List of m-files, with initial comments lines, from `..\Comp\*.m`. This list was printed 05-Dec-2002 15:20:32 by the MakeTex.m function.

contents.m	1208 bytes	28-Jun-2001 15:54:40

```
%Write text describing the m-files in this directory
%
% Arith06     Arithmetic encoder or decoder
% Arith07     Arithmetic encoder or decoder
% entropy     Function returns 0th order entropy of a source.
% eob3        End Of Block Encoding (or decoding) into (from) three sequences
% Huff06      Huffman encoder/decoder with (or without) recursive splitting
% HuffCode    Based on the codeword lengths this function find the Huffman codewords
% HuffLen     Find the lengths of the Huffman code words
% HuffTabLen  Find how many bits we need to store the Huffman Table information
% HuffTree    Make the Huffman-tree from the lengths of the Huffman codes
% JPEGlike    Entropy encoding (or decoding) in a JPEG like manner
% Mat2Vec     Convert an integer matrix to a cell array of vectors,
% TestArith   Test and example of how to use Arith06 and Arith07
% TestBin     Test coding of binary sequence
% TestBin2    Find difference of some coding strategies
% TestHuff    Test and example of how to use Huff06
% TestMat2Vec Test and example of how to use Mat2Vec
% UniQuant    Uniform scalar quantizer (or inverse quantizer) with threshold
```

Arith06.m	19115 bytes	28-Jun-2001 15:54:02

```
% Arith06     Arithmetic encoder or decoder
% Vectors of integers are arithmetic encoded,
% these vectors are collected in a cell array, xC.
% If first argument is a cell array the function do encoding,
% else decoding is done.
% [y, Res] = Arith06(xC);                    % encoding
% y = Arith06(xC);                           % encoding
% xC = Arith06(y);                           % decoding
% ----------------------------------------------------------------
% Arguments:
%  y        a column vector of non-negative integers (bytes) representing
%           the code, 0 <= y(i) <= 255.
%  Res      a matrix that sum up the results, size is (NumOfX+1)x4
%           one line for each of the input sequences, the columns are
%           Res(:,1) - number of elements in the sequence
%           Res(:,2) - unused (=0)
%           Res(:,3) - bits needed to code the sequence
%           Res(:,4) - bit rate for the sequence, Res(:,3)/Res(:,1)
%           Then the last line is total (which include bits needed to store NumOfX)
%  xC       a cell array of column vectors of integers representing the
%           symbol sequences. (should not be to large integers)
%           If only one sequence is to be coded, we must make the cell array
%           like: xC=cell(2,1); xC{1}=x; % where x is the sequence
% ----------------------------------------------------------------
% Note: this routine is extremely slow since it is all Matlab code
% This function do recursive encoding like Huff06.
% An alternative (a perhaps better) aritmethic coder is Arith07,
% which is a more "pure" arithmetic coder
```

```
% SOME NOTES ON THE FUNCTION
% The descrition of the encoding algorithm is in
% chapter 5 of "The Data Compression Book" by Mark Nelson.
% The actual coding algorithm is practical identical, it is a translation
% from C code to MatLab code, but some differences have been made.
% The system model, T, keep record of the symbols that have been encoded.
% Based on this table the probabiltity of each symbol is estimated. Probability
% for symbol m is: (T(m+1)-T(m+2))/T(1)
% The symbols are 0,1,...,M and Escape (M+1), Escape is used to indicate an
% unused symbol, which is then coded by another table, the Tu table.
% POSSIBLE IMPROVEMENTS
% - better decision wether to split a sequence or not
% - for long sequences, update frequency table T=floor(T*a)  (ex: 0.2 < a < 0.9)
%   and do this for every La samples (ex: 100 < La < 5000)
%   We must not set any non-zero probabilities to zero during this adaption!!
% - Display some information (so users know something is happening)
```

Arith07.m 29971 bytes 31-Oct-2002 09:08:10

```
% Arith07     Arithmetic encoder or decoder
% Vectors of integers are arithmetic encoded,
% these vectors are collected in a cell array, xC.
% If first argument is a cell array the function do encoding,
% else decoding is done.
% [y, Res] = Arith07(xC);                    % encoding
% y = Arith07(xC);                           % encoding
% xC = Arith07(y);                           % decoding
% ----------------------------------------------------------------------
% Arguments:
%  y        a column vector of non-negative integers (bytes) representing
%           the code, 0 <= y(i) <= 255.
%  Res      a matrix that sum up the results, size is (NumOfX+1)x4
%           one line for each of the input sequences, the columns are
%           Res(:,1) - number of elements in the sequence
%           Res(:,2) - unused (=0)
%           Res(:,3) - bits needed to code the sequence
%           Res(:,4) - bit rate for the sequence, Res(:,3)/Res(:,1)
%           Then the last line is total (which include bits needed to store NumOfX)
%  xC       a cell array of column vectors of integers representing the
%           symbol sequences. (should not be to large integers)
%           If only one sequence is to be coded, we must make the cell array
%           like: xC=cell(2,1); xC{1}=x; % where x is the sequence
% ----------------------------------------------------------------------
% Note: this routine is extremely slow since it is all Matlab code
% SOME NOTES ON THE FUNCTION
% This function is almost like Arith06, but some important changes have
% been done. Arith06 is buildt almost like Huff06, but this close connection
% is removed in Arith07. This imply that to understand the way Arith06
% works you should read the dokumentation for Huff06 and especially the
% article on Recursive Huffman Coding. To understand how Arith07 works it is
% only confusing to read about the recursive Huffman coder, Huff06.
```

entropy.m 540 bytes 21-Jun-2000 14:42:32

```
% entropy     Function returns 0th order entropy of a source.
%
% H = entropy(S)
% S is probability or count of each symbol
```

```
% S should be a vector of non-negative numbers.
% Ver. 1.0  09.10.97  Karl Skretting
% Ver. 1.1  25.12.98  KS, Signal Processing Project 1998, english version
```

eob3.m	6930 bytes	21-Jun-2000 14:43:48

```
% eob3        End Of Block Encoding (or decoding) into (from) three sequences
% The EOB sequence of numbers (x) is splitted into three sequences,
% (x1, x2, x3), based on previous symbol. The total (x) will have
% L EOB symbol (EOB is 0) for the rest x is one more than y
% The reason to split into several sequences is that the statistics for
% each sequence will be different and this may be exploited in entropy coding
% x = eob3(y);                    % encoding into one sequence
% [x1,x2,x3] = eob3(y);           % encoding into three sequences
% [x,x1,x2,x3] = eob3(y);         % encoding into one sequence and three sequences
% y = eob3(x, N);                 % decoding from one sequence
% y = eob3(x1, x2, x3, N);        % decoding from three sequences
% ---------------------------------------------
% arguments:
%   x       - all symbols in the EOB sequence, this sequence may
%             be splitted into the three following sequence
%             length(x)=length(x1)+length(x2)+length(x3)
%   x1      - the first symbol and all symbols succeeding an EOB symbol
%   x2      - all symbols succeeding a symbol representing zero (in x this is 1),
%             this will never be an EOB symbol (which is 0)
%   x3      - other symbols
%   y       - A matrix, size NxL, of non-negtive integers
%   N       - Length of Block, it is length of column in y,
% ---------------------------------------------
% Note: Number of input arguments indicate encoding or decoding!
%----------------------------------------------------------------------
% Copyright (c) 1999.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  01.01.99  Karl Skretting, Signal Processing Project 1998
% Ver. 1.1  14.01.99  KS, sort rows of y to get rows with fewest
%                     zeros on the top.
% Ver. 1.2  10.03.99  KS, made eob3 based on c_eob
% Ver. 1.3  21.06.00  KS, some minor changes (and moved to ..\comp\ )
%----------------------------------------------------------------------
```

Huff06.m	24949 bytes	12-Nov-2001 12:52:20

```
% Huff06      Huffman encoder/decoder with (or without) recursive splitting
% Vectors of integers are Huffman encoded,
% these vectors are collected in a cell array, xC.
% If first argument is a cell array the function do encoding,
% else decoding is done.
% [y, Res] = Huff06(xC, Level, Speed);                    % encoding
% y = Huff06(xC);                                         % encoding
% xC = Huff06(y);                                         % decoding
% ------------------------------------------------------------------
% Arguments:
%  y        a column vector of non-negative integers (bytes) representing
%           the code, 0 <= y(i) <= 255.
%  Res      a matrix that sum up the results, size is (NumOfX+1)x4
```

```
%           one line for each of the input sequences, the columns are
%           Res(:,1) - number of elements in the sequence
%           Res(:,2) - zero-order entropy of the sequence
%           Res(:,3) - bits needed to code the sequence
%           Res(:,4) - bit rate for the sequence, Res(:,3)/Res(:,1)
%           Then the last line is total (which include bits needed to store NumOfX)
%  xC       a cell array of column vectors of integers representing the
%           symbol sequences. (should not be to large integers)
%           If only one sequence is to be coded, we must make the cell array
%           like: xC=cell(2,1); xC{1}=x; % where x is the sequence
%  Level    How many levels of splitting that is allowed, legal values 1-8
%           If Level=1, no further splitting of the sequences will be done
%           and there will be no recursive splitting.
%  Speed    For complete coding set Speed to 0. Set Speed to 1 to cheat
%           during encoding, y will then be a sequence of zeros only,
%           but it will be of correct length and the other output
%           arguments will be correct.
% ----------------------------------------------------------------------
% SOME NOTES ON THE FUNCTION
% huff06 depends on other functions for Huffman code, and the functions in this file
%  HuffLen     - find length of codewords (HL)
%  HuffTabLen  - find bits needed to store Huffman table information (HL)
%  HuffCode    - find huffman codewords
%  HuffTree    - find huffman tree
```

HuffCode.m	2242 bytes	21-Jun-2000 14:44:18

```
% HuffCode    Based on the codeword lengths this function find the Huffman codewords
% HK = HuffCode(HL,Display);
% HK = HuffCode(HL);
% ----------------------------------------------------------------------
% Arguments:
%  HL     length (bits) for the codeword for each symbol
%         This is usually found by the hufflen function
%  HK     The Huffman codewords, a matrix of ones or zeros
%         the code for each symbol is a row in the matrix
%         Code for symbol S(i) is: HK(i,1:HL(i))
%         ex: HK(i,1:L)=[0,1,1,0,1,0,0,0] and HL(i)=6 ==>
%             Codeword for symbol S(i) = '011010'
%  Display==1  ==> Codewords are displayed on screen, Default=0
% ----------------------------------------------------------------------
%----------------------------------------------------------------------
% Copyright (c) 1999.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  25.08.98  KS: Function made as part of Signal Compression Project 98
% Ver. 1.1  25.12.98  English version of program
%----------------------------------------------------------------------
```

HuffLen.m	3880 bytes	22-Jun-2000 13:11:44

```
% HuffLen     Find the lengths of the Huffman code words
% Based on probability (or number of occurences) of each symbol
% the length for the Huffman codewords are calculated.
%
% HL = hufflen(S);
```

```
% ----------------------------------------------------------------
% Arguments:
%  S  a vector with number of occurences or probability of each symbol
%     Only positive elements of S are used, zero (or negative)
%     elements get length 0.
%  HL length (bits) for the codeword for each symbol
% ----------------------------------------------------------------
% Example:
% hufflen([1,0,4,2,0,1])  =>  ans = [3,0,1,2,0,3]
% hufflen([10,40,20,10])  =>  ans = [3,1,2,3]
%----------------------------------------------------------------------
% Copyright (c) 1999.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  28.08.98  KS: Function made as part of Signal Compression Project 98
% Ver. 1.1  25.12.98  English version of program
% Ver. 1.2  28.07.99  Problem when length(S)==1 was corrected
% Ver. 1.3  22.06.00  KS: Some more exceptions handled
%----------------------------------------------------------------------
```

HuffTabLen.m	4283 bytes	21-Aug-2001 14:23:12

```
% HuffTabLen  Find how many bits we need to store the Huffman Table information
% HLlen = HuffTabLen(HL);
%----------------------------------------------------------------------
% arguments:
%  HL       The codeword lengths, as returned from HuffLen function
%           This should be a vector of integers
%           where  0 <= HL(i) <= 32, 0 is for unused symbols
%           We then have max codeword length is 32
%  HLlen    Number of bits needed to store the table
%----------------------------------------------------------------------
% Function assume that the table information is stored in the following format
%       previous code word length is set to the initial value 2
%       Then we have for each symbol a code word to tell its length
%        '0'            - same length as previous symbol
%        '10'           - increase length by 1, and 17->1
%        '1100'         - reduce length by 1, and 0->16
%        '11010'        - increase length by 2, and 17->1, 18->2
%        '11011'        - One zero, unused symbol (twice for two zeros)
%        '111xxxx'      - set code length to CL=Prev+x (where 3 <= x <= 14)
%                         and if CL>16; CL=CL-16
%       we have 4 unused 7 bit code words, which we give the meaning
%        '1110000'+4bits - 3-18 zeros
%        '1110001'+8bits - 19-274 zeros, zeros do not change previous value
%        '1110010'+4bits - for CL=17,18,...,32, do not change previous value
%        '1111111'      - End Of Table
```

HuffTree.m	2514 bytes	19-Jun-2000 14:04:22

```
% HuffTree    Make the Huffman-tree from the lengths of the Huffman codes
% The Huffman codes are also needed, and if they are known
% they can be given as an extra input argument
% Htree = HuffTree(HL,HK);
% Htree = HuffTree(HL);
% ----------------------------------------------------------------
```

```
% Arguments:
%  HL     length (bits) for the codeword for each symbol
%         This is usually found by the hufflen function
%  HK     The Huffman codewords, a matrix of ones or zeros
%         the code for each symbol is a row in the matrix
%  Htree  A matrix, (N*2)x3, representing the Huffman tree,
%         needed for decoding. Start of tree, root, is Htree(1,:).
%         Htree(i,1)==1 indicate leaf and Htree(i,1)==0 indicate branch
%         Htree(i,2) points to node for left tree if branching point and
%         symbol number if leaf. Note value is one less than symbol number.
%         Htree(i,3) points to node for right tree if branching point
%         Left tree is '0' and right tree is '1'
% ----------------------------------------------------------------
%----------------------------------------------------------------------
% Copyright (c) 1999.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  25.08.98  KS: Function made as part of Signal Compression Project 98
% Ver. 1.1  25.12.98  English version of program
%----------------------------------------------------------------------
```

JPEGlike.m	11527 bytes	21-Jun-2000 14:44:50

```
% JPEGlike    Entropy encoding (or decoding) in a JPEG like manner
% Coding is very similar to sequential Huffman coding as described in
%   William B. Pennebaker, Joan L. Mitchell.
%   JPEG: Still Image Data Compression Standard, chapter 11.1
%   (Van Nostrand Reinhold, New York, USA, 1992, ISBN: 0442012721)
% The number of input arguments decide whether it is encoding or decoding,
% three input argument for decoding, two for encoding.
% [y,Res] = JPEGlike(Speed, W);             % encoding
% y = JPEGlike(Speed, W);                   % encoding
% W = JPEGlike(y, N, L);                    % decoding
%----------------------------------------------------------------------
% arguments:
%   W       The quantized values, a matrix where the first row is the
%           DC component and the following rows are the AC components.
%           We assume AC components (rows) are ordered in ascending frequencies.
%  N        Number of rows in W
%  L        Number of columns in W, [N,L]=size(W)
%  Speed    For complete coding set Speed to 0. Set Speed to 1 to cheat
%           during encoding, y will then be a sequence of zeros only,
%           but it will be of correct length and the other output
%           arguments will be correct.
%  y        a column vector of non-negative integers (bytes) representing
%           the code, 0 <= y(i) <= 255.
%  Res      The results (encoding only)        for DC part       for AC part
%           Number og symbols:                 Res(1)            Res(5)
%           Bits to store Huffman table (HL):  Res(2)            Res(6)
%           Bits to store Huffman symbols:     Res(3)            Res(7)
%           Bits to store additional bits:     Res(4)            Res(8)
%           The total of bits is then:  sum(Res([2:4,6:8]))
%----------------------------------------------------------------------
% Function needs following m-files: HuffLen, HuffCode, HuffTree
```

Mat2Vec.m	8134 bytes	31-Jul-2000 16:40:12

```
% Mat2Vec     Convert an integer matrix to a cell array of vectors,
% several different methods are possible, most of them are non-linear.
% The inverse function is also performed by this function,
% to use this first argument should be a cell array instead of a matrix.
% Examples:
% xC = Mat2Vec(W, Method);           % convert the KxL matrix W to vectors
% xC = Mat2Vec(W, Method, K, L);     % convert the KxL matrix W to vectors
% W = Mat2Vec(xC, Method, K, L);     % convert vectors in xC to a KxL matrix
% ----------------------------------------------------------------------------
% arguments:
%  xC       a cell array of column vectors of integers representing the
%           symbol sequences for matrix W.
%  W        a KxL matrix of integers
%  Method   which method to use when transforming the matrix of quantized
%           values into one or several vectors of integers.
%           The methods that only return non-negative integers in xC are
%           marked by a '+', the others also returns negative integers
%           if W contain negative integers.
%           For Method=10,11,14 and 15 we have K=2,4,8,16,32,64, or 128.
%           The legal methods are
%              0    by columns, direct                          1 seq.
%              1    by columns, run + values                    2 seq.
%              2    by rows, direct                             1 seq.
%              3    by rows, run + values                       2 seq.
%              4 +  EOB coded (by columns)                      1 seq.
%              5 +  EOB coded (by columns)                      3 seq.
%              6 +  by columns, run + values                    2 seq.
%              7 +  by rows, run + values                       2 seq.
%              8    each row, direct                            K seq.
%              9    each row, run + values                    2*K seq.
%             10    each dyadic subband, direct         log2(2*K)seq.
%             11    each dyadic subband, run + values  2*log2(2*K)seq.
%             12 +  each row, direct                            K seq.
%             13 +  each row, run + values                    2*K seq.
%             14 +  each dyadic subband, direct         log2(2*K)seq.
%             15 +  each dyadic subband, run + values  2*log2(2*K)seq.
%  K       size of matrix W, number of rows
%  L       size of matrix W, number of columns
% ----------------------------------------------------------------------------
```

TestArith.m	6240 bytes	21-Aug-2001 14:40:38

```
% TestArith   Test and example of how to use Arith06 and Arith07
%----------------------------------------------------------------------
% Copyright (c) 2000.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  10.04.2001  KS: function made
% Ver. 1.1  28.06.2001  KS: more test signals
%----------------------------------------------------------------------
```

TestBin.m	19534 bytes	16-May-2001 21:15:10

```
% TestBin     Test coding of binary sequence
%             We have a sequence of binary symbols (0/1), where the
% probability of a 1 is p1 and the probability of a 0 is p0=1-p1
```

```
% Direct arithmetic coding (DAC) of this sequence (p1 is given) is compared to
% [adaptive arithmetic coding (AAC) and] count down arithmetic coding (CDAC).
% The value we are interested in is the bit rate, which will be a random
% variable depending on the coding method (DAC/AAC/CDAC),
% the length of the sequence (N), and the probability of a 1 (p1).
% The expectation (and sometimes the distribution) for this random variable
% is interesting.
%----------------------------------------------------------------------
% Copyright (c) 2000.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  14.05.2001  KS: function made
%----------------------------------------------------------------------
```

TestBin2.m	2513 bytes	24-May-2001 21:22:38

```
% TestBin2    Find difference of some coding strategies
%             First arithmetic coding when first storing N
% (we assume that L is already stored)
% Then a variant of this where the probabilities is modified for
% each new entry in x (the sequence)
%----------------------------------------------------------------------
% Copyright (c) 2001.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  24.05.2001  KS: function made
%----------------------------------------------------------------------
```

TestHuff.m	1718 bytes	28-Jun-2001 15:53:56

```
% TestHuff    Test and example of how to use Huff06
%----------------------------------------------------------------------
% Copyright (c) 2000.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  20.06.2000  KS: function made
%----------------------------------------------------------------------
% first make some data we will use in test
```

TestMat2Vec.m	3161 bytes	14-Nov-2000 12:57:42

```
% TestMat2Vec Test and example of how to use Mat2Vec
%             This m-file first make a test signal (an AR-1 signal), x,
% then we take an DCT on the test signal, and organize the
% coefficients in a matrix. This matrix is quantized using
% a unifor quantizer with threshold, this gives a matrix of integers, W.
% The entries in W may be ordered into sequences of integers in many
% different ways, and this is done by Mat2Vec. These sequences are
% then Huffman coded by Huff06.
% We also compare this to the JPEGlike way of compressing W, in JPEGlike.m
```

```
%----------------------------------------------------------------------
% Copyright (c) 2000.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  21.06.2000  KS: function made
%----------------------------------------------------------------------
% first make some data we will use in test
```

UniQuant.m	1876 bytes	01-Dec-1999 13:19:34

```
% UniQuant    Uniform scalar quantizer (or inverse quantizer) with threshold
% Note: Use three arguments for inverse quantizing and
%       four arguments for quantizing.
% Y = UniQuant(X, del, thr, ymax);     % quantizer
% X = UniQuant(Y, del, thr);           % inverse quantizer
% ----------------------------------------------
% arguments:
%   X    - the values to be quantized (or result after inverse
%          quantizer), a vector or matrix with real values.
%   Y    - the indexes for the quantizer cells, the bins are indexed as
%          ..., -3, -2, -1, 0, 1, 2, 3, ...  where 0 is for the zero bin
%   del  - delta i quantizer, size/width of all cells except zero-cell
%   thr  - threshold value, width of zero cell is from -thr to +thr
%   ymax - largest value for y, only used when quantizing
% ----------------------------------------------
%----------------------------------------------------------------------
% Copyright (c) 1999.  Karl Skretting.  All rights reserved.
% Hogskolen in Stavanger (Stavanger University), Signal Processing Group
% Mail:  karl.skretting@tn.his.no   Homepage:  http://www.ux.his.no/~karlsk/
%
% HISTORY:
% Ver. 1.0  27.07.99  Karl Skretting, Signal Processing Project 1999
%                     function made based on c_q1.m
%----------------------------------------------------------------------
```