Review of simulations tools for channel coding

V. Galiano, R. Gandia, V. Herranz & C. Perea
Miguel Hernández University of Elche, Spain

Abstract

Channel coding, including convolutional and Turbo coding, represents a new and very powerful error control technique, which started to have a significant impact in the late 1990s, allowing communication very close to the channel capacity. The powerful error correction capability of channel coding was recognized and accepted for almost all types of channels leading to increased data rates and improved Quality of Service. Many standards, included Digital Video Broadband (DVB), Deep Space Network (DSN), Universal Mobile Telecommunications System (UMTS), WiMAX and others, based on channel coding (convolutional and turbo codes) have already been defined although they are currently under investigation. Authors have been analyzing and reviewing various available tools to evaluate new correcting codes. In the majority of scientific publications, the software used in channel coding simulations is rarely referenced and the results are presented without describing how they have been obtained. In this paper, the authors present a review of simulation software for channel coding systems currently available in the scientific community. This analysis is based on criteria such as simplicity, versatility, flexibility and performance. Our objective is to analyse and describe these properties in available software and conclude by choosing a software tool for the development of new convolutional codes in a future work.
Keywords: convolutional code, turbo code, channel coding, simulation, performance, forward correcting codes.

1 Introduction

With the invention of Turbo Codes [1] and the emergence of turbo-like codes in general, channel coding has finally closed the gap to the capacity for an additive white Gaussian noise (AWGN) channel.

The capacity-approaching performance of turbo-like codes has been a major step forward, but there is still a practical need for improvements in terms of versatility, throughput and simplicity. Important parts of modern wireless communication systems, e.g., HSDPA [2] and IEEE 802.16a [3,4], are techniques such as adaptive modulation and automatic repeat-request (ARQ) coding schemes. These techniques require versatile rate-compatible code schemes in terms of both code rate and block length.

A rate-compatible code family [5] is a set of codes that can be encoded/decoded using a common encoder/decoder. The rate-compatibility restriction requires that the rates are organized in a hierarchy, that is, the high-rate codes are embedded into the lower rate codes of the family. This allows transmission of incremental redundancy in ARQ/FEC schemes and rate variation, changing from low to high error protection within a data frame. Puncturing is an effective strategy for designing rate-compatible parallel concatenated convolutional codes (PCCCs), serially concatenated convolutional codes (SCCCs) and low density parity check code structures [6,7] . The code rate flexibility provided by rate-compatible codes is, however, obtained at the expense of performance losses for the higher rate schemes. To compensate for these losses, a more powerful parent code is required, which in turn leads to more complex designs.

Parent codes are in particular convolutional codes, which have been studied from different points of view. In the last decade several authors, such as [8–12], have introduced different constructions of optimal convolutional codes over any finite fields. Climent *et al.* [13] studied the concatenation of convolutional codes over finite fields from linear systems point of view and, more recently, Devesa *et al.* [14], introduced optimal $1/n$ turbo codes over finite fields, from the same point of view. These advances in coding theory motivates the investigation of channel coding simulations over any finite fields. With this purpose the work is structured as follows: In Section 2 we give the basic background theory of convolutional codes which we need through the paper. We describe the main tools actually available according to criteria as simplicity, usability, performance, and extensibility to be evaluated for the simulation of channel coding in Section 3. To display the software programmability seen in Section 3 and analyze their behavior, we have developed the same simulation using three tools: Toolbox of Matlab, CML and GNUradio, which we describe in Sections 4 and 5. We finally the paper with the Section of Conclusions.

2 Notations and definitions

Let $\mathbb{F} = GF(2)$ be the Galois field of two elements, $\mathbb{F}[D]$ the polynomial ring in the variable D with coefficients in $\mathbb{F}$, $\mathbb{F}(D)$ the field of rational functions over $\mathbb{F}$ and $\mathbb{F}((D))$ the field of Laurent series, that is,

$$\mathbb{F}[D] = \left\{ \sum_{j=0}^{L} a_j D^j \mid L \in \mathbb{N}_0, a_j \in \mathbb{F} \right\} \tag{1}$$

and

$$\mathbb{F}((D)) = \left\{ \sum_{j=l}^{\infty} a_j D^j \mid l \in \mathbb{Z}, a_j \in \mathbb{F} \right\} \tag{2}$$

Many different definitions of convolutional codes can be found throughout the literature [15–17]. In what follows, we describe a rate k/n, (with $k < n$), convolutional code over the field $\mathbb{F} = GF(2)$ as a device which generates the n-tuple

$$v_t = (v_t^{(1)} \cdots v_t^{(n)}) \in \mathbb{F}^n \tag{3}$$

of code bits at time t given the k-tuple

$$u_t = (u_t^{(1)} \cdots u_t^{(n)}) \in \mathbb{F}^k \tag{4}$$

of information bits. The mapping between the information sequence $u = (u_t, u_{t+1}, \ldots)$ and the code sequence $u = (v_t, v_{t+1}, \ldots)$ is determined by $v = uG$, where G denotes the generator matrix. The sequences start at some finite time t_0, and the input k-tuples are zero for $t < t_0$. Moreover, sequences of u_t and v_t can be written as

$$u(D) = \sum_{t=t_0}^{\infty} u_t D^t \in \mathbb{F}^k((D)) \tag{5}$$

and

$$v(D) = \sum_{t=t_0}^{\infty} v_t D^t \in \mathbb{F}^n((D)) \tag{6}$$

respectively. Hence, a convolutional encoder can equivalently be described by a $(k \times n)$ generator matrix $G(D)$ of full rank with polynomial or rational entries such that $v(D) = u(D)G(D)$.

In the following, we adopt the notation used by McEliece [18] and we call a convolutional code $\mathcal{C}$ of rate k/n and degree δ as (n, k, δ)-code, where δ is the degree of $\mathcal{C}$ defined as the number $\delta = \sum_{i=1}^{k} \nu_i$, where ν_i denotes the ith row degree of a basic and minimal generator matrix $G(D) \in \mathbb{F}[D]^{k \times n}$ (that is, $G(D)$ has a polynomial right inverse and the sum $\sum_{i=1}^{k} \nu_i$ attains the minimal value among all generator matrices of the convolutional code $\mathcal{C}$).

A generator matrix and its corresponding encoder is called systematic, whenever all bits in ut appear unchanged in v_t. It is important to note that being systematic is a property of the encoder and not a property of the code, since every code can be encoded using systematic as well as nonsystematic encoders. While both of them generate the same set of codewords, the code sequence $v(D)$ associated with a certain information sequence $u(D)$ is usually different. The state of the encoder at time t is denoted by $x_t = (x_t^{(1)}, \ldots, x_t^{(m)})^T$, where m is the number of memory elements of the encoder.

3 Simulating convolutional codes

Our objective is to obtain an adequate software for the simulation of channel coding. Thus, the authors have been investigating the tools actually available according to criteria as simplicity, usability, performance, and extensibility. This section describes the main tools found to be evaluated with examples in later sections.

3.1 Communication toolbox for Matlab

Communications System Toolbox provides algorithms and tools for the design, simulation, and analysis of communications systems. These capabilities are provided as MATLAB functions, MATLAB System objects, and Simulink blocks. The system toolbox includes algorithms for source coding, channel coding, interleaving, modulation, equalization, synchronization, and channel modeling. Tools are provided for bit error rate analysis, generating eye and constellation diagrams, and visualizing channel characteristics. This toolbox also provides adaptive algorithms that let you model dynamic communications systems that use OFDM, OFDMA, and MIMO techniques. Also, it provides block and convolutional coding and decoding techniques to implement error detection and correction, which are the purpose of our study.

3.2 Coded modulation library

CML (Coded Modulation Library) is an open source toolbox for simulating capacity approaching codes in Matlab. It's available for free at the Iterative Solutions website [19]. Mainly, CML simulates several bit interleaved coded modulations as convolutional codes, Turbocodes, and also modulation as PSK (Phase Shift Keying), QAM (Quadrature Amplitude Modulation) or FSK (Frequency Shift Keying). CML supports real standards as binary UMTS/3GPP, LTE, CSMA2000 or CCSDS, and turbocodes as DVB-RCS or WIMAX IEEE 802.16. CML is mainly a toolbox for prototyping in Matlab but also can be executed in a compiled mode which can be used independently of Matlab. In this way, CML is composed by C-mex files compiled for Windows PC and C sources files for Linux and Mac.

Two commands are mainly used in CML: `CmlSimulate` and `CmlPlot`. The first one runs one or more simulations previously configured in Matlab scripts. We must emphasize that the simulation with CML is mainly based on the proper configuration of the parameters in the script. After the simulation, we will execute `CmlPlot` to plot one or more simulations. Simulations are identified by scenarios. The main content of the scenario file is a structure called `sim_param` which contains an array where each record corresponds to a single distinct simulation. An important aspect that is relevant to achieving our goal is that CML is distributed with source code and can be easily extended and modified.

3.3 GNUradio

GNU Radio [20] is a free software development toolkit that provides the signal processing runtime and processing blocks to implement software radios using readily-available, low-cost external RF hardware and commodity processors. It is widely used in hobbyist, academic and commercial environments to support wireless communications research as well as to implement real-world radio systems. GNU Radio applications are primarily written using the Python programming language, while the supplied, performance-critical signal processing path is implemented in C++ using processor floating point extensions where available. GNU Radio is designed primarily to run on Linux platforms and is distributed with source code so that it can be extended and modified by the scientific community.

3.4 Self-made codes

In a high volume of publications, there is no reference to the simulation software used for error correction simulations and in many others authors implement their own code with programming languages as C. However, source code or any reference is not distributed.

4 Evaluating software

To display the software programmability seen in Section 3 and analyze their behavior, we have developed the same simulation using three tools: Toolbox of Matlab, CML and gnuradio. We should remember that an important aspect of the tool is its extensibility since our ultimate goal is to provide novel simulation tools for nonbinary encoders. We based code generator proposed by Proakis [21] ($G_1 = [133, 171]$) with rate $k = 1/2$ and constraint Length $= 7$. This convolutional code has been widely used in first forward error correcting codes system and so it's widely known by scientific community. Actually in real applications convolutional codes has been replaced by turbocodes [22]. However, as turbo codes are an evolution of convolutional codes, we consider more appropriate an analysis with convolutional codes first. In all simulations, we have used a bit frame with 10^6 elements. We must note that we get a memory error when using Matlab with 10^7 elements and it works right when simulating with CML.

In Figure 1, we show the script which uses Matlab's Toolbox. As we can see, we may also use the Matlab debug mode that allows more comfortable refine the prototype encoder. In lines 1 to 4, we specify the parameters of the simulation, in line 5 we obtain the maximum expected theoretical BER (shown in Figure 4 as *Expected BER*). At line 7 convolutional coding is performed, and after that, noise is added in the transmission channel. Finally, in line 7, the viterbi decoder algorithm is performed.

In the CML software, simulations are structured in a `sim_param` configuration array. After executing with the `CmlSimulate` command, the output in the array

```
codeRate = 1/2;constlen = 7;
codegen = [171 133];
trellis = poly2trellis(constlen, codegen);
dspec = distspec(trellis, 7)
expVitBER = bercoding(EbNo, 'conv', 'hard', codeRate, dspec);
...
msg_enc = convenc(msg_orig, trellis);
...
msg_dec = vitdec(msg_demod, trellis, tblen, 'cont', 'hard');
```

Figure 1: Example of convolutional coding using Matlab Toolbox.

```
 1  record = 1;
 2  sim_param(record).sim_type = 'coded';
 3  sim_param(record).SNR = [0:0.5:8];
 4  sim_param(record).SNR_type = 'Eb/No in dB';
 5  sim_param(record).framesize = 10^6;
 6  sim_param(record).modulation = 'BPSK';
 7  sim_param(record).mod_order = 2;
 8  sim_param(record).mapping = [];
 9  sim_param(record).channel = 'AWGN';
10  sim_param(record).bicm = 1;
11  sim_param(record).demod_type = 0;
12  sim_param(record).linetype = 'k:';
13  sim_param(record).legend = sim_param(record).comment;
14  sim_param(record).g1 = [1 0 1 1 0 1 1
15     1 1 1 1 0 0 1];
16  ...
```

Figure 2: Example of convolutional coding using CML.

sim_state, which is used to plot the performance with CmlPlot. As we can see in the code in Figure 2, line 2 will indicate the type of simulation, and define the range and type of data on lines 2 and 3. In line 14 indicated by the parameter g1, we will use the generator polynomial in the convolution. These simulations are very coupled to the structure defined thus be considered a very rigid tool. However, it is distributed with source so that it can be modified or extended. This makes it an interesting tool that uses a tool as widespread as Matlab but extending its functionality directly from your code.

Finally, we show an example using the library GNURadio. We note that this code is in Python, however Python is an interpreted language very intuitive and similar in many aspects to Matlab so a Matlab user should not have too many problems in its use. As shown in the code shown in Figure 3, the simulation is more laborious than in CML or in the Toolbox of Matlab. Gnuradio is structured in a flow chart in which processes and blocks define data sources and connect between them. Thus, from line 2 to 11 operational blocks are defined which will be subsequently connected to a flowchart indicated in commands from line 13 to 17.

```
 1  ...
 2  f=trellis.fsm(1,2,[121,91])
 3  src = gr.vector_source_s(packet,False)
 4  ...
 5  enc = trellis.encoder_ss(f,0) # initial state = 0
 6  mod = gr.chunks_to_symbols_sf(constellation,dimensionality)
 7  ...
 8  noise = gr.noise_source_f(gr.GR_GAUSSIAN,math.sqrt(N0/2),seed)
 9  ...
10  va = trellis.viterbi_s(f,K,0,-1)
11  dst = gr.vector_sink_s();
12  ...
13  tb.connect (src,b2s,s2fsmi,enc,mod)
14  tb.connect (mod,(add,0))
15  tb.connect (noise,(add,1))
16  tb.connect (add,metrics)
17  tb.connect (metrics,va,fsmi2s,s2b,dst)
18  tb.run()
```

Figure 3: Example of convolutional coding using GNURadio.

Finally, execution is performed by calling `tb.run()` in line 18. In line 2, we define the generator polynomial as a decimal but in Matlab is defined as an octal number. The authors have observed that the programming by gnuradio is certainly more complex but more concise in all processes of communication. GNURadio is based in Python packages and includes a set of libraries compiled in C for performance improvement. However, GNURadio is an open source tool that can also be expanded and modified to incorporate new features.

5 Performance evaluation

Figure 4 shows the results of the simulations with the three tools: Matlab, CML and GNURadio. We also represent the maximum theoretical obtained in the Matlab simulation. As expected, we see in the graph very similar results in all simulations, however the calculation time of these simulations are not so similar and we detected variations that should be analyzed in greater detail. So in Figure 5 we illustrate computing times for simulating the convolutional code described in Section 4. We can note that computational times with the Matlab's Toolbox are lower than others, and CML has better times than GNUradio. However, Matlab Toolbox is not able to execute bigger sizes because we have found memory errors, maybe due to memory management inside the toolbox. Moreover CML is distributed as open source and can be modified and be extended for our purpose. In the other hand, GNURadio requires more time in simulation but we consider it a complete and well structured software that can solve certain issues that can not be reached with others.

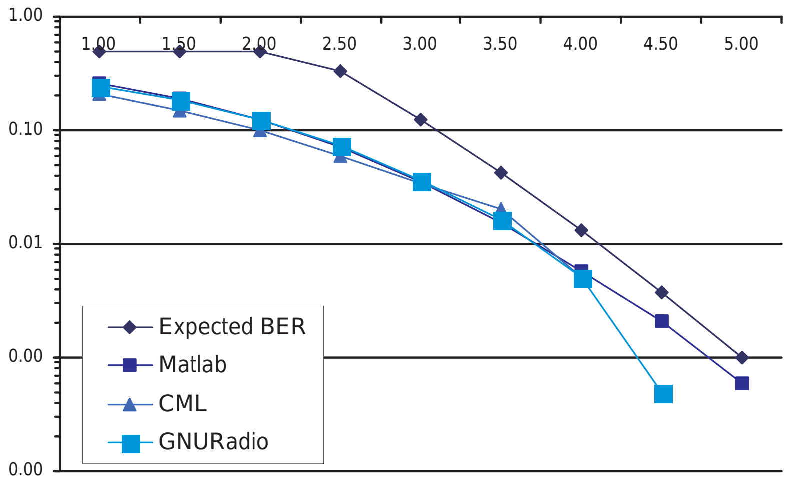

Figure 4: Performance in BER using Matlab, CML and GNURadio.

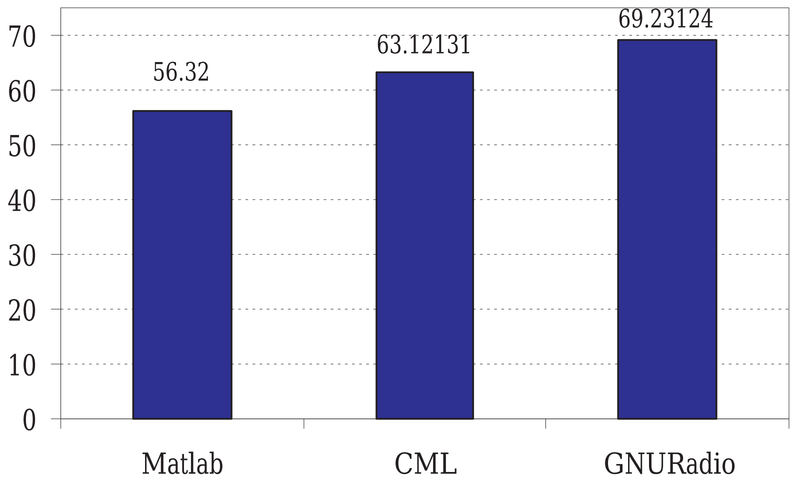

Figure 5: Computing time needed in simulations using Matlab, CML and GNURadio.

6 Conclusions

Due to extensive use of Matlab as prototyping language in the scientific community, we consider the right tool for wider dissemination and validation by other research teams. Notwithstanding the Matlab toolbox libraries are not easily extended, so we consider more interesting the utilization of CML, which is distributed with its source code and it will allow us to implement new convolutional encoders for any field not necessarily binary. In fact, we have now found that any tool allowing non binary fields in convolutional coding simulations. GNURadio also seems a powerful and well structured tool for the simulation, however has greater difficulty in handling its programmability and grater computing times. Our future work is focused to develop new simulation tools for testing new convolutional codes with non binary fields using the selected tool.

Acknowledgements

The work of these authors was partially supported by Spanish grant MTM2011-24858 of the Ministerio de Economía y Competitividad of the Gobierno de España and supported by IPZS01 Proyectos de Investigación UMH-Bancaja.

References

[1] Berrou, C. and Glavieux, A. Near optimum error correcting coding and decoding: Turbo-codes,*IEEE Transactions on Communications*, **44(10)**, pp. 1261–1271, 1996.

[2] Parkvall, S., Englund, E. , Lundevall, M. and Torsner, J. Evolving 3G mobile systems: Broadband and broadcast services in WCDMA, *IEEE Communications Magazine*, vol. **44**, pp. 68-74, 2006.

[3] Ghosh, A., Wolter, D.R., Andrews, J.G., and Chen, R. Broadband wireless access with WiMax/802.16: Current performance benchmarks and future potential, *IEEE Communications Magazine*, **43**, pp. 129-136, 2005.

[4] IEEE standard for local and metropolitan area networks, IEEE 802.16a, 2003.

[5] J. Hagenauer. Rate-compatible punctured convolutional codes (RCPC codes) and their applications.IEEE Transactions on Communications, **36**, pp. 389-400, 1998.

[6] Barbulescu, A.S. and Pietrobon, S.S. Rate compatible turbo codes, *IEE Electron. Lett.*, **31(7)**, pp. 535–536, 1995.

[7] Ha, J., Kim, J., and McLaughlin, S.W. Rate-compatible puncturing of low-density parity-check codes, *IEEE Transactions on Information Theory*, vol. **50**, pp. 2824–2836, 2004.

[8] Rosenthal, J; York, E. BCH convolutional codes. *IEEE Transactions on Information Theory,* **45(6)**, pp.1833-1844, 1999.

[9] Rosenthal, J; Smarandache, R. Maximum distance separable convolutional codes. *Applicable Algebra in Engineering, Communication and Computing*, **10(1)**, pp.15-32, 1999.

[10] Smarandache, R.; Gluesing-Luerssen, H.; Rosenthal, J. Constructions of MDS-convolutional codes. *IEEE Transactions on Information Theory*, **47(5)**, pp. 2045-2049, 2001.

[11] Hutchinson, R; Rosenthal, J; Smarandache, R. Convolutional codes with maximum distance profile. *Systems and Control Letters*, **54(1)**, pp. 53-63, 2005.

[12] Gluesing-Luerssen, H.; Rosenthal, J.; Smarandache, R. Strongly-MDS convolutional codes. *IEEE Transactions on Information Theory*, **52(2)**, pp. 584-598, 2006.

[13] Climent J.J., Herranz V., Perea C. A first approximation of concatenated convolutional codes from linear systems theory viewpoint. *Linear Algebra and its Applications*, **425**, pp. 673-699, 2007.

[14] Devesa A., Herranz V., Perea C. 1/n Turbo Codes with Maximal Effective Distance over any Finite Fields from Linear System Point of View. *Proceedings of the International Conference on Computational and Mathematical Methods in Science and Engineering*. La Manga, Spain, 2012.

[15] Rosenthal, J. Connections between linear systems and convolutional codes,*In Marcus, B; Rosenthal, J. Codes, systems and graphical models (Minneapolis, MN 1999).* New York, pp. 39-66. ISBN 0-387-95173-3

[16] Johannesson, R., and Zigangirov, K.S. *Fundamentals of Convolutional Coding.* New York: IEEE Press, 1999.

[17] Dholakia, A. *Introduction to Convolutional Codes with Applications.* Norwell, MA: Kluwer, 1994.

[18] McEliece, R.J. *The algebraic thery of convolutional codes.* In V. Pless and W.C. Huffman, editors, Handbook of Coding Theory, vol. I, pp. 1065-1138. Elsevier, Amsterdam, 1998

[19] The Coded Modulation Library. Iterative Solutions. http://www.iterativesolutions.com/

[20] Eric Blossom. 2004. *GNU radio: tools for exploring the radio frequency spectrum.* Linux J. 2004, 122 (June 2004).

[21] J.G. Proakis, *Digital Communications*, McGraw-Hill, NewYork, Aug. 2000.

[22] Schlegel, Christian, and Lance Perez. Trellis and turbo coding, Vol. 7. Wiley-IEEE Press, 2004.