E1 244: Detection and Estimation Theory **(Due: 22/06/20)**

Assignment #3

Instructor: Sundeep Chepuri *Name:* Vineeth S, *SR No.:* 16543

Consider a data symbol sequence $\mathbf{s} = [s[0], s[1], ..., s[N_d - 1]]^T$ 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 $\mathbf{w} \sim \mathcal{CN}(0, \sigma_n^2 \mathbf{I})$, 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.

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

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

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

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

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

$$\sum_{n=0}^{N-1} |y[n]|^2 > \gamma''$$

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^2_N distribution.

$$\mathcal{H}_0 : \frac{T(y)}{\sigma_w^2} \sim \chi^2_N$$
$$\mathcal{H}_1 : \frac{T(y)}{\sigma_s^2 + \sigma_w^2} \sim \chi^2_N$$

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

$$P_{FA} = P(T(y) > \gamma"; \mathcal{H}_0)$$
$$P_{FA} = Q_{\chi^2_N}\left(\frac{\gamma"}{\sigma_w^2}\right)$$
$$\gamma" = \sigma_w^2 * Q^{-1}_{\chi^2_N}(P_{FA})$$

And the theoretical detection probability P_D as,

$$P_D = P(T(y) > \gamma"; \mathcal{H}_1)$$
$$= Q_{\chi^2_N}\left(\frac{\gamma"}{\sigma_s^2 + \sigma_w^2}\right)$$

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,

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

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 $n = 0, .., N_c - 1$, 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 $n = 0, .., N_c - 1$, let

$$y[n_1+k_1N] = x_1 + w_{11} \qquad y[n_2+k_2N] = x_2 + w_{21}$$

$$y[n_1+k_1N+N_d] = x_1 + w_{12} \qquad y[n_2+k_2N+N_d] = x_2 + w_{22}$$

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 $|x_1| = \sqrt{Re(x_1)^2 + Im(x_1)^2}$ 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,

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

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 $n=0,..,N_c-1$, 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,

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

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

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

For $n = 0, .., N_c - 1$, let

$$\begin{aligned} y[n_1 + k_1 N] &= w_{11} & y[n_2 + k_2 N] &= w_{21} \\ y[n_1 + k_1 N + N_d] &= w_{12} & y[n_2 + k_2 N + N_d] &= w_{22} \end{aligned}$$

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,

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

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

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

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

Similarly,

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

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,

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

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

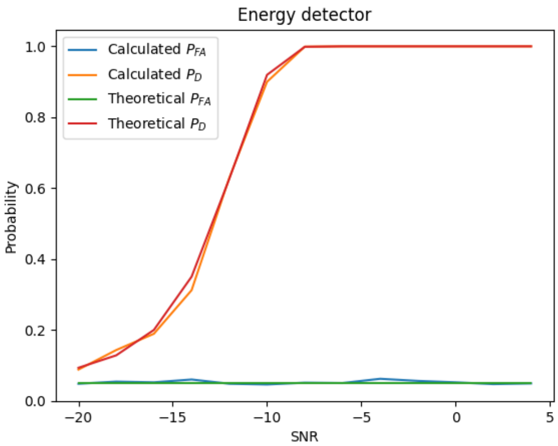
We decide on $\mathcal{H}_1$ if $|T(y)| > \gamma$. Or equivalently if $|T(y)|^2 > \gamma^2$.

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

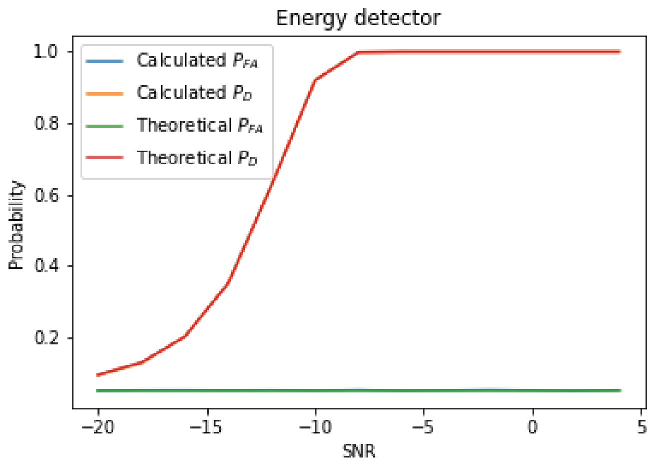
$$P_{FA} = P(|T(y)| > \gamma; \mathcal{H}_0)$$
$$P_{FA} = P(|T(y)|^2 > \gamma^2; \mathcal{H}_0)$$
$$P_{FA} = P(\bar{T}^2(y) + \tilde{T}^2(y) > \gamma^2; \mathcal{H}_0)$$
$$P_{FA} = Q_{\chi_2^2}\left(\frac{\gamma^2}{\frac{N_c}{2K}\sigma_w^4}\right)$$
$$\gamma^2 = \frac{N_c}{2K}\sigma_w^4 * Q_{\chi_2^2}^{-1}(P_{FA})$$

PART B: Energy Detector

Subpart A



(a) For 1000 records

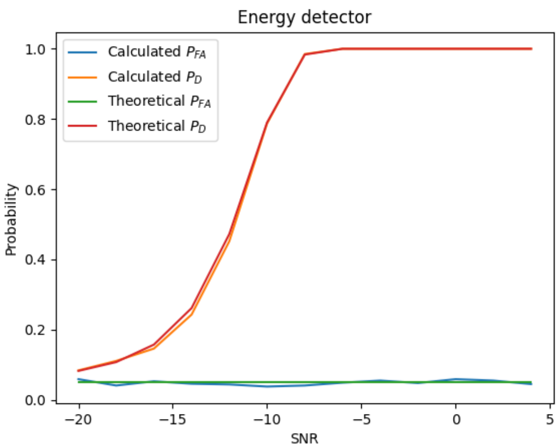


(b) For 100000 records

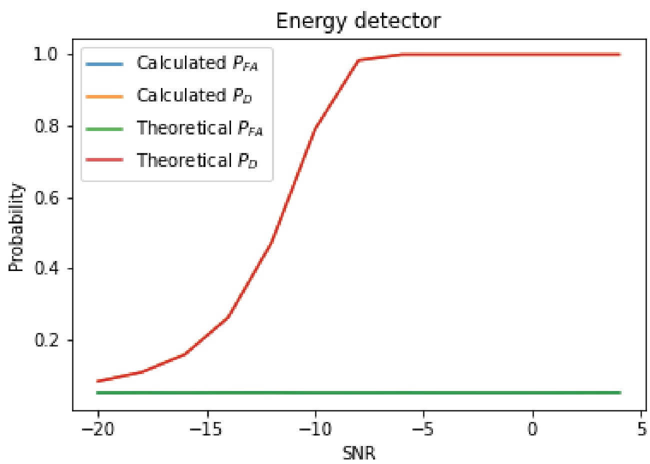
Figure 1: Probability vs SNR

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

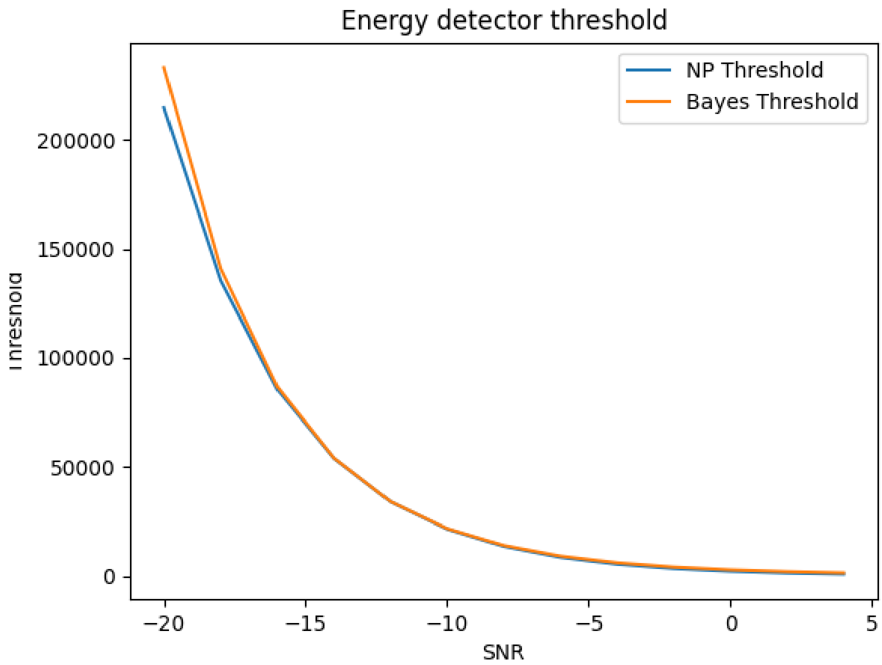


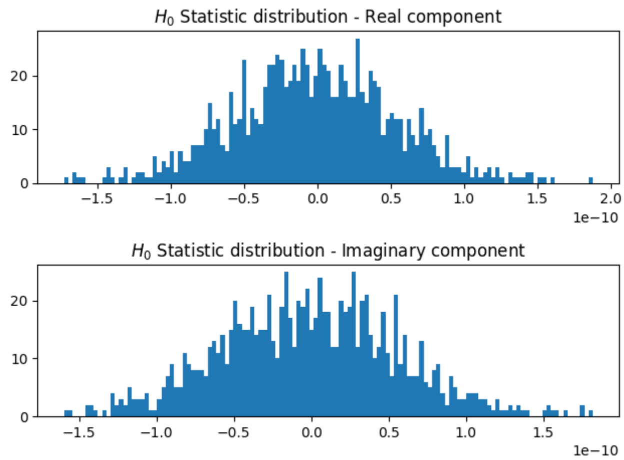
Figure 3: Threshold

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.

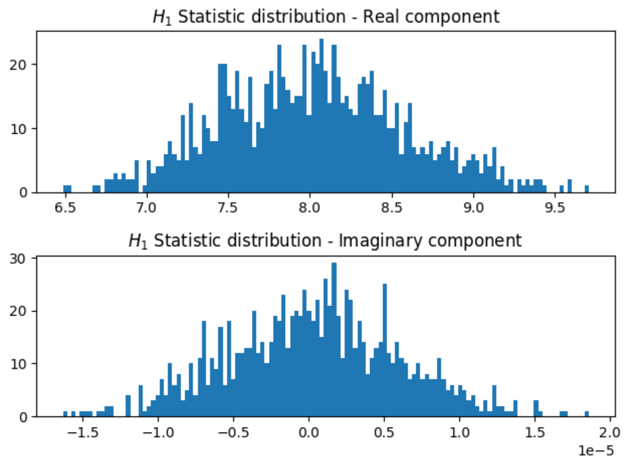
PART B: Cyclostationary Detector

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.

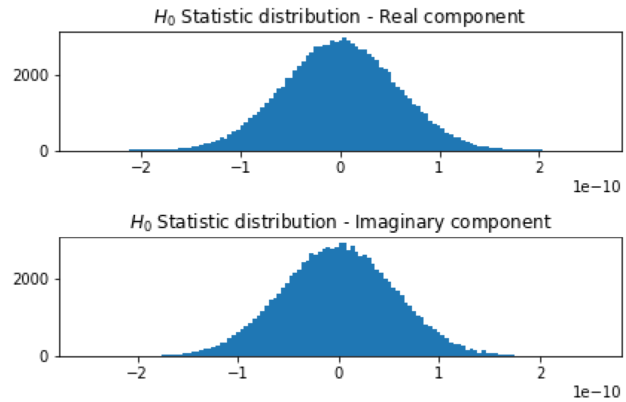


(a) Hypothesis $\mathcal{H}_0$

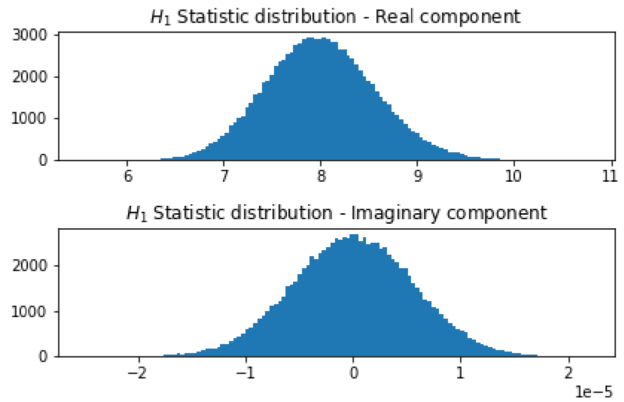


(b) Hypothesis $\mathcal{H}_1$

Figure 4: Distribution for 1000 statistics



(a) Hypothesis $\mathcal{H}_0$



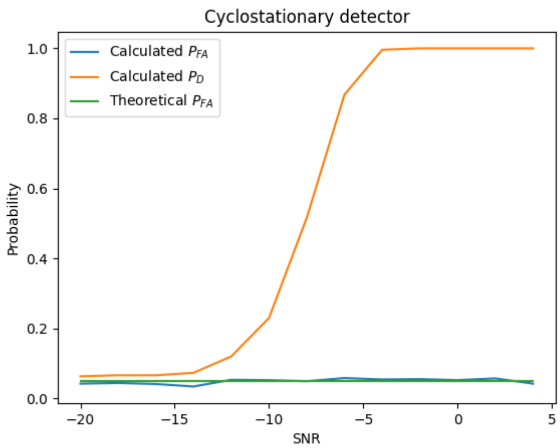
(b) Hypothesis $\mathcal{H}_1$

Figure 5: Distribution for 100000 statistics

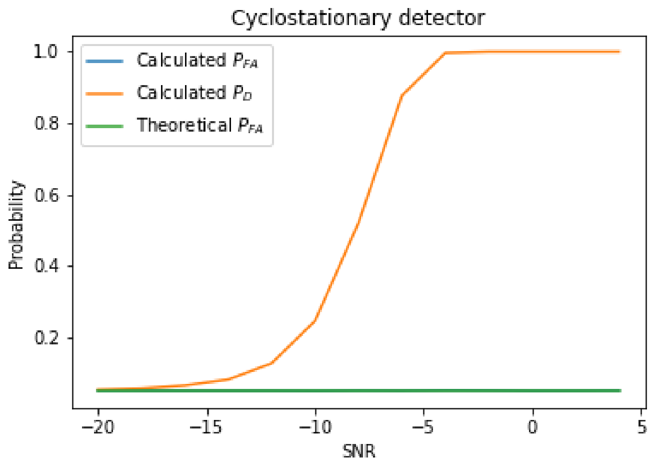
Subpart B

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

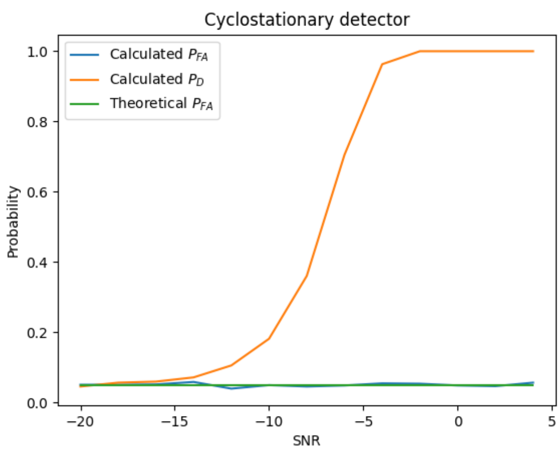


(b) For 100000 records

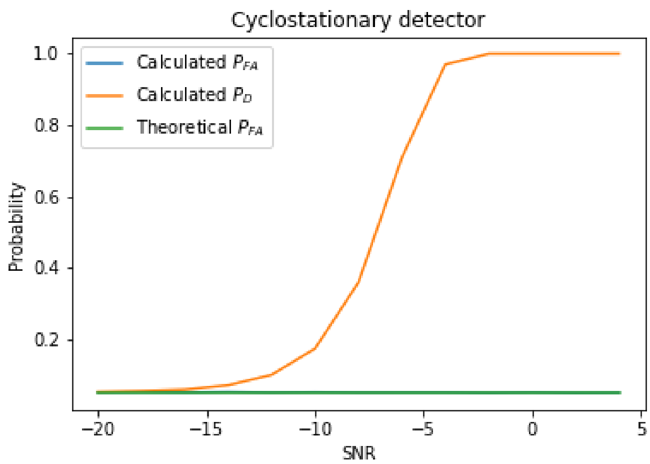
Figure 6: Probability vs SNR

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 me made available, if necessary).

Problem 1A

```python
import numpy as np
from scipy.stats import chi2
import matplotlib.pyplot as plt
from energy_detector.parameters import *


def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
    """
    Generate H0 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H0 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS)

    for ind in range(NUM_STATISTICS):
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = w

        # Calculate test statistic
        T_y[ind] = np.sum(np.square(np.abs(y)))

    return T_y


def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
    """
    Generate H1 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H1 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS)

    for ind in range(NUM_STATISTICS):
        x = sigma_s * np.random.randn(N, 1)
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = x + w

```

```python
        # Calculate test statistic
        T_y[ind] = np.sum(np.square(np.abs(y)))

    return T_y


def main():
    # Define the SNR where performance has to be evaluated
    SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
    N = (K + 1) * (N_c + N_d)

    P_FA_THEO = np.zeros(len(SNR_list))
    P_D_THEO = np.zeros(len(SNR_list))

    P_FA_CALC = np.zeros(len(SNR_list))
    P_D_CALC = np.zeros(len(SNR_list))

    # Evaluate the performance on H0 and H1
    for ind, SNR in enumerate(SNR_list):
        NUM_FALSE_ALARM = 0
        NUM_DETECTION = 0

        sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
        gamma = chi2.isf(q=P_FA, df=N) * (sigma_w ** 2)

        T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
        T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)

        for T in T_y_0:
            if T >= gamma:
                NUM_FALSE_ALARM += 1

        for T in T_y_1:
            if T >= gamma:
                NUM_DETECTION += 1

        P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
        P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS

        P_FA_THEO[ind] = P_FA
        P_D_THEO[ind] = chi2.sf(x=gamma / (sigma_s ** 2 + sigma_w ** 2), df=N)

    # Plot and save the results
    plt.figure()
    plt.plot(SNR_list, P_FA_CALC, label="Calculated $P_{FA}$")
    plt.plot(SNR_list, P_D_CALC, label="Calculated $P_{D}$")
    plt.plot(SNR_list, P_FA_THEO, label="Theoretical $P_{FA}$")
    plt.plot(SNR_list, P_D_THEO, label="Theoretical $P_{D}$")
    plt.xlabel("SNR")
    plt.ylabel("Probability")
    plt.title("Energy detector")
    plt.legend()
    plt.savefig('./results/energy_detector_a.png')
    plt.show()


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

Problem 1B

```python
import numpy as np
from scipy.stats import chi2
import matplotlib.pyplot as plt
from energy_detector.parameters import *


def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
    """
    Generate H0 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H0 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS)

    for ind in range(NUM_STATISTICS):
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = w

        # Calculate test statistic
        T_y[ind] = np.sum(np.square(np.abs(y)))

    return T_y


def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
    """
    Generate H1 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H1 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS)

    for ind in range(NUM_STATISTICS):
        x = sigma_s * np.random.randn(N, 1)
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = x + w

        # Calculate test statistic
        T_y[ind] = np.sum(np.square(np.abs(y)))

    return T_y


def main():
    # Define the SNR where performance has to be evaluated
    SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
    N = (K + 1) * (N_c + N_d)

    P_FA_THEO = np.zeros(len(SNR_list))
```

```python
    P_D_THEO = np.zeros(len(SNR_list))

    P_FA_CALC = np.zeros(len(SNR_list))
    P_D_CALC = np.zeros(len(SNR_list))

    # Evaluate the performance on H0 and H1
    for ind, SNR in enumerate(SNR_list):
        NUM_FALSE_ALARM = 0
        NUM_DETECTION = 0

        sigma_w = sigma_s ** 2 / 10 ** (SNR / 10)
        sigma_w = np.sqrt(sigma_w * 10 ** (SNR_NOISE / 10))
        gamma = chi2.isf(q=P_FA, df=N) * (sigma_w ** 2)

        T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
        T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)

        for T in T_y_0:
            if T >= gamma:
                NUM_FALSE_ALARM += 1

        for T in T_y_1:
            if T >= gamma:
                NUM_DETECTION += 1

        P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
        P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS

        P_FA_THEO[ind] = P_FA
        P_D_THEO[ind] = chi2.sf(x=gamma / (sigma_s ** 2 + sigma_w ** 2), df=N)

    # Plot and save the results
    plt.figure()
    plt.plot(SNR_list, P_FA_CALC, label="Calculated $P_{FA}$")
    plt.plot(SNR_list, P_D_CALC, label="Calculated $P_{D}$")
    plt.plot(SNR_list, P_FA_THEO, label="Theoretical $P_{FA}$")
    plt.plot(SNR_list, P_D_THEO, label="Theoretical $P_{D}$")
    plt.xlabel("SNR")
    plt.ylabel("Probability")
    plt.title("Energy detector")
    plt.legend()
    plt.savefig('./results/energy_detector_b.png')
    plt.show()


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

Problem 1C

```python
import numpy as np
from scipy.stats import chi2
import matplotlib.pyplot as plt
from energy_detector.parameters import *


def main():
```

```python
    # Define the SNR where performance has to be evaluated
    SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
    N = (K + 1) * (N_c + N_d)

    GAMMA_NP = np.zeros(len(SNR_list))
    GAMMA_BD = np.zeros(len(SNR_list))

    for ind, SNR in enumerate(SNR_list):
        sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
        gamma_NP = chi2.isf(q=P_FA, df=N) * (sigma_w ** 2)
        gamma_BD = 2 * (sigma_s ** 2 + sigma_w ** 2) * sigma_w ** 2 / sigma_s ** 2 * (
                    N / 2 * np.log(1 + sigma_s ** 2 / sigma_w ** 2) + np.log(1 - P_H1) - np.log(P_H1))

        GAMMA_NP[ind] = gamma_NP
        GAMMA_BD[ind] = gamma_BD

    # Plot and save the results
    plt.figure()
    plt.plot(SNR_list, GAMMA_NP, label="NP Threshold")
    plt.plot(SNR_list, GAMMA_BD, label="Bayes Threshold")
    plt.xlabel("SNR")
    plt.ylabel("Threshold")
    plt.title("Energy detector threshold")
    plt.legend()
    plt.savefig('./results/energy_detector_c.png')
    plt.show()


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

Problem 1 Parameters

```python
sigma_s = 1
N_d = 32
N_c = 8
K = 50
P_FA = 0.05

SNR = 10
SNR_LOW = -20
SNR_UP = 6
SNR_STEP = 2
SNR_NOISE = 1

P_H1 = 0.2

NUM_STATISTICS = 1000
#NUM_STATISTICS = 100000
```

Problem 2A

```python
import numpy as np
```

```
import matplotlib.pyplot as plt
from cyclostationary_detector.parameters import *


def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
    """
    Generate H0 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H0 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)

    for ind in range(NUM_STATISTICS):
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = w

        # Calculate test statistic
        val = np.complex(0)
        for n in range(N_c):
            for k in range(K):
                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])

        T_y[ind] = 1 / K * val

    return T_y


def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
    """
    Generate H1 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H1 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)

    for ind in range(NUM_STATISTICS):
        x = sigma_s * np.random.randn(N, 1)

        for k in range(K):
            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)]
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = x + w

        # Calculate test statistic
        val = np.complex(0)
        for n in range(N_c):
            for k in range(K):
                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])

        T_y[ind] = 1 / K * val

    return T_y
```

```python

def main():
    N = (K + 1) * (N_c + N_d)
    sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))

    T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
    T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)

    # Plot and save the results
    plt.figure()
    plt.subplot(211)
    plt.hist(np.real(T_y_0), bins=125)
    plt.title("$H_{0}$ Statistic distribution - Real component")
    plt.subplot(212)
    plt.hist(np.imag(T_y_0), bins=125)
    plt.title("$H_{0}$ Statistic distribution - Imaginary component")
    plt.tight_layout()
    plt.savefig('./results/cyclostationary_detector_a_H0.png')
    plt.show()

    plt.figure()
    plt.subplot(211)
    plt.hist(np.real(T_y_1), bins=125)
    plt.title("$H_{1}$ Statistic distribution - Real component")
    plt.subplot(212)
    plt.hist(np.imag(T_y_1), bins=125)
    plt.title("$H_{1}$ Statistic distribution - Imaginary component")
    plt.tight_layout()
    plt.savefig('./results/cyclostationary_detector_a_H1.png')
    plt.show()


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

Problem 2B

```python
import numpy as np
from scipy.stats import chi2
import matplotlib.pyplot as plt
from cyclostationary_detector.parameters import *


def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
    """
    Generate H0 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H0 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)

    for ind in range(NUM_STATISTICS):
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

```

```
        y = w

        # Calculate test statistic
        val = np.complex(0)
        for n in range(N_c):
            for k in range(K):
                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])

        T_y[ind] = 1 / K * val

    return T_y


def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
    """
    Generate H1 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H1 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)

    for ind in range(NUM_STATISTICS):
        x = sigma_s * np.random.randn(N, 1)
        for k in range(K):
            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)]

        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = x + w

        # Calculate test statistic
        val = np.complex(0)
        for n in range(N_c):
            for k in range(K):
                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])

        T_y[ind] = 1 / K * val

    return T_y


def main():
    # Define the SNR where performance has to be evaluated
    SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
    N = (K + 1) * (N_c + N_d)

    P_FA_THEO = np.zeros(len(SNR_list))

    P_FA_CALC = np.zeros(len(SNR_list))
    P_D_CALC = np.zeros(len(SNR_list))

    # Evaluate the performance on H0 and H1
    for ind, SNR in enumerate(SNR_list):
        NUM_FALSE_ALARM = 0
        NUM_DETECTION = 0

        sigma_w = np.sqrt(sigma_s ** 2 / 10 ** (SNR / 10))
        gamma = chi2.isf(q=P_FA, df=2) * N_c / K * (sigma_w ** 4)
```

```
        T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
        T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)

        for T in T_y_0:
            if np.square(np.abs(T)) >= gamma:
                NUM_FALSE_ALARM += 1

        for T in T_y_1:
            if np.square(np.abs(T)) >= gamma:
                NUM_DETECTION += 1

        P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
        P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS

        P_FA_THEO[ind] = P_FA

    # Plot and save the results
    plt.figure()
    plt.plot(SNR_list, P_FA_CALC, label="Calculated $P_{FA}$")
    plt.plot(SNR_list, P_D_CALC, label="Calculated $P_{D}$")
    plt.plot(SNR_list, P_FA_THEO, label="Theoretical $P_{FA}$")
    plt.xlabel("SNR")
    plt.ylabel("Probability")
    plt.title("Cyclostationary detector")
    plt.legend()
    plt.savefig('./results/cyclostationary_detector_b.png')
    plt.show()


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

Problem 2C

```
import numpy as np
from scipy.stats import chi2
import matplotlib.pyplot as plt
from cyclostationary_detector.parameters import *


def generate_statistic_H0(NUM_STATISTICS, sigma_w, N):
    """
    Generate H0 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H0 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)

    for ind in range(NUM_STATISTICS):
        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = w

```

```
        # Calculate test statistic
        val = np.complex(0)
        for n in range(N_c):
            for k in range(K):
                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])

        T_y[ind] = 1 / K * val

    return T_y


def generate_statistic_H1(NUM_STATISTICS, sigma_w, N):
    """
    Generate H1 test statistics
    :param NUM_STATISTICS: Number of statistics to be produced
    :param sigma_w: Std deviation of noise
    :param N: Length of observation vector
    :return: H1 test statistics
    """

    T_y = np.zeros(NUM_STATISTICS, dtype=np.complex)

    for ind in range(NUM_STATISTICS):
        x = sigma_s * np.random.randn(N, 1)
        for k in range(K):
            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)]

        w = sigma_w * np.random.randn(N, 2).view(np.complex128)

        y = x + w

        # Calculate test statistic
        val = np.complex(0)
        for n in range(N_c):
            for k in range(K):
                val += y[n + k * (N_c + N_d)] * np.conjugate(y[n + k * (N_c + N_d) + N_d])

        T_y[ind] = 1 / K * val

    return T_y


def main():
    # Define the SNR where performance has to be evaluated
    SNR_list = np.arange(SNR_LOW, SNR_UP, SNR_STEP)
    N = (K + 1) * (N_c + N_d)

    P_FA_THEO = np.zeros(len(SNR_list))

    P_FA_CALC = np.zeros(len(SNR_list))
    P_D_CALC = np.zeros(len(SNR_list))

    # Evaluate the performance on H0 and H1
    for ind, SNR in enumerate(SNR_list):
        NUM_FALSE_ALARM = 0
        NUM_DETECTION = 0

        sigma_w = sigma_s ** 2 / 10 ** (SNR / 10)
        sigma_w = np.sqrt(sigma_w * 10 ** (SNR_NOISE / 10))
        gamma = chi2.isf(q=P_FA, df=2) * N_c / K * (sigma_w ** 4)

```

```python
        T_y_0 = generate_statistic_H0(NUM_STATISTICS, sigma_w, N)
        T_y_1 = generate_statistic_H1(NUM_STATISTICS, sigma_w, N)

        for T in T_y_0:
            if np.square(np.abs(T)) >= gamma:
                NUM_FALSE_ALARM += 1

        for T in T_y_1:
            if np.square(np.abs(T)) >= gamma:
                NUM_DETECTION += 1

        P_FA_CALC[ind] = NUM_FALSE_ALARM / NUM_STATISTICS
        P_D_CALC[ind] = NUM_DETECTION / NUM_STATISTICS

        P_FA_THEO[ind] = P_FA

    # Plot and save the results
    plt.figure()
    plt.plot(SNR_list, P_FA_CALC, label="Calculated $P_{FA}$")
    plt.plot(SNR_list, P_D_CALC, label="Calculated $P_{D}$")
    plt.plot(SNR_list, P_FA_THEO, label="Theoretical $P_{FA}$")
    plt.xlabel("SNR")
    plt.ylabel("Probability")
    plt.title("Cyclostationary detector")
    plt.legend()
    plt.savefig('./results/cyclostationary_detector_c.png')
    plt.show()


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

Problem 2 Parameters

```python
sigma_s = 1
N_d = 32
N_c = 8
K = 50
P_FA = 0.05

SNR = 100
SNR_LOW = -20
SNR_UP = 6
SNR_STEP = 2
SNR_NOISE = 1

NUM_STATISTICS = 1000
#NUM_STATISTICS = 100000
```