List of m-files, with initial comments lines, from f:\matlab\Comp\*.m. This list was printed 21-Feb-2011 09:29:38 by the MakeTex.m function.

contents.m 981 bytes 21-feb-2011 09:26:00

% Text describing the m-files in directory f:\matlab\Comp
% File generated by mkcontnt.m 21-Feb-2011 09:25:59
% Arith06 Arithmetic encoder or decoder
% Arith07 Arithmetic encoder or decoder
% entropy Function returns first 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
% Mat2Vec Convert an integer matrix to a cell array of vectors,
% TestArith Test and example of how to use Arith06 and Arith07
% TestHuff Test and example of how to use Huff06
% uniquant Uniform scalar quantizer (or inverse quantizer) with threshold

Arith06.m 19115 bytes 28-jun-2001 20:54:02

% 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) arithmetic coder is Arith07,
% which is a more "pure" arithmetic coder
% SOME NOTES ON THE FUNCTION
% The descriotion 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 probability of each symbol is estimated. Probability for symbol \( m \) is: \( \frac{T(m+1) - T(m+2)}{T(1)} \)

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 whether to split a sequence or not
- for long sequences, update frequency table \( T = \text{floor}(T \cdot a) \) (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**

% 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 on Matlab version 5.x and earlier
% SOME NOTES ON THE FUNCTION
% This function is almost like Arith06, but some important changes have
% been done. Arith06 is built almost like Huff06, but this close connection
% is removed in Arith07. This imply that to understand the way Arith06
% works you should read the documentation 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**

% entropy Function returns first 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
% 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
% see also ..\ICTools\myreshape.m (which is mainly for images)
% \[x1,x2,x3\] = eob3(y);  % encoding into three sequences
% [x,x1,x2,x3] = eob3(y);  % encoding into one sequence and three sequences
% \[x,y\] = 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-negative 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.
% % 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\ )
% % Ver. 1.4 08.06.09 KS, warning messages changed
% %----------------------------------------------------------------------

Huff06.m  25888 bytes  22-okt-2010 14:37:30

% 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 19:44:18

% HuffCode  Based on the codeword length 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)=[0,1,0,1,0,0,0] 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  3883 bytes  18-nov-2009 11:53:30

% HuffLen  Find the lengths of the Huffman code words
% Based on probability (or number of occurrences) of each symbol
% the length for the Huffman codewords are calculated.
% % HL  = hufflen(S);
% %-%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arguments:
% S  a vector with number of occurrences 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 26.07.99 Problem when length(S)==1 was corrected
% Ver. 1.3 22.06.00 KS: Some more exceptions handled
%----------------------------------------------------------------------

HuffTabLen.m

% 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

% 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,
Mat2Vec.m

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 \log_2(2*K) seq.
11 each dyadic subband, run + values 2*\log_2(2*K) seq.
12 each row, direct K seq.
13 each row, run + values 2*K seq.
14 each dyadic subband, direct \log_2(2*K) seq.
15 each dyadic subband, run + values 2*\log_2(2*K) seq.
The following ones are for K = 4, 16, 64, 256 or 1024
16 each 2D-dyadic, direct 1+(3/2)\log_2(K) seq.
17 each 2D-dyadic, run+value 2+3*\log_2(K) seq.
18 each 2D-dyadic, direct 1+(3/2)\log_2(K) seq.
19 each 2D-dyadic, run+value 2+3*\log_2(K) seq.
%% 20 + EOB coded (by columns, 2D-dyadic) 3 seq.
%% X size of matrix W, number of rows
%% L size of matrix W, number of columns
%-----------------------------------------------------------------------
%% methods 16-19 added jun 5. 2009, KS

TestArith.m 6257 bytes 22-okt-2010 15:06:18

%% 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
%%-----------------------------------------------------------------------

TestHuff.m 1728 bytes 22-okt-2010 15:08:22

%% 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
%%-----------------------------------------------------------------------

uniquant.m 1880 bytes 22-okt-2010 14:51:34

%% 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

Ver. 1.2  22.10.10  KS: same as  ..\ICTools\uniquant

-----------------------------------------------------------------------