Motivation. What is wrong with block codes?

- Short linear codes which can be represented by trellises are efficient for SISO decoding. Decoding complexity grows exponentially with code length.

- Algebraic codes with hard-decision decoding. These codes are asymptotically bad since their minimum distance does not grow linearly with code length for a fixed code rate. Decoding complexity is proportional to $n^2$ for codes of length $n$. 
What we expect from convolutional codes?

- Convolutional codes: infinite length codes with convenient ML/MAP soft-decision decoding
- Properly terminated convolutional codes can be used for obtaining long codes with linearly growing encoding and decoding complexity of soft decision decoding, for example, turbo-codes.
Convolutional codes

- Block coding: codewords are computed for non-intersecting blocks of information symbols.
- Convolutional coding: code symbols for a certain portion of data depend on the previously encoded and transmitted data.

In other words, in case of convolutional coding we deal with encoders with memory. Newly arrived information symbols partially update the encoder memory and code symbols depend on both the input data and the data in the encoder memory.
shift registers

Block of mod 2 adders

$u_1 \rightarrow 1 \rightarrow 2 \rightarrow \cdots \rightarrow v_1 \rightarrow v_1$

$u_k \rightarrow 1 \rightarrow 2 \rightarrow \cdots \rightarrow v_k \rightarrow v_n$
a) $R = 1/2$, $m = \nu = 2$

$$G(D) = (1 + D^2 \quad 1 + D + D^2)$$

$$G = \begin{pmatrix}
11 & 01 & 11 & 00 & \cdots & \cdots \\
00 & 11 & 01 & 11 & 00 & \cdots \\
\cdots & 00 & 11 & 01 & 11 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

b) $R = 2/3$, $m = 1$, $\nu = 2$

$$G(D) = \begin{pmatrix}
1 + D & 0 & D \\
1 & 1 + D & 1 + D
\end{pmatrix}$$

$$G = \begin{pmatrix}
100 & 101 & 000 & \cdots & \cdots \\
111 & 011 & 000 & \cdots & \cdots \\
\cdots & 100 & 101 & 000 & \cdots \\
\cdots & 111 & 011 & 000 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$
Code parameters

• The sum of lengths of registers $\nu = \sum_{i=1}^{k} \nu_i$ is called constraint length of the encoder,
• Maximum length of registers $m = \max_i \{\nu_i\}$ is called encoder memory
• Connections of memory cells with adders can be determined by binary arrays
  $g_{ij} = (g_{ij0}, g_{ij1}, \ldots, g_{ij\nu_i}), \ i = 1, 2, \ldots, k, \ j = 1, 2, \ldots, n$ where $g_{ijh} = 0$ means “no connection”, and 1 means that $h$-th memory element of $i$-th register is connected with $j$-th output modulo 2 adder. Vectors $g_i$ are called generators of the code.
Generators are presented as *generating polynomials* of the formal variable $D$.

The most famous rate $R = 1/2$ convolutional $(5,7)$-code has generator polynomials:

\[ g_1(D) = 1 + D^2, \]
\[ g_2(D) = 1 + D + D^2. \]

Generator polynomials are elements of a polynomial generator matrix

\[ G(D) = \begin{pmatrix} 1 + D^2 & 1 + D + D^2 \end{pmatrix}. \]
In general, the encoder of rate $R = k/n$ convolutional code is determined by a polynomial generator matrix of size $k \times n$. The polynomial generator matrix corresponds to a scheme based on feedback-free registers or FIR filters. Otherwise elements are presented by ratios of polynomials (IIR filters), e.g. IIR encoder of code (5,7):

$$G(D) = \begin{pmatrix} 1 & \frac{1+D+D^2}{1+D^2} \end{pmatrix}.$$
A convolutional code can be interpreted as a linear code over the field of rational functions (possibly, of infinite degree) with coefficients from a finite field.

Information polynomial has the form

\[ u(D) = (u_1(D) \ldots u_k(D)) = u_0 + u_1D + \ldots, \text{ where } u_i \in \{0, 1\}^k \]

Encoding for the rate \( R = k/n \)-code

\[ v(D) = u(D)G(D). \]

Codeword is a vector of polynomials

\[ v(D) = (v_1(D) \ldots v_n(D)) = v_0 + v_1D + \ldots, \text{ where } v_i \in \{0, 1\}^n \]
Polynomial representation. Example

Let

\[ u = 1 \, 0 \, 0 \, 1 \, 0 \, 1 \ldots \rightarrow u(D) = 1 + D^3 + D^5, \ldots \]

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

Encoding:

\[ v = uG(D) = (v_1(D), v_2(D)) \]

\[ v_1(D) = u(D)(1 + D^2) = 1 + D^2 + D^3 + D^7 + \ldots \]

\[ v_2(D) = u(D)(1 + D + D^2) = 1 + D + D^2 + D^3 + D^4 + D^6 + D^7 + \ldots \]

or

\[ v(D) = (1, 1) + (0, 1)D + (1, 1)D^2 + \ldots \]
For the memory $m$ polynomial encoder, its generator matrix is

$$G(D) = G_0 + G_1 D + \ldots + G_m D^m.$$ 

Polynomials 1, $D$, $D^2$,... form (infinite) basis of the space of information polynomials. The corresponding codewords of convolutional code form a basis of the space of codewords. The binary generator matrix is

$$G = \begin{pmatrix}
G_0 & G_1 & \cdots & G_m & 0_{k,n} & \cdots \\
0_{k,n} & G_0 & G_1 & \cdots & G_m & 0_{k,n} & \cdots \\
0_{k,n} & G_0 & G_1 & \cdots & G_m & 0_{k,n} & \cdots \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\
\end{pmatrix},$$

where $0_{k,n}$ denotes all-zero matrix of size $k \times n$, and $G_i$ are matrix coefficients for $D^i$ in the series expansion of $G(D)$ over degrees of $D$. 
a) \[ G(D) = (1 + D^2) \begin{pmatrix} 1 + D + D^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ \end{pmatrix} D + \begin{pmatrix} 1 & 1 \\ \end{pmatrix} D^2 \]

Binary form:

\[
G = \begin{pmatrix} 11 & 01 & 11 & 00 & \cdots \\ 00 & 11 & 01 & 11 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \end{pmatrix}
\]

b) \[ G(D) = \begin{pmatrix} 1 + D \\ 1 + D \\ \end{pmatrix} \begin{pmatrix} 0 \\ 1 + D \\ 1 + D \\ \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ \end{pmatrix} D + \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ \end{pmatrix} D 
\]

\[
G = \begin{pmatrix} 100 & 101 & 000 & \cdots & \cdots \\ 111 & 011 & 000 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & 100 & 101 & 000 & \cdots \\ \vdots & 111 & 011 & 000 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \end{pmatrix}
\]
Code tree. Example for (5,7)-conv. code

Figure: Graph representation by a code tree
Code trellis. Example for (5,7)-conv. code

Figure: Graph representation by a code trellis
Figure: Graph representation by a state diagram
If a convolutional encoder may generate finite-weight codeword for an infinite-weight information sequence, such encoder is called catastrophic.
Proposition: Rate $R = 1/n$ convolutional encoder is catastrophic if the GCD of $\{g_i(D)\}$, $i = 1, 2, \ldots, n$ is neither 1 nor monomial $D^i$.

Proof.
W.l.o.g. assume that $G_0 \neq 0$. Let $a(D) = \text{GCD}(g_1(D), g_2(D), \ldots, g_n(D))$. Since at least one of generators is not multiple of $D$, then GCD can be written as $a(D) = 1 + Db(D)$, where $b(D) \neq 0$. If input infinite-weight sequence of the encoder is

$$u(D) = 1/a(D) = 1 + Db(D) + D^2 b^2(D) + \ldots$$

then the encoder output $v(D) = (g_1(D)/a(D), g_2(D)/a(D), \ldots, g_n(D)/a(D))$ has finite weight upperbounded by $\sum_{i=1}^{n} (\nu_i + 1)$. 

Example

\[ G(D) = (1 + D \ 1 + D^2), \quad \text{GCD}(g_1(D), g_2(D)) = (1 + D). \]

Infinite-weight input

\[ u(D) = \frac{1}{1 + D} = 1 + D + D^2 + D^3 + \ldots \]

is encoded to

\[ v(D) = [11] + [01]D + [00]D^2 + [00]D^3 + \ldots \]

of weight 3.
Minimum distance $d_f$ of convolutional code is called “free distance”. Properties of free distance:

- Free distance can be found as minimal weight of codewords.
- Free distance is a finite value.
- For finding minimum-weight codeword it is enough to search over the trellis paths diverging from all-zero path at the root of the trellis and do not reach the zero state until their termini (first error sub-tree).
For evaluating convolutional code performance we introduce weight enumerators

\[ T(D) = \sum_{d=1}^{\infty} t_d D^d, \quad F(D) = \sum_{d=1}^{\infty} f_d D^d, \]

where \( t_d \) is the number of weight \( d \) codewords in the first error tree and \( f_d \) is the total number of information symbols equal to 1 in the information sequences encoded to weight \( d \) codewords.
These two functions are used for computing the probability of error event (probability that the ML decoding error appears at moment $t$) and the bit error probability (the portion of decoding errors in infinitely-long transmitted codeword).
Simple bounds which are rather tight for large SNRs:

\[
P_e \leq B(d_f) T(D) \big|_{D=D_0}
\]

\[
P_{eb} \leq \frac{1}{k} B(d_f) F(D) \big|_{D=D_0},
\]

where for the BSC:

\[
B(w) = \frac{1 - p}{1 - 2p} \sqrt{\frac{2}{\pi w}}; \quad D_0 = 2\sqrt{p(1-p)}.
\]

and for the AWGN channel:

\[
B(w) = Q \left( \sqrt{\frac{2wE}{N_0}} \right) e^{wE/N_0}; \quad D_0 = e^{-E/N_0};
\]

\[
Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-\frac{t^2}{2}} dt.
\]

Spectra $T(D), F(D)$ are tabulated for most important code parameters.
Block vs convolutional codes

Figure: Comparison of block and convolutional codes. Block codes: 1) Code (24,12), \( d=8 \); (complexity \( \sim 2^8 \)); 2) Code (48,24), \( d=12 \); (\( \sim 2^{12} \))

Convolutional codes: 3) \( \nu = 4 \), \( g = (35,23) \), \( d_f=7 \); 4) \( \nu = 8 \), \( g = (657,435) \), \( d_f=12 \); 5) \( \nu = 12 \), \( g = (15521,11677) \), \( d_f=16 \)
Let $a_0, a_1, a_2, \ldots$ be a sequence of real numbers then

$$g(D) = \sum_{i=0}^{\infty} a_i D^i,$$

where $D$ is a formal variable, is called combinatorial generating function.

Consider two generating functions $g(D) = \sum_{i=0}^{\infty} a_i D^i$ and $f(D) = \sum_{i=0}^{\infty} b_i D^i$ then their product $g(D)f(D) = \sum_{i=0}^{\infty} c_i D^i = \sum_i \sum_{(k,j), k+j=i} a_k b_j D^i,$

$c_i = \sum_{k=0}^{\infty} a_k b_{i-k}$.

Thus, the sequence $c_0, c_1, c_2, \ldots$ is a convolution of the sequences $a_0, a_1, \ldots$ and $b_0, b_1, \ldots$. 
Weight enumerators

Figure: State transition diagrams a) State diagram with edge g.f., b) State diagram with two-variable g.f for edge weight and the corresponding information weight
\begin{align*}
\begin{cases}
  g_0(D) & = D^2 g_1(D); \\
  g_1(D) & = D g_2(D) + D g_3(D); \\
  g_2(D) & = D^2 + g_1(D); \\
  g_3(D) & = D g_2(D) + D g_3(D). 
\end{cases}
\end{align*}

\[ g_0(D) = \frac{D^5}{1 - 2D} = D^5 + 2D^6 + 4D^7 + \ldots. \]
\[ T(D, I) = \sum_{d=1}^{\infty} \sum_{i=1}^{\infty} t(d, i) D^d I^i. \]

\[
\begin{align*}
g_0(D, I) &= D^2 g_1(D, I); \\
g_1(D, I) &= D g_2(D, I) + D g_3(D, I); \\
g_2(D, I) &= D^2 I + D g_1(D, I); \\
g_3(D, I) &= D l g_2(D) + D l g_3(D, I). \\
\end{align*}
\]

Solution:

\[ T(D, I) = g_0(D, I) = \\
= \frac{D^5 I}{1 - 2DI} \\
= D^5 I + 2D^6 I^2 + 4D^7 I^3 + 8D^8 I^4 \ldots \]
The functions $T(D)$ and $T(D, I)$ are called transfer function and extended transfer function for node error. We introduce the bit error transfer function

\[ F(D) = \sum_{d=1}^{\infty} f_d D^d, \quad f_d = \sum_{i=1}^{\infty} it(d, i). \]

where $f_d$ is the total number of nonzero information bits corresponding to all codewords of weight $d$ in the first-error subtree.

\[ F(D) = \sum_{d=1}^{\infty} \sum_{i=1}^{\infty} it(d, i) D^d = \frac{\partial}{\partial I} T(D, I) \bigg|_{I=1}. \]

For the (5,7) code

\[ T(D, I) = D^5 I + 2D^6 I^2 + 4D^7 I^3 + 8D^8 I^4 + \ldots \]

\[ F(D) = D^5 + 4D^6 + 12D^7 + 32D^8 + \ldots \].
Termination is used when a finite number of bits $k$ has to be transmitted. Options are:

<table>
<thead>
<tr>
<th>Solution</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truncation</td>
<td>No rate loss but the last bits are not protected</td>
</tr>
<tr>
<td>Zero-tail terminating</td>
<td>All bits are protected at the cost of rate loss</td>
</tr>
<tr>
<td>Tail-biting</td>
<td>All bits are protected. No rate loss. Higher decoding complexity</td>
</tr>
</tbody>
</table>
Zero-tail terminated codes

\[ G = (L + m)n_0 \]

\[ R = R_c \frac{L}{L+m} < R_c \]

\[ k = Lk_0 \text{ information bits followed by } \nu \text{ zeros} \]

Convolutional encoder \( R_c = \frac{k_0}{n_0} \)

Figure: Zero-tail terminated convolutional code
Tail-biting codes

$k = Lk_0$ is the number of information bits. Initially, $\nu$ bits are in the convolutional encoder register. After $k$ encoding steps the final state of the register coincides with the initial state.

Encoder memory $\nu = mk_0$ bits

Convolutional encoder $R_c = k_0/n_0$

Code bits $n = Ln_0$

Figure: TB convolutional code
Example.
Let $R = 1/2$, and $G(D) = (1, 1 + D + D^2)$. Binary generator matrix for (10, 5) TB-code
\[
G = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 1
\end{pmatrix}
\]

$d_{\text{min}} = 4$.  

Irina Bocharova, Tartu, 2022, Page 33 of 38
Figure: TB convolutional code trellis

Codewords correspond to path from any state at level 0 to the same state at level $L = n/n_0 = 5$. 
Algorithm 1: Brute-force ML decoding for TB codes

Input: Channel output $y$

Output: Candidate codeword $c$

1. Let $c_0 \leftarrow 0$. $\mu_0 = p(y|c_0)$;
2. for States $s = 0$ to $2^\nu - 1$ do
3. Assign zero metric to $s$ and $-\infty$ metric to all other states.
4. Use the Viterbi algorithm for finding ML candidate path $c$
   with metric $\mu = p(y|c)$ to the state $s$ at level $L$.
5. if $\mu > \mu_0$ then
6. $\mu_0 = \mu$, $c_0 = c$
8. return $c_0$

Complexity: Number of VA attempts is $2^\nu$, complexity of each attempt is $2^\nu$, the overall complexity is of order $2^{2\nu}$
**Table:** Search results and upper bounds on the TB trellis decoding complexity

<table>
<thead>
<tr>
<th>Code</th>
<th>Search result</th>
<th>Upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>(24,12,8)</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>(48,24,12)</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>(92,46,16)</td>
<td>26</td>
<td>46</td>
</tr>
<tr>
<td>(52,26,10)</td>
<td>12</td>
<td>26</td>
</tr>
</tbody>
</table>
A linear block code $C$ of length $n = ml$ over a finite field $GF(q)$ is called a quasi-cyclic code of index $l$ if for every codeword $c \in C$ there exists a number $l$ such that the codeword obtained by $l$ cyclic shifts is also a codeword in $C$. That is,

$$c = (c_0, \ldots, c_{n-1}) \in C \rightarrow c' = (c_{n-l}, \ldots, c_0, \ldots c_{n-l-1}) \in C.$$

The binary $(6, 3)$ code with generator matrix

$$G = \begin{pmatrix}
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1
\end{pmatrix} \quad (1)$$

is a quasi-cyclic code with $l = 2$. To ease the visualization we can write the shifts as blocks,

$$G = \begin{pmatrix}
11 & 01 & 00 \\
00 & 11 & 01 \\
01 & 00 & 11
\end{pmatrix} \quad (2)$$
For memoryless channel ML decoding is reduced to computing and comparing additive metrics of codewords. Metric formula depends on the channel model (Hamming distance, scalar product, etc).

By using trellis representation of the code ML decoding is reduced to search of the shortest path over the graph.

Minimal trellis of the code, is unique (if time axis is fixed).

ML decoding is implemented by using the Viterbi algorithm.