Complete the following tasks. You should base your implementation on the template .m files that have been distributed via e-mail. The .m files with the word Template in their name contain “TODOs” for you. These files should be renamed with your first name in place of the word Template, the TODOs should be completed, and the .m files should be sent back to me via e-mail (use the e-mail address esp@eecs.ku.edu).

1. **Specify the Trellis Tables for the Differential Encoder.** In this project, we will be using the differential encoder (DE) shown in Figure 1. This encoder will serve as the “inner code” of a serial concatenated convolutional code (SCCC). The SCCC encoder is shown in Figure 2 (the symbol Π denotes an interleaver). The “outer code” for the SCCC system is the now-familiar (5,7) convolutional code (CC). We already have a trellis table for the outer code from a previous project. We need this trellis information for the inner code as well. In the file RunBerSimulationScccTemplate.m, you will find a “TODO” for specifying the trellis of the DE.

2. **Implement the SCCC Decoder using the Log-Based SISO Algorithm.** A block diagram of the SCCC decoder is shown in Figure 3. In the e-mail distribution, you already have implementations of the major blocks in this system. You also have a template function for the decoder. Your task then is to “wire” it all together properly. The data vectors that are passed back and forth between the SISO modules are in the form of log likelihood ratios (LLRs). In the case of the log-based SISO (Algorithm 7), the LLRs are computed in their exact form, so the scale factors $K_1$ and $K_2$ should be set to unity. The soft input $\lambda(u; I)$ for both the inner and outer SISOs should be initialized to zero before the first iteration (for later iterations, $\lambda(u; I)$ for the inner SISO will be non-zero). You do not need to run a BER simulation; however, the template simulation file is set up to do so. You just need to verify that the SCCC decoder works properly. The log-based SISO should be implemented as a MATLAB function with the following syntax:

   
   \[ [L_{uO}, L_{cO}] = \text{SisoLogXXX}(L_{uI}, L_{cI}, \text{TERM}, sS, \text{ue}, ce, sE, eR); \]

3. **Implement the SCCC Decoder using the Max–Log SISO Algorithm.** Repeat the previous task using the modified “max–log” SISO (Algorithm 8) instead of the original SISO. In this case, the LLRs are computed only approximately, and the algorithm tends to “overestimate” these LLRs. Thus, there is a noticeable performance improvement if the LLRs are scaled down via $K_1$ and $K_2$. There is no analytical method for determining $K_1$ and $K_2$; however, values of 0.75 offer good performance and they also allow for a simple hardware implementation that uses shifting and adding instead of dedicated multipliers. Once again, you do not need to run a BER simulation, but you can verify that the BER performance of the max–log SISO is worse than the original SISO. The max–log SISO should be implemented as a MATLAB function with the following syntax:

   \[ [L_{uO}, L_{cO}] = \text{SisoMaxLogXXX}(L_{uI}, L_{cI}, \text{TERM}, sS, \text{ue}, ce, sE, eR); \]
Figure 1: Differential encoder.

Figure 2: Serial concatenated convolutional encoder.

Figure 3: Serial concatenated convolutional decoder.
Algorithm 7 The Log-Based SISO Algorithm.

1: **Input:** The *a priori* LLRs for \( \mathbf{u} = \{u_0, u_1, \ldots, u_{T-1}\} \), \( \lambda(\mathbf{u}; 1) = \{\lambda_i(u_i; 1)\}_{i=0}^{k_0T-1} \), \( k_0T \) LLRs in total.
2: **Input:** The *a priori* LLRs for \( \mathbf{c} = \{c_0, c_1, \ldots, c_{T-1}\} \), \( \lambda(\mathbf{c}; 1) = \{\lambda_i(c_i; 1)\}_{i=0}^{n_0T-1} \), \( n_0T \) LLRs in total.
3: **Input:** A Boolean flag \( \text{TERM} \) to indicate if the trellis is terminated.
4: **Outputs:** The *a posteriori* LLRs \( \lambda(\mathbf{u}; O) \) and \( \lambda(\mathbf{c}; O) \).
5: **Initialization:**
6: if \( \text{TERM} == \text{TRUE} \) then,
7: \( A_{-1}(0) = 0; \) and \( A_{-1}(s) = -\infty \) for \( 1 \leq s \leq N_s - 1 \);
8: \( B_{T-1}(0) = 0; \) and \( B_{T-1}(s) = -\infty \) for \( 1 \leq s \leq N_s - 1 \);
9: else
10: \( A_{-1}(s) = 0 \) for \( 0 \leq s \leq N_s - 1 \);
11: \( B_{T-1}(s) = 0 \) for \( 0 \leq s \leq N_s - 1 \);
12: end if;
13: **Main Algorithm:**
14: The edge metric is \( \gamma_t(e) \triangleq p_t(U(e); 1) + p_t(C(e); 1) = \frac{1}{2} \left[ \lambda_t(u_i; 1) a(\mathbf{u}(e))^T + \lambda_t(c(i); 1) a(\mathbf{c}(e))^T \right] \)
15: \( = \frac{1}{2} \left[ \sum_{j=0}^{k_0-1} \lambda_{k_0t+j}(u; 1) a(u(j)(e)) + \sum_{j=0}^{n_0-1} \lambda_{n_0t+j}(c; 1) a(c(j)(e)) \right] \) for all \( t \in \{0, 1, \ldots, T-1\} \) and \( e \in \{0, 1, \ldots, N_s - 1\} \)
16: for \( t = 0, 1, \ldots, T - 2 \) do 
17: \( A_t(s) = \sum_{ \mathbf{c}^t(e) = i } A_{t-1}(s^t(e)) + \gamma_t(e); \)
18: end for
19: end for
20: for \( t = T - 2, 1, 0 \) do 
21: for \( s = 0, 1, \ldots, N_s - 1 \) do
22: \( B_t(s) = \sum_{ \mathbf{c}^t(e) = i } B_{t+1}(s^t(e)) + \gamma_{t+1}(e); \)
23: end for
24: end for
25: for \( t = 0, 1, \ldots, T - 1 \) do 
26: for \( j = 0, 1, \ldots, k_0 - 1 \) do
27: \( \lambda_{k_0t+j}(u; O) = \sum_{ \mathbf{u}^t(e) = 1 } A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)) - \sum_{ \mathbf{e}^t(e) = 0 } A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)); \)
28: end for
29: for \( j = 0, 1, \ldots, n_0 - 1 \) do
30: \( \lambda_{n_0t+j}(c; O) = \sum_{ \mathbf{c}^t(e) = 1 } A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)) - \sum_{ \mathbf{e}^t(e) = 0 } A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)); \)
31: end for
32: end for
33: The "extrinsic" version of an output LLR (if desired) is obtained by normalizing the output LLR by the respective input LLR, i.e. \( \lambda(\mathbf{u}; O) \sim \lambda(\mathbf{u}; 1). \)
Algorithm 8 The “Max–Log” SISO Algorithm. The terms “LLR” and “pdf” are used below; however, precise scaling of these inputs (i.e. knowledge and use of the factor $L_r$) is not necessary for this algorithm to work properly.

1. **Input:** The a priori LLRs for $u = \{u_0, u_1, \ldots, u_{T-1}\}$, $\lambda(u;1) = \{\lambda_i(u;1)\}_{i=0}^{k_0-1}$, $k_0 T$ LLRs in total.
2. **Input:** The a priori LLRs for $c = \{c_0, c_1, \ldots, c_{T-1}\}$, $\lambda(c;1) = \{\lambda_i(c;1)\}_{i=0}^{n_0-1}$, $n_0 T$ LLRs in total.
3. **Input:** A Boolean flag TERM to indicate if the trellis is terminated.
4. **Outputs:** The a posteriori LLRs $\lambda(u;O)$ and $\lambda(c;O)$.
5. **Initialization:**
6. **if** TERM == TRUE **then**
7. \[ A_{-1}(0) = 0; \quad \text{and} \quad A_{-1}(s) = -\infty \quad \text{for} \quad 1 \leq s \leq N_s - 1; \]
8. \[ B_{T-1}(0) = 0; \quad \text{and} \quad B_{T-1}(s) = -\infty \quad \text{for} \quad 1 \leq s \leq N_s - 1; \]
9. **else**
10. \[ A_{-1}(s) = 0 \quad \text{for} \quad 0 \leq s \leq N_s - 1; \]
11. \[ B_{T-1}(s) = 0 \quad \text{for} \quad 0 \leq s \leq N_s - 1; \]
12. **end if**;
13. **Main Algorithm:**
14. The edge metric is $\gamma_t(e) \triangleq p_t(U(e);1) + p_t(C(e);1) = \frac{1}{T} \left[ \lambda_t(u;1)a(u(e)) + \lambda_t(c;1)a(c(e)) \right]$ for all $t \in \{0,1,\ldots,T-1\}$ and $e \in \{0,1,\ldots,N_E-1\}$
15. **for** $t = 0,1,\ldots,T-2$ **do** // Forward recursion
16. **for** $s = 0,1,\ldots,N_s - 1$ **do**
17. \[ A_t(s) = \max_{e \in s^t(e) = 1} \left\{ A_{t-1}(s^t(e)) + \gamma_t(e) \right\}; \]
18. **end for**
19. **end for**
20. **for** $t = T - 2,\ldots,1,0$ **do** // Backward recursion
21. **for** $s = 0,1,\ldots,N_s - 1$ **do**
22. \[ B_t(s) = \max_{e \in s^t(e) = 1} \left\{ B_{t+1}(s^t(e)) + \gamma_{t+1}(e) \right\}; \]
23. **end for**
24. **end for**
25. **for** $t = 0,1,\ldots,T - 1$ **do** // Completion Step
26. **for** $j = 0,1,\ldots,k_0 - 1$ **do**
27. \[ \lambda_{k_0t+j}(u;O) = \max_{c \in u^{(j)}(e) = 1} \left\{ A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)) \right\} - \max_{c \in u^{(j)}(e) = 0} \left\{ A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)) \right\}; \]
28. **end for**
29. **for** $j = 0,1,\ldots,n_0 - 1$ **do**
30. \[ \lambda_{n_0t+j}(c;O) = \max_{c \in c^{(j)}(e) = 1} \left\{ A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)) \right\} - \max_{c \in c^{(j)}(e) = 0} \left\{ A_{t-1}(s^t(e)) + \gamma_t(e) + B_t(s^t(e)) \right\}; \]
31. **end for**
32. **end for**
33. **end for**
34. The “extrinsic” version of an output LLR (if desired) is obtained by normalizing the output LLR by the respective input LLR, i.e. $\gamma(u;O) = \lambda(u;O) - \Lambda(u;1)$.