

## ***Welcome to the Iterative Solutions Coded Modulation Library (ISCML).***

Documentation last updated on June 25, 2006.

### Contents:

- (1) Licensing information
- (2) List of currently supported features.
- (3) A quick start guide
- (4) The `sim_param` structure format.
- (5) The `sim_state` structure format.
- (6) Compiling mex files.
- (7) Features that will be supported in the (near) future.
- (8) Revision History

### (1) Licensing information

The Iterative Solutions Coded Modulation Library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

A copy of the GNU Lesser General Public License is found in the `./cml/documentation` directory as the `license.txt` file.

### (2) Currently supported features.

- Simulation of any complex (i.e. two dimensional) modulation, including BPSK, QPSK, M-ary PSK, QAM, HEX, and APSK.
- Simulation of FSK modulation, both orthogonal (integer  $h$ ) and nonorthogonal, and with coherent and noncoherent demodulation.
- Monte Carlo calculation of the Shannon capacity of any complex M-ary modulation or FSK modulation. For any modulation, either the Coded Modulation (CM) capacity or the Bit Interleaved Coded Modulation (BICM) capacity can be found. Capacity can be found for AWGN or "fully-interleaved" (ergodic) Rayleigh fading.
- Monte Carlo calculation of the information outage probability of either unconstrained (Gaussian input) or constellation constrained modulation over a Rayleigh block fading channel.
- Simulation of coded M-ary modulation, using convolutional codes (NSC or RSC), turbo codes (PCCC), cyclic block codes, or LDPC codes (conforming to the DVB-S2 standard).
- Generation of S-random interleavers.
- Support for the cdma2000, UMTS, and CCSDS turbo codes.
- Support for HSDPA.
- Iterative demodulation and decoding, also known as BICM with Iterative Decoding (BICM-ID).
- Simulations run until either a specified number of errors are logged or a specified number of trials are run. When a simulation is halted and then rerun, it will pick up where it left off (unless a flag is set telling it to restart the simulation).
- Generation of throughput results for hybrid-ARQ systems, including HSDPA.

- Plotting of the results through a simple interface that only requires the entry of which simulations to plot by entering the indices of their records in the scenario file. The following types of plots are generated:
  - Shannon capacity vs/  $E_s/N_0$  and  $E_b/N_0$ .
  - Bit error rate (BER) vs.  $E_s/N_0$  and  $E_b/N_0$  for uncoded and coded modulation.
  - Symbol error rate (SER) vs.  $E_s/N_0$  and  $E_b/N_0$  for uncoded modulation.
  - Frame error rate (FER) vs.  $E_s/N_0$  and  $E_b/N_0$  for coded modulation and outage probability.
  - Normalized throughput vs.  $E_s/N_0$
- Ability to run on a cluster of computers running the *Frontier* compute engine.
- Ability to run simulations locally as a stand-alone DOS executable.

### (3) Quick start guide

- Starting with version 1.6.0, there are three files compressed using the *rar* utility. A client for decompressing these files can be found at [www.rarlabs.com](http://www.rarlabs.com)
  - First extract the `cml.X.X.X.rar` file, making sure that the directory structure is preserved. There will be a main directory `./cml` with several subdirectories under it. This directory will contain the source code and functions required to run the simulations, but will not contain any simulated data.
  - If you would like access to the library of simulated data (output of many hours of simulation runs), then extract the `output.X.X.X.rar` file. This will create a directory `./output`, which should be placed into the `./cml` directory.
  - If you wish to run on the grid (using Frontier) or as a stand-alone executable, extract the `grid.X.X.X.rar` file and place the `./grid` directory into the `./cml` directory.
- Start matlab (version 6.5 or higher) and `cd` to the `./cml` directory.
- From within matlab, run:
 

```
CmlStartup
```

This updates the matlab path, and writes the "CmlHome.mat" file into the scenarios directory. You need to type this in everytime you start matlab (after `cd`'ing to this directory).
- Type:
 

```
edit UncodedScenarios
```

This file contains an array of `sim_param` structures. Each record in this array corresponds to a single simulation. Note the record number of a few scenarios that you would like to see plotted, for instance scenarios 1, 3, 5, and 11
- To plot scenarios 1, 3, 5, and 11 type (assuming you have downloaded the `output.X.X.X.rar` file and placed the `./output` directory into the `./cml` directory):
 

```
[sim_param, sim_state] = CmlPlot('UncodedScenarios', [1 3 5 11]);
```
- To run a set of scenarios, for instance 5 and 10, type:
 

```
[sim_param, sim_state] = CmlSimulate('UncodedScenarios', [5 10]);
```
- In order to run new simulations, you may add new elements to the `sim_param` array, and then run `CmlSimulate` (if you run `CmlPlot` before first generating a simulation result using `CmlSimulate`, then nothing will appear in your plot). You can create your own Scenarios files, containing descriptions of your own simulations. A description of each member of the `sim_param` structure is in the next section.

### (4) The `sim_param` structure format.

The parameters associated with the scenario to be simulated are stored in the `sim_param` structure. Both the `CmlPlot` and `CmlSimulate` functions begin by calling the function `ReadScenario`, which reads the `sim_param` values stored in the scenario file.

Each `sim_param` structure contains the following members, some of which may be empty (i.e. set to `[]`) if they are not relevant to the particular scenario:

- `sim_type`: The type of simulation, can be:  
     'uncoded' for uncoded modulation,  
     'coded' for coded modulation,  
     'capacity' for Shannon capacity,  
     'outage' to calculate outage probability in block fading,  
     'throughput' to plot the throughput of hybrid-ARQ.  
     'bwcapacity' to determine the min Eb/No as a function of modulation index h under a bandwidth constraint  
     'minSNRvsB'  
     'exit' to generate an EXIT curve (not yet supported).
- `SNR_type`: Set to either 'Eb/No in dB' or 'Es/No in dB'. Use Es/No for capacity simulations, while coded and uncoded modulation simulations may use either.
- `framesize`: The frame size. If uncoded, capacity, or exit, this is the number of symbols per frame. If coded, this is the number of information bits per frame (except for LDPC codes, where this is the number of code bits).
- `bicm`: Flag for Bit Interleaved Coded Modulation  
     = 0 for no BICM, used primarily for calculating the CM capacity.  
     = 1 for BICM but noniterative demodulation.  
     = 2 for BICM-ID, i.e. iterative demodulation and decoding.
- `modulation`: The type of modulation. Must be one of the following (case insensitive):  
     'BPSK', 'QPSK', 'PSK', 'QAM', 'APSK', 'HEX', or 'HSDPA'  
     Other modulations can be supported by modifying the `Mod2D` function.
- `mod_order`: The modulation order, i.e. number of symbols in the signal set. Must be a power of 2, except that values of 0 and -1 are allowed in "outage" simulations to represent 2-D unconstrained modulation (=0) or 1-D unconstrained modulation (=1).
- `mapping`: How the bits are mapped to symbols. Must be one of the following:  
     'gray', 'Antigray', 'quasigray', 'SP', 'SSP', 'MSEW', 'huangITNr1', 'huangITNr2', 'huangLetterNr1', or 'huangLetterNr2'.
- `h`: The modulation index (for FSK modulation only).
- `demod_type`: A flag indicating how the `max_star` function is implemented within the demodulator:  
     = 0 For linear-log-MAP algorithm, i.e. correction function is a straight line.  
     = 1 For max-log-MAP algorithm (i.e.  $\max^*(x,y) = \max(x,y)$  ), i.e. correction function = 0.  
     = 2 For Constant-log-MAP algorithm, i.e. correction function is a constant.  
     = 3 For log-MAP, correction factor from small nonuniform table and interpolation.  
     = 4 For log-MAP, correction factor uses C function calls.
- `code_configuration`: The code configuration (if any)  
     = 0 for a single convolutional code  
     = 1 for a turbo code (parallel concatenated convolutional code)  
     = 2 for a LDPC code  
     = 3 for HSDPA
- `g1`: The generator for the first convolutional encoder

- `nsc_flag1`: A flag indicating if the first encoder is NSC or RSC  
     = 0 for recursive systematic convolutional (RSC); required if turbo.  
     = 1 for nonsystematic convolutional (NSC)
- `pun_pattern1`: The puncturing pattern for the second encoder.
- `tail_pattern1`: The puncturing pattern for the second encoder's tail.
- `g2`: The generator for the second convolutional encoder
- `nsc_flag2`: A flag indicating if the second encoder is NSC or RSC  
     = 0 for recursive systematic convolutional (RSC); required if turbo.  
     = 1 for nonsystematic convolutional (NSC)
- `pun_pattern2`: The puncturing pattern for the second encoder.
- `tail_pattern2`: The puncturing pattern for the second encoder's tail.
- `decoder_type`: A flag indicating how the `max_star` function is implemented within the decoder:  
     = 0 For linear-log-MAP algorithm, i.e. correction function is a straight line.  
     = 1 For max-log-MAP algorithm (i.e.  $\max(x,y) = \max(x,y)$  ), i.e. correction function = 0.  
     = 2 For Constant-log-MAP algorithm, i.e. correction function is a constant.  
     = 3 For log-MAP, correction factor from small nonuniform table and interpolation.  
     = 4 For log-MAP, correction factor uses C function calls.
- `max_iterations`: The maximum number of demodulator or decoder iterations. Assumed to be  
     =1 if noniterative coding and modulation is simulated.
- `code_interleaver`: The function that must be invoked to generate the code interleaver for a  
     turbo code. Note that this is indirect, as storing the actual interleaver takes up too much space. A  
     saved interleaver could be called by setting this to 'load interleaver\_name.mat' where  
     'interleaver\_name.mat' is the filename containing the interleaver coefficients. Note that a length K  
     interleaver should contain the coefficients 0 through K-1 (numbering uses the C convention of  
     starting at 0, not the matlab convention of starting with 1).
- `parity_check_matrix`: The function that must be invoked to generate the parity check matrix  
     for an LDPC code. Note that this is indirect, as storing the actual parity check matrix takes up too  
     much space.
- `channel`: The channel type. Can be either "AWGN", "Rayleigh", or "block".
- `blocks_per_frame`: Used by "outage" simulations or in block fading; number of fading blocks  
     per codeword.
- `N_IR`: Size of the virtual IR buffer (used by HSDPA only).
- `P`: Number of physical channels (used by HSDPA only).
- `X_set`: Sequence of redundancy versions (used by HSDPA only).
- `combining_type`: Used by "outage" simulations; may be 'code' or 'diversity'
- `rate`: The spectral efficiency, defined to be number of data bits divided by number of symbols per  
     frame (this parameter is calculated with the `ReadScenario` function).
- `csi_flag`: Flag used by the FSK demodulator  
     = 0 for coherent reception (default)  
     1 for noncoherent reception w/ perfect amplitude estimates  
     2 for noncoherent reception without amplitude estimates
- `bwconstraint`: Bandwidth constraint. Used only by the 'bwcapacity' simulation type.
- `bwdatabase`: Location of the bandwidth database file. Used only by the 'bwcapacity' simulation  
     type.
- `SNR`: A vector containing the SNR points to test. New points can be added or deleted prior to  
     resuming a simulation.
- `filename`: Name of the file storing the state.
- `comment`: A text comment describing the scenario.

- `legend`: What gets printed in the figure legened.
- `linetype`: What type of line to use when plotting.
- `plot_iterations`: A vector specifying which iterations to plot.
- `save_rate`: The simulation state is saved every "save\_rate" frames or once all frames have been simulated for a particular SNR point (if not defined, default is 100).
- `reset`: Reset flag.  
= 0 to resume,  
= 1 to restart,
- `max_trials`: Vector containing maximum number of frames to simulate at each SNR point. Simulation at that SNR point halts once this number of frames has been simulated. This number can be incremented prior to resuming the simulation.
- `minBER`: Scalar value representing the targeted minimum BER for the simulation. If the simulation is noniterative, it halts once it is reached (no more SNR points considered); For simulations of iterative reception (turbo codes, BICM-ID), the next SNR point will be run, but with fewer iterations (only as many as needed to reach minBER at the last SNR point).
- `minFER`: Used by "outage" simulations; lowest FER to simulate (halts once reached).
- `max_frame_errors`: Vector containing maximum number of frame errors at each SNR point. Simulation at that SNR point halts once this number of frame errors has been reached. This number can be incremented prior to resuming the simulation, in order to add more confidence to the simulation (which should smoth out the corresponding curve).
- `compiled_mode`: When set to 1, uses the compiled version of SingleSimulate.
- `input_filename`: Used by "outage" simulations with constellation constrained modulation; name of file containing AWGN capacity results.
- `trial_size`: Used by "outage" simulations; is the number of codewords per simulation trial.
- `scenarios`: Used by "throughput" type; is a list of simulation records used to compute the throughput.

#### (5) The `sim_state` structure format.

The state/result of a simulation are stored in the `sim_state` structure. Let "SNRpoints" be the number of SNRpoints simulated (length of `sim_param.SNR` vector), let "iter" be the maximum number of iterations (`max_iterations`), and let "MIpoints" be the number of input mutual information points simulated to create an EXIT curve (for a capacity simulation this is just one, namely input MI = zero).

The `sim_state` structure has the following members:

- `trials`: iter by SNRpoints matrix containing the number of frames simulated.
- `capacity_sum`: MIpoints by SNRpoints matrix containing the sum of capacities of all simulated symbols.
- `capacity_avg`: MIpoints by SNRpoints matrix containing the average capacity of the simulated symbols.
- `frame_errors`: iter by SNRpoints matrix containing the number of frame errors logged. Only updated for "coded" and "outage" simulation types.
- `symbol_errors`: Length SNRpoints vector containing the number of symbol errors logged. Only updated for uncoded modulation.
- `bit_errors`: iter by SNRpoints matrix containing the number of bit errors logged.
- `FER`: iter by SNRpoints matrix containing the frame error rate. Only updated for coded modulation.

- SER: Length SNRpoints vector containing the symbol error rate. Only updated for uncoded modulation.
- BER: iter by SNRpoints matrix containing the bit error rate.
- throughput: The throughput of hybrid-ARQ, only used by “throughput” sim\_type.
- min\_rate: The minimum rate that achieves the bandwidth constraint
- min\_EsNodB: The minimum EsNo (in dB)
- min\_EbNodB: The minimum EbNo (in dB)

#### (6) Compiling mex files.

The mex source files in the ./cml/source directory have been compiled for use on windows platforms. The compiled .dll files are in the ./cml/mex directory. If you are using a different platform, you will need to compile the source files and make sure they get placed in the ./cml/mex directory. This can be done by executing the following statement for each .c file in the ./cml/source directory:

```
>> mex -outdir ../mex <filename.c>
```

If this is the first time that you have run mex on this system, you may need to set it up by running:

```
>> mex -setup
```

A pair of makefiles is in the ./cml/source directory to automate the compilation of all mex files. Use make for matlab 6.5, 7.0, or 7.1, and use make72 for matlab 7.2.

#### (7) Features that will be supported in the (near) future:

- Ability to generate EXIT curves.
- Support for serially concatenated convolutional codes (SCCCs).
- Support for fully asymmetric PCCC codes (currently both constituent encoders need to have the same constraint length).
- Support for the DVB-RCS and WiMax duobinary CRSC turbo codes.
- Support for space-time modulation over a MIMO channel.
- Sliding window SISO decoder.
- A Graphical User Interface (GUI) for managing the Scenario File.
- Ability to request additional iterations to be simulated when resuming a simulation of an iteratively decoded system.

#### (8) Revision History

Version 1.0.0: Oct. 2, 2005

- Initial Release

Version 1.0.1: Oct. 17, 2005

- The statment "if (errors(max\_iterations))" on line 245 causes capacity simulations to exit with an error stating that variable "errors" is not defined. Fixed.
- The files containing results for QPSK capacity in AWGN (QPSK.mat) and Rayleigh fading (QPSKRayleigh.mat) were corrupt. These simulations were rerun.
- "CmlSimulate" has been updated to allow BICM capacity simulations for BPSK modulation (although it will give the same result as the CM capacity simulation for BPSK).

- "CmlPlot" also plots the frame error rate of "coded" simulations and the symbol error rate for "uncoded" simulations.
- The Quick Start guide in the readme.txt file has been updated to make it clear that you must first run "CmlSimulate" to generate a curve, then run "CmlPlot" to view it.

Version 1.1.0: Nov. 11, 2005

- "CmlPlot", "CmlSimulate", and "ReadScenario" can now read from multiple scenario files. The new calling syntax is  

```
[sim_param, sim_state] = ReadScenario( scenario_filename1, cases1 ,
scenario_filename2, cases2, ... )
```

and there is no limit on the number of scenario files (it can be as few as one up to however many you want).
- The "CmlScenarios" file no longer exists. It has been broken into several scenario files, namely "CapacityScenarios", "Cdma2000Scenarios", "ConvolutionalScenarios", "DVBS2Scenarios", and "UncodedScenarios"
- A new set of scenarios for the UMTS turbo code, called "UmtsScenarios" has been included. Results for all 48 simulations listed in UmtsScenarios are included in the ./cml/output/UMTS directory.
- **LDPC Codes:** A more general class of LDPC codes can now be handled, instead of just the DVB-S2 codes. The "EncodeDVBS2" function no longer exists and has been replaced with "LdpcEncode". "InitializeDVBS2" now returns the H\_1 matrix (in H\_rows, H\_cols sparse format) rather than the full H matrix. The "MpDecode" function can be given either the full H matrix or just the H\_1 matrix.
- **Save Rate:** A new parameter is added called "sim\_param.save\_rate", which determines how often the simulation is saved. Simulations are now saved once every "sim\_param.save\_rate" frames or once the final frame for a particular SNR has been simulated. The simulation is no longer saved when an error occurs, as this tends to bias the simulation when it is halted.

Version 1.2.0: Nov. 29, 2005

- **HSDPA:** Support for HSDPA, which is implemented through function pairs "HarqMatch/HarqDematch", "RateMatch/RateDematch", and "BitCollection/BitDecollection". A HSDPA simulation is specified by setting "sim\_param.code\_configuration = 3". HSDPA simulations require that "sim\_param.N\_IR", "sim\_param.X\_set", and "sim\_param.P" be defined. Also, "Mod2D" has been updated to include the two HSDPA modulations.
- Simulation results for HSDPA have been generated. The scenario file is "HsdpaScenarios.m", located in the ./cml/scenarios subdirectory, and the results are stored in ./cml/output/HSDPA.
- **Demos:** A directory devoted to demos (./cml/demos) has been created. This directory contains a demo for HSDPA (HsdpaDemo) and also a script that plots HsdpaThroughput curves (HsdpaThroughput). "CmlStartup.m" has been updated so that ./cml/demos will be on the path.
- "CmlSimulate" no longer contains the main simulation logic. All it does now is determine what type of simulation is being run and then calls one of the following new functions: "CapacitySimulate", "UncodedSimulate", and "CodedSimulate".
- Likewise, the main logic in "ReadScenario" has been divided into new functions "CapacityRead", "UncodedRead", and "CodedRead".
- **Stand-alone Turbo Encoder and Decoder:** The turbo encoding and decoding logic has been stripped out of "CmlEncode" and "CmlDecode" and made into new functions "TurboEncode" and "TurboDecode".
- "CodedRead" calculates the following values: rate, symbols\_per\_frame, and code\_bits\_per\_frame, and saves them in the sim\_param structure. These values are used within "CmlPlot" to covert

between Es/No (dB) and Eb/No (dB). Now the BER, SER, and FER of coded systems are plotted against not only Eb/No (dB) but also Es/No (dB).

- **UMTS Puncturing:** "Puncture" and "Depuncture" have been modified so that the tail comes out according to the ordering in the UMTS standard.
- **Help on c-mex Functions:** Information about each mex function can now be obtained by typing "help <function\_name>" at the matlab prompt. This was achieved by creating a .m file for each mex file, where the .m file only contains the help information. These .m files are placed in new directory "./cml/mexhelp".

Version 1.2.1: Dec. 5, 2005

- "RateMatch" and "RateDematch" are now implemented as c-mex functions.
- A divide-by-zero problem in "HsdpaThroughput" has been fixed, so now the throughput results are accurate even at low SNR

Version 1.2.2: Dec. 13, 2005.

- **HSDPA H-SET 6:** Support for HSDPA H-SET 6, which encodes two turbo words in parallel and send them simultaneously over 2\*sim\_param.P physical channels. In order to provide this support, "TurboEncode" can now accept a data vector whose length is an integer multiple of the interleaver length. If  $\text{length}(\text{data}) = B * \text{length}(\text{code\_interleaver})$ , then "TurboEncode" will return a matrix with B rows, one for each codeword. Likewise, "TurboDecode" can accept an input (input\_decoder\_c) comprised of B rows, in which case it will decode each row as an independent codeword (although the error count is a single vector that indicates the sum of errors across all codewords). Also, "HarqMatch" has been modified to recognize when the data must be segmented into multiple turbocodewords (whenever the data length after CRC encoding is greater than 5114), and the complementary operations have been updated in "HarqDematch".
- Simulation results for H-SET 6 in AWGN and fully-interleaved (ergodic) Rayleigh fading have been run. See the "HsdpaScenarios" file (these are records 17 through 32).

Version 1.3: Dec. 26, 2005

- **Outage Probability:** Can calculate the information outage probability in Rayleigh block fading using Monte Carlo integration. This required the specification of a new sim\_type called "outage". In block fading, each block undergoes independent fading, but the channel is AWGN for the duration of the block. So to compute the instantaneous capacity, the capacity of each block is found (for code combining) or the SNRs are added and the corresponding capacity computed (for diversity combining). Capacity is computed for an unconstrained (Gaussian input) using the equation  $\text{capacity} = \log_2(1 + \gamma)$ , where  $\gamma$  is the instantaneous SNR. Whenever the capacity drops below the rate, an outage is logged. Outage simulations are handled by the function OutageSimulate, and the simulation is set up by the OutageRead function. Constellation constrained modulation requires that an AWGN capacity simulation be already run, and capacity values are read from the results using the function CapacityTableLookup (implemented as both .m and c-mex).
- **Rayleigh Block Fading Channel for Coded Modulation:** A new channel type called "block" has been defined for "coded" simulations. This is Rayleigh block fading, and the number of blocks per codeword is represented by sim\_param.blocks\_per\_frame.
- **Throughput of Hybrid-ARQ:** Throughput curves can now be drawn by the CmlPlot function. This requires a new sim\_type called "throughput", although it is not actually a simulation. Instead, it uses results from "coded" simulations, where each simulation corresponds to a fixed number of block transmissions. A new variable sim\_state.throughput is used to represent the throughput.



- **HsdpaThroughput obsolete:** The HsdpaThroughput function has been deleted, the same results can be obtained by running:  

```
[sim_param, sim_state] = CmlPlot('ThroughputScenarios', 1:10 );
```
- Setting save\_rate to the default value (100) has been moved from the simulation functions (CodedSimulate, UncodedSimulate, CapacitySimulate) to the read functions (CodedRead, UncodedRead, CapacityRead).
- CodedRead no longer calculates code\_bits\_per\_frame, data\_bits\_per\_frame, and symbols\_per\_frame, since these values are calculated by CodedRead and are now members of the sim\_param structure.

Version 1.3.1: Jan. 11, 2006

- **ConvEncode:** Fixed bug at the end of ConvEncode: “free(rsc\_tail)” changed to “free(tail)”.
- **turbo.h:** Broke turbo.h into two header files: convolutional.h and siso.h. This way ConvEncode does not need to include the decoding functions in siso.h.
- **Comments:** All comments with the C++ syntax ( // and comment until end of line) were changed to the C syntax ( /\* and \*/ ).
- Recompiled the c-mex functions for windows.

Version 1.4.0: Feb. 26, 2006

- **InitializeDVBS2:** The entered code rate only needs to be within 0.01 of the desired value. So, for instance, rate 1/3 could be entered as rate = 0.33. If the rate and length is not valid, an error message is returned to the user.
- **DVBS2Scenarios:** Now all 21 DVBS2 LDPC codes are represented as scenarios in this file. Simulation results have been run down to at least BER =  $10^{-2}$  for all of these cases, and future library updates will include results to lower BER.
- **CmlSimulate** and **SingleSimulate:** The CmlSimulate function has been revised so that inside the main loop it calls a new function called SingleSimulate. SingleSimulate is in charge of simulating a single scenario case, and branches to CapacitySimulate, UncodedSimulate, CodedSimulate, or OutageSimulate depending on how sim\_param.sim\_type is set. This modification was made to facilitate support for grid computing (which will be released in a future update).
- **CmlSimulate** and **CodedSimulate:** The code\_param structure is now created inside CmlSimulate instead of in CodedSimulate.
- **UncodedSimulate:** Bug fix. The line  
condition3 = ~mod( sim\_state.trials(snrpoint),save\_rate );  
has been changed to  
condition3 = ~mod( sim\_state.trials(snrpoint),sim\_param.save\_rate );

Version 1.4.1: Mar.4, 2006

- **BlockcodeScenarios:** This new scenario file gives examples of APP decoding of cyclic block codes.
- **Filenaming:** It is not obvious to some users that every record must have a distinct filename. To make this more obvious, the following statement is placed at the bottom of each Scenario File:

```
% To add a new record, cut and paste one of the above records.
% Change record number to be one higher than the last used.
% Modify parameters as desired.
%
% Important: Each record must have a unique filename. It is recommended
% that for each new record you set
% sim_param(record).filename = strcat( data_directory, base_name, int2str(record), '.mat' );
```

Version 1.5.0: Apr. 20, 2006

- Support for orthogonal FSK Modulation.
- UncodedSimulate and CodedSimulate have been consolidated into the single function “ModSimulate”

Version 1.5.1: Apr. 23, 2006

- Compiled version of SingleSimulate will run without needing matlab. This was a necessary step for getting CML to run on the grid.

Version 1.5.2: Apr. 24, 2006

- Fixed a problem with turbo-coded BICM-ID.
- Saves results to temporary file, then uses movefile (this prevents problems with system crashing during save).

Version 1.5.3: May 10, 2006

- Support for nonorthogonal full-response FSK ( $h < 1$ ).
- New function “Modulate” combines the functionality of FskModulate and Mod2D. Create2D has been replaced with CreateConstellation, which is able to create FSK constellations.

Version 1.6: June 5, 2006

- Functions that read scenarios (CapacityRead, CodedRead, OutageRead, UncodedRead) have been combined into a single function, SingleRead
- Phased out sim\_param.reset = -1; no longer creates “backup” files because the fix instituted in version 1.5.2 has eliminated the corrupted file problem.
- Added “bwcapacity” and “minSNRvsB” simulation types.
- CmlPlot is able to plot results even if sim\_param.reset = 1.
- “ThroughputRead” replaced with “CalculateThroughput”; “throughput” simulation types must first be first run using CmlSimulate, prior to running CmlPlot
- “IntializeSimParam” replaced with “DefineStructures”
- Packaged using three separate files, one containing the source/runtime code (cml.X.X.X.rar), one containing the library of simulated data (output.X.X.X.rar) and the other for grid support (grid.X.X.X.rar).

Version 1.6.1: June 27, 2006

- Added support for the CCSDS turbo code through the inclusion of the CreateCcsdsInterleaver c-mex function and the CcsdsScenarios.m file.
- Added the PN scrambling to HSDPA (implemented as function PnGenerator.m) and corrected a bug in the HarqMatching and HarqDematching functions whereby the offset “a” used by the two rate matching blocks during the first rate matching were backwards.
- The HsdpaDemo function has been updated to use the CreateConstellation and Modulation functions (which became outdated with release 1.5.3).
- Added commas between the two output arguments in the DefineStructures.m function.
- In SingleRead, the matrices BER, FER, SER, and capacity\_avg are initialized to all-zeros before the save file is read.
- All existing scenario files have been updated so that if their output directory does not exist, it will create it.
- CmlStartup adds ./cml/grid/mat to the path only if it exists.

Version 1.6.2: July 5, 2006

- InitializeCodeParam has been revised to properly set up the turbo code used by HSDPA.

- CmlStartup senses if it is being run on a windows pc or a unix/linux machine. If on windows, it uses “\” between directories; if on unix, it uses “/”.
- Likewise, the scenario files sense if on a pc or unix machine to determine whether to separate directory names with “/” or “\”. If the output directory specified in the scenario file does not exist, it will be created.
- The make72 and make files in the ./cml/source directory have been merged into a single makefile (simply called make). The new makefile senses if the matlab version is 7.2 or if it is earlier than 7.2, and makes appropriate adjustments based on the matlab version. If run on unix, then the old compiled mex files are deleted before they are recompiled.
- All .c and .h files have an end-of-line at the end of the file (absences of the end-of-line causes some C compilers to flag a warning message).