Serial Concatenated Convolutional Codes with Iterative Decoding
EECS 869: Error Control Coding
Fall 2017

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 $\boldsymbol{\lambda}(u;I)$ for both the inner and outer SISOs should be initialized to zero before the first iteration (for later iterations, $\boldsymbol{\lambda}(u;I)$ 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] = SisoLogXXX(L_uI, L_cI, TERM, sS, 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] = SisoMaxLogXXX(L_uI, L_cI, TERM, sS, ue, ce, sE, eR);
```

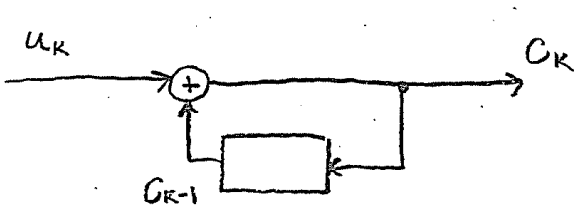

Figure 1: Differential encoder.

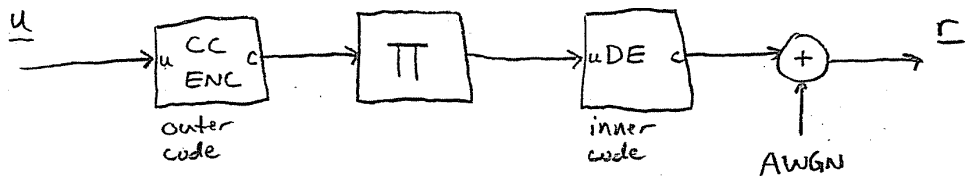

Figure 2: Serial concatenated convolutional encoder.

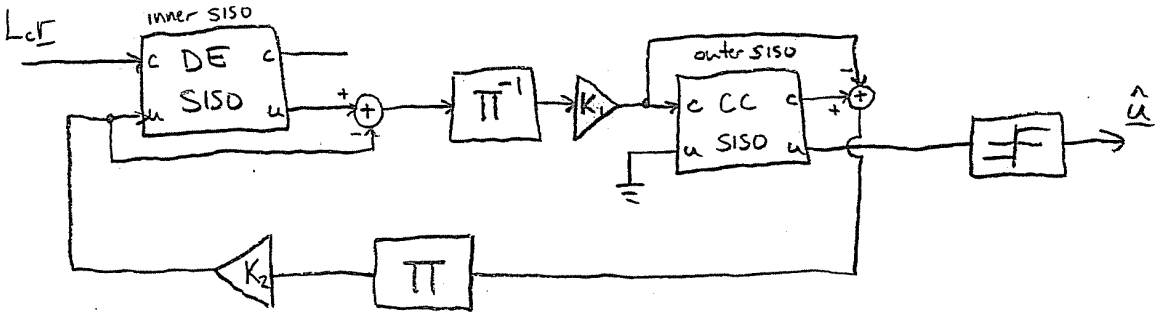

Figure 3: Serial concatenated convolutional decoder.

Algorithm 7 The Log-Based SISO Algorithm.

Input: The *a priori* LLRs for $\mathbf{u} = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{T-1}\}$, $\boldsymbol{\lambda}(u;\mathrm{I}) = \{\lambda_i(u;\mathrm{I})\}_{i=0}^{k_0 T-1}$, $k_0 T$ LLRs in total.
Input: The *a priori* LLRs for $\mathbf{c} = \{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{T-1}\}$, $\boldsymbol{\lambda}(c;\mathrm{I}) = \{\lambda_i(c;\mathrm{I})\}_{i=0}^{n_0 T-1}$, $n_0 T$ LLRs in total.
Input: A Boolean flag TERM to indicate if the trellis is terminated.
Outputs: The *a posteriori* LLRs $\boldsymbol{\lambda}(u;\mathrm{O})$ and $\boldsymbol{\lambda}(c;\mathrm{O})$.
Initialization:
if TERM == TRUE **then**,
    $A_{-1}(0) = 0$; and $A_{-1}(s) = -\infty$ for $1 \le s \le N_\mathrm{s} - 1$;
    $B_{T-1}(0) = 0$; and $B_{T-1}(s) = -\infty$ for $1 \le s \le N_\mathrm{s} - 1$;
else
    $A_{-1}(s) = 0$ for $0 \le s \le N_\mathrm{s} - 1$;
    $B_{T-1}(s) = 0$ for $0 \le s \le N_\mathrm{s} - 1$;
end if;
Main Algorithm:
The edge metric is $\gamma_t(e) \triangleq p_t(U(e);\mathrm{I}) + p_t(C(e);\mathrm{I}) = \frac{1}{2}\left[\boldsymbol{\lambda}_t(u;\mathrm{I})a(\mathbf{u}(e))^T + \boldsymbol{\lambda}_t(c;\mathrm{I})a(\mathbf{c}(e))^T\right]$
$= \frac{1}{2}\left[\sum_{j=0}^{k_0-1}\lambda_{k_0 t+j}(u;\mathrm{I})a(u^{(j)}(e)) + \sum_{j=0}^{n_0-1}\lambda_{n_0 t+j}(c;\mathrm{I})a(c^{(j)}(e))\right]$ for all $t \in \{0,1,\ldots,T-1\}$ and $e \in \{0,1,\ldots,N_\mathrm{E}-1\}$
for $t = 0,1,\ldots,T-2$ **do** // Forward recursion
    for $s = 0,1,\ldots,N_\mathrm{s}-1$ **do**
        $A_t(s) = \hat{\sum_{e:s^\mathrm{E}(e)=s}} A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e)$;
    end for
end for
for $t = T-2,\ldots,1,0$ **do** // Backward recursion
    for $s = 0,1,\ldots,N_\mathrm{s}-1$ **do**
        $B_t(s) = \hat{\sum_{e:s^\mathrm{S}(e)=s}} B_{t+1}(s^\mathrm{E}(e)) + \gamma_{t+1}(e)$;
    end for
end for
for $t = 0,1,\ldots,T-1$ **do** // Completion Step
    for $j = 0,1,\ldots,k_0-1$ **do**
        $\lambda_{k_0 t+j}(u;\mathrm{O}) = \hat{\sum_{e:u^{(j)}(e)=1}} A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e)) \; - \hat{\sum_{e:u^{(j)}(e)=0}} A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e))$;
    end for
    for $j = 0,1,\ldots,n_0-1$ **do**
        $\lambda_{n_0 t+j}(c;\mathrm{O}) = \hat{\sum_{e:c^{(j)}(e)=1}} A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e)) \; - \hat{\sum_{e:c^{(j)}(e)=0}} A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e))$;
    end for
end for
The "extrinsic" version of an output LLR (if desired) is obtained by normalizing the output LLR by the respective input LLR, i.e. $\boldsymbol{\lambda}(u;\mathrm{O}) - \boldsymbol{\lambda}(u;\mathrm{I})$.

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_c) is not necessary for this algorithm to work properly.

Input: The *a priori* LLRs for $\mathbf{u} = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{T-1}\}$, $\boldsymbol{\lambda}(u;\mathrm{I}) = \{\lambda_i(u;\mathrm{I})\}_{i=0}^{k_0 T-1}$, $k_0 T$ LLRs in total.

Input: The *a priori* LLRs for $\mathbf{c} = \{\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{T-1}\}$, $\boldsymbol{\lambda}(c;\mathrm{I}) = \{\lambda_i(c;\mathrm{I})\}_{i=0}^{n_0 T-1}$, $n_0 T$ LLRs in total.

Input: A Boolean flag TERM to indicate if the trellis is terminated.

Outputs: The *a posteriori* LLRs $\boldsymbol{\lambda}(u;\mathrm{O})$ and $\boldsymbol{\lambda}(c;\mathrm{O})$.

Initialization:

if TERM == TRUE **then**,

$A_{-1}(0) = 0$; and $A_{-1}(s) = -\infty$ for $1 \le s \le N_\mathrm{S} - 1$;

$B_{T-1}(0) = 0$; and $B_{T-1}(s) = -\infty$ for $1 \le s \le N_\mathrm{S} - 1$;

else

$A_{-1}(s) = 0$ for $0 \le s \le N_\mathrm{S} - 1$;

$B_{T-1}(s) = 0$ for $0 \le s \le N_\mathrm{S} - 1$;

end if;

Main Algorithm:

The edge metric is $\gamma_t(e) \triangleq p_t(U(e);\mathrm{I}) + p_t(C(e);\mathrm{I}) = \frac{1}{2}\left[\boldsymbol{\lambda}_t(u;\mathrm{I})a(\mathbf{u}(e))^T + \boldsymbol{\lambda}_t(c;\mathrm{I})a(\mathbf{c}(e))^T\right]$

$= \frac{1}{2}\left[\sum_{j=0}^{k_0-1}\lambda_{k_0 t+j}(u;\mathrm{I})a(u^{(j)}(e)) + \sum_{j=0}^{n_0-1}\lambda_{n_0 t+j}(c;\mathrm{I})a(c^{(j)}(e))\right]$ for all $t \in \{0,1,\ldots,T-1\}$ and $e \in \{0,1,\ldots,N_\mathrm{E}-1\}$

for $t = 0,1,\ldots,T-2$ **do** // Forward recursion

for $s = 0,1,\ldots,N_\mathrm{S}-1$ **do**

$A_t(s) = \max_{e:s^\mathrm{E}(e)=s}\left\{A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e)\right\}$;

end for

end for

for $t = T-2,\ldots,1,0$ **do** // Backward recursion

for $s = 0,1,\ldots,N_\mathrm{S}-1$ **do**

$B_t(s) = \max_{e:s^\mathrm{S}(e)=s}\left\{B_{t+1}(s^\mathrm{E}(e)) + \gamma_{t+1}(e)\right\}$;

end for

end for

for $t = 0,1,\ldots,T-1$ **do** // Completion Step

for $j = 0,1,\ldots,k_0-1$ **do**

$\lambda_{k_0 t+j}(u;\mathrm{O}) = \max_{e:u^{(j)}(e)=1}\left\{A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e))\right\} - \max_{e:u^{(j)}(e)=0}\left\{A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e))\right\}$;

end for

for $j = 0,1,\ldots,n_0-1$ **do**

$\lambda_{n_0 t+j}(c;\mathrm{O}) = \max_{e:c^{(j)}(e)=1}\left\{A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e))\right\} - \max_{e:c^{(j)}(e)=0}\left\{A_{t-1}(s^\mathrm{S}(e)) + \gamma_t(e) + B_t(s^\mathrm{E}(e))\right\}$;

end for

end for

The "extrinsic" version of an output LLR (if desired) is obtained by normalizing the output LLR by the respective input LLR, i.e. $\boldsymbol{\lambda}(u;\mathrm{O}) - \boldsymbol{\lambda}(u;\mathrm{I})$.