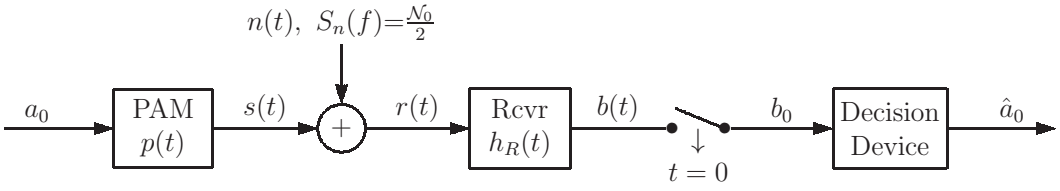
Lab 7: PAM with Noise, Symbol Error Probability, Convolutional Coding

1 Introduction

When transmitting digital data, an important measure for the goodness of the communication system is the probability of symbol error $P_s(\mathcal{E})$. This depends on the signal-to-noise ratio (SNR), the size and geometry of the signal set, and also on how the receiver filter $h_R(t)$ is chosen. To eliminate the dependency on the receiver implementation, the performance of different signal sets is usually compared assuming that $h_R(t)$ is a matched filter. In that case, and under the further assumption of additive white Gaussian noise, $P_s(\mathcal{E})$ can be expressed as a function of $E_b/\mathcal{N}_0$, the energy per bit divided by the PSD of the noise, and the normalized minimum distance $d_{\min}$ of the signal set. Now suppose that $P_s(\mathcal{E})$ came out to be 10^{-3}, but transmitting sensitive data requires that $P_s(\mathcal{E}) < 10^{-12}$. That is an opportunity to apply error control coding. There are two main classes of error control codes, block codes and convolutional codes. In the example given above a convolutional code might be used to bring $P_s(\mathcal{E})$ from 10^{-3} down to 10^{-5} or 10^{-6}. A block code could then be used to clean up any remaining errors and achieve the goal of $P_s(\mathcal{E}) = 10^{-12}$.

1.1 Probability of Error

Consider the situation of transmitting a single, discrete-valued symbol a_0 over an additive white Gaussian noise (AWGN) channel, as shown in the following block diagram.



The transmitted signal is $s(t) = a_0\, p(t)$, and the received signal is $r(t) = a_0\, p(t) + n(t)$. The receiver consists of a filter $h_R(t)$, followed by a sampler at $t = 0$ and a decision device that outputs an estimate $\hat{a}_0$ of the (most likely) transmitted symbol a_0. Since

$$b(t) = h_R(t) * \big(a_0\, p(t) + n(t)\big) = a_0 \int_{-\infty}^{\infty} H_R(f)\, P(f)\, e^{j2\pi ft}\, df + \int_{-\infty}^{\infty} h_R(\mu)\, n(t-\mu)\, d\mu\,,$$

it is quite easy to see that, in the absence of noise,

$$E[b_0] = E[a_0] \int_{-\infty}^{\infty} H_R(f)\, P(f)\, df = \gamma\, E[a_0]\,,$$

$$E[|b_0|^2] = E[|a_0|^2] \left| \int_{-\infty}^{\infty} H_R(f)\, P(f)\, df \right|^2 = |\gamma|^2\, E[|a_0|^2]\,,$$

where

$$\gamma = \int_{-\infty}^{\infty} H_R(f)\, P(f)\, df\,,$$

is a (possibly complex-valued) constant. Note that when $h_R(t)$ is a matched filter, then

$$h_R(t) = \frac{p^*(-t)}{\int_{-\infty}^{\infty} |p(\mu)|^2\, d\mu} \quad \Longleftrightarrow \quad H_R(f) = \frac{P^*(f)}{\int_{-\infty}^{\infty} |P(\nu)|^2\, d\nu}\,,$$

and in this case $\gamma = 1$. In the presence of noise alone (setting $a_0 = 0$)

$$R_b(\tau) = \int_{-\infty}^{\infty} S_n(f)\, |H_R(f)|^2\, e^{j2\pi f\tau}\, df \quad \Longrightarrow \quad \sigma_b^2 = R_b(0) = \frac{\mathcal{N}_0}{2} \int_{-\infty}^{\infty} |H_R(f)|^2\, df\,,$$

where $S_n(f) = \mathcal{N}_0/2$ was substituted in the second equation. When $h_R(t)$ is a matched filter, then

$$\sigma_b^2 = \frac{\mathcal{N}_0}{2\int_{-\infty}^{\infty} |p(\mu)|^2\, d\mu} = \frac{\mathcal{N}_0}{2\int_{-\infty}^{\infty} |P(\nu)|^2\, d\nu}\,.$$

Thus, b_0 is a random variable with mean $E[b_0]$ and variance σ_b^2.

Binary Case. Suppose a_0 can only take on two distinct values, say $a_0 \in \{A, B\}$ with equal probability. Then, assuming $n(t)$ is white Gaussian noise, b_0 is a random variable with mean either γA or γB (with the constant γ as defined previously) and variance σ_b^2. The conditional pdfs of b_0, for a given value of a_0 are

$$f_{b_0}(\beta|a_0{=}A) = \frac{1}{\sqrt{2\pi\sigma_b^2}}\, e^{-(\beta-\gamma A)^2/2\sigma_b^2}\,, \quad \text{and} \quad f_{b_0}(\beta|a_0{=}B) = \frac{1}{\sqrt{2\pi\sigma_b^2}}\, e^{-(\beta-\gamma B)^2/2\sigma_b^2}\,.$$

The decision rule for a **maximum likelihood (ML)** receiver upon receiving $b_0 = \beta$ is:

$$\text{Decide } \hat{a}_0{=}A \text{ iff} \qquad f_{b_0}(\beta|a_0{=}A) > f_{b_0}(\beta|a_0{=}B)\,,$$

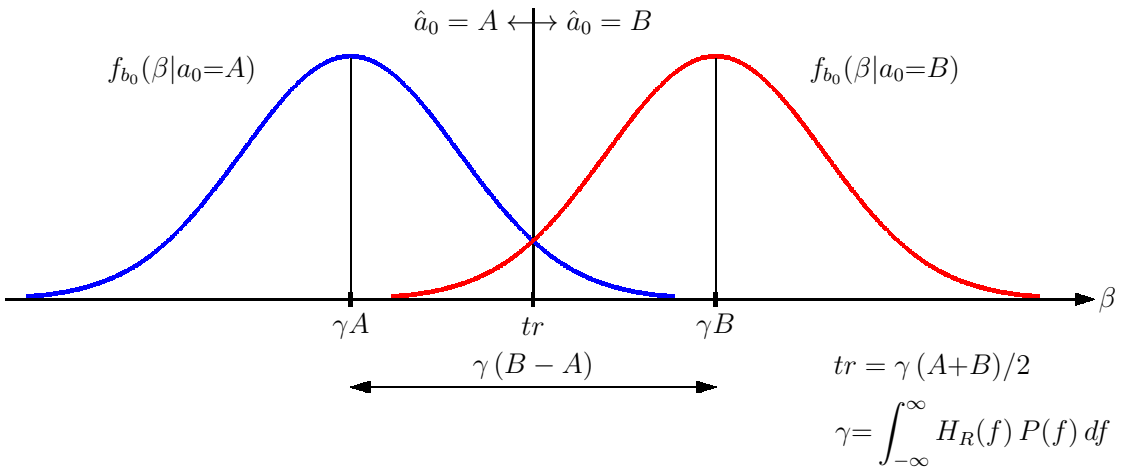
else decide $\hat{a}_0 = B$. Substituting the conditional pdfs yields: Decide $\hat{a}_0 = A$ iff

$$\frac{1}{\sqrt{2\pi\sigma_b^2}}\, e^{-(\beta-\gamma A)^2/2\sigma_b^2} > \frac{1}{\sqrt{2\pi\sigma_b^2}}\, e^{-(\beta-\gamma B)^2/2\sigma_b^2} \quad \Longrightarrow \quad \frac{(\beta-\gamma A)^2}{2\sigma_b^2} < \frac{(\beta-\gamma B)^2}{2\sigma_b^2}\,.$$

Note the reversal of the inequality after taking the logarithm on both sides and dropping the minus sign. Multiplying both sides out and canceling common terms yields: Decide $\hat{a}_0 = A$ iff

$$2\beta\,(B-A) < \gamma\,(B^2 - A^2) \quad \Longrightarrow \quad \beta < tr = \frac{\gamma\,(B+A)}{2}\,.$$

The conditional pdfs, their intersection, and the decision regions for $\hat{a}_0$ are shown graphically in the figure below.



An error is made by the decision device if either $a_0{=}A$ and $\hat{a}_0{=}B$, or $a_0{=}B$ and $\hat{a}_0{=}A$. The symbol error probabilities $P_s(\mathcal{E})$ are defined as follows

$$\begin{aligned} P_s(\mathcal{E}|a_0{=}A) &: \text{ Probability of symbol error given } A \text{ was sent,} \\ P_s(\mathcal{E}|a_0{=}B) &: \text{ Probability of symbol error given } B \text{ was sent,} \\ P_s(\mathcal{E}) = \tfrac{1}{2}\big(P_s(\mathcal{E}|a_0{=}A) + P_s(\mathcal{E}|a_0{=}B)\big) &: \text{ Probability of symbol error.} \end{aligned}$$

Note that the last expression assumes that $a_0{=}A$ and $a_0{=}B$ are equally likely. To compute $P_s(\mathcal{E}|a_0{=}A)$, integrate over the conditional Gaussian pdf $f_{b_0}(\beta|a_0 = A)$ from the decision threshold $tr = \gamma(A+B)/2$ to ∞ to obtain

$$\begin{aligned} P_s(\mathcal{E}|a_0{=}A) &= \int_{tr}^{\infty} f_{b_0}(\beta|a_0{=}A)\,d\beta = \frac{1}{\sqrt{2\pi\sigma_b^2}}\int_{tr}^{\infty} e^{-(\beta-\gamma A)^2/2\sigma_b^2}\,d\beta \\ &= \frac{1}{\sqrt{2\pi}}\int_{(tr-\gamma A)/\sigma_b}^{\infty} e^{-\mu^2/2}\,d\mu = Q\Big(\frac{tr-\gamma A}{\sigma_b}\Big) = Q\Big(\frac{\gamma(B-A)}{2\sigma_b}\Big)\,, \end{aligned}$$

where the substitution

$$\mu = \frac{\beta - \gamma A}{\sigma_b} \qquad \Longrightarrow \qquad \frac{d\mu}{d\beta} = \frac{1}{\sigma_b}\,,$$

was used for the second line and $tr = \gamma(A+B)/2$ was used for the last equality. The Q **function** that was used on the second line is defined as and upper bounded by

$$Q(x) = \frac{1}{\sqrt{2\pi}}\int_x^{\infty} e^{-\mu^2/2}\,d\mu\,, \qquad Q(x) \le \frac{1}{2}\,e^{-x^2/2}\,, \quad x \ge 0\,.$$

Note that

$$1 - Q(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-\mu^2/2}\,d\mu = \frac{1}{\sqrt{2\pi}}\int_{\infty}^{-x} e^{-\nu^2/2}\,(-d\nu) = \frac{1}{\sqrt{2\pi}}\int_{-x}^{\infty} e^{-\nu^2/2}\,d\nu = Q(-x)\,,$$

where the substitution $\nu = -\mu$ was used for the second equality. The Q function is related to the **complementary error function erfc(x)** by

$$Q(x) = \frac{1}{2}\,\mathrm{erfc}\Big(\frac{x}{\sqrt{2}}\Big)\,, \qquad \text{where} \qquad \mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-\mu^2}\,d\mu\,.$$

For $x \geq 0$ the complementary error function is upper bounded by $\mathrm{erfc}(x) \leq e^{-x^2}$.

Returning to the computation of $P_s(\mathcal{E})$, the conditional error probability $P_s(\mathcal{E}|a_0{=}B)$ is obtained as

$$\begin{aligned} P_s(\mathcal{E}|a_0{=}B) &= \int_{-\infty}^{tr} f_{b_0}(\beta|a_0{=}B)\,d\beta = \frac{1}{\sqrt{2\pi\sigma_b^2}}\int_{-\infty}^{tr} e^{-(\beta-\gamma B)^2/2\sigma_b^2}\,d\beta \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{(tr-\gamma B)/\sigma_b} e^{-\mu^2/2}\,d\mu = 1 - Q\Big(\frac{tr-\gamma B}{\sigma_b}\Big) = Q\Big(\frac{\gamma B - tr}{\sigma_b}\Big) \\ &= Q\Big(\frac{\gamma(B-A)}{2\sigma_b}\Big) = P_s(\mathcal{E}|a_0{=}A)\,. \end{aligned}$$

Therefore, the unconditional probability of symbol (bit) error in the case of binary equally likely symbols $a_0 \in \{A, B\}$ is

$$P_s(\mathcal{E}) = Q\Big(\frac{\gamma D_{AB}}{2\sigma_b}\Big) \leq \frac{1}{2}\,e^{-\gamma^2 D_{AB}^2/8\sigma_b^2}\,, \qquad D_{AB} = |B-A|\,,$$

with

$$\gamma = \int_{-\infty}^{\infty} H_R(f)P(f)\,df\,, \qquad \text{and} \qquad \sigma_b^2 = \frac{\mathcal{N}_0}{2}\int_{-\infty}^{\infty}|H_r(f)|^2\,df\,,$$

where $p(t) \Leftrightarrow P(f)$ is the PAM pulse as seen by the receiver and $h_R(t) \Leftrightarrow H_R(f)$ is the receiver filter. In the case of a matched filter this becomes

$$H_R(f) = \frac{P^*(f)}{\int_{-\infty}^{\infty}|P(\nu)|^2\,d\nu} \qquad \Longrightarrow \qquad P_s(\mathcal{E}) = Q\Big(\frac{D_{AB}}{2\sigma_b}\Big) \leq \frac{1}{2}\,e^{-D_{AB}^2/8\sigma_b^2}\,.$$

The crucial observation to be made here is that the probability of symbol error depends only on the **distance** $D_{AB} = |B-A|$ between the signal points A and B (as seen at the receiver input by a normalized matched filter at the optimum sampling time) and the noise power σ_b^2 at the output of the receiver filter.

***M*-ary Case.** Assume now that a_0 can take on M different values $A_1, A_2, \ldots, A_M$ with equal probability. In this case the decision rule when receiving $b_0 = \beta$ is: Decide $\hat{a}_0 = A_i$ iff

$$f_{b_0}(\beta|a_0{=}A_i) > f_{b_0}(\beta|a_0{=}A_j) \quad \text{for all } j \neq i\,.$$

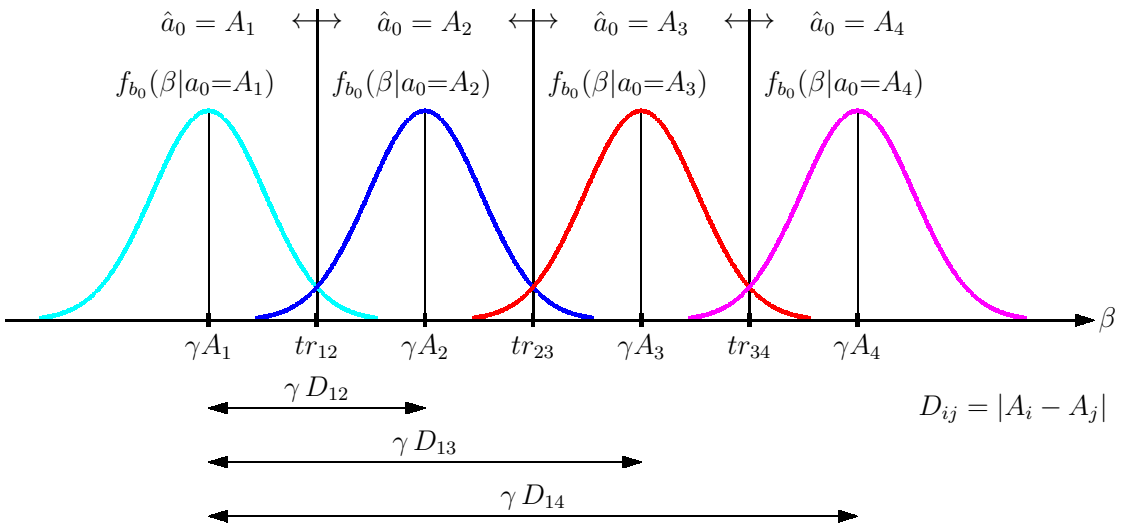
In additive white Gaussian noise this translates into: Decide $\hat{a}_0 = A_i$ iff

$$(\beta - \gamma A_i)^2 < (\beta - \gamma A_j)^2 \qquad \Longrightarrow \qquad D(\beta, \gamma A_i) < D(\beta, \gamma A_j) \quad \text{for all } j \neq i\,,$$

where

$$D(x,y) = |x-y|\,,$$

denotes the (Euclidean) distance between two points x and y. A graphical illustration for $M=4$ is shown in the following figure.



Thus, when receiving $b_0 = \beta$, the decision device measures the distance $|\beta - \gamma A_j|$ between β and all possible signal values γA_j and decides $\hat{a}_0 = A_i$ iff $|\beta - \gamma A_i|$ is the shortest distance.

The conditional probability of symbol error given $a_0{=}A_i$ is transmitted and $b_0{=}\beta$ is received is

$$\begin{aligned} P_s(\mathcal{E}|a_0{=}A_i) &= Pr\Big\{ \bigcup_{\substack{j=1\\ j\neq i}}^{M} D(\beta,\gamma A_j) < D(\beta,\gamma A_i)\big|a_0{=}A_i\Big\} \\ &\leq \sum_{\substack{j=1\\ j\neq i}}^{M} Pr\big\{D(\beta,\gamma A_j) < D(\beta,\gamma A_i)\big|a_0{=}A_i\big\} = \sum_{\substack{j=1\\ j\neq i}}^{M} Q\Big(\frac{\gamma\, D_{ij}}{2\sigma_b}\Big)\,, \end{aligned}$$

where $D_{ij} = |A_i - A_j|$. The upper bound is obtained by using the **union bound**, which says that for two (or more) events E_1 and E_2

$$Pr\{E_1 \cup E_2\} \leq Pr\{E_1\} + Pr\{E_2\}\,,$$

with equality iff E_1 and E_2 are mutually exclusive. The last equality in the expression for $P_s(\mathcal{E}|a_0{=}A_i)$ comes from recognizing that $Pr\big\{D(\beta,\gamma A_j) < D(\beta,\gamma A_i)\big|a_0{=}A_i\big\}$ is the same as the conditional probability of a symbol error in the binary case with $A = A_i$ and $B = A_j$. Finally, the unconditional probability of symbol error for equally likely M-ary signals can be expressed as

$$P_s(\mathcal{E}) = \frac{1}{M}\sum_{i=1}^{M} P_s(\mathcal{E}|a_0{=}A_i) \leq \frac{1}{M}\sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M} Q\Big(\frac{\gamma\, D_{ij}}{2\sigma_b}\Big) \approx \frac{2K}{M}\, Q\Big(\frac{\gamma\, D_{\min}}{2\sigma_b}\Big)\,,$$

where $D_{\min}$ is the minimum distance between any two signal points in the signal set and K is the number of distinct signal pairs which are $D_{\min}$ apart, e.g., $D_{\min} = 2$ and $K = 3$ if $M = 4$ and $A_i \in \{-3, -1, +1, +3\}$. The approximation is justified since

$$Q(x_1) \gg Q(x_2) \qquad \text{if} \quad x_1 < x_2 \,.$$

For example, $Q(2\sqrt{2}) = 0.0023$, but $Q(4) = 0.0000317$.

1.2 $E_b/\mathcal{N}_0$ and Normalized Minimum Distance

Consider again the block diagram of transmitting a single symbol $a_0 \in \{A_1, A_2, \ldots A_M\}$ using PAM with a pulse $p(t)$ over a AWGN channel. Assume that the receiver filter $h_R(t)$ is a matched filter, i.e.,

$$h_R(t) = \frac{p^*(-t)}{\int_{-\infty}^{\infty} |p(\mu)|^2 \, d\mu} \quad \Longleftrightarrow \quad H_R(f) = \frac{P^*(f)}{\int_{-\infty}^{\infty} |P(\nu)|^2 \, d\nu} \,,$$

so that

$$E[b_0] = E[a_0] \,, \qquad E[|b_0|^2] = E[|a_0|^2] \,, \qquad \text{and} \qquad \sigma_b^2 = \frac{\mathcal{N}_0}{2\int_{-\infty}^{\infty} |p(\mu)|^2 \, d\mu} \,.$$

A natural definition for signal-to-noise ratio (SNR) that is also easy to measure in practice is

$$\text{SNR:} \qquad \frac{E_b}{\mathcal{N}_0} = \frac{E_s}{\mathcal{N}_0 \, \log_2 M} \,,$$

where E_b is the average energy per bit, E_s is the average symbol energy, M is the number of possible symbols, and $\mathcal{N}_0$ is the PSD (in W/Hz) of the white Gaussian noise. In the absence of noise the average symbol energy at the receiver input is

$$E_s = E\Big[\int_{-\infty}^{\infty} |a_0 \, p(t)|^2 \, dt\Big] = E[|a_0|^2] \int_{-\infty}^{\infty} |p(t)|^2 \, dt \qquad \Longrightarrow \quad \int_{-\infty}^{\infty} |p(t)|^2 \, dt = \frac{E_s}{E[|a_0|^2]} \,.$$

Thus, the noise power can be expressed as

$$\sigma_b^2 = \frac{\mathcal{N}_0 \, E[|a_0|^2]}{2E_s} = \frac{\mathcal{N}_0 \, E[|a_0|^2]}{2E_b \, \log_2 M} \,.$$

Normalized Minimum Distance d_{min}. Consider first the binary case with $a_0 \in \{A, B\}$. Since there are only two signals the (actual, unnormalized) minimum distance is clearly

$$D_{\min} = D_{AB} = |A - B| \,.$$

Thus, by making $|A-B|$ large, $P_s(\mathcal{E}) = Q\big(D_{\min}/(2\sigma_b)\big)$ can be made as small as desired. The catch, however, is that this increases the average energy $E[|a_0|^2] = (|A|^2+|B|^2)/2$ (assuming a_0=A and a_0=B are equally likely) required when using the signal set. To be able to make

fair comparisons between different signal sets, distances should therefore be normalized so that they become a property of the geometric arrangement of the points in the signal set, independent of scaling factors. This leads to the following definition.

Definition: For binary signals $a_0 \in \{A, B\}$ the normalized minimum distancce $d_{\min}$ is defined as

$$d_{\min} = \frac{D_{\min}}{\sqrt{E[|a_0|^2]}}\,, \qquad \text{where} \quad D_{\min} = |A - B|\,.$$

For equally likely signals $E[|a_0|^2] = (|A|^2 + |B|^2)/2$. For polar binary signaling ($A = -B$) with equally likely signals $d_{\min} = 2$.

For M-ary signals it is desirable to normalize the signal set distances in such a way that comparisons can be made between signal sets with different numbers of signals. Thus, distance is normalized with respect to average energy per bit, rather than directly with respect to the average symbol energy $E[|a_0|^2]$, as specified in the next definition.

Definition: For M-ary signals $a_0 \in \{A_1, A_2, \ldots, A_M\}$ the normalized (with respect to average energy per bit) minimum distance is defined as

$$d_{\min} = \frac{D_{\min}}{\sqrt{E[|a_0|^2]/\log_2 M}}\,, \qquad \text{where} \quad D_{\min} = \min_{i,j \neq i} |A_i - A_j|\,.$$

For equally likely signals $E[|a_0|^2] = (|A_1|^2 + |A_2|^2 + \ldots + |A_M|^2)/M$. For equidistant polar 4-ary signaling ($A_1, A_4 = \pm 3V$, $A_2, A_3 = \pm V$ for some V) with equally likely signals $d_{\min} = 2\sqrt{2}/\sqrt{(}5) = 1.265$.

Probability of Error in Terms of $d_{\min}$ and $E_b/\mathcal{N}_0$. Using the expression

$$\sigma_b^2 = \frac{\mathcal{N}_0\, E[|a_0|^2]}{2E_b \log_2 M} \qquad \Longrightarrow \qquad 2\sigma_b = \sqrt{\frac{2\mathcal{N}_0}{E_b}}\sqrt{\frac{E[|a_0|^2]}{\log_2 M}}\,,$$

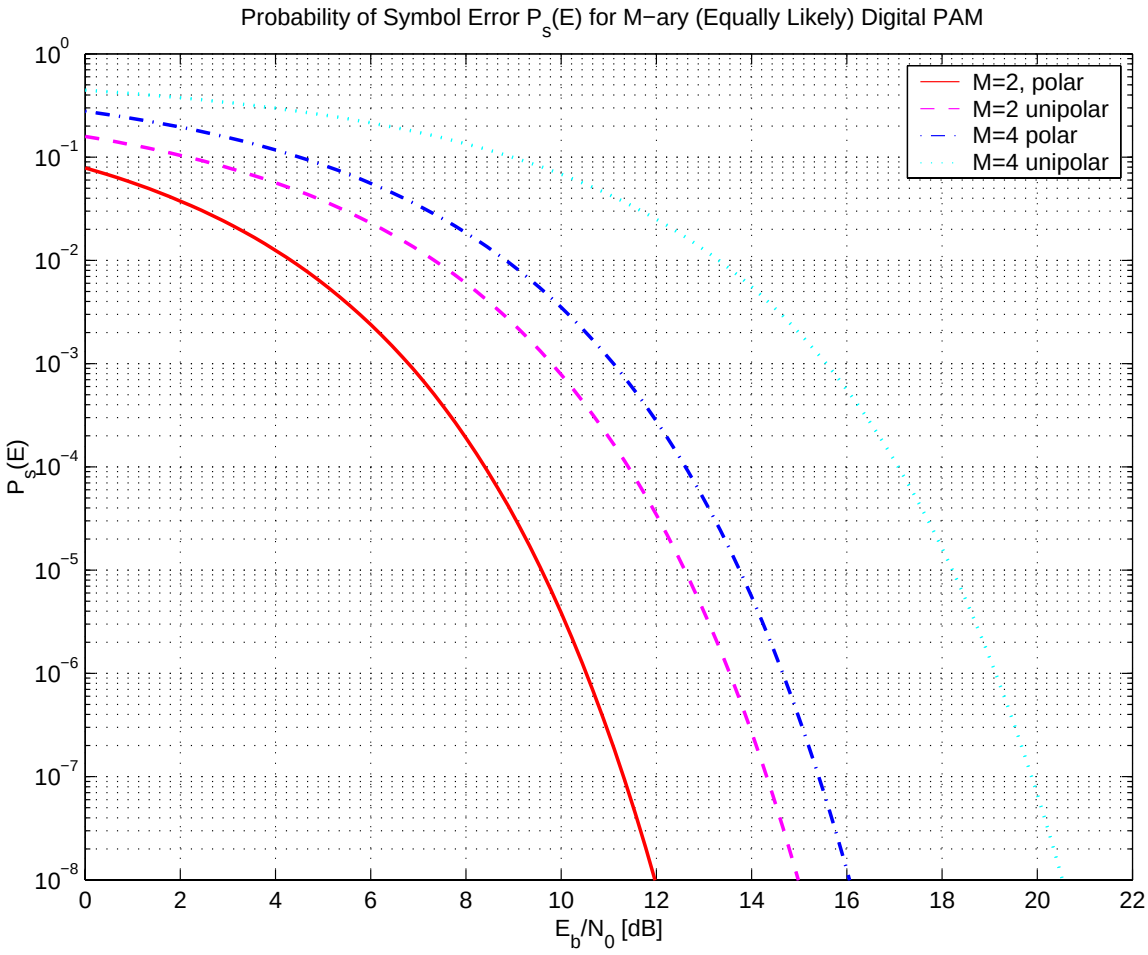
in the formula for $P_s(\mathcal{E})$ in the binary case with equally likely signals yields

$$P_s(\mathcal{E}) = Q\Big(\frac{D_{\min}}{2\sigma_b}\Big) \qquad \Longrightarrow \qquad P_s(\mathcal{E}) = Q\Big(d_{\min}\sqrt{\frac{E_b}{2\mathcal{N}_0}}\Big)\,.$$

For the M-ary case with equally likely signals $P_s(\mathcal{E})$ becomes

$$P_s(\mathcal{E}) \leq \frac{2K}{M}\, Q\Big(\frac{D_{\min}}{2\sigma_b}\Big) \qquad \Longrightarrow \qquad P_s(\mathcal{E}) \leq \frac{2K}{M}\, Q\Big(d_{\min}\sqrt{\frac{E_b}{2\mathcal{N}_0}}\Big)\,,$$

where K is the number of distinct pairs of signals normalized distance $d_{\min}$ apart. The following graph shows $P_s(\mathcal{E})$ for polar and unipolar digital binary PAM (exact $P_s(\mathcal{E})$) and for equidistant polar 4-ary digital PAM (upper bound on $P_s(\mathcal{E})$) versus $E_b/\mathcal{N}_0$.



Note that polar binary signaling performs about 3dB better in $E_b/\mathcal{N}_0$ than unipolar binary, about 4dB better than polar equidistant $M = 4$ signaling, and almost 9dB better than unipolar equidistant $M = 4$ signaling.

1.3 Introduction to Error Control Coding

Error control coding schemes add redundancy to a transmitted data sequence. Linear codes typically generate n code symbols for every k information or data symbols. Thus, the code rate is $R = k/n < 1$ and the goal is to use the redundancy $r = n - k$ to detect and/or correct transmission errors. There are two major classes of linear codes, **block codes** and **convolutional codes**. Linear block codes are specified by giving their generator matrix $\mathbf{G}$ which is a $k \times n$ matrix, e.g.,

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix},$$

defines a binary block code of rate $R = 1/2$ with $n = 6$ and $k = 3$. The codewords $\mathbf{c}$ of this code are obtained from the datawords $\mathbf{d}$ by computing $\mathbf{c} = \mathbf{d}\,\mathbf{G}$ where, since the code

is binary, all operations are performed modulo 2 (i.e., $0+0=1+1=2$, $0+1=1+0=1$, $0\cdot 0=0\cdot 1=1\cdot 0=0$, $1\cdot 1=1$). As an example, the codewords of the code whose $\mathbf{G}$ is given above are

d	**c**	**d**	**c**
000	000000	100	100110
001	001011	101	101101
010	010101	110	110011
011	011110	111	111000

The single most important goodness criterion of a code is its **minimum distance**, i.e., the smallest distance (according to some distance measure) between any two distinct codewords $\mathbf{c}_i$ and $\mathbf{c}_j$ of the code. A popular distance measure is defined as follows.

Definition: The **Hamming distance** $d_H(\mathbf{c}_i, \mathbf{c}_j)$ between two vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ of the same length is equal to the number of positions in which $\mathbf{c}_i$ and $\mathbf{c}_j$ differ. For instance, $d_H(010101, 011110) = 3$.

A related quantity is defined next.

Definition: The **Hamming weight** $w_H(\mathbf{c})$ of a vector $\mathbf{c}$ is the number of nonzero components of $\mathbf{c}$. For example, $w_H(110011) = 4$.

For a linear blockcode the minimum Hamming distance $d_{\min}$ between any two nonzero codewords of the code is equal to the minimum weight of any nonzero codeword of the code. For the binary code whose $\mathbf{G}$ was given above $d_{\min} = 3$, which means that the code can be used to **detect** any single or double transmission error, or to **correct** any single transmission error. In general, a code with minimum distance d_{min} can either detect all patterns of $d_{\min} - 1$ or fewer errors, or it can correct all patterns of $\lfloor (d_{\min} - 1)/2 \rfloor$ or fewer errors.

The other big class of codes, the convolutional codes, passes k semi-infinte data sequences or vectors $\mathbf{d}^{(1)}, \ldots, \mathbf{d}^{(k)}$ through a set of $k \times n$ filters to produce n code sequences or vectors $\mathbf{c}^{(1)}, \ldots, \mathbf{c}^{(n)}$ which are also semi-infinite. The convolution between the data sequences and the $k \times n$ filter matrix is what gives these codes their name. The important special class of convolutional codes when $k = 1$ is treated in the following section.

1.4 Convolutional Codes

A general convolutional encoder has k inputs and n outputs and is characterized by a transfer function matrix $\mathbf{G}(D)$ of size $k \times n$. Such an encoder produces a convolutional code with rate $R = k/n < 1$ and redundancy $r = n - k > 0$. Here only the important special case of binary rate $1/n$ convolutional codes is considered. In this case $\mathbf{G}(D)$ is of the form

$$\mathbf{G}(D) = \left[g^{(1)}(D) \; g^{(2)}(D) \; \ldots \; g^{(n)}(D) \right],$$

where each of the **generator polynomials** $g^{(\ell)}(D)$ is a polynomial of degree m in the indeterminate D, i.e.,

$$g^{(\ell)}(D) = g_0^{(\ell)} + g_1^{(\ell)}\, D + \ldots + g_m^{(\ell)}\, D^m = \sum_{i=0}^{m} g_i^{(\ell)}\, D^i \,.$$

The largest degree m of any of the $g^{(\ell)}(D)$ polynomials in $\mathbf{G}(D)$ is called the **maximal memory order** of the encoder. The constraint length K of a convolutional encoder is generally defined as $K = m + 1$, where m is the maximal memory order. The D stands for "delay" and plays the same role as z^{-1} for discrete time systems. In fact, to relate this back to linear systems, you can think of the polynomials $g^{(\ell)}(D)$ as system functions and, if you replace D by z^{-1} you can write

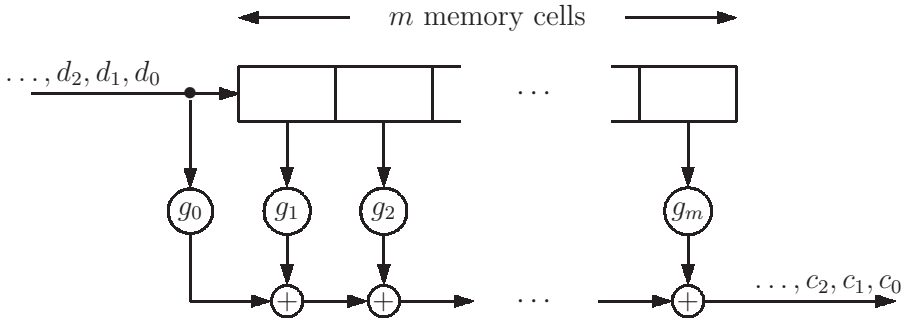
$$g^{(\ell)}(z^{-1}) = g_0^{(\ell)} + g_1^{(\ell)}\, z^{-1} + \ldots + g_m^{(\ell)}\, z^{-m} = \sum_{i=0}^{m} g_i^{(\ell)}\, z^{-i} \,.$$

Note however that, since the codes considered here are binary, the coefficients $g_i^{(\ell)}$ are binary, i.e., $g_i^{(\ell)} \in \{0, 1\}$.

Next, let d_i with $d_i \in \{0, 1\}$ denote a binary data sequence and let $\{g_i\} = \{g_0, g_1, \ldots, g_m\}$ denote the sequence of coefficients of one of the generator polynomials $g(D)$. Then a code sequence c_i is obtained from the convolution

$$c_i = d_i * g_i = \sum_{j=0}^{m} g_j\, d_{i-j} \,, \quad i = 0, 1, 2, \ldots \,, \qquad \text{where} \quad d_\ell = 0 \text{ for } \ell < 0 \,.$$

Note that, for a binary code, this convolution is computed **modulo 2**. The convolution $d_i * g_i$ can be implemented conveniently using a shift register of length m as shown in the figure below.



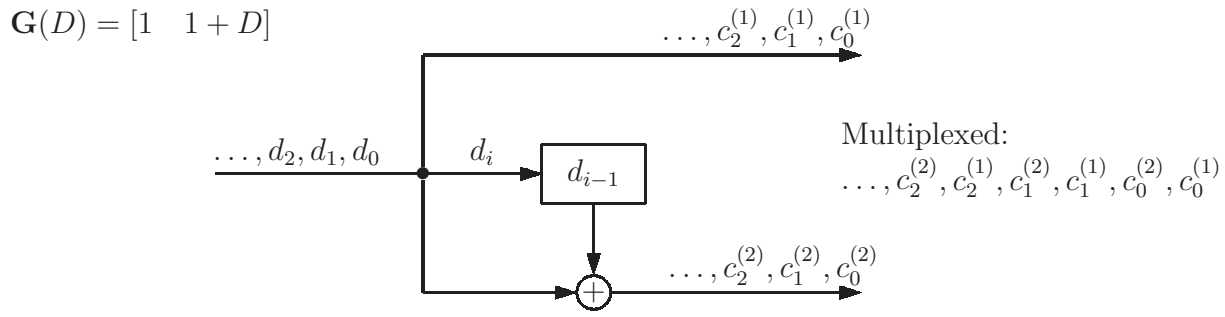
For a rate 1/2 convolutional code the transfer function matrix is $\mathbf{G}(D) = [g^{(1)}(D)\ g^{(2)}(D)]$, and thus two code sequences $c_i^{(1)}$ and $c_i^{(2)}$ of the same length as the data sequence d_i are produced as follows

$$c_i^{(1)} = \sum_{j=0}^{m} g_j^{(1)}\, d_{i-j} \,, \qquad c_i^{(2)} = \sum_{j=0}^{m} g_j^{(2)}\, d_{i-j} \,, \qquad i = 0, 1, 2, \ldots \,,$$

where again $d_\ell = 0$ for $\ell < 0$. The code sequences $c_i^{(1)}$ are then multiplexed to form a single code sequence c_i of twice the length of the data sequence, i.e.,

$$\{c_i\} = \{c_0^{(1)}, c_0^{(2)}, c_1^{(1)}, c_1^{(2)}, c_2^{(1)}, c_2^{(2)}, \ldots\} .$$

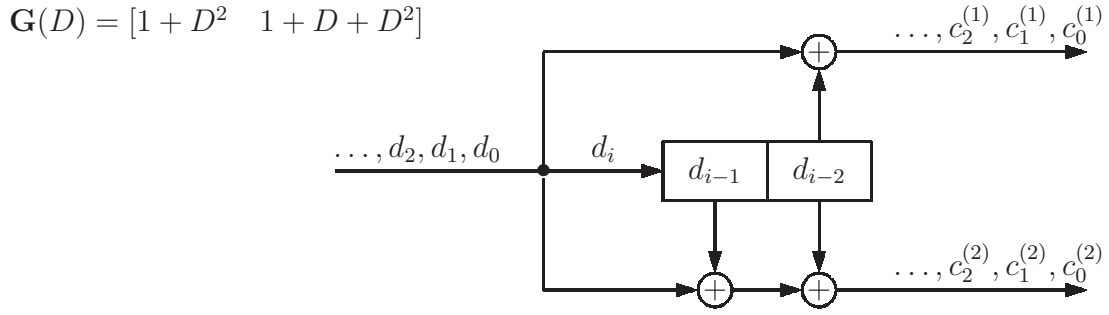
The simplest example of a binary convolutional encoder is shown in the next figure.



Examples of data sequences and resulting code sequences (after multiplexing) for this encoder are:

$\{d_i\}$	$\{c_i\}$	$w(\{c_i\})$
$000\ldots$	$000000\ldots$	0
$100\ldots$	$110100\ldots$	3
$0100\ldots$	$00110100\ldots$	3
$1100\ldots$	$11100100\ldots$	4
$00100\ldots$	$0000110100\ldots$	3
$10100\ldots$	$1101110100\ldots$	6
$11100\ldots$	$1110100100\ldots$	5

The last column shows the (Hamming) weight of the codewords. Since convolutional codes are linear, the minimum weight is equal to the minimum distance of the code. For the above code the **minimum free distance**, as it is called for convolutional codes, is $d_{free} = 3$. Another example of a rate $R = 1/2$ convolutional encoder is shown in the next figure.

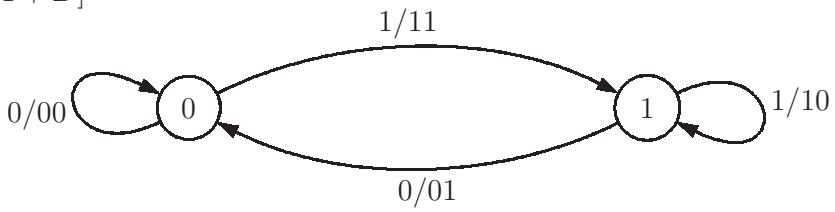


This code has more memory, and thus it has the potential to perform better (i.e., detect and/or correct more errors) than the previous one that had only one memory cell. Indeed, looking at a few data sequences and the resulting code sequences shows that $w(\{c_i\}) \geq 5$ for nonzero $\{c_i\}$:

$\{d_i\}$	$\{c_i\}$	$w(\{c_i\})$
$0000\ldots$	$00000000\ldots$	0
$1000\ldots$	$11011100\ldots$	5
$01000\ldots$	$0011011100\ldots$	5
$11000\ldots$	$1110101100\ldots$	6
$001000\ldots$	$000011011100\ldots$	5
$101000\ldots$	$110100011100\ldots$	6
$111000\ldots$	$111001101100\ldots$	7

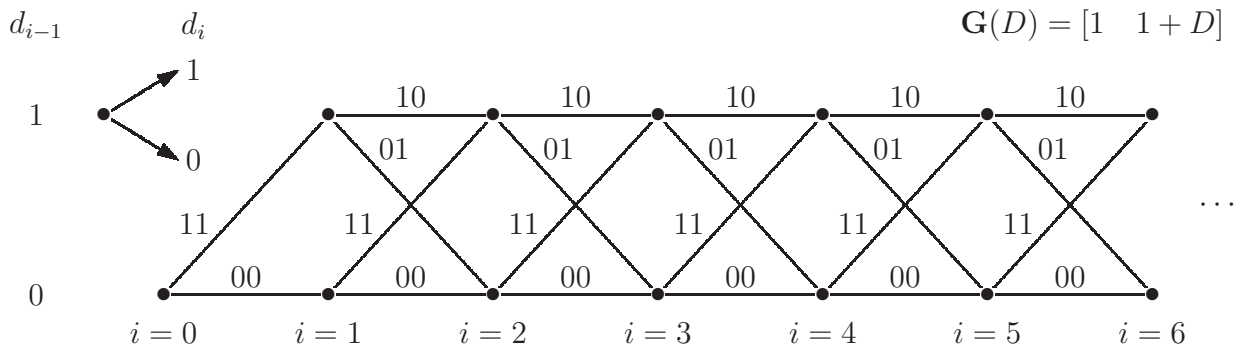
Convolutional encoders have a finite number of memory cells and, if the input is restricted to a finite number of possible values, they can be modeled as **finite state machines**. The binary encoder with a single memory cell has just two states, either 0 or 1. The following figure shows the state transition diagram for this encoder.

$\mathbf{G}(D) = [1 \quad 1+D]$

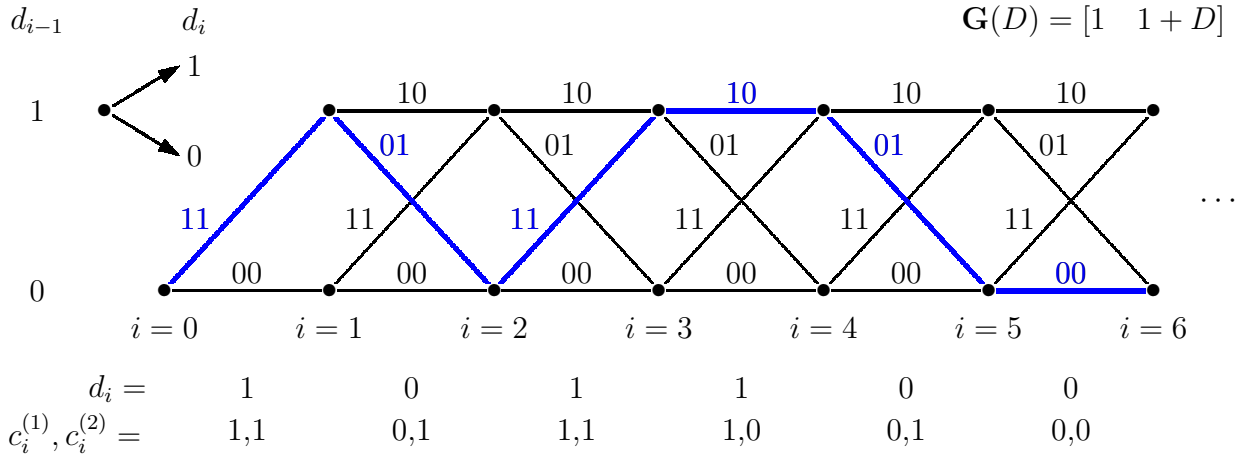


The transitions are labeled in the form $d_i/c_i^{(1)}, c_i^{(2)}$. Thus, to encode the data sequence $d_i = 1, 1, 0, 0, \ldots$ you start in state 0, then you go to state 1, outputting code bits 11 along the way. Then, for the second one in the data string, you go along the self-loop at state 1, outputting code bit 10. The first 0 in the data sequence then gets you back from state 1 to state 0, while producing code bits 01. The second 0 in d_i then corresponds to using the self-loop around state 0, which produces code bits 00. Thus, the data sequence $1, 1, 0, 0, \ldots$ gets encoded into code sequence $1, 1, 1, 0, 0, 1, 0, 0, \ldots$.

The encoder state transition diagram is a compact way to characterize the encoder and to generate code sequences from data sequences. But sometimes it is useful to be able to see the different encoder states and transitions between states expanded over time. A nice graphical tool to do this is a trellis diagram as shown in the figure below for the binary convolutional encoder with a single memory cell.



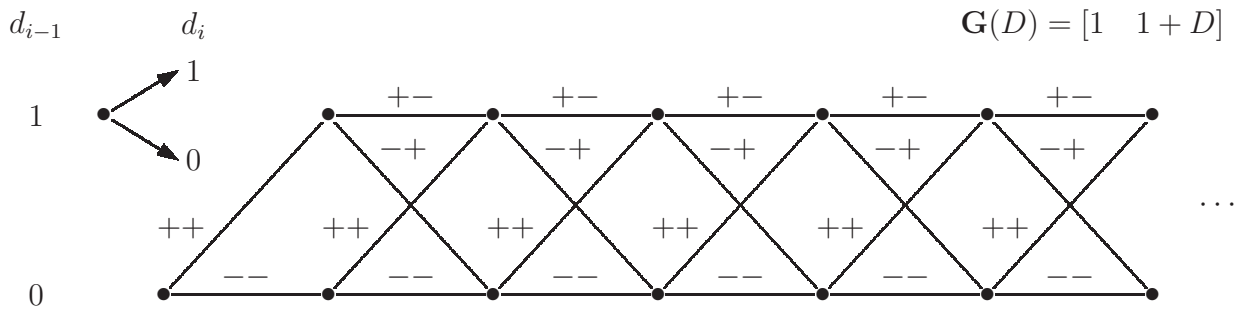
The encoder state d_{i-1} is labeled on the left side of the trellis. The lower row of dots corresponds to state 0 and the upper row corresponds to state 1. The branches between states are labeled with the codeword bits that the encoder outputs when it makes the corresponding transition between states. In any state, when the data input is a 1, the encoder leaves the state along the upper branch and produces the corresponding code bits. Conversely, when the data input is a 0, the lower branch leading out of the state is taken and the code bits associated with that branch are output. In the following trellis the path corresponding to the data sequence $d_i = 1, 0, 1, 1, 0, 0, \ldots$ is highlighted.



For convolutional encoders with more memory cells the trellis can become quite complex because the number of states grows exponentially with a inear increase in encoder memory. Nevertheless, the trellis is a very valuable tool not only for encoding, but also for decoding convolutional codes, as shown in the next section.

1.5 Viterbi Algorithm

One of the advantages of using convolutional codes (rather than block codes) is that **soft-decision decoding**, i.e., decoding based directly on the unquantized sampled values b_i after matched filtering is relatively easy to accomplish. The trellis diagram is a very convenient tool for organizing the decoding process. Assuming polar binary ($0 \rightarrow -1$, $1 \rightarrow +1$) signaling, the first step is to relabel the trellis with $-$ for logical 0 and $+$ for logical 1 as shown below for the encoder with transfer function matrix $\mathbf{G}(D) = [1 \; 1+D]$.



If all possible code sequences are equally likely, then the task of a ML decoder is to find the most likely one. If $\underline{A}^{(\ell)}$, $\ell = 1, 2, 3, \ldots$ denotes "legal" polar code sequences (corresponding to a path in the trellis) and $\hat{\underline{a}}$ denotes the estimated polar code sequence, then the ML decision rule is: Upon receiving $\underline{b} = \underline{\beta}$, decide $\hat{\underline{a}}{=}\underline{A}^{(\ell)}$ iff

$$f_{\underline{b}}(\underline{\beta}|\underline{a}{=}\underline{A}^{(\ell)}) > f_{\underline{b}}(\underline{\beta}|\underline{a}{=}\underline{A}^{(j)}) \qquad \text{for all } j \neq \ell\,.$$

If the noise is Gaussian, then $f_{\underline{b}}(\underline{\beta}|\underline{a}{=}\underline{A}^{(\ell)})$ is a multivariate Gaussian density function. Assuming additive white Gaussian noise (AWGN), the noise samples in the sequence b_i are statistically independent which implies that

$$f_{\underline{b}}(\underline{\beta}|\underline{a}{=}\underline{A}^{(\ell)}) = \prod_{i\geq 0} f_{b_i}(\beta_i|a_i{=}A_i^{(\ell)})\,.$$

That is, the multivariate Gaussian density becomes a product of one-dimensional Gaussian random variables. Taking the negative logarithm on both sides converts the product into a sum

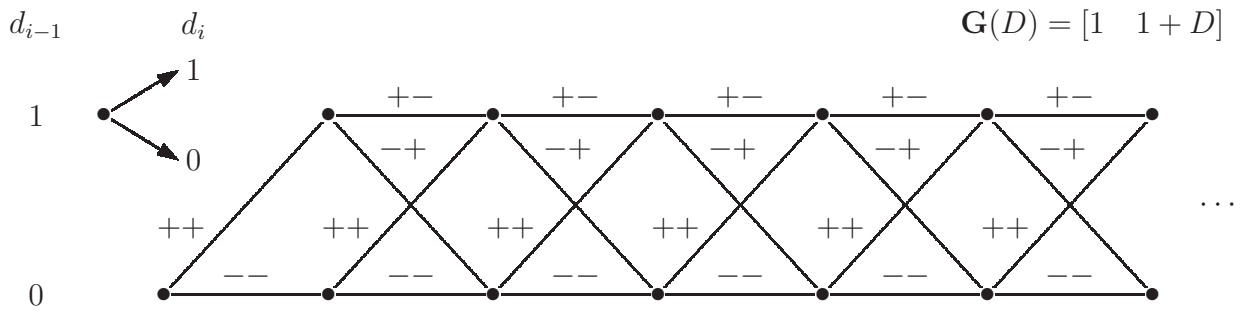
$$-\log\left(f_{\underline{b}}(\underline{\beta}|\underline{a}{=}\underline{A}^{(\ell)})\right) = \sum_{i\geq 0} -\log\left(f_{b_i}(\beta_i|a_i{=}A_i^{(\ell)})\right)\,.$$

Apart from the proportionality factor $1/\sqrt{2\pi\sigma_b^2}$, the negative logarithm of a one-dimensional Gaussian density function with mean A_i and variance σ_b^2 is $(\beta - A_i)^2/(2\sigma_b^2)$. Therefore, the ML decision rule can be rephrased as: Upon receiving $\underline{b} = \underline{\beta}$, decide $\hat{\underline{a}}{=}\underline{A}^{(\ell)}$ iff

$$DM(\underline{\beta}, \underline{A}^{(\ell)}) = \sum_{i\geq 0} (\beta_i - A_i^{(\ell)})^2 < \sum_{i\geq 0} (\beta_i - A_i^{(j)})^2 = DM(\underline{\beta}, \underline{A}^{(j)}) \qquad \text{for all } j \neq \ell\,.$$

The quantity $= DM(\underline{\beta}, \underline{A}^{(\ell)})$ is called a **distance metric** because it measures the (Euclidean) distance between $\underline{\beta}$ and $\underline{A}^{(\ell)}$. The distance metric is the sum of the **branch metrics** $(\beta_i - A_i^{(\ell)})^2$.

The **Viterbi algorithm** is a ML decoding algorithm for convolutional codes that uses the trellis diagram of the encoder to organize the computation of the distance metrics $= DM(\underline{\beta}, \underline{A}^{(j)})$ for all (polar) code sequences $\underline{A}^{(j)}$ that are competitors for the smallest distance metric. To get started for the rate 1/2 code with $\mathbf{G}(D) = [1\ 1{+}D]$, write the received samples b_i pairwise under the trellis diagram as shown in the example below.



$b_i^{(1)}, b_i^{(2)} =$ $+0.2, +0.7,\ -0.9, +2.0,\ +0.2, +0.8,\ +0.1, -0.4,\ +0.1, +0.7,\ -0.1, -1.4$

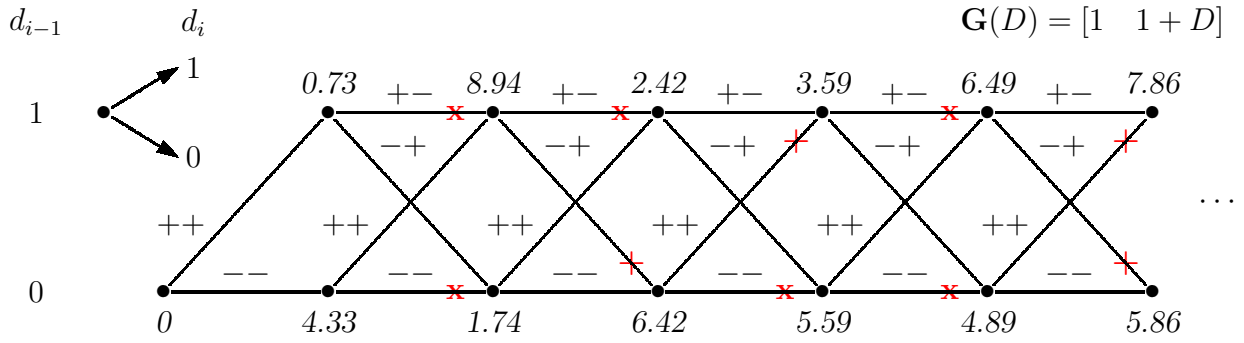
Then, for each section of the trellis, compute all possible branch metrics which take the form $(b_i^{(1)} - A_{2i}^{(j)})^2 + (b_i^{(2)} - A_{2i+1}^{(j)})^2$. Thus, for the first section of the trellis the branch metrics from bottom to top are

$$(0.2 - (-1))^2 + (0.7 - (-1))^2 = 4.33 \quad \text{and} \quad (0.2 - 1)^2 + (0.7 - 1)^2 = 0.73\,.$$

For the second section one similarly obtains (from bottom to top)

$$(-0.9 - (-1))^2 + (2.0 - (-1))^2 = 9.01\,, \qquad (-0.9 - 1)^2 + (2.0 - 1)^2 = 4.61\,,$$
$$(-0.9 - (-1))^2 + (2.0 - 1)^2 = 1.01\,, \qquad (-0.9 - 1)^2 + (2.0 - (-1))^2 = 12.61\,.$$

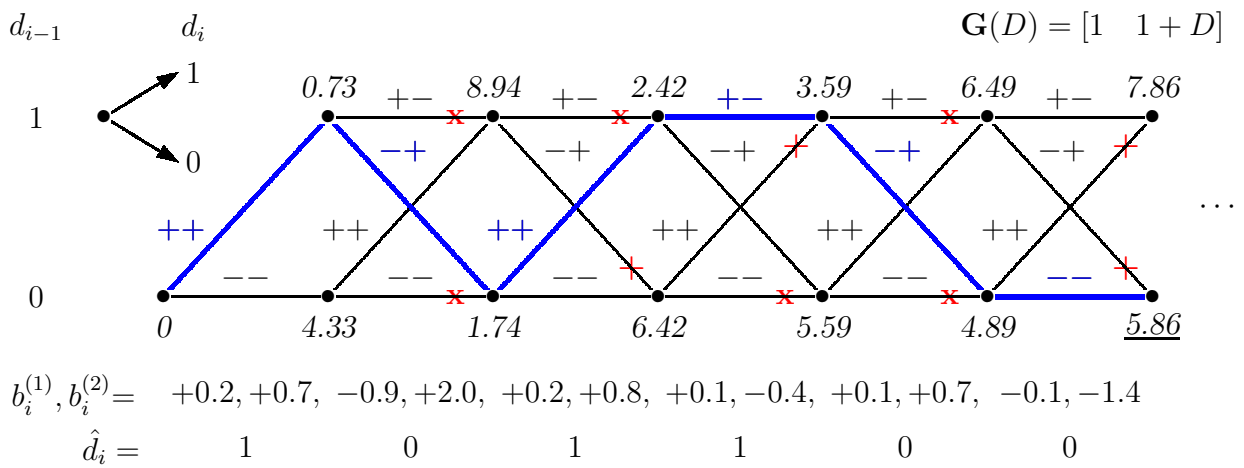
The Viterbi algorithm computes **partial** distance metrics on a section-by-section basis of the trellis by adding the current branch metrics to the previous partial distance metrics. Every time two branches merge into a node, only the path with the smaller partial distance metric survives and the other path is crossed out. This is shown in the following figure where the numbers above the nodes are the partial distance metrics of the surviving path that leads to the node.



$b_i^{(1)}, b_i^{(2)} =$ $+0.2, +0.7,\ -0.9, +2.0,\ +0.2, +0.8,\ +0.1, -0.4,\ +0.1, +0.7,\ -0.1, -1.4$

Initially, the partial distance metric is set to 0. At the end of the first section the partial distance metrics are 4.33 at the bottom and 0.73 at the top. These are just simply the branch metrics of the first section that were computed above. For the partial distance metric at the end of the second section there are two competitors. At the bottom these are

4.33+9.01=13.34 and 0.73+1.01=1.74. The second metric is smaller and this path survives while the other one is crossed out. At the top the competing metrics are 4.33+4.61=8.94 and 0.73+12.61=13.34. Here the first metric is smaller and thus the top branch is crossed out. The algorithm continues in this fashion until the end of the received sequence is reached. Then the path with the smallest metric, 5.86 in this example, is declared the winner. To determine the winning code sequence and thus the most likely transmitted data sequence, trace back through the trellis from the node with the smallest metric, as shown in the next figure.



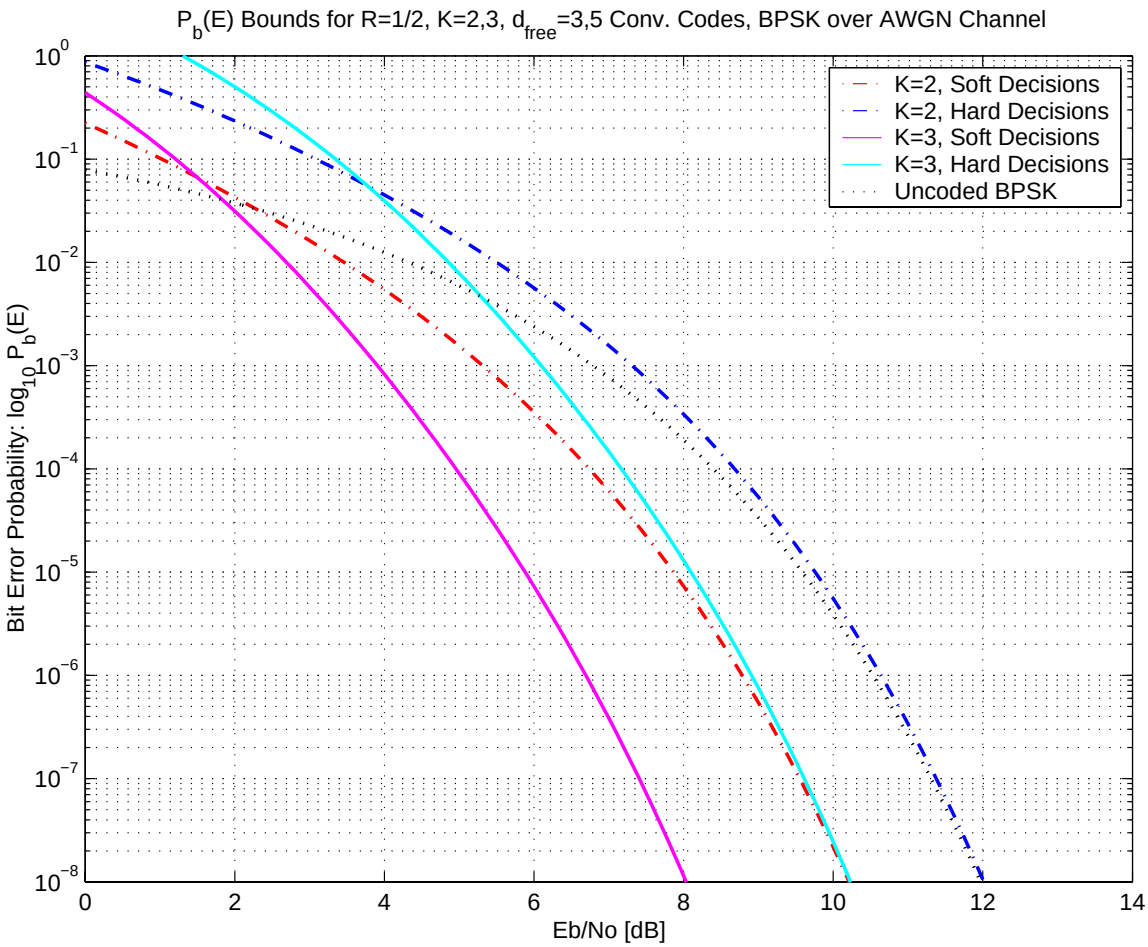
For real-time data transmissions it is not practical to have to wait with the ML decision until a whole code sequence is received. Luckily it turns out that the performance of the algorithm is still nearly optimal if decisions for past trellis sections are output once the partial distance metric computations have moved ahead far enough. For the encoder with $\mathbf{G}(D) = [1\ 1+D]$ which uses only one bit of memory, a very good decision can be made for trellis sections which are about 5 sections before the one that is currently processed.

1.6 Probability of Error for Hard- and Soft-Decision Decoding

The graph below the probability of bit error $P_b(\mathcal{E})$ for the $K = 2$ and $K = 3$ convolutional encoders with transfer function matrices

$$\mathbf{G}(D) = [1 \quad 1+D]\,, \quad \text{and} \quad \mathbf{G}(D) = [1+D^2 \quad 1+D+D^2]\,,$$

For both hard-decision and soft-decision decoding. For comparison, $P_b(\mathcal{E})$ for uncoded BPSK (binary phase shift keying or polar binary PAM) is also shown in the graph.



2 Lab Experiments

E1. Probability of Error (Uncoded PAM). (a) Use the `pam12` function to generate a random polar binary ($a_n \in \{-1, +1\}$) PAM signal of length N with specified `ptype` and `pparms`. Then add noise corresponding to a desired SNR $E_b/\mathcal{N}_0$ in dB and use the `pamrcvr10` function to receive the noisy PAM signal. Compare the transmitted and the received binary data d_n and $\hat{d}_n$ and count the number N_{err} of errors to obtain the probability of symbol (bit) errors as $P_s\mathcal{E} = N_{\text{err}}/N$. The Python script below is a starting point for finding $P_s\mathcal{E}$ using simulation for PAM communication with `ptype='rect'`.

```
# File PsEsim_x01.py
# Simulation of probability of symbol error Ps(E) for
# data transmission using uncoded PAM
from pylab import *
import pamfun

# ***** Parameters *****
Fs = 1000                       # Sampling rate
FB = 100                        # Baud rate FB
N = 100000                      # Number of symbols
EbNodB = 6                      # Specified SNR Eb/No in dB
ptype, pparms = 'rect',[]       # Pulse type/parameters
an_set = [-1,+1]                # Set of possible an values
M = len(an_set)                 # Number of signal levels
# ***** Compute Eb for given p(t) and signal constellation *****
# >>>>> Add your code here <<<<<
# ***** Generate PAM signal using random data *****
dn = array(floor(2*rand(N)),int)   # Random binary data signal
an = 2*dn-1                     # Polar binary sequence
tt, st = pamfun.pam12(an, FB, Fs, ptype, pparms)  # PAM signal
# ***** Generate Gaussian noise signal *****
nt = randn(len(tt))             # Gaussian noise
# >>>>> Compute An such that rt has desired SNR Eb/No <<<<<
rt = st + An*nt                 # Noisy PAM signal
# ***** PAM signal receiver *****
dly = 0
bn, bt, ixn = pamfun.pamrcvr10(tt, rt, [FB, dly], ptype, pparms)
dnhat = array(zeros(len(bn)),int)
ix = where(bn > 0)[0]
dnhat[ix] = ones(len(ix))       # Received binary data, quantized
# ***** Compare dn, dnhat and compute Ps(E) *****
# >>>>> Add your code here <<<<<
```

Note that F_s=1000 Hz is used (rather than F_s=8000 or F_s=44100 Hz) to shorten the simulation time and to avoid problems due to finite numerical precision for larger values of N ($N \approx 1,000,000$ or more) that are needed when $P_s(\mathcal{E})$ is small (10^{-5} or less). Your task is to complete the above Python script by computing the value of `Eb` and of the noise amplitude `An` so that `rt` has the specified $E_b/\mathcal{N}_0$ value in dB. The probability of symbol error $P_s(\mathcal{E})$ is then obtained from the comparison between `dn` and `dnhat`. To compute E_b, generate a PAM signal corresponding to each of the values that a_n can take on (specified in `an_set`) and then compute the average symbol energy E_s assuming that all signal values are equally likely. The average bit energy is then obtained as $E_b = E_s/\log_2 M$. To determine $\mathcal{N}_0$ for the noise `An*nt`, note that `nt` generated by `randn` is white Gaussian noise. Since `nt` is sampled with rate F_s, its spectrum extends from $-F_s/2$ to $F_s/2$ and thus the noise power can be computed in two ways as

```
Pnt = mean(np.power(An*nt,2.0))    # Noise power in time domain
Pnt = Fs*No/2                      # Noise power in freq domain
```

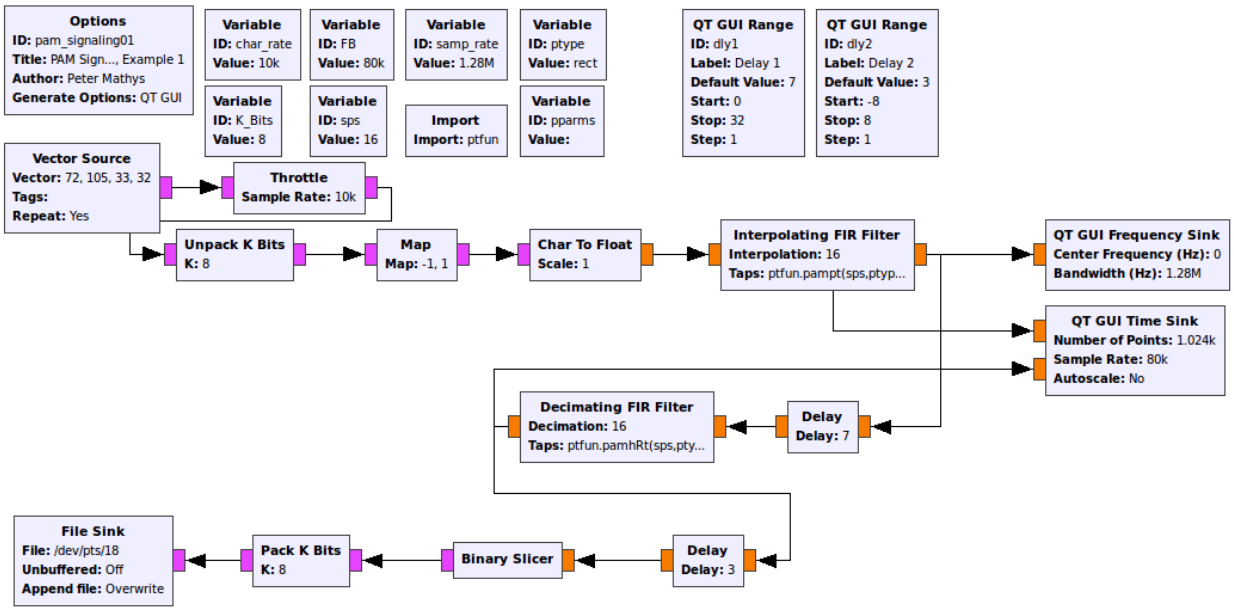
From this it follows that

```
No = 2*mean(np.power(An*nt,2.0))/float(Fs)  # One-sided white noise PSD
```

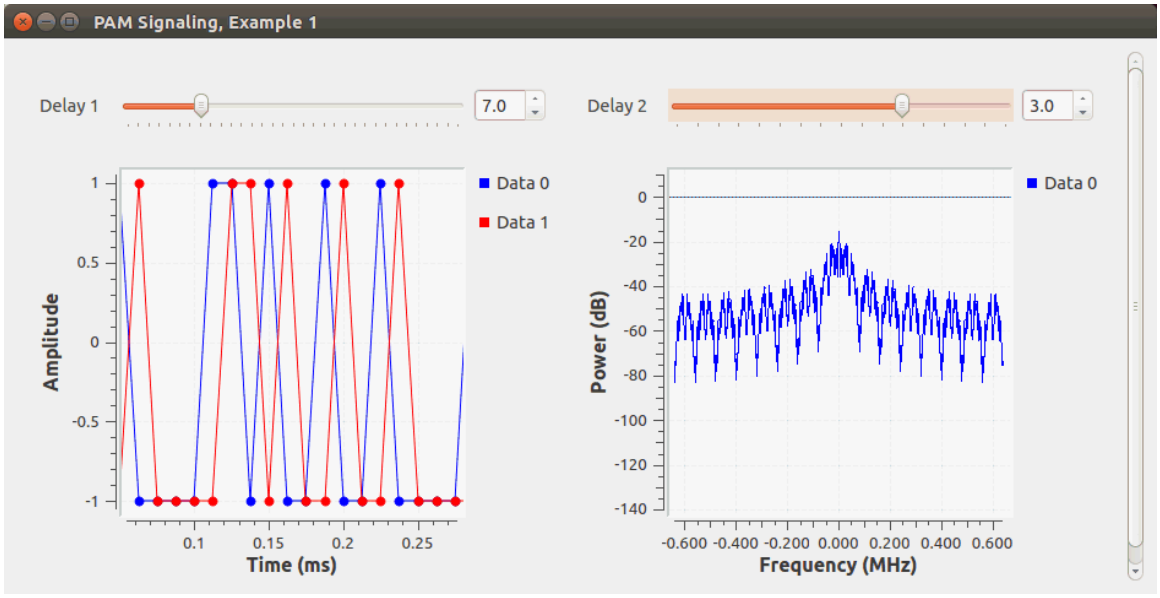
Run the script file to determine $P_s(\mathcal{E})$ for $E_b/\mathcal{N}_0 = 2, 4, 6, 8$ dB. Increase N as necessary so that $P_s(\mathcal{E})$ is computed from at least $10 \ldots 50$ simulated errors. Compare the results with the $P_s(\mathcal{E})$ graph given in the introduction. You should find very good agreement between the results from the simulations and the analytically computed values. If your $P_s(\mathcal{E})$ values differ by more than a factor of 2 from the theoretical values, you are doing something wrong in your simulations.

(b) Determine $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ (for SNRs of 2,4,6,8 dB) using the script file from (a) for polar binary PAM with `'man'` and `'tri'` pulse types. Do both have the same $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ characteristic as the rectangular pulse type? If not, why not?

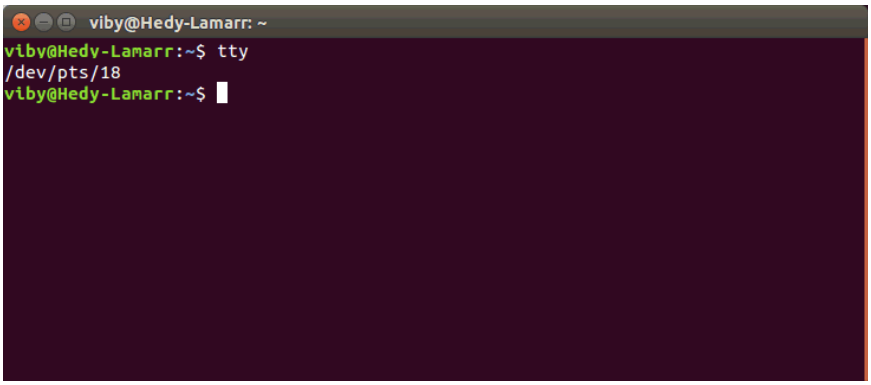
(c) PAM Transmission/Reception in GNU Radio. Build the following flowgraph consisting of a message source, a PAM transmitter, a matched filter (MF) receiver, and a message (file) sink in GNU Radio. The specifications of the PAM pulses $p(t)$ come from the `ptfun` module that you created earlier. You will have to add the MF responses $h_R(t)$ as a function `pamhRt` to this module.



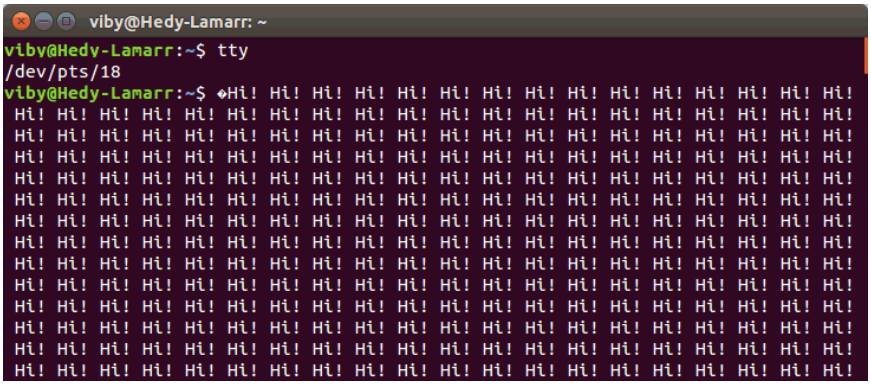
The result of running the flowgraph when everything is adjusted properly is shown in the screen snapshot below.



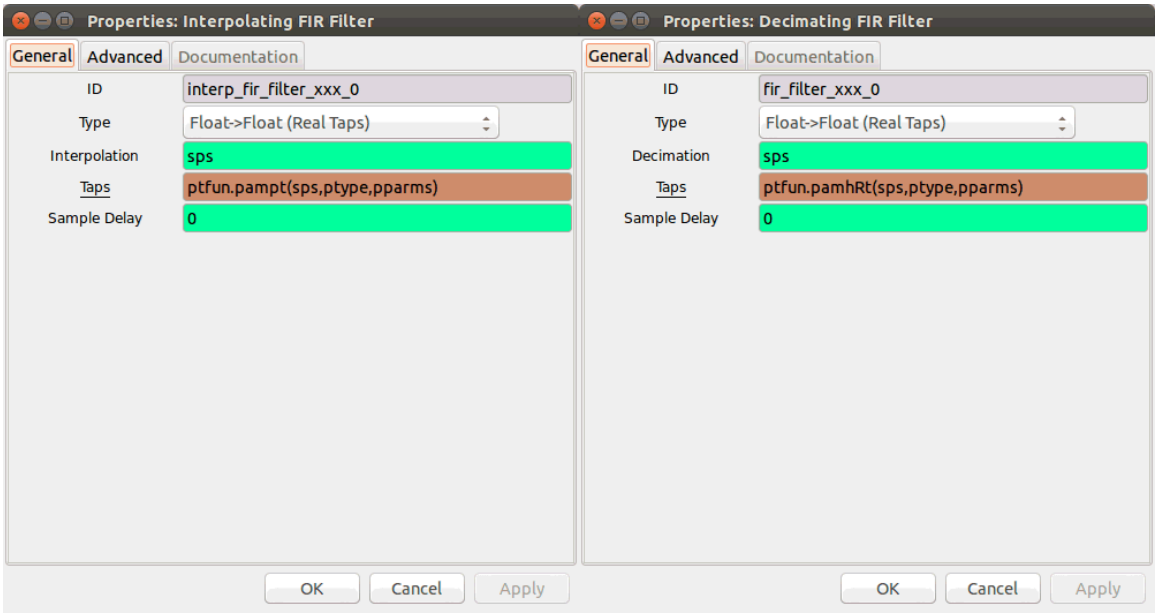
The File Sink at the bottom left of the flowgraph is used to display the received ASCII text in a terminal window. To accomplish this, start a terminal window (press Alt-Ctrl-T) in Linux and type `tty` to get the `tty` device number as shown in the example below.



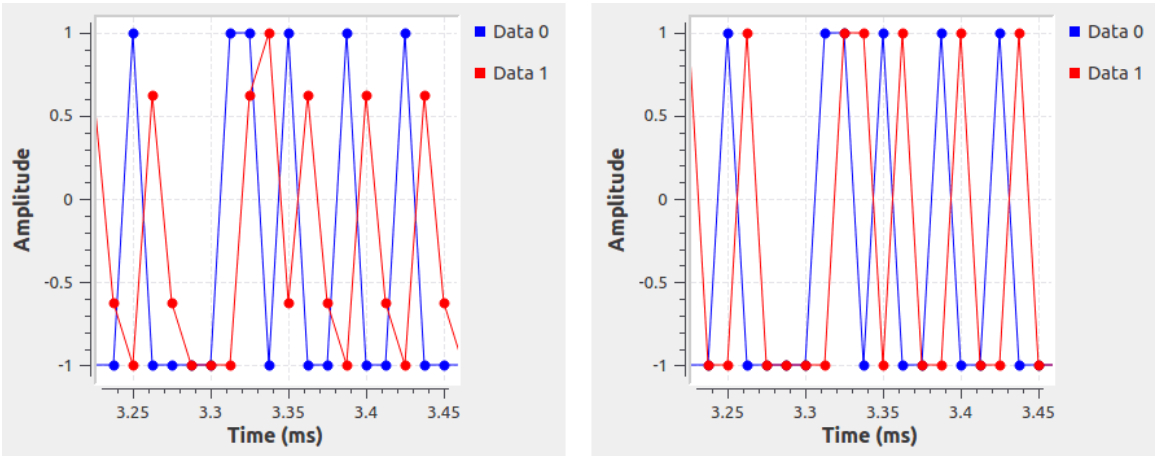
Enter the device number (18 in the example above) in the File field of the File Sink block. If the delay in front of the Binary Slicer and the Pack K Bits blocks is adjusted correctly, then the `tty` terminal should display the ASCII text that is transmitted from the Vector Source through the PAM transmission system. Here is a screen snapshot of the correct output.



The settings for the filter taps of the PAM pulse generator (Interpolating FIR Filter block) and the matched filter (MF) receiver (Decimating FIR Filter block) are shown below. Note that you will need to import the `ptfun` module with the `pampt` and the `pamhRt` functions for the generation of the correct filter taps.

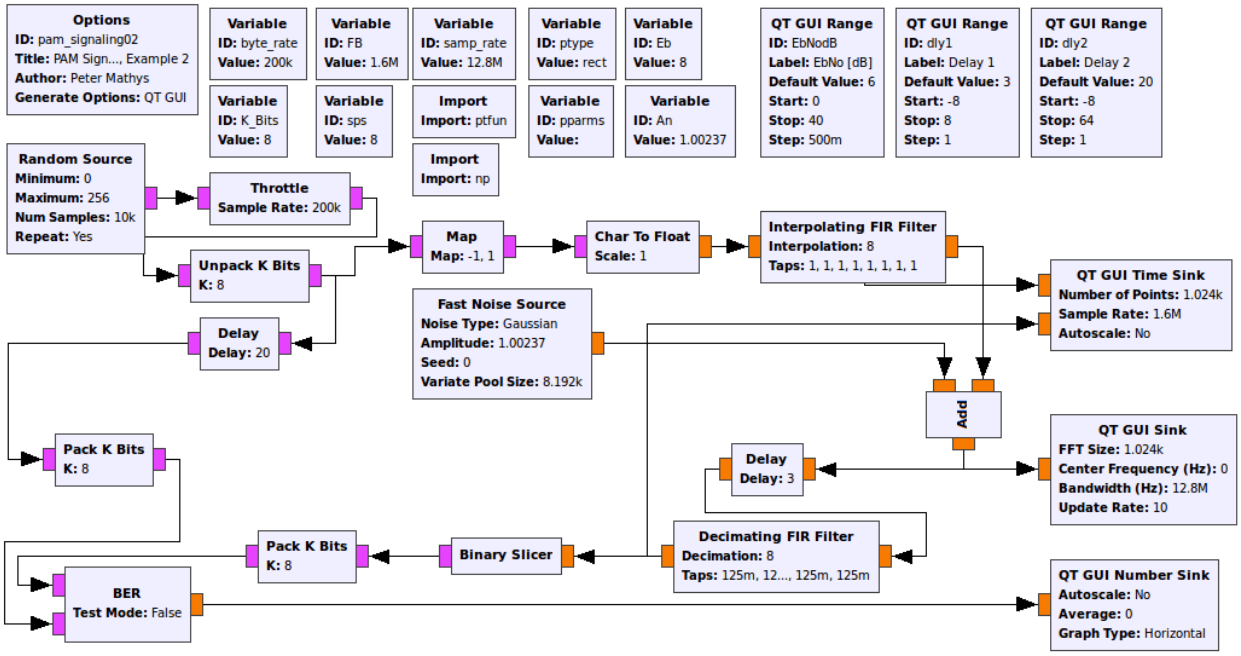


To adjust the first of the two Delay blocks (between the Interpolating and the Decimating FIR Filter blocks), look at the time domain signals before and after the two FIR filters. The graphs below show the output signals, sampled after the matched filter for an incorrect delay on the left and a correct delay on the right, in both cases for a rectangular shaped $p(t)$.

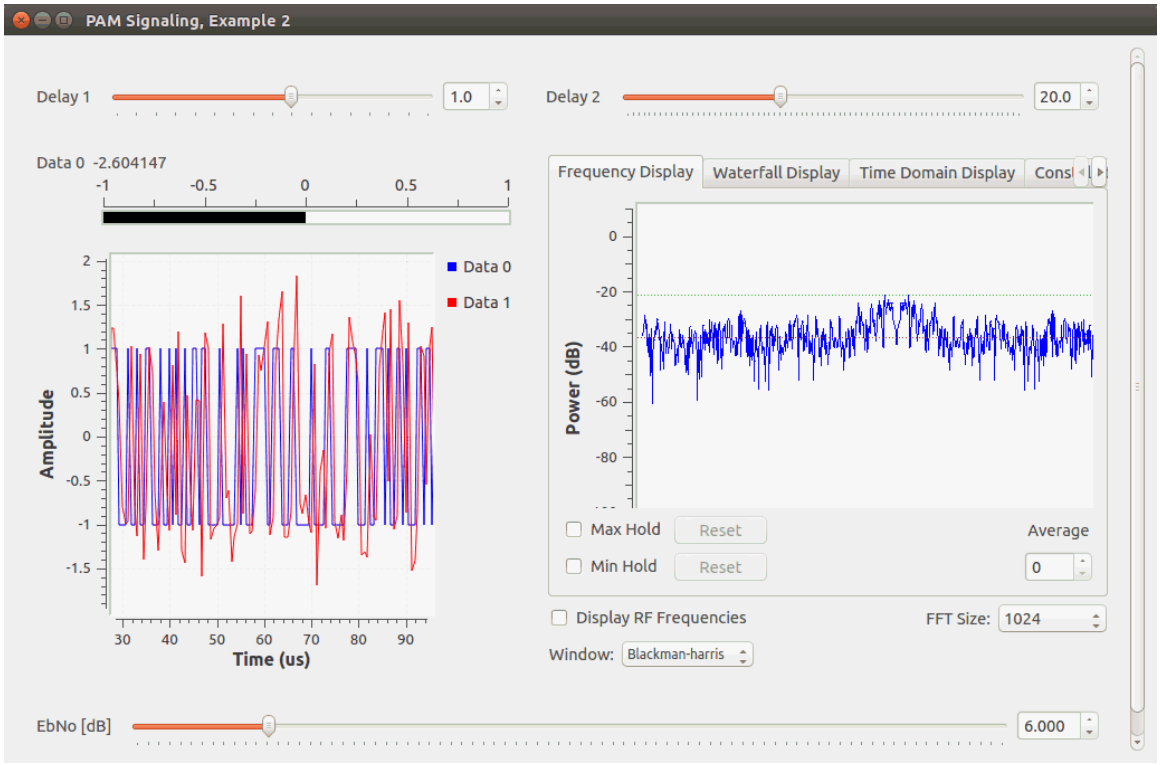


Your task for this experiment is to find the correct values for both delays (`dly1` and `dly2`) such that the correct ASCII code message is displayed in the terminal device specified in the File Sink for PAM communication using both `ptype='rect'`, `pparms=[]`, and `ptype='rrcf'`, `pparms=[10,0.4]`.

(d) Bit Error Measurements for PAM in GNU Radio. Modify the GNU Radio flowgraph in part (c) by replacing the Vector Source with a Random Source, then adding Gaussian noise to the received PAM signal, and finally replacing the File Sink by a BER (bit error rate) measurement block, as shown below.



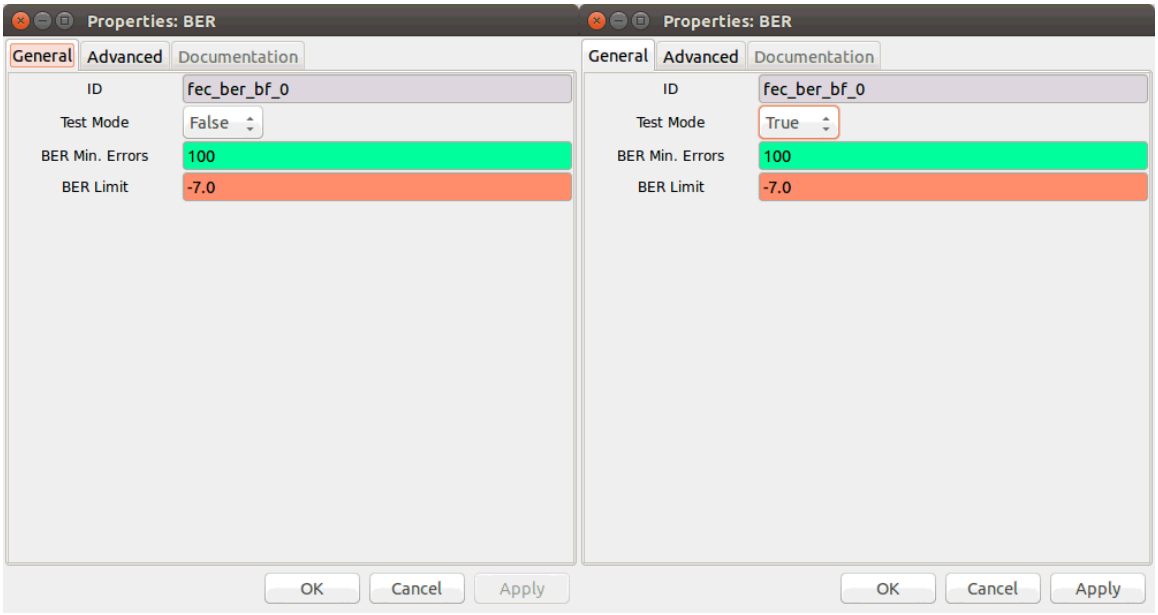
A typical display resulting from running the flowgraph with the delays adjusted properly and a SNR of $E_b/N_0 = 6$ dB is shown in the next screen snapshot. The Data 0 output of -2.604147 means that the probability of symbol error at the given SNR is $10^{-2.604147} = 2.488 \times 10^{-3}$ which is in very good agreement with the graph of $P_s(\mathcal{E})$ versus E_b/N_0 for polar binary PAM given in the introduction.



Besides adjusting delays `dly1` and `dly2` correctly, it is of crucial importance to adjust the Fast Noise Source Amplitude A_n (or `An`) properliy to obtain correct $\log_{10}(P_s(\mathcal{E}))$ readings at the output of the BER block. Since we are operating with binary data in discrete time, $E_b = E_s = \sum_n |p_n|^2$ where p_n are the DT samples of the PAM pulse $p(t)$. The noise power from the Fast Noise Source is $P_n = A_n^2$ and, using the DTFT and Parseval's relation, we also have $P_n = \mathcal{N}_0/2$ in the (normalized) frequency domain. Thus $\mathcal{N}_0 = 2\,A_n^2$ and therefore $E_b/\mathcal{N}_0 = (\sum_n |p_n|^2)/(2\,A_n^2)$, from which A_n can be computed for a given $E_b/\mathcal{N}_0$.

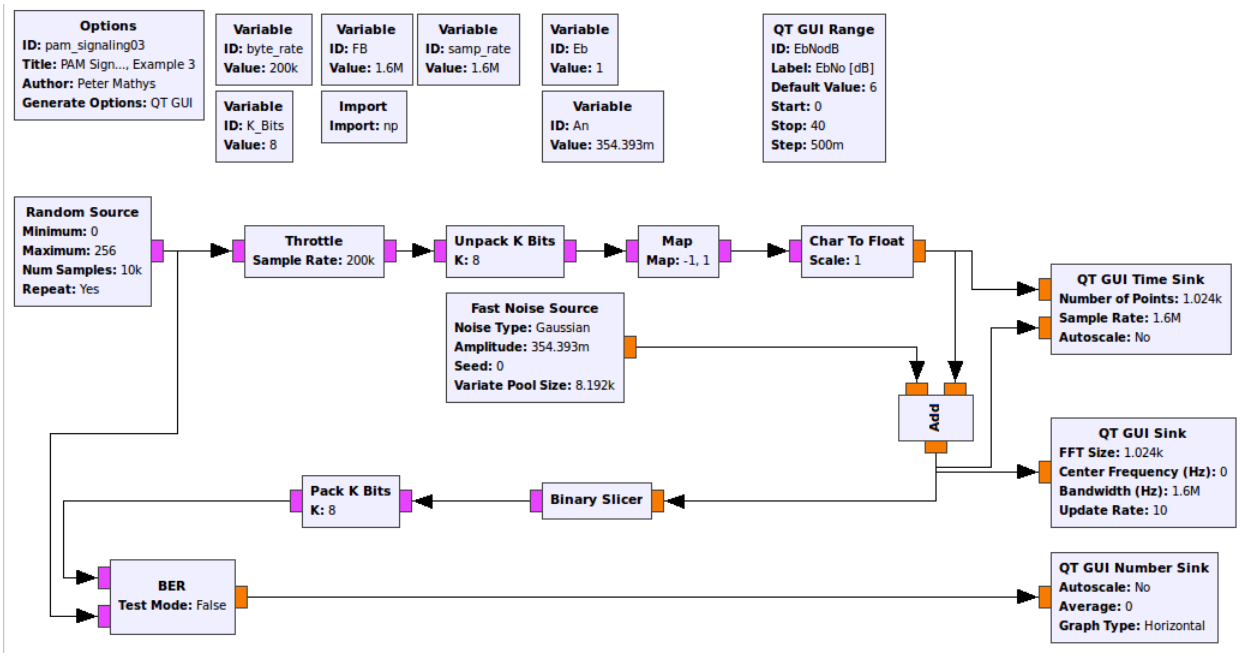
Your task for this experiment is to measure $\log_{10}(P_s(\mathcal{E}))$ and determine $P_s(\mathcal{E})$ for $E_b/\mathcal{N}_0$ values of 2,4,6,8 dB for the following scenarios using `ptype='rect'` PAM pulses at the transmitter and (i) a matched filter for the receiver, (ii) a trapezoidal LPF with cutoff frequencies $2\,F_B$, F_B, and $F_B/2$ for the receiver. Use $k \approx 10$ and $\alpha \approx 0.2$ for the trapezoidal filter.

The BER block has a Test Mode property that can be set to either True or False (False is the default) as shown below.

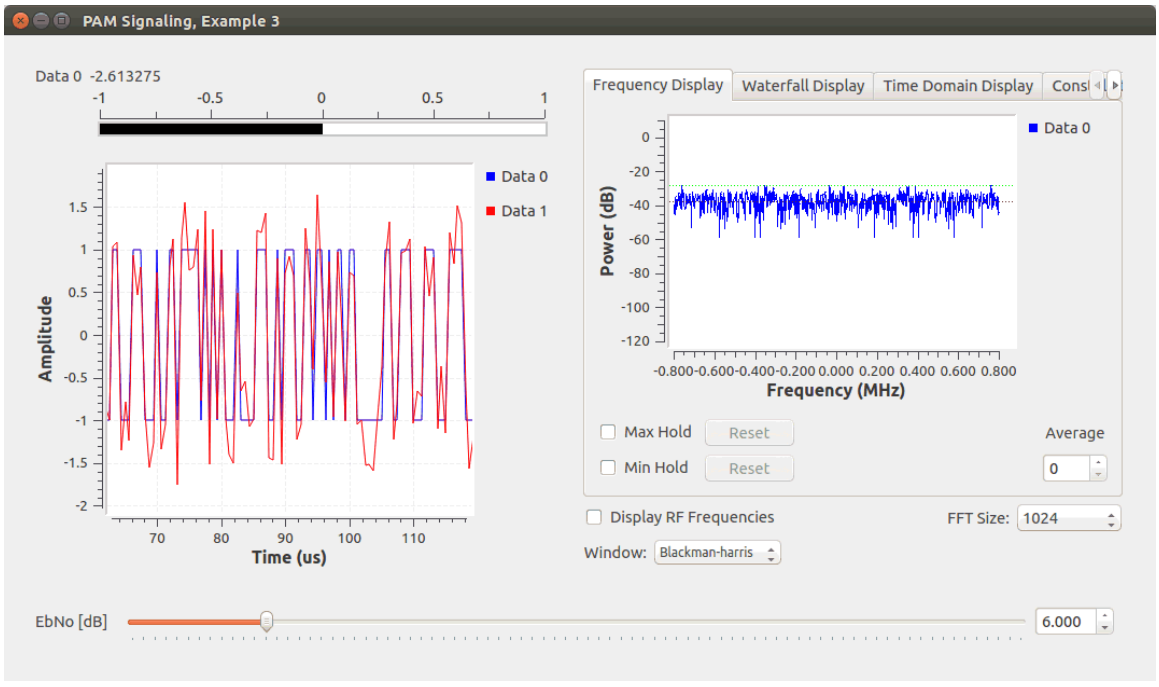


In Test Mode: False, the BER block continuously updates $\log_{10}(P_s(\mathcal{E}))$ which is useful for observing whether $P_s(\mathcal{E})$ increases or decreases as system parameters, such as $E_b/\mathcal{N}_0$, are changed. However, it takes a long time in this mode to reach steady-state after changes because there is no provision for a reset of the previously acquired history, except for restarting the flowgraph. For specific and precise measurements it is best to have all parameters (such as $E_b/\mathcal{N}_0$) set to the desired values before starting the flowgraph and to set the BER Test Mode to True. In this mode the flowgraph will run until the specified (in the BER Min. Errors property) number of errors has occurred and then the resulting $\log_{10}(P_b(\mathcal{E}))$ is displayed in the Reports Panel below the main flowgraph window.

(e) Faster Bit Error Measurements in GNU Radio. You may have noticed in part (d) that the BER measurements are somwhat slow due to the generation of the actual PAM waveform and its processing in the matched filter at the receiver. If you only need to make BER measurements with respect to additive white Gaussian noise and not with respect to the effect that $p(t)$ and $h_R(t)$ have on the BER, the GNU Radio flowchart shown below can be used for a faster measurement of $P_s(\mathcal{E})$.



A typical example of the graphs displayed when running the flowgraph is shown below. Note that now the time domain graph has only one sample per transmitted bit and the corresponding spectrum is essentially flat because of the random data used and the fact that $p_n = \delta_n$.



Use the flowgraph shown above to measure $P_s(\mathcal{E})$ in BER Test Mode: True with BER Min. Errors set to 1000 for both polar and unipolar binary PAM with $E_b/\mathcal{N}_0 = 2, 4, 6, 8$ dB. Note

that you will have to modify the flowgraph for unipolar binary PAM. Compare the results with the $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ graph in the introduction and with the measurements you made in the previous experiments of E1.

E2. Convolutional Encoder/Decoder. (a) Start a Python module `ccfun.py` for convolutional coding functions and write a function called `ccencod10` that implements a binary rate $1/n$ convolutional encoder with data input sequence d_i, encoded output sequence c_i (n filter outputs multiplexed), and transfer function matrix $\mathbf{G}(D) = [g_1(D) \ \ldots \ g_n(D)]$. Initially n code sequences $c_i^{(1)}, \ldots, c_i^{(n)}$ of the same length as the data sequence are produced by convolving d_i with $\{g_i^{(1)}\}, \ldots, \{g_i^{(n)}\}$. Since the code is binary, the convolution is computed using modulo 2 arithmetic. Then the n code sequences are multiplexed to produce a single binary code sequence c_i whose length is n times the length of the data sequence. More specifically

$$\{c_i\} = \{c_0^{(1)}, c_0^{(2)}, \ldots c_0^{(n)}, c_1^{(1)}, c_1^{(2)}, \ldots c_1^{(n)}, c_2^{(1)}, \ldots\}$$

The header of `ccencod10` is shown below.

```
def ccencod10(di, GD, trim=True):
    """
    Binary rate 1/n convolutional encoder with transfer
    function matrix G(D), V 1.0
    >>>>> ci = ccencod10(di, GD, trim) <<<<<
    where  ci:   multiplexed binary code sequence
           di:   binary (unipolar) data sequence
           GD:   array of n encoder polynomials
           trim: if True, trim ci to n*len(di)

    Examples:  GD = [[1,0],[1,1]]  for G(D) = [1 1+D]
               GD = [[1,0,1],[1,1,1]]  for G(D) = [1+D^2 1+D+D^2]
               GD = [[1,0,1,1],[1,1,0,1],[1,1,1,1]]
                    for G(D) = [1+D^2+D^3 1+D+D^3 1+D+D^2+D^3]
    """
```

Generate a few code sequences and compare them to the table of codewords in the introduction to check the basic functionality of the encoder. For example, when `GD = [1 0;1 1]` then

$$\{d_i\} = \{1, 0, 1, 1, 0, 0, \ldots\} \quad \Longrightarrow \quad \{c_i\} = \{1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, \ldots\}$$

(b) The function `ccdecod10` shown below implements a Viterbi decoder for the rate 1/2 convolutional code generated by the transfer function matrix $\mathbf{G}(D) = [1 \ 1+D]$. It is assumed that the received code sequence `bn` (at the output of the matched filter after sampling) is **polar** binary ({-A,+A}) with additive white Gaussian noise.

```
def ccdecod10(bi):
    """
    Viterbi decoder for binary rate 1/2 convolutional code received
    as polar (binary 0 -> -1, binary 1 -> +1) PAM signal from AWGN
    channel.
    Version 1.0 for encoder transfer function matrix G(D) = [1 1+D]
    >>>>> dihat, DM = ccdecod10(bi) <<<<<
    where  dihat: ML estimate of binary data sequence
           DM:    array of final distance metrics
           bi:    received noisy polar binary (+A/-A) sequence
    """
    n, K, m = 2, 2, 1          # Rate 1/n, constraint len K, memory m
    N = int(floor(len(bi)/float(n)))  # Number of codeword frames
    CBM = [[-1,-1],[+1,+1],[-1,+1],[+1,-1]]  # Code-bit (-1/+1) matrix
    DM = array([0,1000])       # (Initial) distance metrics
    dA = array(zeros((2,1)),int)  # Competing data sequence array
    ix2 = array([0,0,1,1],int) # State indexes (doubled)
    for i in range(N):
        bDM = np.power(outer(ones(4),bi[n*i:n*(i+1)]) - CBM,2.0)
        bDM = dot(bDM,ones((n,1)))  # Branch distance metrics
        bDM = reshape(bDM,size(bDM))  # Convert to 1d array
        tDM = DM[ix2] + bDM    # Tentative distance metrics
        tDM = reshape(tDM,(2,2))  # Reshape for path elimination
        DM = amin(tDM,axis=0)  # Select paths with smaller metric
        ix = argmin(tDM,axis=0)  # Indexes of smaller metric paths
        dA = hstack((dA[ix,:],array([[0],[1]])))
                               # Competing data sequence update
    dA = dA[:,1:]              # Discard first (dummy) column
    ix = argmin(DM)            # Index of smallest metric
    dihat = dA[ix,:]           # ML-decoded data sequence
    dihat = reshape(array(dihat),size(dihat))  # Convert to 1d array
    return dihat, DM
```

Test `ccencod10` together with `ccdecod10` to make sure that `dihat` is equal to `di` in the absence of errors in `bi`. Note that the `ci` at the output of `ccencod10` are unipolar binary, whereas `bi` is expected to be polar binary.

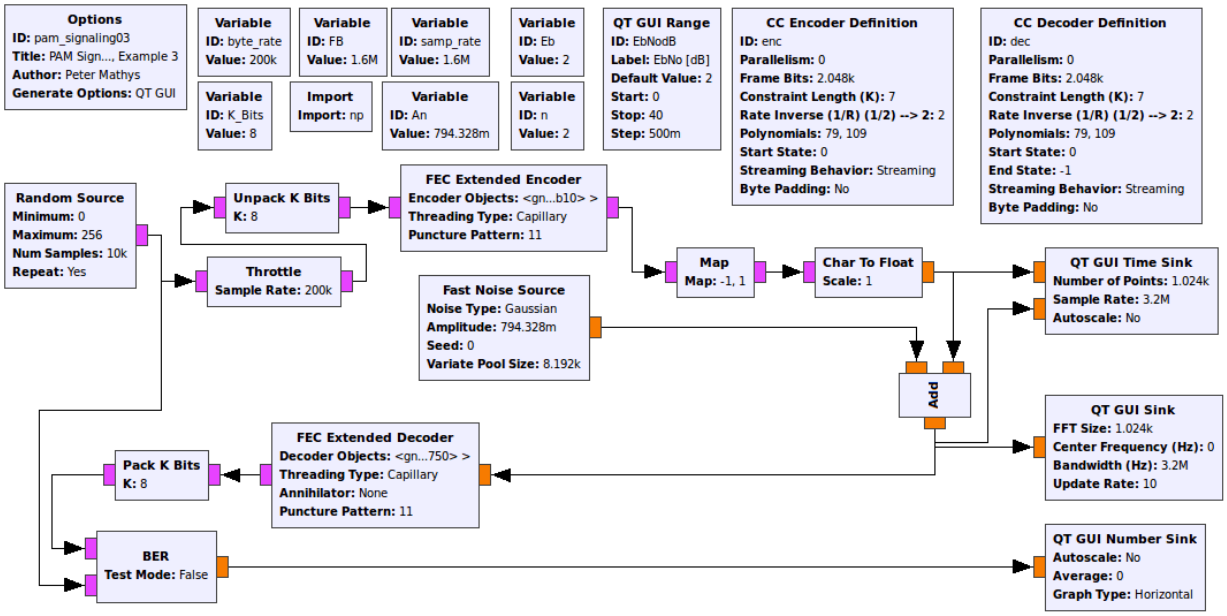
(c) Determine $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ for polar binary PAM with rectangular $p(t)$ in conjunction with error control coding using the convolutional encoder and decoder with $\mathbf{G}(D) = [1\ 1+D]$ from parts (a) and (b). Note that, since the code has rate 1/2, E_b now is the average energy of **two** (coded) PAM pulses. Determine $P_s(\mathcal{E})$ for $E_b/\mathcal{N}_0$ values of $2, 4, 6, 8$ dB and compare them with the values in the $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ graph for convolutional codes given in the introduction. Choose N such that $P_s(\mathcal{E})$ is computed from at least $10 \ldots 50$ simulated errors. You should find very good agreement between the results from the simulations and the analytically computed values in the graph.

(d) Modifiy the program you used for (c) to determine $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ for hard-decision decoding and find again $P_s(\mathcal{E})$ for $E_b/\mathcal{N}_0$ values of $2, 4, 6, 8$ dB. Estimate the amount of

coding gain (in dB) that soft-decision decoding attains compared to hard-decision decoding for $P_s(\mathcal{E}) \approx 10^{-3}$.

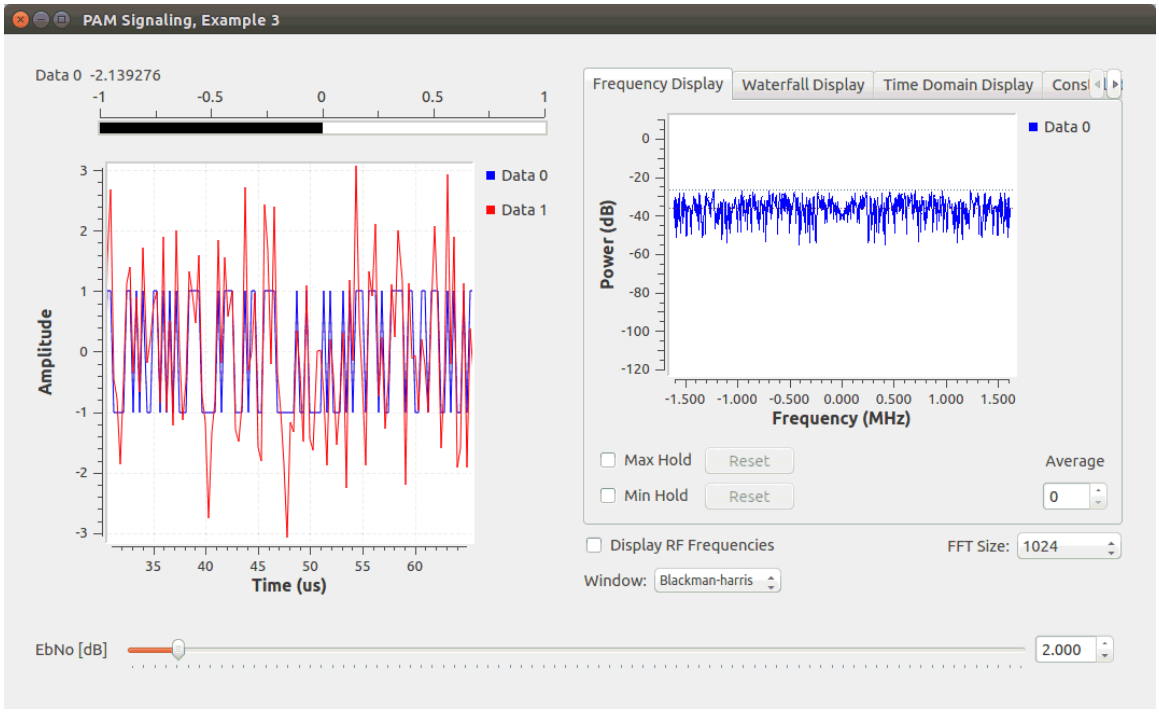
(e) The signal in `pamsig701.wav` is a convolutionally encoded (rate 1/2 encoder with $\mathbf{G}(D) = [1\ 1+D]$) PAM signal with additive white Gaussian noise, rectangular $p(t)$ and $F_B = 100$ baud. Estimate the SNR $E_b/\mathcal{N}_0$ in dB and try to recover the ASCII text (8 bit, LSB first) in `pamsig701.wav`.

(f) Concolutional Encoding and Decoding in GNU Radio. The flowchart below can be used for BER measurements with convolutional encoding and decoding on a AWGN channel.



The decoder in the flowchart will only work for the rate 1/2 code with encoder polynomials [79,109] (the convolutional code that was/is used for the Voyager mission which is now the Voyager Interstellar Mission). There are other convolutional encoder/decoder blocks in GNU Radio under Trellis Coding which are much more general, but also more difficult to set up.

The screen snapshot below shows the results that are obtained when the flowgraph is run for $E_b/\mathcal{N}_0 = 2$ dB.



Determine $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ for SNRs of 2,3,4,5 dB for both soft-decision and hard-decision decoding.

E3. Probability of Error for Additional Communication Scenarios. (Experiment for ECEN 5002, optional for ECEN 4652) **(a)** Modify E1(a) so that you can determine $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ (for SNRs of 6,8,10,12 dB) for rectangular $p(t)$ PAM signals with $M = 4$. Generate a iid random data sequence a_n with $a_n \in \{-3, -1, +1, +3\}$, assuming all 4 symbols are equally likely. Note that the average symbol energy E_s is obtained by averaging the energy for pulses with amplitudes -3,-1,+1,+3. The average bit energy is then $E_s/\log_2 M = E_s/2$. Compare your results with the $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ graph given in the introduction. Under what circumstances would it be better to use $M = 4$ than $M = 2$?

(b) PAM with More Powerful Convolutional Code. Write a more general Viterbi decoder function in Python that can be used for any rate $1/n$ convolutional code specified by a transfer function matrix $G(D) = [g^{(1)}(D)\ g^{(2)}(D) \ldots g^{(n)}(D)]$. The header of this function, called `ccdecod20`, to be included in the `ccfun` module is shown below.

```
def ccdecod20(bi, GD):
    """
    Viterbi decoder for binary rate 1/n convolutional code with
    transfer function matrix G(D), received as polar (-1,+1)
    PAM signal from AWGN channel. Version 2.0
    >>>>> dihat, DM = ccdecod20(bi, GD) <<<<<
    where  dihat: ML estimate of binary data sequence
           DM:    array of final distance metrics
           bi:    received noisy polar binary (+A/-A) sequence
           GD:    array of n encoder polynomials

    Examples:  GD = [[1,0],[1,1]]  for G(D) = [1 1+D]
               GD = [[1,0,1],[1,1,1]]  for G(D) = [1+D^2 1+D+D^2]
               GD = [[1,0,1,1],[1,1,0,1],[1,1,1,1]]
                    for G(D) = [1+D^2+D^3 1+D+D^3 1+D+D^2+D^3]
    """
```

Test `ccencod10` together with `ccdecod20` to make sure that `dihat` is equal to `di` in the absence of errors in `bi`. Note that the `ci` at the output of `ccencod10` are unipolar binary, whereas `bi` is expected to be polar binary.

(c) Determine $P_s(\mathcal{E})$ versus $E_b/\mathcal{N}_0$ for polar binary PAM with rectangular $p(t)$ in conjunction with error control coding using the convolutional encoder and decoder with $\mathbf{G}(D) = [1 + D^2\ 1 + D + D^2]$ from part (b). Note that, since the code has rate 1/2, E_b now is the average energy of **two** (coded) PAM pulses. Determine $P_s(\mathcal{E})$ for $E_b/\mathcal{N}_0$ values of $2, 3, 4, 5, 6$ dB and compare them with the symbol error probabilities for uncoded binary polar PAM in E1 and coded binary polar PAM in E2.

(d) The signal in `pamsig702.wav` is a convolutionally encoded (rate 1/2 encoder with $\mathbf{G}(D) = [1 + D^2\ 1 + D + D^2]$) PAM signal with additive white Gaussian noise, rectangular $p(t)$ and unknown F_B. Estimate the SNR $E_b/\mathcal{N}_0$ in dB, determine F_B, and try to recover the ASCII text (8 bit, LSB first) in `pamsig702.wav`.

