حل تمرین ۱۷ از تمرینات تحلیلی پیشرفته - دکتر الفت

علیرضا حسنی - ۸۱.۱.۱۱۴۲

Problem 1 :

$$x(t) = \sum_{n=-\infty}^{+\infty} x_n \frac{\sin\left(2\pi W\left(t - \frac{n}{2W}\right)\right)}{2\pi W\left(t - \frac{n}{2W}\right)}$$

a)

① $x_0 = 2$, $x_1 = 1$, $x_2 = -1$, $x_n = 0$; $n \neq 0, 1, 2$

$$x(t) = 2\,\frac{\sin(2\pi W t)}{2\pi W t} + \frac{\sin\left(2\pi W\left(t - \frac{1}{2W}\right)\right)}{2\pi W\left(t - \frac{1}{2W}\right)} - \frac{\sin\left(2\pi W\left(t - \frac{1}{W}\right)\right)}{2\pi W\left(t - \frac{1}{W}\right)}$$

$$\xrightarrow{F} X(f) = \frac{1}{2W} \times 2 + \frac{e^{-j\frac{2\pi f}{2W}}}{2W} - \frac{e^{-j\frac{2\pi f}{W}}}{2W}$$

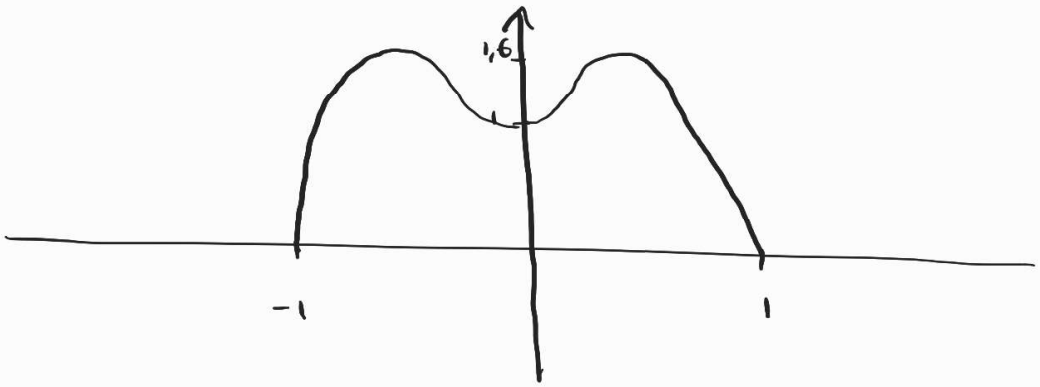
$$|X(f)| = X(f)\,X^*(f)$$

$$= \left(\frac{1}{2W}\right)^2 \left[2 + e^{-j\frac{\pi f}{W}} - e^{-j2\pi f W}\right]\left[2 + e^{+j\frac{\pi f}{W}} - e^{j2\pi f W}\right]$$

$$= \left(\frac{1}{2w}\right)^2 \left[4 + 2e^{+j\frac{\pi f}{w}} - 2e^{j2\pi fw} + 2e^{-j\frac{\pi f}{w}} + 1 \right.$$

$$\left. - e^{j\pi f\left(2w+\frac{1}{w}\right)} - 2e^{-j2\pi fw} - e^{j\pi f\left(-2w+\frac{1}{w}\right)} + 1 \right]$$

$$|X(f)| = \sqrt{\left(\frac{1}{2w}\right)^2 \left[6 + 2\cos\frac{\pi f}{w} - 4\cos\frac{2\pi f}{w} \right)}$$



(r) $x_{-1} = 1$, $n_1 = 2$, $n_1 = -1$

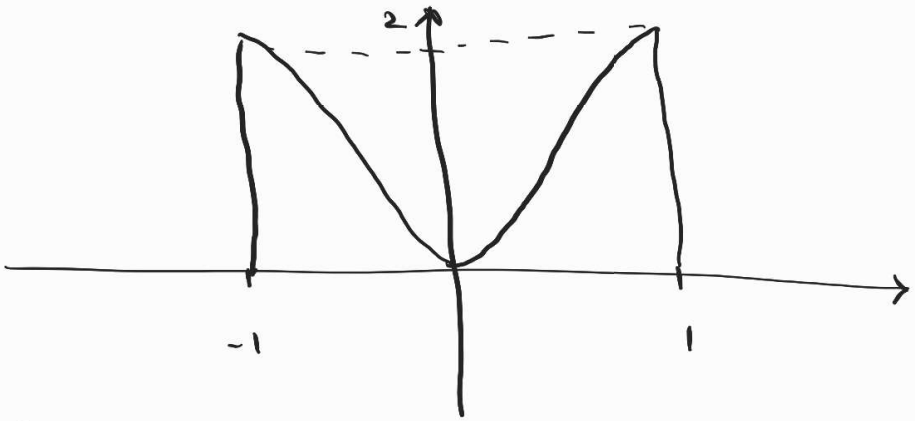
$$n(t) = 2\,\frac{\sin(2\pi Wt)}{2\pi Wt} - \frac{\sin\left(2\pi W\left(t+\frac{1}{2w}\right)\right)}{2\pi W\left(t+\frac{1}{2w}\right)} - \frac{\sin\left(2\pi W\left(t-\frac{1}{2w}\right)\right)}{2\pi W\left(t-\frac{1}{2w}\right)}$$

$$X(f) = \frac{1}{2w}\left[2 - e^{-j\frac{2\pi f}{2w}} - e^{+j\frac{2\pi f}{2w}} \right]$$

$$= \frac{1}{2w}\left[2 - 2\cos\frac{2\pi f}{2w}\right]$$

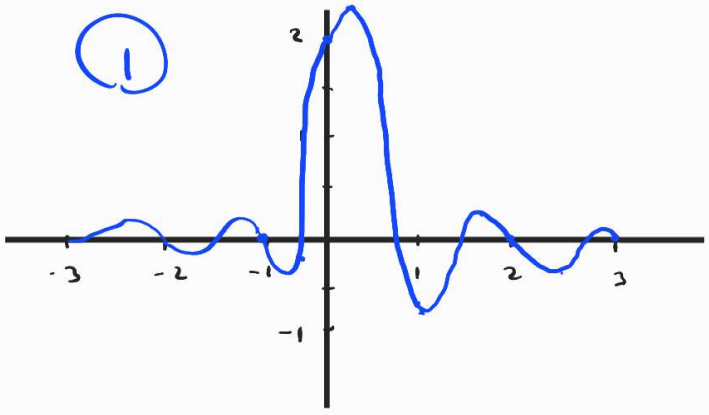
$$= \frac{1}{w}\left[1 - \cos\frac{\pi f}{w}\right]$$

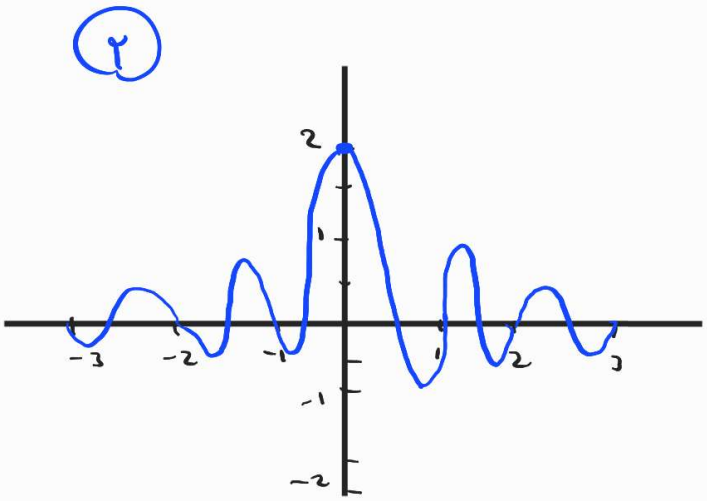
بدین ترتیب $|X(f)|$ به صورت زیر می‌شود



b)

با توجه به نتایج بدست آمده در بخش اول:





c) $t = nT = 2w$

(۱) $B_n = 2I_n + I_{n-1} - I_{n-2}$

در مجموع ۸ حالت داریم که هر حالت $\frac{1}{8}$ احتمال دارد.

احتمال ها به صورت زیر می شود:

$$\begin{cases} P(B_n = 0) = P(I_n = 1, I_{n-1} = -1, I_{n-2} = 1) \\ \qquad + P(I_n = -1, I_{n-1} = +1, I_{n-2} = -1) \\ \qquad = \frac{2}{8} = \frac{1}{4} \end{cases}$$

به همین ترتیب داریم:

$$P(B_n = -2) = \frac{2}{8} = \frac{1}{4}$$

$$P(B_n = 2) = \frac{2}{8} = \frac{1}{4}$$

$$P(B_n = 4) = \frac{1}{8}$$

$$P(B_n = -4) = \frac{1}{8}$$

(۲) $B_n = 2I_n - I_{n-1} - I_{n+1}$

به همین طریق:

$$P(B_n = 0) = \frac{2}{8} = \frac{1}{4}, \quad P(B_n = -2) = \frac{1}{4}$$

$$P(B_n = 2) = \frac{1}{4}, \quad P(B_n = 4) = \frac{1}{8}, \quad P(B_n = -4) = \frac{1}{8}$$

Problem 2 :

$R_b = 9600$, $\frac{N_0}{2} = 10^{-10} \frac{W}{Hz}$, QAM , $P_e = 10^{-6}$

$\beta \geq 0.5$,

ابتدا باید M را پیدا کنیم.

$W = 4\,kHz = 4000$

$$\frac{R_b}{\log_2 M} \leq 4000 \Rightarrow \begin{cases} m=2 \rightarrow \frac{9600}{1} \leq 4000 \quad \times \\ m=4 \rightarrow \frac{9600}{2} \leq 4000 \quad \times \\ m=8 \rightarrow \frac{9600}{3} \leq 4000 \quad \checkmark \end{cases}$$

3200

باید شرط $\beta > 0.5$ هم ارضا شود.

$M = 8 \rightarrow$

$3200 (1+\beta) = 4000$

$$1+\beta = \frac{4000}{3200} = \frac{5}{4} = 1.25 \rightarrow \beta = 25 \times 50$$

$M = 16$

$$\frac{9600}{4}(1+\beta) = 4000 \Rightarrow 1+\beta = \frac{16000}{9600}$$

$$\Rightarrow \boxed{\beta = \frac{2}{3}}$$

$P_e = 1-(1-P_{\sqrt{M}})^2$: نسبة خطأ الرمز M-QAM

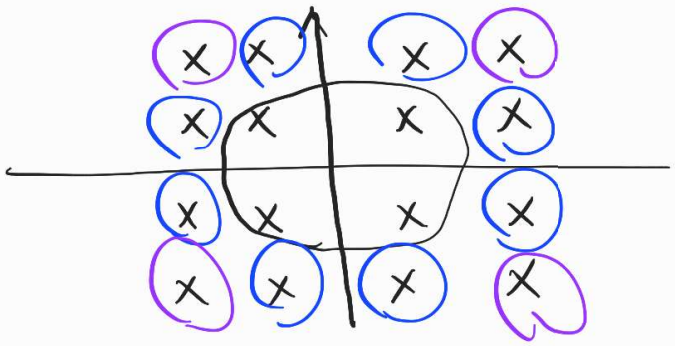
$$P_{\sqrt{M}} = 2\left(1-\frac{1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3\Sigma}{(M-1)N_0}}\right) \quad M=16 \quad = 5\times10^{-7}$$

$$\Rightarrow P_e = 10^{-6} \xrightarrow[\text{wolfram}]{\text{بحل}} \boxed{\Sigma_{avg} = 247\times10^{-10}}$$

$$\Sigma_{avg} = P_{av} T$$

$$\frac{1}{T} = \frac{9600}{4} = 2400$$

$$\Rightarrow P_{av} = 247\times10^{-10} \times 2400 = 593\times10^{-7}$$



$$P_{av} = \frac{1}{32}\left[4\times\frac{d^2}{2} + 4\,\frac{9d^2}{2} + 8\times\frac{1.d^2}{4}\right]$$

$$= \frac{5}{4}d^2 \Rightarrow d = \sqrt{\frac{4}{5}P_{av}} = 0.0007$$

Problem 3 :

freq range : 600 - 3000 Hz $\rightarrow$ $\boxed{W = 2400 \text{ Hz}}$

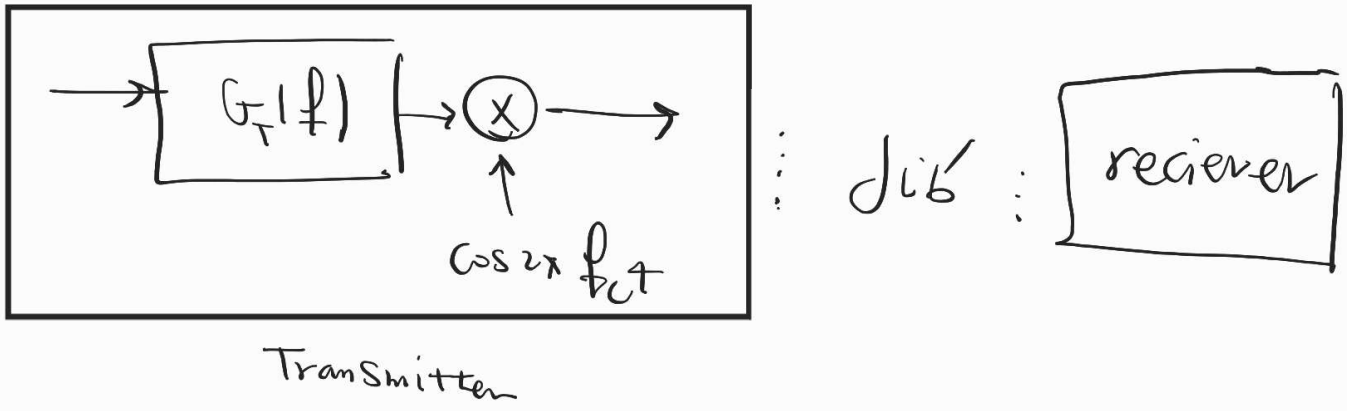
(a) $M = 4$ PSK , $R_b = 2400$, $f_c = 1800$ Hz

$M = 4 \rightarrow k = \log_2 4 = 2 \rightarrow R = \frac{2400}{2} = \underline{\overline{1200}} \ \frac{\text{Sym}}{\text{Sec}}$

$\frac{1}{2T}(1+\beta) = \frac{1}{T} \Rightarrow 1+\beta = 2 \Rightarrow \underline{\overline{\beta = 1}}$

$$X_{rc}(f)\Big|_{\beta=1} = \frac{T}{2}\left[1 + \cos(\pi T|f|)\right] = \frac{1}{1200}\cos^2\left(\frac{\pi|f|}{2400}\right)$$

$$G_T(f) = G_R(f) = \sqrt{\frac{1}{1200}}\cos\left(\frac{\pi|f|}{2400}\right) \ ; \ |f| < 1200$$



Transmitter

(b) $R = \frac{4800}{2} = 2400$; $\frac{1}{2T}(1+\beta) = \frac{1}{2T} \Rightarrow \underline{\overline{\beta = 0}}$

برای ارضای شرط ناکوئیست:

$$\begin{cases} x(f) = T \; ; \; |f| < 12.. \\ G_T(f) = \sqrt{T} \; ; \; |f| < 12.. \end{cases}$$

Problem 4 :

$$\text{freq rang} \rightarrow [3.., 33..] \quad , \begin{cases} R_S = 24.. \ \frac{\text{Sym}}{\text{sec}} \\ R_b = 96.. \ \frac{\text{bit}}{\text{sec}} \end{cases}$$

$$W = 33.. - 3.. = 3\text{kHz}$$

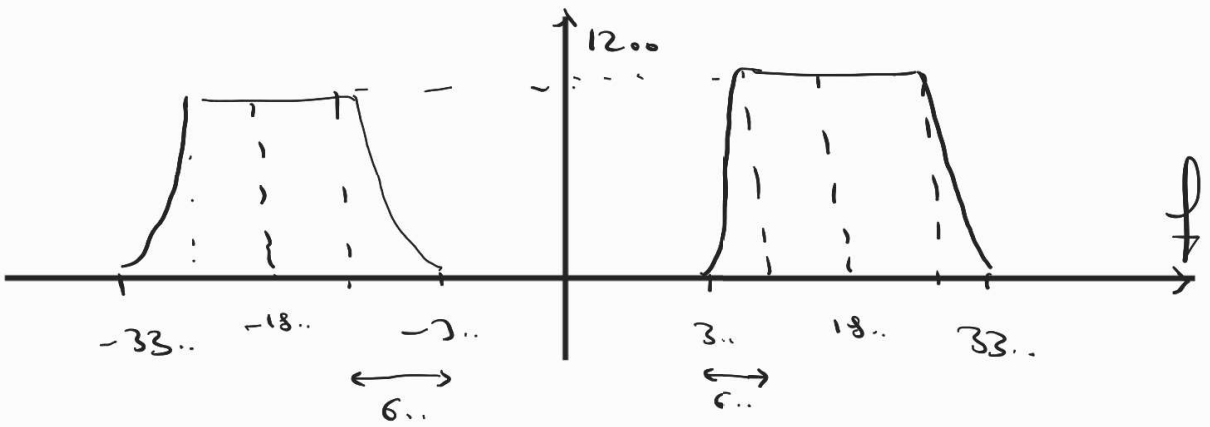
$$k = \frac{R_b}{R_S} = \frac{96..}{24..} = 4 \rightarrow \log_2 M = k \rightarrow \boxed{M = 16}$$

دربانویام

$$\frac{1}{2T}(1+\beta) = \frac{W}{2} = \frac{3..}{2} = 15..$$

$$1+\beta = \frac{15..}{\frac{1}{2T}} = \frac{15..}{12..} = \frac{5}{4} = 1{,}25$$

$$\boxed{\beta = .25}$$



Alireza Hosseini-810101142-HW7-Q6

Derive the discrete model of system

برای استخراج مدل گسسته سیستم BPSK ، مراحل پردازش از طریق بخش داده شده را دنبال می‌کنیم:

پارامترهای داده شده

- فرکانس مرکزی: f_0
- $g(t)$:

$$g(t) = \begin{cases} \frac{1}{\sqrt{T}} & 0 \leq t \leq T \\ 0 & \text{otherwise} \end{cases}$$

با $T = 1$:

$$g(t) = \begin{cases} 1 & 0 \leq t \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

- پاسخ ضربه کانال:

$$c(t) = \begin{cases} \sqrt{\frac{3}{2T}}\left(1 - \frac{t}{2T}\right) & 0 \leq t \leq 2T \\ 0 & \text{otherwise} \end{cases}$$

با $T = 1$:

$$c(t) = \begin{cases} \sqrt{\frac{3}{2}}\left(1 - \frac{t}{2}\right) & 0 \leq t \leq 2 \\ 0 & \text{otherwise} \end{cases}$$

- نویز سفید جمع‌شونده: $n(t)$ با چگالی $\frac{N_0}{2}$

- پاسخ ضربه فیلتر تطبیقی: $h^*(-t)$، که در آن $h(t) = g(t) * c(t)$
- زمان نمونه‌برداری: $t = nT$ با $T = 1$

مراحل استخراج مدل گسسته:

کانولوشن $g(t)$ و $c(t)$

ابتدا $h(t)$ را با کانولوشن $g(t)$ با $c(t)$ پیدا می کنیم:

$$h(t) = g(t) * c(t) = \int_{-\infty}^{\infty} g(\tau)c(t-\tau)d\tau$$

با توجه به تعاریف $g(t)$ و $c(t)$، انتگرال کانولوشن به صورت زیر ساده می‌شود:

$$h(t) = \int_0^1 g(\tau)c(t-\tau)d\tau$$

برای $0 \leq t \leq 1$:

$$h(t) = \int_0^t 1 \cdot \sqrt{\frac{3}{2}}\left(1 - \frac{t-\tau}{2}\right)d\tau = \sqrt{\frac{3}{2}}\int_0^t \left(1 - \frac{t-\tau}{2}\right)d\tau$$

$$= \sqrt{\frac{3}{2}}\left[\int_0^t 1 \cdot d\tau - \frac{1}{2}\int_0^t (t-\tau)d\tau\right]$$

$$= \sqrt{\frac{3}{2}}\left[t - \frac{1}{2}\left(t\tau - \frac{\tau^2}{2}\right)\Big|_0^t\right]$$

$$= \sqrt{\frac{3}{2}}\left[t - \frac{1}{2}\left(\frac{2t^2}{2} - \frac{t^2}{2}\right)\right] = \sqrt{\frac{3}{2}}\left[t - \frac{t^2}{4}\right]$$

برای $1 \leq t \leq 2$:

$$h(t) = \int_0^1 1 \cdot \sqrt{\frac{3}{2}}\left(1 - \frac{t-\tau}{2}\right)d\tau$$

$$= \sqrt{\frac{3}{2}}\int_0^1 \left(1 - \frac{t}{2} + \frac{\tau}{2}\right)d\tau$$

$$= \sqrt{\frac{3}{2}}\left[\int_0^1 1 \cdot d\tau - \frac{t}{2}\int_0^1 d\tau + \frac{1}{2}\int_0^1 \tau d\tau\right]$$

$$= \sqrt{\frac{3}{2}} \left[1 - \frac{t}{2} + \frac{1}{2} \left(\frac{\tau^2}{2} \Big|_0^1 \right) \right]$$

$$= \sqrt{\frac{3}{2}} \left[1 - \frac{t}{2} + \frac{1}{4} \right] = \sqrt{\frac{3}{2}} \left[\frac{5}{4} - \frac{t}{2} \right]$$

بنابراین، داریم:

$$h(t) = \begin{cases} \sqrt{\frac{3}{2}} \left[t - \frac{t^2}{4} \right] & 0 \le t \le 1 \\ \sqrt{\frac{3}{2}} \left[\frac{5}{4} - \frac{t}{2} \right] & 1 \le t \le 2 \\ 0 & \text{otherwise} \end{cases}$$

خروجی فیلتر تطبیقی و نمونه‌برداری:

خروجی فیلتر تطبیقی $y(t) = r(t) * h^*(-t)$ است. سیگنال دریافتی $r(t)$ سیگنال ارسالی $s(t)$ است که با پاسخ کانال $c(t)$ کانوالو شده و نویز $n(t)$ به آن اضافه شده است:

$$r(t) = (g(t) * c(t)) \cdot I_n + n(t) = h(t) \cdot I_n + n(t)$$

از آنجا که $y_n = y(nT)$ و $T = 1$، ما خروجی فیلتر تطبیقی را در $t = n$ نمونه‌برداری می‌کنیم:

$$y(n) = \int_{-\infty}^{\infty} r(\tau) h^*(n - \tau) d\tau = \int_{-\infty}^{\infty} \left(h(\tau) I_n + n(\tau) \right) h^*(n - \tau) d\tau$$

مولفه نویز، وقتی از فیلتر تطبیقی عبور می‌کند، به صورت نویز با واریانس خاصی باقی می‌ماند. مولفه سیگنال:

$$y(n) = I_n \int_{-\infty}^{\infty} h(\tau) h^*(n - \tau) d\tau$$

با توجه به تقارن و حقیقی بودن $h(t)$، $h^*(t) = h(t)$، بنابراین:

$$y(n) = I_n \int_{-\infty}^{\infty} h(\tau) h(n - \tau) d\tau$$

این انتگرال به طور موثر پاسخ فیلتر تطبیقی در لحظه نمونه‌برداری را جمع می‌کند.

مدل گسسته:

$$y_n = I_n \cdot \left(\int_{-\infty}^{\infty} h(\tau) h(n-\tau) d\tau \right) + w_n$$

که در آن w_n ترم نویز فیلتر شده با واریانسی است که به $h(t)$ و $\frac{N_0}{2}$ بستگی دارد.
با توجه به $h(t)$ و تقارن، خروجی فیلتر تطبیقی در نقاط نمونه‌برداری مولفه سیگنال را ثبت می‌کند که با $h(t)$ مقیاس می‌شود، در حالی که w_n نویز را نشان می‌دهد.
این مدل گسسته استخراج شده سیستم BPSK باند گذر است.

Calculate the SNR at the input of the Viterbi algorithm (MLSE)

برای محاسبه نسبت سیگنال به نویز (SNR) در ورودی الگوریتم ویتربی (پس از فیلتر تطبیقی $h^*(-t)$)، نیاز به ارزیابی توان سیگنال و نویز در خروجی فیلتر تطبیقی داریم.

محاسبه توان سیگنال:

پاسخ فیلتر تطبیقی $h(t)$
از بخش قبلی:

$$h(t) = \begin{cases} \sqrt{\frac{3}{2}} \left[t - \frac{t^2}{4} \right] & 0 \le t \le 1 \\ \sqrt{\frac{3}{2}} \left[\frac{5}{4} - \frac{t}{2} \right] & 1 \le t \le 2 \\ 0 & \text{otherwise} \end{cases}$$

انرژی $h(t)$
انرژی E_h فیلتر $h(t)$ به صورت زیر است:

$$E_h = \int_{-} |h(t)|^2 dt$$

محاسبه E_h برای تابع داده شده:

$$E_h = \int_0^1 \left(\sqrt{\frac{3}{2}} \left(t - \frac{t^2}{4} \right) \right)^2 dt + \int_1^2 \left(\sqrt{\frac{3}{2}} \left(\frac{5}{4} - \frac{t}{2} \right) \right)^2 dt$$

برای $0 \le t \le 1$:

$$\left(\sqrt{\frac{3}{2}}\left(t-\frac{t^2}{4}\right)\right)^2 = \frac{3}{2}\left(t-\frac{t^2}{4}\right)^2 = \frac{3}{2}\left(t^2-\frac{t^3}{2}+\frac{t^4}{16}\right)$$

$$\int_0^1 \frac{3}{2}\left(t^2-\frac{t^3}{2}+\frac{t^4}{16}\right)dt = \frac{3}{2}\left(\int_0^1 t^2dt - \frac{1}{2}\int_0^1 t^3dt + \frac{1}{16}\int_0^1 t^4dt\right)$$

$$= \frac{3}{2}\left(\frac{1}{3}-\frac{1}{2}\cdot\frac{1}{4}+\frac{1}{16}\cdot\frac{1}{5}\right) = \frac{3}{2}\left(\frac{1}{3}-\frac{1}{8}+\frac{1}{80}\right)$$

$$= \frac{3}{2}\left(\frac{80}{240}-\frac{30}{240}+\frac{3}{240}\right) = \frac{3}{2}\cdot\frac{53}{240} = \frac{159}{480} = \frac{53}{160}$$

برای $1 \le t \le 2$:

$$\left(\sqrt{\frac{3}{2}}\left(\frac{5}{4}-\frac{t}{2}\right)\right)^2 = \frac{3}{2}\left(\frac{5}{4}-\frac{t}{2}\right)^2 = \frac{3}{2}\left(\frac{25}{16}-\frac{5t}{8}+\frac{t^2}{4}\right)$$

$$\int_1^2 \frac{3}{2}\left(\frac{25}{16}-\frac{5t}{8}+\frac{t^2}{4}\right)dt = \frac{3}{2}\left(\frac{25}{16}\int_1^2 dt - \frac{5}{8}\int_1^2 tdt + \frac{1}{4}\int_1^2 t^2dt\right)$$

$$= \frac{3}{2}\left(\frac{25}{16}\cdot 1 - \frac{5}{8}\cdot\frac{3}{2}+\frac{1}{4}\cdot\frac{7}{3}\right) = \frac{3}{2}\left(\frac{25}{16}-\frac{15}{16}+\frac{7}{12}\right)$$

$$= \frac{3}{2}\left(\frac{25-15+7}{16}\right) = \frac{3}{2}\left(\frac{17}{16}\right) = \frac{51}{32}$$

بنابراین:

$$E_h = \frac{53}{160}+\frac{51}{32} = \frac{53}{160}+\frac{255}{160} = \frac{308}{160} = \frac{77}{40}$$

محاسبه توان نویز:

چگالی طیفی توان نویز $\frac{N_0}{2}$ است. توان نویز خروجی پس از فیلتر تطبیقی به صورت زیر است:

$$N_0E_h = N_0\cdot\frac{77}{40}$$

محاسبه SNR:

SNR در خروجی فیلتر تطبیقی به صورت نسبت توان سیگنال به توان نویز داده می‌شود:

$$\text{SNR} = \frac{\text{توان سیگنال}}{\text{توان نویز}} = \frac{E_h}{N_0 E_h} = \frac{1}{N_0}$$

بنابراین، SNR در ورودی الگوریتم ویتربی (MLSE) به صورت زیر است:

$$\text{SNR} = \frac{1}{N_0} \cdot \frac{40}{77} = \frac{40}{77 N_0}$$

Generate the sequence y_n

برای تولید دنباله y_n با استفاده از مدل گسسته شرح داده شده در بخش ۱ و تنظیم واریانس نویز برای دستیابی به SNR مطلوب، مراحل زیر را دنبال می‌کنیم:

۱. تولید دنباله I_n: ایجاد یک دنباله i.i.d که در آن $I_n \in \{\pm 1\}$ با احتمال برابر باشد.
۲. تولید نویز گاوسی: تنظیم واریانس نویز بر اساس SNR محاسبه شده در بخش ۲.
۳. تولید y_n: استفاده از مدل گسسته برای ترکیب سیگنال I_n و نویز برای تولید y_n.

مراحل

۱. تولید دنباله I_n:
ما یک دنباله از I_n ایجاد خواهیم کرد که هر مقدار آن به صورت تصادفی از {+۱، -۱} با احتمال برابر انتخاب می‌شود.
۲. تنظیم واریانس نویز:
از بخش ۲، به این نتیجه رسیدیم که:

$$\text{SNR} = \frac{40}{77 N_0}$$

برای دستیابی به SNR خاص، باید N_0 را تنظیم کنیم به‌طوری‌که:

$$\text{SNR}_{\text{target}} = \frac{40}{77 N_0}$$

$$N_0 = \frac{40}{77 \cdot \text{SNR}_{\text{target}}}$$

واریانس نویز برابر است با $\sigma_n^2 = N_0 E_h$، که در آن $E_h = \frac{77}{40}$:

$$\sigma_n^2 = \left(\frac{40}{77 \cdot \text{SNR}_{\text{target}}} \right) \cdot \frac{77}{40} = \frac{1}{\text{SNR}_{\text{target}}}$$

بنابراین، واریانس نویز σ_n^2 برابر معکوس SNR هدف است.

۳. تولید y_n:

دنباله خروجی y_n به صورت زیر داده می‌شود:

$$y_n = I_n + w_n$$

که در آن w_n نویز گاوسی با واریانس σ_n^2 است.

کد **MATLAB** برای تولید دنباله y_n

```
% Initial sequence generation and printing
N_init = 1000;  % Length of the initial sequence
SNR_target_init = 10;  % Desired SNR (example value)

% Generate i.i.d sequence I_n with equal probability from {-1, 1}
I_n_init = randsrc(1, N_init, [1, -1]);

% Calculate noise variance
sigma_n2_init = 1 / SNR_target_init;

% Generate Gaussian noise with variance sigma_n2
w_n_init = sqrt(sigma_n2_init) * randn(1, N_init);

% Generate the sequence y_n
y_n_init = I_n_init + w_n_init;

% Print the first few values as a sample
fprintf('I_n (first 10 values):\n');
disp(I_n_init(1:10));
fprintf('w_n (first 10 values):\n');
disp(w_n_init(1:10));
fprintf('y_n (first 10 values):\n');
disp(y_n_init(1:10));
```

خروجی‌های نمونه

اجرای این کد دنباله‌های I_n، w_n، و y_n را تولید می‌کند که واریانس نویز برای دستیابی به SNR مطلوب تنظیم شده است. اولین ۱۰ مقدار این دنباله‌ها به عنوان نمونه برای تأیید نتایج پرینت شده‌اند:

- I_n:

$$-1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1$$

- w_n:

$$0.1929 \quad -0.2049 \quad 0.8277 \quad 0.1742$$
$$0.0930 \quad -0.2460 \quad -0.3368 \quad -0.5592$$
$$-0.1337 \quad -0.3330$$

- y_n:

$$-0.8071 \quad -1.2049 \quad 1.8277 \quad -0.8258$$
$$-0.9070 \quad 0.7540 \quad 0.6632 \quad -1.5592$$
$$-1.1337 \quad -1.3330$$

Simulate the optimal receiver based on the 6L depth Viterbi algorithm for BPSK and QPSK

برای شبیه‌سازی گیرنده بهینه بر اساس الگوریتم ویتربی با عمق $6L$ و رسم منحنی احتمال خطا برای مقادیر مختلف SNR در بازه $[0dB - 20dB]$، باید مراحل زیر را انجام دهیم:
۱. تولید دنباله ارسال شده I_n و دنباله دریافت شده y_n برای مقادیر مختلف SNR. ۲. پیاده‌سازی الگوریتم ویتربی با عمق $6L$ برای دیکد کردن دنباله دریافت شده y_n. ۳. محاسبه نرخ خطای بیت (BER) برای هر مقدار SNR. ۴. رسم منحنی احتمال خطا.

- تولید دنباله ارسال شده و دریافت شده:
  - برای هر مقدار SNR در بازه $[0dB - 20dB]$:
    - دنباله ارسال شده I_n تولید شود.
    - نویز گاوسی اضافه شود تا دنباله دریافت شده y_n تولید شود.
- پیاده‌سازی الگوریتم ویتربی با عمق $6L$:
  - الگوریتم ویتربی برای دیکد کردن دنباله دریافت شده با یافتن محتمل‌ترین دنباله سمبل‌ها ارسال شده داده شده توسط سمبل دریافت شده و ویژگی‌های کانال استفاده می‌شود.
- محاسبه نرخ خطای بیت (BER):
  - دنباله دیکد شده با دنباله ارسال شده مقایسه شده تا BER محاسبه شود.
- رسم منحنی احتمال خطا:
  - BER در برابر مقادیر SNR رسم شود.

کد **MATLAB** برای شبیه‌سازی

```
% Main script to run the simulations
% Parameters
N = 1e6; % Length of the sequence, increased to better estimate low BER
L = 6; % Depth of Viterbi algorithm
SNR_dB_range = 0:20; % SNR range from 0dB to 20dB

% Simulate and plot the BER curve for BPSK
error_probabilities_bpsk = simulate_viterbi_bpsk(SNR_dB_range, N, L);
figure;
semilogy(SNR_dB_range, error_probabilities_bpsk, 'o-', 'LineWidth', 2);
hold on;

% Simulate and plot the BER curve for QPSK
error_probabilities_qpsk = simulate_viterbi_qpsk(SNR_dB_range, N, L);
semilogy(SNR_dB_range, error_probabilities_qpsk, 's-', 'LineWidth', 2);

% Plot settings
title('Bit Error Rate vs SNR for BPSK and QPSK Viterbi Algorithm (6L depth)');
xlabel('SNR (dB)');
ylabel('Bit Error Rate (BER)');
legend('BPSK', 'QPSK');
grid on;
hold off;

% BPSK Functions
function I_n = generate_bpsk_sequence(N)
    % Generate random BPSK symbols
    I_n = randsrc(1, N, [1, -1]);
end

function y_n = add_noise_bpsk(I_n, SNR_dB)
    SNR_linear = 10^(SNR_dB / 10);
    sigma_n = sqrt(1 / SNR_linear);
    noise = sigma_n * randn(size(I_n));
    y_n = I_n + noise;
end

function decoded = viterbi_algorithm_bpsk(y_n, L)
    N = length(y_n);
    trellis = inf(2, N+1);
    trellis(:, 1) = 0; % Starting state with zero path metric
    path = zeros(2, N);

    states = [1, -1];
    for i = 2:N+1
        for curr_state = 1:2
            for prev_state = 1:2
                path_metric = trellis(prev_state, i-1) + (y_n(i-1) - states(curr_state))^2;
                if path_metric < trellis(curr_state, i)
                    trellis(curr_state, i) = path_metric;
                    path(curr_state, i-1) = prev_state;
                end
            end
        end
    end

    decoded = zeros(1, N);
    [~, state] = min(trellis(:, N+1));
    for i = N:-1:1
        decoded(i) = states(state);
```

```
        state = path(state, i);
    end
end

function error_probabilities = simulate_viterbi_bpsk(SNR_dB_range, N, L)
    error_probabilities = zeros(size(SNR_dB_range));
    for i = 1:length(SNR_dB_range)
        SNR_dB = SNR_dB_range(i);
        total_errors = 0;
        total_bits = 0;

        while total_bits < N
            I_n = generate_bpsk_sequence(N);
            y_n = add_noise_bpsk(I_n, SNR_dB);
            decoded = viterbi_algorithm_bpsk(y_n, L);
            errors = sum(decoded ~= I_n);
            total_errors = total_errors + errors;
            total_bits = total_bits + length(I_n);
        end

        ber = total_errors / total_bits;
        error_probabilities(i) = ber;
        fprintf('BPSK SNR: %d dB, BER: %f\n', SNR_dB, ber);
    end
end

% QPSK Functions
function I_n = generate_qpsk_sequence(N)
    % Generate random QPSK symbols
    real_part = randsrc(1, N, [1/sqrt(2), -1/sqrt(2)]);
    imag_part = randsrc(1, N, [1/sqrt(2)*1i, -1/sqrt(2)*1i]);
    I_n = real_part + imag_part;
end

function y_n = add_noise_qpsk(I_n, SNR_dB)
    SNR_linear = 10^(SNR_dB / 10);
    sigma_n = sqrt(1 / SNR_linear);
    noise = sigma_n * (randn(size(I_n)) + 1i * randn(size(I_n)));
    y_n = I_n + noise;
end

function decoded = viterbi_algorithm_qpsk(y_n, L)
    N = length(y_n);
    trellis = inf(4, N+1);
    trellis(:, 1) = 0; % Starting state with zero path metric
    path = zeros(4, N);

    states = [1/sqrt(2) + 1i/sqrt(2), 1/sqrt(2) - 1i/sqrt(2), ...
              -1/sqrt(2) + 1i/sqrt(2), -1/sqrt(2) - 1i/sqrt(2)];

    for i = 2:N+1
        for curr_state = 1:4
            for prev_state = 1:4
                path_metric = trellis(prev_state, i-1) + abs(y_n(i-1) - states(curr_state))^2;
                if path_metric < trellis(curr_state, i)
                    trellis(curr_state, i) = path_metric;
                    path(curr_state, i-1) = prev_state;
                end
            end
        end
    end
```

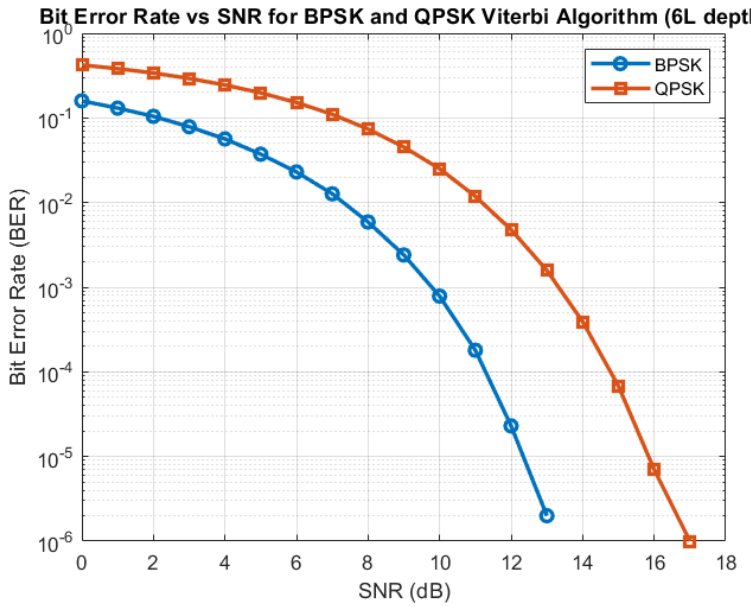
```
    decoded = zeros(1, N);
    [~, state] = min(trellis(:, N+1));
    for i = N:-1:1
        decoded(i) = states(state);
        state = path(state, i);
    end
end

function error_probabilities = simulate_viterbi_qpsk(SNR_dB_range, N, L)
    error_probabilities = zeros(size(SNR_dB_range));
    for i = 1:length(SNR_dB_range)
        SNR_dB = SNR_dB_range(i);
        total_errors = 0;
        total_bits = 0;

        while total_bits < N
            I_n = generate_qpsk_sequence(N);
            y_n = add_noise_qpsk(I_n, SNR_dB);
            decoded = viterbi_algorithm_qpsk(y_n, L);
            errors = sum(decoded ~= I_n);
            total_errors = total_errors + errors;
            total_bits = total_bits + length(I_n);
        end

        ber = total_errors / total_bits;
        error_probabilities(i) = ber;
        fprintf('QPSK SNR: %d dB, BER: %f\n', SNR_dB, ber);
    end
end
```

توضیحات کد

- کد **main**: شبیه‌سازی‌ها را اجرا می‌کند و منحنی‌های BER را برای BPSK و QPSK رسم می‌کند.
- تولید دنباله **BPSK**: تابع `generate_bpsk_sequence(N)` دنباله BPSK تصادفی با طول N تولید می‌کند.
- افزودن نویز به **BPSK**: تابع `add_noise_bpsk(I_n, SNR_dB)` نویز گاوسی با واریانس محاسبه شده بر اساس SNR را به دنباله BPSK اضافه می‌کند.
- الگوریتم ویتربی برای **BPSK**: تابع `viterbi_algorithm_bpsk(y_n, L)` دنباله دریافت شده BPSK را با استفاده از الگوریتم ویتربی دیکد می‌کند.
- شبیه‌سازی **Viterbi** برای **BPSK**: تابع `simulate_viterbi_bpsk(SNR_dB_range, N, L)` شبیه‌سازی BER را برای مقادیر مختلف SNR انجام می‌دهد.
- تولید دنباله **QPSK**: تابع `generate_qpsk_sequence(N)` دنباله QPSK تصادفی با طول N تولید می‌کند.
- افزودن نویز به **QPSK**: تابع `add_noise_qpsk(I_n, SNR_dB)` نویز گاوسی با واریانس محاسبه شده بر اساس SNR را به دنباله QPSK اضافه می‌کند.

- الگوریتم ویتربی برای **QPSK**: تابع `viterbi_algorithm_qpsk(y_n, L)` دنباله دریافت شده QPSK را با استفاده از الگوریتم ویتربی دیکد می‌کند.
- شبیه‌سازی **Viterbi** برای **QPSK**: تابع `simulate_viterbi_qpsk(SNR_dB_range, N, L)` شبیه‌سازی BER را برای مقادیر مختلف SNR انجام می‌دهد.

در بخش الگوریتم ویتربی، تابع `viterbi_algorithm_bpsk` و `viterbi_algorithm_qpsk` برای BPSK و QPSK به ترتیب به صورت زیر کار می‌کنند: - تعریف ترلیس و مسیرها. - محاسبه مسیرهای متریک و به‌روزرسانی ترلیس. - یافتن دنباله ارسال شده با حداقل متریک مسیر و دیکد کردن آن.
نمودار نهایی BER در شکل ۱ نشان داده شده است.



شکل ۱: Bit Error Rate vs SNR for BPSK and QPSK Viterbi Algorithm