1 Background Material

Complete the following tasks. You should submit an e-mail with three .m file attachments (use the e-mail address esp@eecs.ku.edu).

1. Implement a Function to Create Look-up Tables for the Trellis for Convolutional Codes. You may assume that we are limited to rate 1/n feedforward non-systematic codes. You need to create look-up tables that are sorted according to the left-hand edge index, \( e^l \); tables are needed for the starting state, \( s^l(e^l) \), the message bit, \( m(e^l) \), the code bits, \( c(e^l) \), the ending state, \( s^e(e^l) \), and the right-hand edge index, \( e^r(e^l) \). Additionally, one look-up table is required to be sorted according to the right-hand edge index; this table is the left-hand edge index, \( e^l(e^r) \) (so that the algorithm can trace backwards).

You should implement this as a MATLAB function with the following syntax:

```matlab
[sS, me, ce, sE, eR, eL_eR] = CreateCcTrellisXXX(G);
```

where \( G \) is a \((ν + 1) \times n\) matrix that specifies \( n \) encoder functions with a constraint length (memory order) of \( ν \). For the \((5,7)\) code, we have a transfer function matrix

\[ G(D) = \begin{bmatrix} 1 + D^2 & 1 + D + D^2 \end{bmatrix} \]

and so we specify \( G \) in MATLAB as

```matlab
G = [1 1; 1 1];
```

As for the output arguments, each is an \( N_e \times 1 \) MATLAB vector (except for \( ce \), which is an \( N_e \times n \) matrix), where \( N_e \) is the number of edges in the trellis. Because these elements are in order of increasing left-hand edge index, the specification for \( e^l \) is implied and does not need to be stated explicitly.

2. Implement the Hard-Decision Viterbi Algorithm for Convolutional Codes. You should implement this as a MATLAB function with the following syntax:

```matlab
m_hat = VaCcHdXXX(r, TERM, sS, me, ce, eR, eL_eR);
```

where \( r \) is a \( 1 \times nL \) MATLAB vector containing hard decisions from the BSC (i.e., 0's and 1's) and \( TERM \) is a Boolean flag (i.e., a 0 or a 1) to indicate if the trellis is terminated. The metric increment for the hard-decision VA is

\[ y_t(e) = d_H(r, c(e)) \]

where \( d_H(\cdot, \cdot) \) denotes Hamming distance. For the \((5,7)\) code, the metric increment is 0, 1, or 2. The objective of the VA is to minimize this metric. The algorithm you need to implement is shown in Algorithm 3.

3. Implement the Soft-Decision Viterbi Algorithm for Convolutional Codes. You should implement this as a MATLAB function with the following syntax:

```matlab
m_hat = VaCcSdXXX(r, TERM, sS, me, ae, eR, eL_eR);
```
where \( r \) is a \( 1 \times nL \) MATLAB vector containing soft decisions from the AWGN channel (i.e., noisy +1's and -1's). The metric increment for the soft-decision VA is

\[
\gamma_t(e) = \sum_{i=0}^{n-1} r_i^{(i)} a^{(i)}(e) = a(e) r_t^T
\]

where \( a(e) = 2c(e) - 1 \) is the antipodal version of \( c(e) \) and \((\cdot)^T\) is the transpose operator. Over time, the increments \( r_t a^T(e) \) add up and result in the correlation between \( r \) and \( a \). The objective of the VA is to maximize the correlation, so the appropriate changes must be made to the VA operations. The algorithm you need to implement is shown in Algorithm 5.

4. **BER Simulation.** Verify the correct operation of your VA implementations by running a BER simulation (using the "wrapper file" provided) for the (5,7) code. The expected result is shown above. You should run your simulation over the range of \( E_b/N_0 \) values from 0 to 5 dB.

5. **Channel Capacity.** Plot the (5,7) code performance by adding two data points to your capacity plot, one for hard decision decoding and one for soft decision decoding.
The Viterbi algorithm (VA) for hard-decision inputs.

1: **Input:** The hard-decision received signal $r = \{r_0, r_1, \ldots, r_{T-1}\}$.
2: **Input:** A Boolean flag TERM to indicate if the trellis is terminated.
3: **Output:** The estimated symbol sequence $\hat{m} = \{\hat{m}_0, \hat{m}_1, \ldots, \hat{m}_{T-1}\}$.

4: **Initialization:**
5: if TERM == TRUE then
6:   $A_{-1}(0) = 0$; and $A_{-1}(s) = +\infty$ for $1 \leq s \leq N_s - 1$;
7: else
8:   $A_{-1}(s) = 0$ for $0 \leq s \leq N_s - 1$;
9: end if;

10: **Main Algorithm:**
11: The edge metric is $\gamma_t(e) = d_h(r_t, c(e))$ for all $t \in \{0, 1, \ldots, T-1\}$ and $e \in \{0, 1, \ldots, N_e - 1\}$
12: for $t = 0, 1, \ldots, T-1$ do // Forward recursion with metric and survivor updates
13:   for $s = 0, 1, \ldots, N_s - 1$ do
14:     $A_t(s) = \min_{e:s(e)=s} \{A_{t-1}(s^e(e)) + \gamma_t(e)\}$;
15:     $T_t(s) = \arg \min_{e:s(e)=s} \{A_{t-1}(s^e(e)) + \gamma_t(e)\}$;
16:   end for
17: end for
18: if TERM == TRUE then // Final global survivor at the end of the transmission
19:   $\hat{x}_{T-1}^e = 0$;
20: else
21:   $\hat{x}_{T-1}^e = \arg \min_{0 \leq s \leq N_e - 1} \{A_{T-1}(s)\}$;
22: end if;
23: for $t = T-1, \ldots, 1$ do // Traceback operation to identify output sequence
24:   $\hat{m}_t = m(T_t(\hat{x}_t^e))$;
25:   $\hat{x}_{t-1}^e = s^e(T_t(\hat{x}_t^e))$;
26: end for
Algorithm 2 The Viterbi algorithm (VA) for soft-decision inputs.

1: **Input:** The soft-decision received signal \( r = \{ r_0, r_1, \ldots, r_{T-1} \} \).
2: **Input:** A Boolean flag \( \text{TERM} \) to indicate if the trellis is terminated.
3: **Output:** The estimated symbol sequence \( \hat{m} = \{ \hat{m}_0, \hat{m}_1, \ldots, \hat{m}_{T-1} \} \).
4: **Initialization:**
5: if \( \text{TERM} == \text{TRUE} \) then,
6: \( A_{-1}(0) = 0 \); and \( A_{-1}(s) = -\infty \) for \( 1 \leq s \leq N - 1 \);
7: else
8: \( A_{-1}(s) = 0 \) for \( 0 \leq s \leq N - 1 \);
9: end if;
10: **Main Algorithm:**
11: The edge metric is \( \gamma_t(e) = \sum_{i=0}^{n-1} r_t^{(i)} a^{(i)}(e) = a(e)r_t^T \) for all \( t \in \{ 0, 1, \ldots, T-1 \} \) and \( e \in \{ 0, 1, \ldots, N_e - 1 \} \).
12: for \( t = 0, 1, \ldots, T - 1 \) do  // Forward recursion with metric and survivor updates
13: for \( s = 0, 1, \ldots, N - 1 \) do
14: \( A_t(s) = \max_{e:s(e)=s} \{ A_{t-1}(s^e) + \gamma_t(e) \} \);
15: \( T_t(s) = \arg \max_{e:s(e)=s} \{ A_{t-1}(s^e) + \gamma_t(e) \} \);
16: end for
17: end for
18: if \( \text{TERM} == \text{TRUE} \) then,  // Final global survivor at the end of the transmission
19: \( \hat{s}_{T-1}^E = 0 \);  
20: else
21: \( \hat{s}_{T-1}^E = \arg \max_{0 \leq s \leq N - 1} \{ A_{T-1}(s) \} \);
22: end if;
23: for \( t = T - 1, \ldots, 1, 0 \) do  // Traceback operation to identify output sequence
24: \( \hat{m}_t = m(T_t(\hat{s}_t^E)) \);
25: \( \hat{s}_{t-1}^E = s^E(T_t(\hat{s}_t^E)) \);
26: end for
Algorithm 3: The VA for hard-decision inputs using indexes for the left- and right-hand sides of the trellis.

1: **Input:** The hard-decision received signal $r = \{r_0, r_1, \ldots, r_{T-1}\}$.
2: **Input:** A Boolean flag TERM to indicate if the trellis is terminated.
3: **Output:** The estimated symbol sequence $\hat{m} = \{\hat{m}_0, \hat{m}_1, \ldots, \hat{m}_{T-1}\}$.
4: **Initialization:**
5: if TERM == TRUE then,
6: $A_{-1}(0) = 0$; and $A_{-1}(s^t) = +\infty$ for $1 \leq s^t \leq N_s - 1$;
7: else
8: $A_{-1}(s^t) = 0$ for $0 \leq s^t \leq N_s - 1$;
9: end if;
10: **Main Algorithm:**
11: The edge metric is $y_t(e^t) = d_H(r_t, c(e^t))$ for all $t \in \{0, 1, \ldots, T - 1\}$ and $e \in \{0, 1, \ldots, N_e - 1\}$
12: for $t = 0, 1, \ldots, T - 1$ do // Forward recursion with metric and survivor updates
13: for $e^t = 0, 1, \ldots, N_e - 1$ do // Compute the edge metrics, store them in right-edge order
14: $M_t(e^t(e^t)) = A_{t-1}(s^t(e^t)) + y_t(e^t)$;
15: end for
16: for $s^t = 0, 1, \ldots, N_s - 1$ do // Determine survivors at the ending states
17: $A_t(s^t) = \min_{2^k s^k \leq e^t \leq 2^k (s^t+1) - 1} \{M_t(e^k)\}$;
18: $T_t(s^t) = \arg\min_{2^k s^k \leq e^t \leq 2^k (s^t+1) - 1} \{M_t(e^k)\}$;
19: end for
20: end for
21: if TERM == TRUE then, // Final global survivor at the end of the transmission
22: $\hat{s}^E_{T-1} = 0$;
23: else
24: $\hat{s}^E_{T-1} = \arg\min_{0 \leq s^t \leq N_s - 1} \{A_{T-1}(s^t)\}$;
25: end if;
26: for $t = T - 1, \ldots, 1, 0$ do // Traceback operation to identify output sequence
27: $\hat{e}^t_0 = e^t(T_t(\hat{s}^E_t))$;
28: $\hat{m}_t = m(\hat{e}^t_0)$;
29: $\hat{s}^E_{t-1} = s^t(\hat{e}^t_0)$;
30: end for
Algorithm 4: The VA for soft-decision inputs using indexes for the left- and right-hand sides of the trellis.

1: **Input:** The soft-decision received signal \( \mathbf{r} = \{ r_0, r_1, \ldots, r_{T-1} \} \).
2: **Input:** A Boolean flag TERM to indicate if the trellis is terminated.
3: **Output:** The estimated symbol sequence \( \hat{\mathbf{m}} = \{ \hat{m}_0, \hat{m}_1, \ldots, \hat{m}_{T-1} \} \).
4: **Initialization:**
5: if TERM == TRUE then
   6: \( A_{-1}(0) = 0; \) and \( A_{-1}(s^k) = -\infty \) for \( 1 \leq s^k \leq N_s - 1 \);
7: end if;
8: **Main Algorithm:**
9: for \( t = 0, 1, \ldots, T-1 \) do // Forward recursion with metric and survivor updates
10: \( M_t(e^t) =\)
11: end for
12: for \( s^k = 0, 1, \ldots, N_s - 1 \) do // Compute the edge metrics, store them in right-edge order
13: \( s^k \leq e^k \leq 2^k(s^k+1)-1 \) \( M_t(e^k) \) for all \( e^k \in \{0, 1, \ldots, N_e-1\} \)
14: end for
15: for \( s^k = 0, 1, \ldots, N_s - 1 \) do // Determine survivors at the ending states
16: \( T_t(s^k) =\)
17: end for
18: for \( t = T-1, \ldots, 1, 0 \) do // Traceback operation to identify output sequence
19: \( \hat{e}_t =\)
20: end for
Algorithm 5 The VA for soft-decision inputs with finite traceback.

1: **Input:** The soft-decision received signal \( \mathbf{r} = \{r_0, r_1, \ldots, r_{T-1}\} \).
2: **Input:** An integer \( \delta \) to specify the finite traceback length.
3: **Input:** A Boolean flag \( \text{TERM} \) to indicate if the trellis is terminated.
4: **Output:** The estimated symbol sequence \( \hat{\mathbf{m}} = \{\hat{m}_0, \hat{m}_1, \ldots, \hat{m}_{T-1}\} \).
5: **Initialization:**
6: if \( \text{TERM} == \text{TRUE} \) then
7: \( A_{-1}(0) = 0 \); and \( A_{-1}(s^k) = -\infty \) for \( 1 \leq s^k \leq N_s - 1 \);
8: else
9: \( A_{-1}(s^k) = 0 \) for \( 0 \leq s^k \leq N_s - 1 \);
10: end if;
11: **Main Algorithm:**
12: The edge metric is \( y_t(e^t) = \sum_{i=0}^{n-1} r_t^{(i)} a^{(i)}(e^t) = a(e^t) r_t^T \) for all \( t \in \{0, 1, \ldots, T-1\} \) and \( e \in \{0, 1, \ldots, N_e-1\} \)
13: for \( t = 0, 1, \ldots, T - 1 \) do // Forward recursion with metric and survivor updates
14: for \( e^t = 0, 1, \ldots, N_e-1 \) do // Compute the edge metrics, store them in right-edge order
15: \( M_t(e^t) = A_{t-1}(s^k(e^t)) + y_t(e^t) \);
16: end for
17: for \( s^k = 0, 1, \ldots, N_s - 1 \) do // Determine survivors at the ending states
18: \( A_t(s^k) = \max_{2^ks^k \leq e^t \leq 2^k(s^k+1)-1} \{M_t(e^k)\}; \)
19: \( T_t(s^k) = \arg \max_{2^ks^k \leq e^t \leq 2^k(s^k+1)-1} \{M_t(e^k)\}; \)
20: end for
21: if \( t \geq \delta \) then // Traceback operation to identify the output symbol with delay \( \delta \)
22: \( \hat{s}_t^E = \arg \max_{0 \leq s^k \leq N_s - 1} A_t(s^k); \)
23: for \( t' = t, t-1, \ldots, t - \delta + 1 \) do
24: \( \hat{s}_{t'-1}^E = s^k(T_{t'}(\hat{s}_{t'}^E)); \)
25: end for
26: \( \hat{\mathbf{m}}_{T-\delta} = \mathbf{m}(e^t(T_t(\hat{s}_{t-\delta}^E))); \)
27: end if;
28: end for
29: if \( \text{TERM} == \text{TRUE} \) then // Final global survivor at the end of the transmission
30: \( \hat{s}_{T-1}^E = 0; \)
31: else
32: \( \hat{s}_{T-1}^E = \arg \max_{0 \leq s^k \leq N_s - 1} A_{T-1}(s^k); \)
33: end if;
34: for \( t = T - 1, T - 2, \ldots, T - \delta \) do // Complete the traceback up to the most recent symbol
35: \( \hat{e}_t^E = e^t(T_t(\hat{s}_t^E)); \)
36: \( \hat{\mathbf{m}}_t = \mathbf{m}(\hat{e}_t^E); \)
37: \( \hat{s}_{t-1}^E = s^k(\hat{e}_t^E); \)
38: end for