^*]E[w_{22}] \\ &= 0 \end{aligned}$$

For $n_1 = n_2$ and $k_1 = k_2$,

$$\begin{aligned} E[z] &= E[w_{11}w_{12}^*w_{11}^*w_{12}] \\ &= E[|w_{11}|^2|w_{12}|^2] \\ &= \sigma_w^4 \end{aligned}$$

Hence we have,

$$\begin{aligned} E[|T(y)|^2] &= \frac{1}{K^2} \sum_{n_1=0}^{N_c-1} \sum_{k_1=0}^{K-1} \sum_{n_2=0}^{N_c-1} \sum_{k_2=0}^{K-1} E[(y[n_1 + k_1N]y^*[n_1 + k_1N + N_d]y^*[n_2 + k_2N]y[n_2 + k_2N + N_d])] \\ &= \frac{1}{K^2} [N_c K \sigma_w^4] \\ &= \frac{1}{K} N_c \sigma_w^4 \end{aligned}$$

Summarizing all data we have,

$$\begin{aligned} E[T] &= 0 \\ E[T^2(y)] &= \frac{1}{K} N_c \sigma_w^4 &= E[\bar{T}^2 + j2\bar{T}\tilde{T} - \tilde{T}^2] \\ E[|T(y)|^2] &= \frac{1}{K} N_c \sigma_w^4 &= E[\bar{T}^2 - \tilde{T}^2] \end{aligned}$$

Solving we have,

$$\begin{aligned} E[\bar{T}] &= 0 \\ E[\tilde{T}] &= 0 \\ E[\bar{T}\tilde{T}] &= 0 \\ E[\bar{T}^2] &= \frac{N_c}{2K} \sigma_w^4 \\ E[\tilde{T}^2] &= \frac{N_c}{2K} \sigma_w^4 \end{aligned}$$

Finally we have,

$$\begin{aligned} E[\bar{T}] &= 0 \\ Var(\bar{T}) &= \frac{N_c}{2K} \sigma_w^4 \\ E[\tilde{T}] &= 0 \\ Var(\tilde{T}) &= \frac{N_c}{2K} \sigma_w^4 \\ Cov(\bar{T}\tilde{T}) &= 0 \end{aligned}$$

Problem 2B Neyman-Pearson Detector

We decide on \mathcal{H}_1 if $|T(y)| > \gamma$. Or equivalently if $|T(y)|^2 > \gamma^2$.

For a specified P_{FA} we can find the NP threshold as,

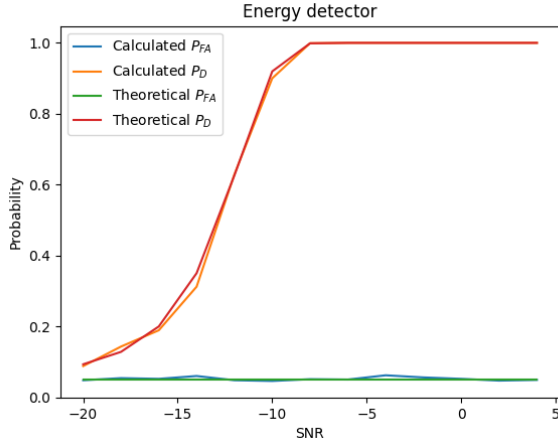
$$P_{FA} = P(|T(y)| > \gamma; \mathcal{H}_0)$$

$$P_{FA} = P(|T(y)|^2 > \gamma^2; \mathcal{H}_0)$$

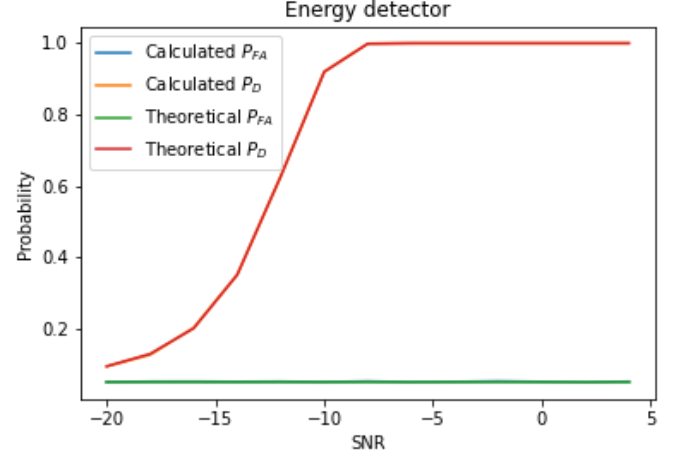
$$P_{FA} = P(\bar{T}^2(y) + \tilde{T}^2(y) > \gamma^2; \mathcal{H}_0)$$

$$P_{FA} = Q_{\chi^2_2} \left(\frac{\gamma^2}{\frac{N_c}{2K} \sigma_w^4} \right)$$

$$\gamma^2 = \frac{N_c}{2K} \sigma_w^4 * Q_{\chi^2_2}^{-1}(P_{FA})$$

PART B: Energy Detector**Subpart A**

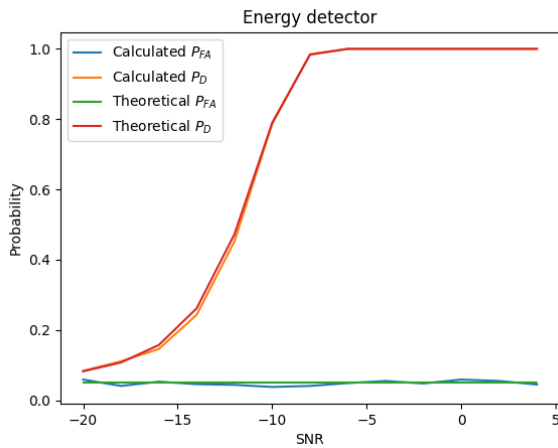
(a) For 1000 records



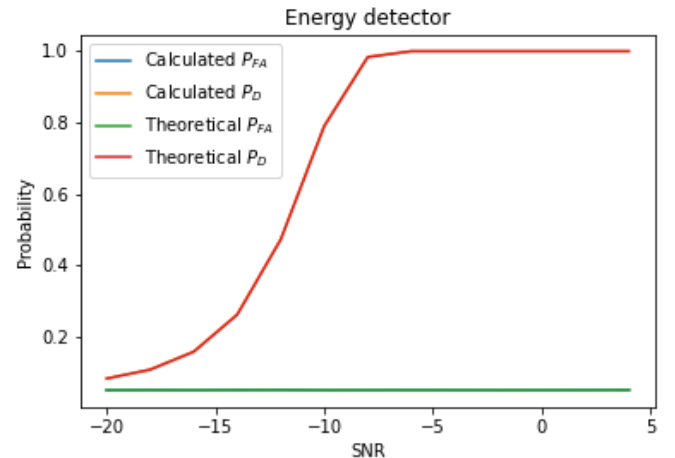
(b) For 100000 records

Figure 1: Probability vs SNR

From the plot Figure(1), we can easily observe that the theoretical P_d and calculated P_d increases with the SNR value. We can also verify that P_{FA} is under 0.05. Figure(1a) represents the detector performance with 1000 statistics and Figure(1b) represents the detector performance with 100000 statistics. We can see that with increased number of test statistics, we get the ideal detector performance.

Subpart B

(a) For 1000 records



(b) For 100000 records

Figure 2: Probability vs SNR

From the plot Figure(2), we can observe a clear drop in theoretical P_d and calculated P_d compared to the previous case when we have noisy estimates.. However the performance drop is negligible for higher values

of SNR — owing to the smaller noise component. Figure(2a) represents the detector performance with 1000 statistics and Figure(2b) represents the detector performance with 100000 statistics. We can see that with increased number of test statistics, we get the ideal detector performance.

Subpart C

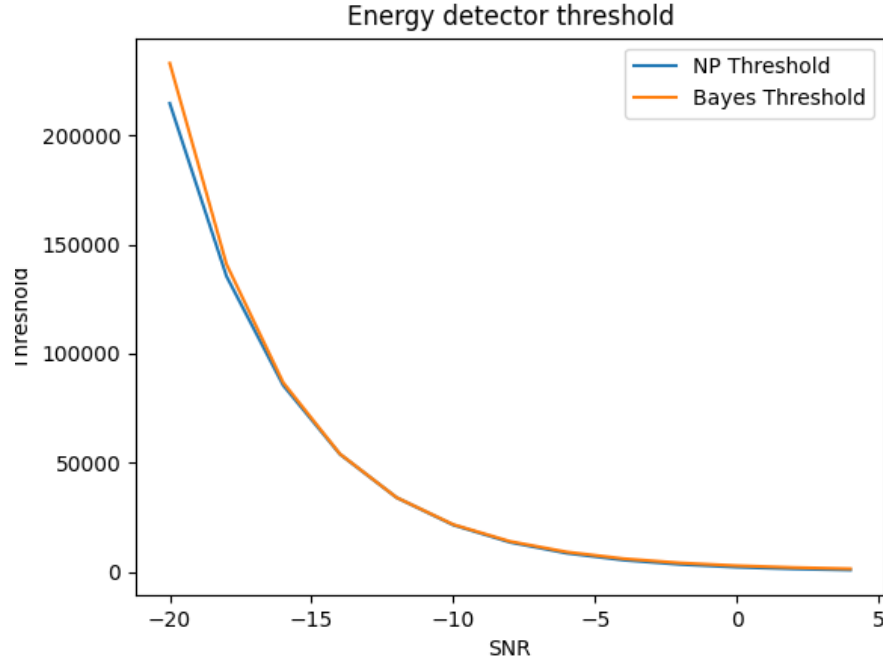


Figure 3: Threshold

Figure(3) shows the threshold variation with respect to SNR for Neyman-Pearson detector and Bayes detector. We can observe that the NP threshold is slightly lesser than Bayes threshold for lower values of SNR. As the value of SNR increases, the noise component in the observed signal decreases, and hence the variance of observed signal decreases. As a result, the detector threshold decreases.

PART B: Cyclostationary Detector

Subpart A

From the plots Figure(4) and Figure(5), we can verify that the means and variances match with the same of that of the complex Gaussian distribution we derived. Figure(4) represents the detector performance with 1000 statistics and Figure(5) represents the detector performance with 100000 statistics. We can see that with increased number of test statistics, we get the ideal probability distribution.

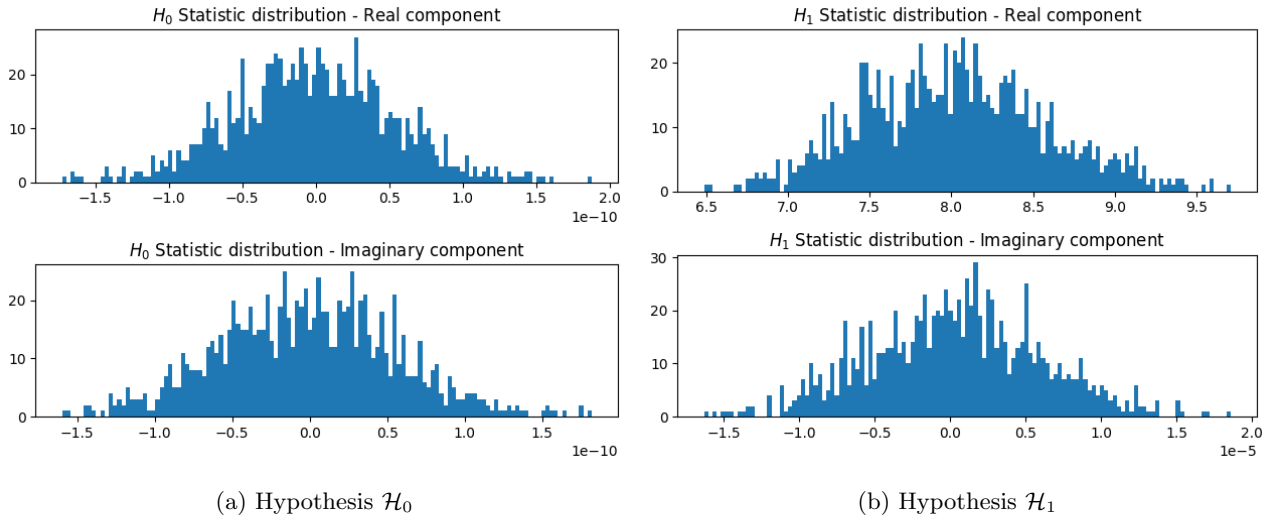


Figure 4: Distribution for 1000 statistics

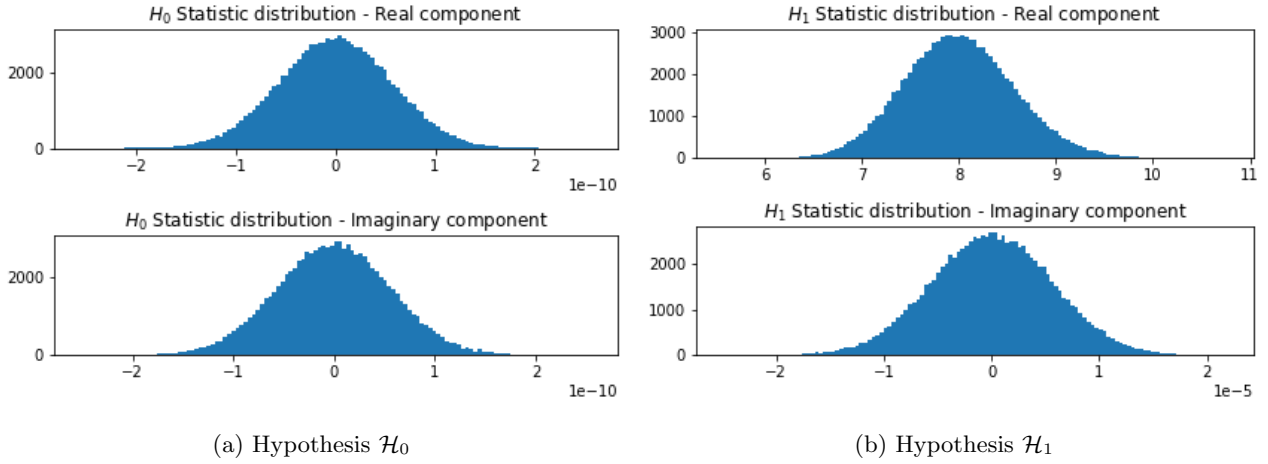
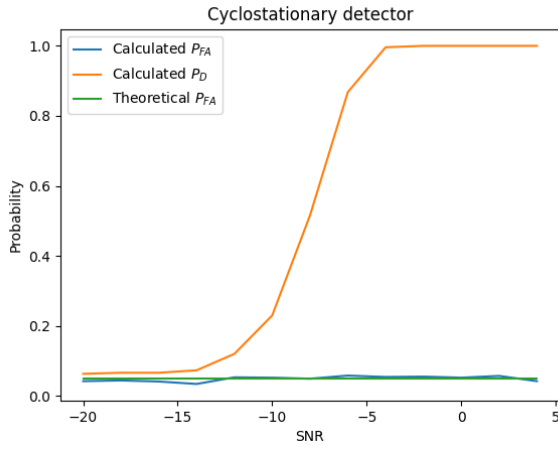


Figure 5: Distribution for 100000 statistics

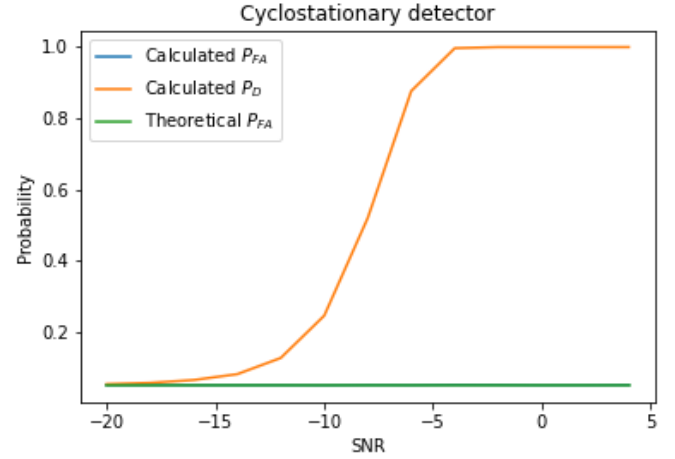
Subpart B

From the plot Figure(6), we can easily observe that the calculated P_d increases with the SNR value. We can also verify that P_{FA} is under 0.05. Figure(6a) represents the detector performance with 1000 statistics and Figure(6b) represents the detector performance with 100000 statistics. We can see that with increased number of test statistics, we get the ideal detector performance.

Comparing energy detector performance with cyclostationary detector performance, we can observe that energy detector gives much higher values of P_D for the same SNR value. Also, energy detector achieves higher values of P_D for lower SNR value. For example, energy detector achieves P_D values close to 1 around -10dB whereas cyclostationary detector achieves it around -5dB. For lower values of SNR, energy detector clearly outperforms cyclostationary detector by a huge margin.



(a) For 1000 records

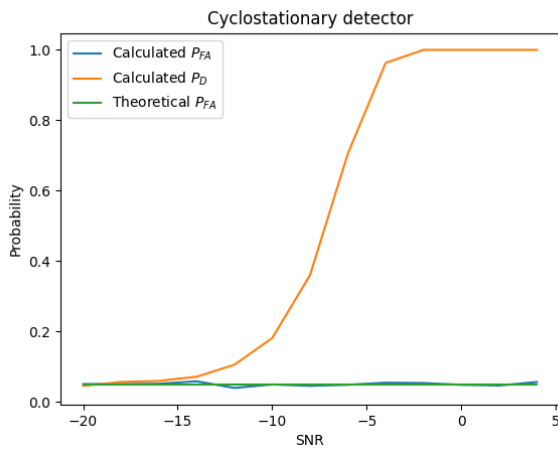


(b) For 100000 records

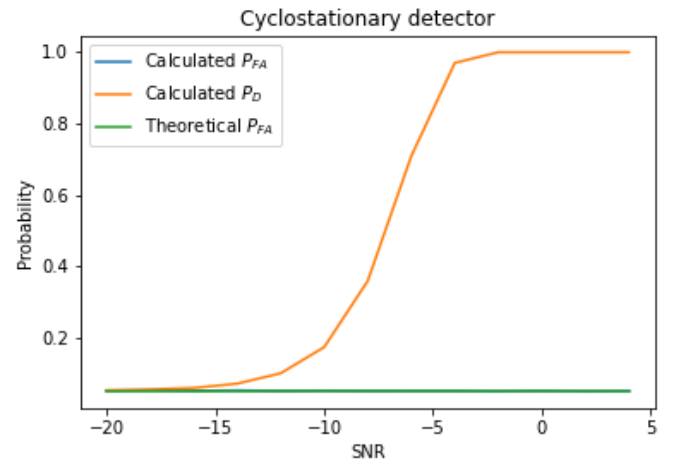
Figure 6: Probability vs SNR

Subpart C

From the plot Figure(7), we can observe a clear drop in calculated P_d compared to the previous case. However the performance drop is negligible for higher values of SNR — owing to the smaller noise component. We can also observe that the drop in performance for cyclostationary detector is lesser compared to same case with energy detector and hence it is less prone to noisy estimates. Figure(7a) represents the detector performance with 1000 statistics and Figure(7b) represents the detector performance with 100000 statistics. We can see that with increased number of test statistics, we get the ideal detector performance.



(a) For 1000 records



(b) For 100000 records

Figure 7: Probability vs SNR

PART C: Appendices (Python codes)

(Solution) The entire program is written in six python files. The codes are provided below and are also available at <https://github.com/vineeths96/Spectrum-sensing-for-cognitive-radio> GitHub repository. (Repository access is private as of now. Access can be made available, if necessary).

Problem 1A

```

1  import numpy as np
2  from scipy.stats import chi2
3  import matplotlib.pyplot as plt
4  from energy_detector.parameters import *
5
6
7  def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
8      """
9      Generate H0 test statistics
10     :param NUM_STATISTICS: Number of statistics to be produced
11     :param sigma_w: Std deviation of noise
12     :param N: Length of observation vector
13     :return: H0 test statistics
14     """
15
16     T_y = np.zeros(NUM_STATISTICS)
17
18     for ind in range(NUM_STATISTICS):
19         w = sigma_w * np.random.randn(N, 2).view(np.complex128)
20
21         y = w
22
23         # Calculate test statistic
24         T_y[ind] = np.sum(np.square(np.abs(y)))
25
26     return T_y
27
28
29  def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
30      """
31      Generate H1 test statistics
32     :param NUM_STATISTICS: Number of statistics to be produced
33     :param sigma_w: Std deviation of noise
34     :param N: Length of observation vector
35     :return: H1 test statistics
36     """
37
38     T_y = np.zeros(NUM_STATISTICS)
39
40     for ind in range(NUM_STATISTICS):
41         x = sigma_s * np.random.randn(N, 1)
42         w = sigma_w * np.random.randn(N, 2).view(np.complex128)
43
44         y = x + w
45

```

```

46         # Calculate test statistic
47         T_y[ind] = np.sum(np.square(np.abs(y)))
48
49     return T_y
50
51
52 def main():
53     # Define the SNR where performance has to be evaluated
54     SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
55     N = (K + 1) * (N_c + N_d)
56
57     P_FA_THEO = np.zeros(len(SNR_list))
58     P_D_THEO = np.zeros(len(SNR_list))
59
60     P_FA_CALC = np.zeros(len(SNR_list))
61     P_D_CALC = np.zeros(len(SNR_list))
62
63     # Evaluate the performance on H0 and H1
64     for ind, SNR in enumerate(SNR_list):
65         NUM_FALSE_ALARM = 0
66         NUM_DETECTION = 0
67
68         sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
69         gamma = chi2.isf(q=P_FA, df=N) * (sigma_w ** 2)
70
71         T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
72         T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)
73
74         for T in T_y_0:
75             if T >= gamma:
76                 NUM_FALSE_ALARM += 1
77
78         for T in T_y_1:
79             if T >= gamma:
80                 NUM_DETECTION += 1
81
82         P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
83         P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS
84
85         P_FA_THEO[ind] = P_FA
86         P_D_THEO[ind] = chi2.sf(x=gamma / (sigma_s ** 2 + sigma_w ** 2), df=N)
87
88     # Plot and save the results
89     plt.figure()
90     plt.plot(SNR_list, P_FA_CALC, label="Calculated  $P_{FA}$ ")
91     plt.plot(SNR_list, P_D_CALC, label="Calculated  $P_D$ ")
92     plt.plot(SNR_list, P_FA_THEO, label="Theoretical  $P_{FA}$ ")
93     plt.plot(SNR_list, P_D_THEO, label="Theoretical  $P_D$ ")
94     plt.xlabel("SNR")
95     plt.ylabel("Probability")
96     plt.title("Energy detector")
97     plt.legend()
98     plt.savefig('./results/energy_detector_a.png')
99     plt.show()
100
101
102 if __name__ == '__main__':
103     main()

```

Problem 1B

```

1  import numpy as np
2  from scipy.stats import chi2
3  import matplotlib.pyplot as plt
4  from energy_detector.parameters import *
5
6
7  def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
8      """
9      Generate H0 test statistics
10     :param NUM_STATISTICS: Number of statistics to be produced
11     :param sigma_w: Std deviation of noise
12     :param N: Length of observation vector
13     :return: H0 test statistics
14     """
15
16     T_y = np.zeros(NUM_STATISTICS)
17
18     for ind in range(NUM_STATISTICS):
19         w = sigma_w * np.random.randn(N, 2).view(np.complex128)
20
21         y = w
22
23         # Calculate test statistic
24         T_y[ind] = np.sum(np.square(np.abs(y)))
25
26     return T_y
27
28
29  def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
30      """
31      Generate H1 test statistics
32      :param NUM_STATISTICS: Number of statistics to be produced
33      :param sigma_w: Std deviation of noise
34      :param N: Length of observation vector
35      :return: H1 test statistics
36      """
37
38     T_y = np.zeros(NUM_STATISTICS)
39
40     for ind in range(NUM_STATISTICS):
41         x = sigma_s * np.random.randn(N, 1)
42         w = sigma_w * np.random.randn(N, 2).view(np.complex128)
43
44         y = x + w
45
46         # Calculate test statistic
47         T_y[ind] = np.sum(np.square(np.abs(y)))
48
49     return T_y
50
51
52  def main():
53     # Define the SNR where performance has to be evaluated
54     SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
55     N = (K + 1) * (N_c + N_d)
56
57     P_FA_THEO = np.zeros(len(SNR_list))

```

```

58     P_D_THEO = np.zeros(len(SNR_list))
59
60     P_FA_CALC = np.zeros(len(SNR_list))
61     P_D_CALC = np.zeros(len(SNR_list))
62
63     # Evaluate the performance on H0 and H1
64     for ind, SNR in enumerate(SNR_list):
65         NUM_FALSE_ALARM = 0
66         NUM_DETECTION = 0
67
68         sigma_w = sigma_s ** 2 / 10 ** (SNR / 10)
69         sigma_w = np.sqrt(sigma_w * 10 ** (SNR_NOISE / 10))
70         gamma = chi2.isf(q=P_FA, df=N) * (sigma_w ** 2)
71
72         T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
73         T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)
74
75         for T in T_y_0:
76             if T >= gamma:
77                 NUM_FALSE_ALARM += 1
78
79         for T in T_y_1:
80             if T >= gamma:
81                 NUM_DETECTION += 1
82
83         P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
84         P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS
85
86         P_FA_THEO[ind] = P_FA
87         P_D_THEO[ind] = chi2.sf(x=gamma / (sigma_s ** 2 + sigma_w ** 2), df=N)
88
89     # Plot and save the results
90     plt.figure()
91     plt.plot(SNR_list, P_FA_CALC, label="Calculated $P_{FA}$")
92     plt.plot(SNR_list, P_D_CALC, label="Calculated $P_{D}$")
93     plt.plot(SNR_list, P_FA_THEO, label="Theoretical $P_{FA}$")
94     plt.plot(SNR_list, P_D_THEO, label="Theoretical $P_{D}$")
95     plt.xlabel("SNR")
96     plt.ylabel("Probability")
97     plt.title("Energy detector")
98     plt.legend()
99     plt.savefig('./results/energy_detector_b.png')
100    plt.show()
101
102
103    if __name__ == '__main__':
104        main()

```

Problem 1C

```

1  import numpy as np
2  from scipy.stats import chi2
3  import matplotlib.pyplot as plt
4  from energy_detector.parameters import *
5
6
7  def main():

```



```

8     # Define the SNR where performance has to be evaluated
9     SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
10    N = (K + 1) * (N_c + N_d)
11
12    GAMMA_NP = np.zeros(len(SNR_list))
13    GAMMA_BD = np.zeros(len(SNR_list))
14
15    for ind, SNR in enumerate(SNR_list):
16        sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
17        gamma_NP = chi2.isf(q=P_FA, df=N) * (sigma_w ** 2)
18        gamma_BD = 2 * (sigma_s ** 2 + sigma_w ** 2) * sigma_w ** 2 / sigma_s ** 2 * (
19            N / 2 * np.log(1 + sigma_s ** 2 / sigma_w ** 2) + np.log(1 - P_H1) - np.log(P_H1))
20
21        GAMMA_NP[ind] = gamma_NP
22        GAMMA_BD[ind] = gamma_BD
23
24    # Plot and save the results
25    plt.figure()
26    plt.plot(SNR_list, GAMMA_NP, label="NP Threshold")
27    plt.plot(SNR_list, GAMMA_BD, label="Bayes Threshold")
28    plt.xlabel("SNR")
29    plt.ylabel("Threshold")
30    plt.title("Energy detector threshold")
31    plt.legend()
32    plt.savefig('./results/energy_detector_c.png')
33    plt.show()
34
35
36 if __name__ == '__main__':
37     main()

```

Problem 1 Parameters

```

1  sigma_s = 1
2  N_d = 32
3  N_c = 8
4  K = 50
5  P_FA = 0.05
6
7  SNR = 10
8  SNR_LOW = -20
9  SNR_UP = 6
10 SNR_STEP = 2
11 SNR_NOISE = 1
12
13 P_H1 = 0.2
14
15 NUM_STATISTICS = 1000
16 #NUM_STATISTICS = 100000

```

Problem 2A

```

1  import numpy as np

```

```

2 import matplotlib.pyplot as plt
3 from cyclostationary_detector.parameters import *
4
5
6 def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
7     """
8     Generate H0 test statistics
9     :param NUM_STATISTICS: Number of statistics to be produced
10    :param sigma_w: Std deviation of noise
11    :param N: Length of observation vector
12    :return: H0 test statistics
13    """
14
15    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)
16
17    for ind in range(NUM_STATISTICS):
18        w = sigma_w * np.random.randn(N, 2).view(np.complex128)
19
20        y = w
21
22        # Calculate test statistic
23        val = np.complex(0)
24        for n in range(N_c):
25            for k in range(K):
26                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])
27
28        T_y[ind] = 1 / K * val
29
30    return T_y
31
32
33 def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
34     """
35     Generate H1 test statistics
36     :param NUM_STATISTICS: Number of statistics to be produced
37     :param sigma_w: Std deviation of noise
38     :param N: Length of observation vector
39     :return: H1 test statistics
40     """
41
42    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)
43
44    for ind in range(NUM_STATISTICS):
45        x = sigma_s * np.random.randn(N, 1)
46
47        for k in range(K):
48            x[k * (N_c + N_d): k * (N_c + N_d) + N_c] = x[k * (N_c + N_d) + N_d: (k + 1) * (N_c + N_d)]
49        w = sigma_w * np.random.randn(N, 2).view(np.complex128)
50
51        y = x + w
52
53        # Calculate test statistic
54        val = np.complex(0)
55        for n in range(N_c):
56            for k in range(K):
57                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])
58
59        T_y[ind] = 1 / K * val
60
61    return T_y
62

```

```

63
64 def main():
65     N = (K + 1) * (N_c + N_d)
66     sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
67
68     T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
69     T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)
70
71     # Plot and save the results
72     plt.figure()
73     plt.subplot(211)
74     plt.hist(np.real(T_y_0), bins=125)
75     plt.title("$H_{0}$ Statistic distribution - Real component")
76     plt.subplot(212)
77     plt.hist(np.imag(T_y_0), bins=125)
78     plt.title("$H_{0}$ Statistic distribution - Imaginary component")
79     plt.tight_layout()
80     plt.savefig('./results/cyclostationary_detector_a_H0.png')
81     plt.show()
82
83     plt.figure()
84     plt.subplot(211)
85     plt.hist(np.real(T_y_1), bins=125)
86     plt.title("$H_{1}$ Statistic distribution - Real component")
87     plt.subplot(212)
88     plt.hist(np.imag(T_y_1), bins=125)
89     plt.title("$H_{1}$ Statistic distribution - Imaginary component")
90     plt.tight_layout()
91     plt.savefig('./results/cyclostationary_detector_a_H1.png')
92     plt.show()
93
94
95 if __name__ == '__main__':
96     main()

```

Problem 2B

```

1 import numpy as np
2 from scipy.stats import chi2
3 import matplotlib.pyplot as plt
4 from cyclostationary_detector.parameters import *
5
6
7 def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
8     """
9     Generate H0 test statistics
10    :param NUM_STATISTICS: Number of statistics to be produced
11    :param sigma_w: Std deviation of noise
12    :param N: Length of observation vector
13    :return: H0 test statistics
14    """
15
16    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)
17
18    for ind in range(NUM_STATISTICS):
19        w = sigma_w * np.random.randn(N, 2).view(np.complex128)
20

```

```

21     y = w
22
23     # Calculate test statistic
24     val = np.complex(0)
25     for n in range(N_c):
26         for k in range(K):
27             val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])
28
29     T_y[ind] = 1 / K * val
30
31     return T_y
32
33
34 def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
35     """
36     Generate H1 test statistics
37     :param NUM_STATISTICS: Number of statistics to be produced
38     :param sigma_w: Std deviation of noise
39     :param N: Length of observation vector
40     :return: H1 test statistics
41     """
42
43     T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)
44
45     for ind in range(NUM_STATISTICS):
46         x = sigma_s * np.random.randn(N, 1)
47         for k in range(K):
48             x[k * (N_c + N_d): k * (N_c + N_d) + N_c] = x[k * (N_c + N_d) + N_d: (k + 1) * (N_c + N_d)]
49
50         w = sigma_w * np.random.randn(N, 2).view(np.complex128)
51
52         y = x + w
53
54         # Calculate test statistic
55         val = np.complex(0)
56         for n in range(N_c):
57             for k in range(K):
58                 val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])
59
60         T_y[ind] = 1 / K * val
61
62     return T_y
63
64
65 def main():
66     # Define the SNR where performance has to be evaluated
67     SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
68     N = (K + 1) * (N_c + N_d)
69
70     P_FA_THEO = np.zeros(len(SNR_list))
71
72     P_FA_CALC = np.zeros(len(SNR_list))
73     P_D_CALC = np.zeros(len(SNR_list))
74
75     # Evaluate the performance on H0 and H1
76     for ind, SNR in enumerate(SNR_list):
77         NUM_FALSE_ALARM = 0
78         NUM_DETECTION = 0
79
80         sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
81         gamma = chi2.isf(q=P_FA, df=2) * N_c / K * (sigma_w ** 4)

```

```

82
83     T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
84     T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)
85
86     for T in T_y_0:
87         if np.square(np.abs(T)) >= gamma:
88             NUM_FALSE_ALARM += 1
89
90     for T in T_y_1:
91         if np.square(np.abs(T)) >= gamma:
92             NUM_DETECTION += 1
93
94     P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
95     P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS
96
97     P_FA_THEO[ind] = P_FA
98
99     # Plot and save the results
100    plt.figure()
101    plt.plot(SNR_list, P_FA_CALC, label="Calculated  $P_{FA}$ ")
102    plt.plot(SNR_list, P_D_CALC, label="Calculated  $P_D$ ")
103    plt.plot(SNR_list, P_FA_THEO, label="Theoretical  $P_{FA}$ ")
104    plt.xlabel("SNR")
105    plt.ylabel("Probability")
106    plt.title("Cyclostationary detector")
107    plt.legend()
108    plt.savefig('./results/cyclostationary_detector_b.png')
109    plt.show()
110
111
112 if __name__ == '__main__':
113     main()

```

Problem 2C

```

1  import numpy as np
2  from scipy.stats import chi2
3  import matplotlib.pyplot as plt
4  from cyclostationary_detector.parameters import *
5
6
7  def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
8      """
9      Generate H0 test statistics
10     :param NUM_STATISTICS: Number of statistics to be produced
11     :param sigma_w: Std deviation of noise
12     :param N: Length of observation vector
13     :return: H0 test statistics
14     """
15
16     T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)
17
18     for ind in range(NUM_STATISTICS):
19         w = sigma_w * np.random.randn(N, 2).view(np.complex128)
20
21         y = w
22

```

```

23     # Calculate test statistic
24     val = np.complex(0)
25     for n in range(N_c):
26         for k in range(K):
27             val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])
28
29     T_y[ind] = 1 / K * val
30
31     return T_y
32
33
34 def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
35     """
36     Generate H1 test statistics
37     :param NUM_STATISTICS: Number of statistics to be produced
38     :param sigma_w: Std deviation of noise
39     :param N: Length of observation vector
40     :return: H1 test statistics
41     """
42
43     T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)
44
45     for ind in range(NUM_STATISTICS):
46         x = sigma_s * np.random.randn(N, 1)
47         for k in range(K):
48             x[k * (N_c + N_d): k * (N_c + N_d) + N_c] = x[k * (N_c + N_d) + N_d: (k + 1) * (N_c + N_d)]
49
50         w = sigma_w * np.random.randn(N, 2).view(np.complex128)
51
52         y = x + w
53
54         # Calculate test statistic
55         val = np.complex(0)
56         for n in range(N_c):
57             for k in range(K):
58                 val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])
59
60         T_y[ind] = 1 / K * val
61
62     return T_y
63
64
65 def main():
66     # Define the SNR where performance has to be evaluated
67     SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
68     N = (K + 1) * (N_c + N_d)
69
70     P_FA_THEO = np.zeros(len(SNR_list))
71
72     P_FA_CALC = np.zeros(len(SNR_list))
73     P_D_CALC = np.zeros(len(SNR_list))
74
75     # Evaluate the performance on H0 and H1
76     for ind, SNR in enumerate(SNR_list):
77         NUM_FALSE_ALARM = 0
78         NUM_DETECTION = 0
79
80         sigma_w = sigma_s ** 2 / 10 ** (SNR / 10)
81         sigma_w = np.sqrt(sigma_w * 10 ** (SNR_NOISE / 10))
82         gamma = chi2.isf(q=P_FA, df=2) * N_c / K * (sigma_w ** 4)
83

```

```

84     T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
85     T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)
86
87     for T in T_y_0:
88         if np.square(np.abs(T)) >= gamma:
89             NUM_FALSE_ALARM += 1
90
91     for T in T_y_1:
92         if np.square(np.abs(T)) >= gamma:
93             NUM_DETECTION += 1
94
95     P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
96     P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS
97
98     P_FA_THEO[ind] = P_FA
99
100     # Plot and save the results
101     plt.figure()
102     plt.plot(SNR_list, P_FA_CALC, label="Calculated  $P_{FA}$ ")
103     plt.plot(SNR_list, P_D_CALC, label="Calculated  $P_D$ ")
104     plt.plot(SNR_list, P_FA_THEO, label="Theoretical  $P_{FA}$ ")
105     plt.xlabel("SNR")
106     plt.ylabel("Probability")
107     plt.title("Cyclostationary detector")
108     plt.legend()
109     plt.savefig('./results/cyclostationary_detector_c.png')
110     plt.show()
111
112
113 if __name__ == '__main__':
114     main()

```

Problem 2 Parameters

```

1  sigma_s = 1
2  N_d = 32
3  N_c = 8
4  K = 50
5  P_FA = 0.05
6
7  SNR = 100
8  SNR_LOW = -20
9  SNR_UP = 6
10 SNR_STEP = 2
11 SNR_NOISE = 1
12
13 NUM_STATISTICS = 1000
14 #NUM_STATISTICS = 100000

```
