Convolutional codes simulation using Matlab

EUGEN PETAC
Ovidius University of Constantza,
ROMANIA
ABDEL-RAHMAN ALZOUBAIDI
Mutah University
JORDAN

Abstract: In order to reduce the effects of random and burst errors in transmitted signal it is necessary to use error-control coding. We researched some possibilities of such coding using the MATLAB Communications Toolbox. There are two types of codes available Linear Block Codes and Convolutional Codes. In block coding the coding algorithm transforms each piece (block) of information into a code word part of which is a generated structured redundancy. Convolutional code uses an extra parameter (memory). This puts an extra constraint on the code. Convolutional codes operate on serial data, one or a few bits at a time. This paper describes basic aspects of Convolutional codes and illustrates Matlab encoding and decoding implementations. Convolutional codes are often used to improve the performance of radio and satellite links.

Key words: - Convolutional codes, error-control coding, radio and satellite links.

1. Introduction

Convolutional codes are commonly specified by three parameters (n,k,m): n = number of output bits; k = number of input bits; m = number of memory registers. The quantity k/n called the code rate, is a measure of the efficiency of the code. Commonly k and n parameters range from 1 to 8, m from 2 to 10 and the code rate from 1/8 to 7/8 except for deep space applications where code rates as low as 1/100 or even longer have been employed.

Often the manufacturers of convolutional code chips specify [1] the code by parameters (n,k,L), The quantity L is called the constraint length of the code and is defined by Constraint Length, $L = k\ (m-1)$. The constraint length L represents the number of bits in the encoder memory that affect the generation of the n output bits. The constraint length L is also referred to by the capital letter K, which can be confusing with the lower case k, which represents the number of input bits. In some books K is defined as equal to product the of k and m. Often in commercial spec, the codes are specified by (r, K), where r = the code rate k/n and K is the constraint length. The constraint length K however is equal to L – 1, as defined in this paper.

Even though a convolutional coder accepts a fixed number of message symbols and produces a fixed number of code symbols, its computations depend not only on the current set of input symbols but on some of the previous input symbols.

In general, a rate $R=k/n$, $k \le n$, convolutional encoder input (information sequence) is a sequence of binary k-tuples, $u = ..,u_{-1},\ u_0,\ u_1,\ u_2,...,$ where $u_i = (u_i^{(1)}...u_i^{(k)})$. The output (code sequence) is a sequence of binary n-tuples, $v = ..,v_{-1},\ v_0,\ v_1,\ v_2,...,$ where $v_i = (v_i^{(1)}...v_i^{(n)})$. The sequences must start at a finite (positive or negative) time and may or may not end.

The relation between the information sequences and the code sequences is determined by the equation

$$v = uG\ ,$$

where

$$G = \begin{pmatrix} G_0 & G_1 & \dots & G_m & & & \\ & G_0 & G_1 & \dots & G_m & & \\ & & G_0 & G_1 & \dots & G_m & \\ & & & \dots & \dots & \dots & \dots \end{pmatrix}$$

is the semi-infinite generator matrix, and where the sub-matrices G_i , $0 \le i \le m$, are binary kXn matrices. The arithmetic in $v = uG$ is carried out over the binary field, F_2 , and the parts left blank in the generator matrix G are assumed to be filled in with zeros. The right hand side of $v= uG$ defines a discrete-time convolution between u and

$g = (G_0 G_1 ... G_m)$, hence, the name convolutional codes [2].

As in many other situations where convolutions appear it is convenient to express the sequences in some sort of transform. In information theory and coding theory [3], [4] it is common to use the delay operator *D*, the *D*-transform. The information and code sequences becomes

$$u(D) = ... + u_{-1}D^{-1} + u_0 + u_1 D + u_2 D^2 + ...$$

and

$$v(D) = ... + v_{-1}D^{-1} + v_0 + v_1 D + v_2 D^2 + ...$$

They are related through the equation

$$v(D) = u(D)G(D),$$

where

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

is the generator matrix.

The set of polynomial matrices is a special case of the rational generator matrices. Hence, instead of having finite impulse response in the encoder, as for the polynomial case, we can allow periodically repeating infinite impulse responses. To make the formal definitions for this case it is easier to start in the D-domain.

Let $F_2((D))$ denote the field of binary Laurent series. The element

$$x(D) = \sum_{i=r}^{\infty} x_i D^i \in F_2((D)), r \in Z,$$

contains at most finitely many negative powers of *D*. similarly, let $F_2[D]$ denote the ring of binary polynomials.

A polynomial

$$x(D) = \sum_{i=0}^{t} x_i D^i \in F_2[D], t \in Z^+,$$

contains no negative powers of *D* and only finitely many positive.

Given a pair of polynomials $x(D), y(D) \in F_2[D]$, where $y(D) \neq 0$, we can obtain the element $x(D)/y(D) \in F_2((D))$ by long division. All non-zero ratios $x(D)/y(D)$ are invertible, so they form the field of binary rational functions, $F_2(D)$, which is a sub-field of $F_2((D))$.

A rate $R = k/n$ (binary) convolutional transducer over the field of rational functions $F_2(D)$ is a linear mapping

$$\tau : F_2^k((D)) \to F_2^n((D))$$
$$u(D) \to v(D)$$

which can be represented as

$$v(D)=u(D)G(D),$$

where $G(D)$ is a k X n transfer function matrix of rank *k* with entries in $F_2(D)$ and $v(D)$ is called the code sequence corresponding to the information sequence $u(D)$.

A rate $R = k/n$ convolutional code ***C*** over F_2 is the image set of a rate $R = k/n$ convolutional transducer. We will only consider realizable (causal) transfer function matrices, which we call generator matrices. A transfer function matrix of a convolutional code is called a generator matrix if it is realizable (causal).

It follows from the definitions that a rate $R = k/n$ convolutional code ***C*** with the k X n generator matrix $G(D)$ is the row space of $G(D)$ over $F((D))$. Hence, it is the set of all code sequences generated by the convolutional generator matrix, $G(D)$.

A rate $R = k/n$ convolutional encoder of a convolutional code with rate $R = k/n$ generator matrix $G(D)$ over $F_2(D)$ is a realization by linear sequential circuits of $G(D)$.

2. Convolutional encoder simulation

The Convolutional Encoder block encodes a sequence of binary input vectors to produce a sequence of binary output vectors. This block can process multiple symbols at a time. If the encoder takes *k* input bit streams (that is, can receive 2^k possible input symbols), then this block's input vector length is $L*k$ for some positive integer *L*. Similarly, if the encoder produces *n* output bit streams (that is, can produce 2^n possible output symbols), then this block's output vector length is $L*n$. The input can be a sample-based vector with $L = 1$, or a frame-based column vector with any positive integer for *L*. For a variable in the MATLAB workspace [5], [6] that contains the trellis structure, we put its name as the **Trellis structure** parameter. This way is preferable because it causes Simulink [5] to spend less time updating the diagram at the beginning of each simulation, compared to the usage in the next bulleted item. For specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, we used a *poly2trellis* command within the **Trellis structure** field. For example, for an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), we have used the **Trellis structure** parameter to *poly2trellis*(*7*,[*171 133*],*171*).

The encoder registers begin in the all-zeros state. We configured the encoder so that it resets its registers to the all-zeros state during the course of the simulation: The value **None** indicates that the encoder never resets; The value **On each frame** indicates that the encoder resets at the beginning of each frame, before processing the next frame of input data; The value **On nonzero Rst input** causes the block to have a second input port, labeled *Rst*. The signal at the *Rst* port is a scalar signal. When it is nonzero, the encoder resets before processing the data at the first input port.

3. Convolutional decoder simulation

3.1. Viterbi Decoder

The Viterbi Decoder block [7], [1] decodes input symbols to produce binary output symbols. This block can process several symbols at a time for faster performance. If the convolutional code uses an alphabet of 2^n possible symbols, then this block's input vector length is $L*n$ for some positive integer L. Similarly, if the decoded data uses an alphabet of 2^k possible output symbols, then this block's output vector length is $L*k$. The integer L is the number of frames that the block processes in each step. The input can be either a sample-based vector with $L = 1$, or a frame-based column vector with any positive integer for L.

The entries of the input vector are either bipolar, binary, or integer data, depending on the **Decision type** parameter: **Unquantized -** Real numbers; **Hard Decision -** *0, 1*; **Soft Decision -** Integers between 0 and 2^k-1, where k is the **Number of soft decision bits** parameter, with 0 for most confident decision for logical zero and 2^k-1, most confident decision for logical one. Other values represent less confident decisions.

If the input signal is frame-based, then the block has three possible methods for transitioning between successive frames. The **Operation mode** parameter controls which method the block uses: In **Continuous** mode, the block saves its internal state metric at the end of each frame, for use with the next frame. Each traceback path is treated independently. In **Truncated** mode, the block treats each frame independently. The traceback path starts at the state with the best metric and always ends in the all-zeros state. This mode is appropriate when the corresponding Convolutional Encoder block has its **Reset** parameter set to **On each frame**. In **Terminated** mode, the block treats each frame independently, and the traceback path always starts and ends in the all-zeros state. This mode is appropriate when the uncoded message signal (that is, the input to the corresponding Convolutional Encoder block) has enough zeros at the end of each frame to fill all memory registers of the encoder. If the encoder has k input streams and constraint length vector *constr* (using the polynomial description), then "enough" means *k**max(*constr-1*). In the special case when the frame-based input signal contains only one symbol, the **Continuous** mode is most appropriate.

The **Traceback depth** parameter, *D*, influences the decoding delay. The decoding delay is the number of zero symbols that precede the first decoded symbol in the output. If the input signal is sample-based, then the decoding delay consists of *D* zero symbols. If the input signal is frame-based and the **Operation mode** parameter is set to **Continuous**, then the decoding delay consists of *D* zero symbols. If the **Operation mode** parameter is set to **Truncated** or **Terminated**, then there is no output delay and the **Traceback depth** parameter must be less than or equal to the number of symbols in each frame. If the code rate is *1/2*, then a typical **Traceback depth** value is about five times the constraint length of the code.

The reset port is usable only when the **Operation mode** parameter is set to **Continuous**. Checking the **Reset input** check box causes the block to have an additional input port, labeled *Rst*. When the *Rst* input is nonzero, the decoder returns to its initial state by configuring its internal memory as follows: Sets the all-zeros state metric to zero; Sets all other state metrics to the maximum value; Sets the traceback memory to zero; Using a reset port on this block is analogous to setting the **Reset** parameter in the Convolutional Encoder block to **On nonzero Rst input**.

3.2. APP Decoder

The APP Decoder block [8] performs a posteriori probability (APP) decoding of a convolutional code. The input $L(u)$ represents the sequence of log-likelihoods of encoder input bits, while the input L(c) represents the sequence of log-likelihoods of code bits. The outputs *L*(u) and *L*(*c*) are updated versions of these sequences, based on information about the encoder. If the convolutional code uses an alphabet of 2^n possible symbols, then this block's *L*(*c*) vectors have length *Q***n* for some positive integer *Q*. Similarly, if the decoded data uses an alphabet of 2^k possible output symbols, then this block's *L*(*u*) vectors have length *Q***k*. The integer *Q* is the number of frames that the block processes in each step.

The inputs can be either: Sample-based vectors having the same dimension and orientation, with $Q = 1$; Frame-based column vectors with any positive integer for *Q*.

To define the convolutional encoder that produced the coded input, we have used the **Trellis structure** MATLAB parameter. We tested two ways: The name as the **Trellis structure** parameter, for a variable in the MATLAB workspace that contains the trellis structure. This way is preferable because it causes Simulink to spend less time updating the diagram at the beginning of each simulation, compared to the usage in the next bulleted item; For specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, we used a *poly2trellis* command within the **Trellis structure** field. For example, for an encoder with a constraint length of 7, code generator polynomials of *171* and *133* (in octal numbers), and a feedback connection of *171* (in octal), we used the **Trellis structure** parameter to *poly2trellis*(*7*,[*171 133*],*171*.

To indicate how the encoder treats the trellis at the beginning and end of each frame, it's necessary to set the **Termination method** parameter to either **Truncated** or **Terminated**. The **Truncated** option indicates that the encoder resets to the all-zeros state at the beginning of each frame, while the **Terminated** option indicates that the encoder forces the trellis to end each frame in the all-zeros state.

We can control part of the decoding algorithm using the **Algorithm** parameter. The **True APP** option implements a posteriori probability. To gain speed, both the **Max*** and **Max** options approximate expressions by other quantities. The **Max** option uses $\max\{a_i\}$ as the approximation, while the **Max*** option uses $\max\{a_i\}$ plus a correction term. The **Max*** option enables the **Scaling bits** parameter in the mask. This parameter is the number of bits by which the block scales the data it processes internally. We have used this parameter to avoid losing precision during the computations. It is especially appropriate for implementation uses fixed-point components.

4. Conclusions

In these work we have constructed and tested in Maple convolutional encoders and decoders of various types, rates, and memories. Convolutional codes are fundamentally different from other classes of codes, in that a continuous sequence of message bits is mapped into a continuous sequence of encoder output bits. It is well-known in the literature and practice that these codes achieve a larger coding gain than that with block coding with the same complexity. The encoder operating at a rate *1*/*n* bits/symbol, may be viewed as a finite-state machine that consists of an M-stage shift register with prescribed connections to *n* modulo-*2* adders, and a multiplexer that serializes the outputs of the adders.

References:

[1] Viterbi, Andrew J. "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes." *IEEE Journal on Selected Areas in Communications*, vol. 16, February 1998. 260-264.

[2] Gitlin, Richard D., Jeremiah F. Hayes, and Stephen B. Weinstein. *Data Communications Principles*. New York: Plenum, 1992.

[3] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.

[4] Pless,V., *Introduction to the Theory of Error-Correcting Codes*, 3rd ed. New York: John Wiley & Sons, 1998.

[5] Matlab Documentation, http://www.math.niu.edu/help/math/matlab/

[6] Matlab Online Reference Documentation, http://www.utexas.edu/math/Matlab/Manual/ReferenceTOC.html

[7] Heller, Jerrold A. and Irwin Mark Jacobs. "Viterbi Decoding for Satellite and Space Communication." *IEEE Transactions on Communication Technology*, vol. COM-19, October 1971. 835-848.

[8] Höst S., Johannesson R., Zyablov V. V., and Skopintsev O. “Generator matrices for binary woven convolutional codes”, Proceedings of 6th Intern. Workshop on Algebraic and Comb. Coding Theory, pages 142_146, Pskov, Russia, Sept. 6_12 1998.