Digital Image Processing

BB1603391 116033910045 修宇亮

Problem 7 Requirement 1

7. Transform image compression (test image lenna.tif)

(a) Investigate image compression based on DCT. Divide the image into 8-by-8 subimages, compute the two-dimensional discrete cosine transform of each subimage, compress the test image to different qualities by discarding some DCT coefficients based on zonal mask and threshold mask and using the inverse discrete cosine transform with fewer transform coefficients. Display the original image, the reconstructed images and the difference images.

Problem solution 1

im2jpeg.m

MATLAB

```
function y = im2jpeg(x, quality)
%IM2JPEG Compresses an image using a JPEG approximation.
%   Y = IM2JPEG(X, QUALITY) compresses image X based on 8 x 8 DCT
%   transforms, coefficient quantization, and Huffman symbol
%   coding. Input QUALITY determines the amount of information that
%   is lost and compression achieved.  Y is an encoding structure
%   containing fields:
%
%      Y.size      Size of X
%      Y.numblocks Number of 8-by-8 encoded blocks
%      Y.quality   Quality factor (as percent)
%      Y.huffman   Huffman encoding structure, as returned by
%                  MAT2HUFF
%
%   See also JPEG2IM.
```

```
%   Copyright 2002-2006 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.5 $  $Date: 2006/07/15 20:44:34 $
%
%   Revised: 3/20/06 by R.Woods to correct an 'eob' coding problem, to
%   check the 'quality' input for <= 0, and to fix a warning when
%   struct y is created.

error(nargchk(1, 2, nargin));             % Check input arguments
if ndims(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
   error('The input must be a UINT8 image.');
end
if nargin ==2 && quality <= 0
   error('Input parameter QUALITY must be greater than zero.');
end
if nargin < 2
   quality = 1;   % Default value for quality.
end

m = [16 11  10  16  24  40  51  61        % JPEG normalizing array
     12  12  14  19  26  58  60  55       % and zig-zag redordering
     14  13  16  24  40  57  69  56       % pattern.
     14  17  22  29  51  87  80  62
     18  22  37  56  68  109 103 77
     24  35  55  64  81  104 113 92
     49  64  78  87  103 121 120 101
     72  92  95  98  112 100 103 99] * quality;

order = [1 9  2  3  10 17 25 18 11 4  5  12 19 26 33  ...
        41 34 27 20 13 6  7  14 21 28 35 42 49 57 50  ...
        43 36 29 22 15 8  16 23 30 37 44 51 58 59 52  ...
        45 38 31 24 32 39 46 53 60 61 54 47 40 48 55  ...
        62 63 56 64];

[xm, xn] = size(x);                    % Get input size.
x = double(x) - 128;                   % Level shift input
t = dctmtx(8);                         % Compute 8 x 8 DCT matrix

% Compute DCTs of 8x8 blocks and quantize the coefficients.
y = blkproc(x, [8 8], 'P1 * x * P2', t, t');
y = blkproc(y, [8 8], 'round(x ./ P1)', m);

y = im2col(y, [8 8], 'distinct');  % Break 8x8 blocks into columns
xb = size(y, 2);                       % Get number of blocks
y = y(order, :);                       % Reorder column elements

eob = max(y(:)) + 1;                   % Create end-of-block symbol
```

```
r = zeros(numel(y) + size(y, 2), 1);
count = 0;
for j = 1:xb                          % Process 1 block (col) at a time
   i = max(find(y(:, j)));            % Find last non-zero element
   if isempty(i)                      % No nonzero block values
      i = 0;
   end
   p = count + 1;
   q = p + i;
   r(p:q) = [y(1:i, j); eob];         % Truncate trailing 0's, add EOB,
   count = count + i + 1;             % and add to output vector
end

r((count + 1):end) = [];              % Delete unused portion of r

y            = struct;
y.size       = uint16([xm xn]);
y.numblocks = uint16(xb);
y.quality    = uint16(quality * 100);
y.huffman    = mat2huff(r);
```

jpeg2im.m

MATLAB

```
function x = jpeg2im(y)
%JPEG2IM Decodes an IM2JPEG compressed image.
%   X = JPEG2IM(Y) decodes compressed image Y, generating
%   reconstructed approximation X.  Y is a structure generated by
%   IM2JPEG.
%
%   See also IM2JPEG.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.4 $  $Date: 2003/10/26 18:39:08 $

error(nargchk(1, 1, nargin));                % Check input arguments

m = [16 11  10  16  24  40  51  61           % JPEG normalizing array
     12  12  14  19  26  58  60  55          % and zig-zag reordering
     14  13  16  24  40  57  69  56          % pattern.
     14  17  22  29  51  87  80  62
     18  22  37  56  68  109 103 77
     24  35  55  64  81  104 113 92
     49  64  78  87  103 121 120 101
     72  92  95  98  112 100 103 99];

```

```
order = [1 9  2  3  10 17 25 18 11 4  5  12 19 26 33  ...
        41 34 27 20 13 6  7  14 21 28 35 42 49 57 50  ...
        43 36 29 22 15 8  16 23 30 37 44 51 58 59 52  ...
        45 38 31 24 32 39 46 53 60 61 54 47 40 48 55  ...
        62 63 56 64];
rev = order;                                % Compute inverse ordering
for k = 1:length(order)
   rev(k) = find(order == k);
end

m = double(y.quality) / 100 * m;      % Get encoding quality.
xb = double(y.numblocks);             % Get x blocks.
sz = double(y.size);
xn = sz(2);                           % Get x columns.
xm = sz(1);                           % Get x rows.
x = huff2mat(y.huffman);              % Huffman decode.
eob = max(x(:));                      % Get end-of-block symbol

z = zeros(64, xb);   k = 1;           % Form block columns by copying
for j = 1:xb                          % successive values from x into
   for i = 1:64                       % columns of z, while changing
      if x(k) == eob                  % to the next column whenever
         k = k + 1;   break;          % an EOB symbol is found.
      else
         z(i, j) = x(k);
         k = k + 1;
      end
   end
end

z = z(rev, :);                                 % Restore order
x = col2im(z, [8 8], [xm xn], 'distinct');     % Form matrix blocks
x = blkproc(x, [8 8], 'x .* P1', m);           % Denormalize DCT
t = dctmtx(8);                                 % Get 8 x 8 DCT matrix
x = blkproc(x, [8 8], 'P1 * x * P2', t', t);   % Compute block DCT-1
x = uint8(x + 128);                            % Level shift
```

mat2huff.m

MATLAB

```
function y = mat2huff(x)
%MAT2HUFF Huffman encodes a matrix.
%   Y = MAT2HUFF(X) Huffman encodes matrix X using symbol
%   probabilities in unit-width histogram bins between X's minimum
%   and maximum values. The encoded data is returned as a structure
%   Y:
%      Y.code   The Huffman-encoded values of X, stored in
```

```
%                a uint16 vector.  The other fields of Y contain
%                additional decoding information, including:
%       Y.min    The minimum value of X plus 32768
%       Y.size   The size of X
%       Y.hist   The histogram of X
%
%   If X is logical, uint8, uint16, uint32, int8, int16, or double,
%   with integer values, it can be input directly to MAT2HUFF. The
%   minimum value of X must be representable as an int16.
%
%   If X is double with non-integer values---for example, an image
%   with values between 0 and 1---first scale X to an appropriate
%   integer range before the call. For example, use Y =
%   MAT2HUFF(255*X) for 256 gray level encoding.
%
%   NOTE: The number of Huffman code words is round(max(X(:))) -
%   round(min(X(:))) + 1.  You may need to scale input X to generate
%   codes of reasonable length.  The maximum row or column dimension
%   of X is 65535.
%
%   See also HUFF2MAT.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.5 $  $Date: 2003/11/21 15:21:12 $

if ndims(x) ~= 2 | ~isreal(x) | (~isnumeric(x) & ~islogical(x))
   error('X must be a 2-D real numeric or logical matrix.');
end

% Store the size of input x.
y.size = uint32(size(x));

% Find the range of x values and store its minimum value biased
% by +32768 as a UINT16.
x = round(double(x));
xmin = min(x(:));
xmax = max(x(:));
pmin = double(int16(xmin));
pmin = uint16(pmin + 32768);     y.min = pmin;

% Compute the input histogram between xmin and xmax with unit
% width bins, scale to UINT16, and store.
x = x(:)';
h = histc(x, xmin:xmax);
if max(h) > 65535
    h = 65535 * h / max(h);
end
```

```
h = uint16(h);    y.hist = h;

% Code the input matrix and store the result.
map = huffman(double(h));           % Make Huffman code map
hx = map(x(:) - xmin + 1);          % Map image
hx = char(hx)';                     % Convert to char array
hx = hx(:)';
hx(hx == ' ') = [];                 % Remove blanks
ysize = ceil(length(hx) / 16);      % Compute encoded size
hx16 = repmat('0', 1, ysize * 16); % Pre-allocate modulo-16 vector
hx16(1:length(hx)) = hx;            % Make hx modulo-16 in length
hx16 = reshape(hx16, 16, ysize);    % Reshape to 16-character words
hx16 = hx16' - '0';                 % Convert binary string to decimal
twos = pow2(15:-1:0);
y.code = uint16(sum(hx16 .* twos(ones(ysize, 1), :), 2))';
```

huff2mat.m

MATLAB

```
function x = huff2mat(y)
%HUFF2MAT Decodes a Huffman encoded matrix.
%   X = HUFF2MAT(Y) decodes a Huffman encoded structure Y with uint16
%   fields:
%      Y.min    Minimum value of X plus 32768
%      Y.size   Size of X
%      Y.hist   Histogram of X
%      Y.code   Huffman code
%
%   The output X is of class double.
%
%   See also MAT2HUFF.

%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.5 $  $Date: 2003/11/21 13:17:50 $

if ~isstruct(y) | ~isfield(y, 'min') | ~isfield(y, 'size') | ...
       ~isfield(y, 'hist') | ~isfield(y, 'code')
   error('The input must be a structure as returned by MAT2HUFF.');
end

sz = double(y.size);   m = sz(1);   n = sz(2);
xmin = double(y.min) - 32768;             % Get X minimum
map = huffman(double(y.hist));            % Get Huffman code (cell)

% Create a binary search table for the Huffman decoding process.
% 'code' contains source symbol strings corresponding to 'link'
```

```
% nodes, while 'link' contains the addresses (+) to node pairs for
% node symbol strings plus '0' and '1' or addresses (-) to decoded
% Huffman codewords in 'map'. Array 'left' is a list of nodes yet to
% be processed for 'link' entries.

code = cellstr(char('', '0', '1'));     % Set starting conditions as
link = [2; 0; 0];    left = [2 3];       % 3 nodes w/2 unprocessed
found = 0;    tofind = length(map);      % Tracking variables

while length(left) & (found < tofind)
   look = find(strcmp(map, code{left(1)}));    % Is string in map?
   if look                                  % Yes
      link(left(1)) = -look;                % Point to Huffman map
      left = left(2:end);                   % Delete current node
      found = found + 1;                    % Increment codes found

   else                                     % No, add 2 nodes & pointers
      len = length(code);                   % Put pointers in node
      link(left(1)) = len + 1;

      link = [link; 0; 0];                  % Add unprocessed nodes
      code{end + 1} = strcat(code{left(1)}, '0');
      code{end + 1} = strcat(code{left(1)}, '1');

      left = left(2:end);                   % Remove processed node
      left = [left len + 1 len + 2];        % Add 2 unprocessed nodes
   end
end

x = unravel(y.code', link, m * n);      % Decode using C 'unravel'
x = x + xmin - 1;                       % X minimum offset adjust
x = reshape(x, m, n);                   % Make vector an array
```

huffman.m

MATLAB

```
function CODE = huffman(p)
%HUFFMAN Builds a variable-length Huffman code for a symbol source.
%   CODE = HUFFMAN(P) returns a Huffman code as binary strings in
%   cell array CODE for input symbol probability vector P. Each word
%   in CODE corresponds to a symbol whose probability is at the
%   corresponding index of P.
%
%   Based on huffman5 by Sean Danaher, University of Northumbria,
%   Newcastle UK. Available at the MATLAB Central File Exchange:
%   Category General DSP in Signal Processing and Communications.

```

```
%   Copyright 2002-2004 R. C. Gonzalez, R. E. Woods, & S. L. Eddins
%   Digital Image Processing Using MATLAB, Prentice-Hall, 2004
%   $Revision: 1.5 $  $Date: 2003/10/26 18:37:16 $

% Check the input arguments for reasonableness.
error(nargchk(1, 1, nargin));
if (ndims(p) ~= 2) | (min(size(p)) > 1) | ~isreal(p) | ~isnumeric(p)
   error('P must be a real numeric vector.');
end

% Global variable surviving all recursions of function 'makecode'
global CODE
CODE = cell(length(p), 1);  % Init the global cell array

if length(p) > 1            % When more than one symbol ...
   p = p / sum(p);          % Normalize the input probabilities
   s = reduce(p);           % Do Huffman source symbol reductions
   makecode(s, []);         % Recursively generate the code
else
   CODE = {'1'};            % Else, trivial one symbol case!
end;

%-------------------------------------------------------------------%
function s = reduce(p);
% Create a Huffman source reduction tree in a MATLAB cell structure
% by performing source symbol reductions until there are only two
% reduced symbols remaining

s = cell(length(p), 1);

% Generate a starting tree with symbol nodes 1, 2, 3, ... to
% reference the symbol probabilities.
for i = 1:length(p)
   s{i} = i;
end

while numel(s) > 2
   [p, i] = sort(p);    % Sort the symbol probabilities
   p(2) = p(1) + p(2);  % Merge the 2 lowest probabilities
   p(1) = [];           % and prune the lowest one

   s = s(i);            % Reorder tree for new probabilities
   s{2} = {s{1}, s{2}}; % and merge & prune its nodes
   s(1) = [];           % to match the probabilities
end

%-------------------------------------------------------------------%
function makecode(sc, codeword)
```

```
% Scan the nodes of a Huffman source reduction tree recursively to
% generate the indicated variable length code words.

% Global variable surviving all recursive calls
global CODE

if isa(sc, 'cell')                     % For cell array nodes,
   makecode(sc{1}, [codeword 0]);     % add a 0 if the 1st element
   makecode(sc{2}, [codeword 1]);     % or a 1 if the 2nd
else                                   % For leaf (numeric) nodes,
   CODE{sc} = char('0' + codeword);   % create a char code string
end
```

unravel.c

C

```c
/*===================================================================
* unravel.c
* Decodes a variable length coded bit sequence (a vector of
* 16-bit integers) using a binary sort from the MSB to the LSB
* (across word boundaries) based on a transition table.
*==================================================================*/
#include "mex.h"
void unravel(uint16_T *hx, double *link, double *x,
     double xsz, int hxsz)
{
   int i = 15, j = 0, k = 0, n = 0;      /* Start at root node, 1st */
                                         /* hx bit and x element */
   while (xsz - k)   {                   /* Do until x is filled */
   if (*(link + n) > 0)    {             /* Is there a link? */
      if ((*(hx + j) >> i) & 0x0001)     /* Is bit a 1? */
         n = *(link + n);                /* Yes, get new node */
      else n = *(link + n) - 1;          /* It's 0 so get new node */
      if (i) i--; else {j++; i = 15;}    /* Set i, j to next bit */
      if (j > hxsz)                      /* Bits left to decode? */
         mexErrMsgTxt("Out of code bits ???");
   }
   else    {                             /* It must be a leaf node */
      *(x + k++) = - *(link + n);        /* Output value */
       n = 0;     }                      /* Start over at root */
   }
   if (k == xsz - 1)                     /* Is one left over? */
      *(x + k++) = - *(link + n);
}
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
```

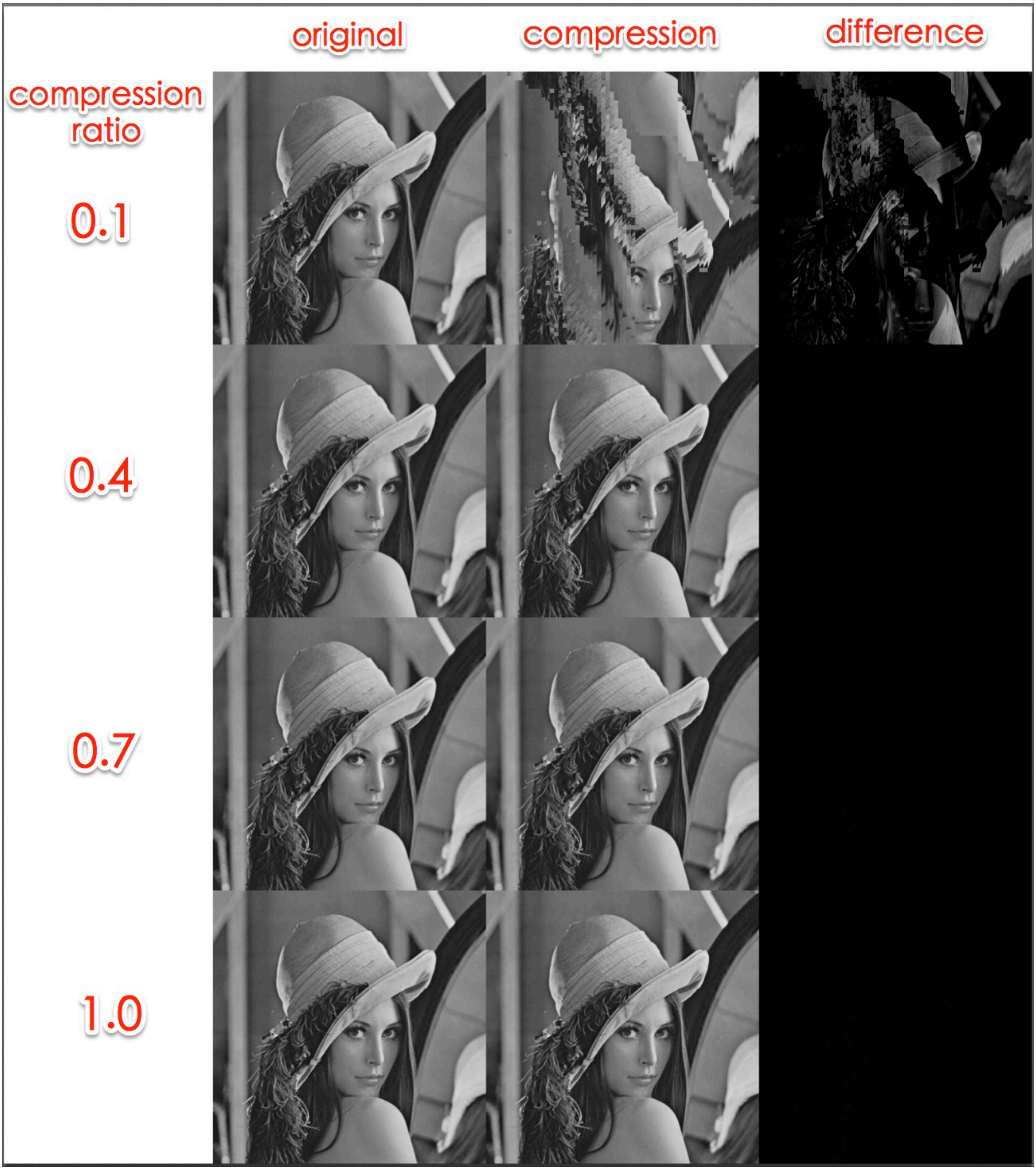
```
    double *link, *x, xsz;
    uint16_T *hx;
    int hxsz;

    /* Check inputs for reasonableness */
    if (nrhs != 3)
       mexErrMsgTxt("Three inputs required.");
    else if (nlhs > 1)
       mexErrMsgTxt("Too many output arguments.");

    /* Is last input argument a scalar? */
    if(!mxIsDouble(prhs[2])  || mxIsComplex(prhs[2])  ||
          mxGetN(prhs[2]) * mxGetM(prhs[2]) != 1)
       mexErrMsgTxt("Input XSIZE must be a scalar.");

    /* Create input matrix pointers and get scalar */
    hx = (uint16_T *) mxGetData(prhs[0]);
    link = (double *) mxGetData(prhs[1]);
    xsz = mxGetScalar(prhs[2]);          /* returns DOUBLE */

    /* Get the number of elements in hx */
    hxsz = mxGetM(prhs[0]);

    /* Create 'xsz' x 1 output matrix */
    plhs[0] = mxCreateDoubleMatrix(xsz, 1, mxREAL);

    /* Get C pointer to a copy of the output matrix */
    x = (double *) mxGetData(plhs[0]);

    /* Call the C subroutine */
    unravel(hx, link, x, xsz, hxsz);
}

```

prob7.m

MATLAB

```
img = imread('lenna.tif');
[M,N] = size(img);
all_in = zeros(4*M,3*N);
for k=0.1:0.3:1
    img_comp = im2jpeg(img,k);
    img_rec = jpeg2im(img_comp);
    img_diff = imsubtract(img,img_rec);
    all_in(round((k-0.1)/0.3)*M+1:round((k-0.1)/0.3)*M+M,1:N*3)=[img,img_rec,img_diff]
end
imshow(mat2gray(all_in));
```

Result



Problem 7 Requirement 2

(b) Investigate image compression based on wavelets. Consider four types of wavelets:

Haar: $h0=\left[\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right]$, $h1=\left[\frac{1}{\sqrt{2}},-\frac{1}{\sqrt{2}}\right]$

Daubechies: 8-tap

n	$g_0(n)$	$h_0(n)$	$g_1(n)$	$h_1(n)$
0	0.23037781	$g_0(7-n)$	$(-1)^n h_0(n)$	$g_1(7-n)$
1	0.71484657			
2	0.63088076			
3	-0.02798376			
4	-0.18703481			
5	0.03084138			
6	0.03288301			
7	-0.01059740			

Symlet: 8-tap

n	$g_0(n)=h_\varphi(n)$	$h_0(n)$	$g_1(n)$	$h_1(n)$
0	0.0322	$g_0(7-n)$	$(-1)^n h_0(n)$	$g_1(7-n)$
1	-0.0126			
2	-0.0992			
3	0.2979			
4	0.8037			
5	0.4976			
6	-0.0296			
7	-0.0758			

n	$h_0(n)$	$h_1(n)$	n	$h_0(n)$	$h_1(n)$
0	0	0	9	0.8259	0.4178
1	0.0019	0	10	0.4208	0.0404
2	-0.0019	0	11	-0.0941	-0.0787
3	-0.017	0.0144	12	-0.0773	-0.0145
4	0.0119	-0.0145	13	0.0497	0.0144
5	0.0497	-0.0787	14	0.0119	0
6	-0.0773	0.0404	15	-0.017	0
7	-0.0941	0.4178	16	-0.0019	0
8	0.4208	-0.7589	17	0.0010	0

The biorthogonal Cohen-Daubechies-Feauveau:

$$g_0(n) = (-1)^{n+1} h_1(n), \qquad g_1(n) = (-1)^n h_0(n)$$

Decompose the test image by wavelets to 3 levels, truncate the wavelet coefficients to 0 below some threshold. And reconstruct image from the left coefficients. Display the wavelet transforms of the image, the reconstructed images and the difference images.

(Consult Chapter 7 for more technique details).

Wavelet Family Short Name	Wavelet Family Name
'haar'	Haar wavelet
'db'	Daubechies wavelets
'sym'	Symlets
'coif'	Coiflets
'bior'	Biorthogonal wavelets
'rbio'	Reverse biorthogonal wavelets
'meyr'	Meyer wavelet
'dmey'	Discrete approximation of Meyer wavelet
'gaus'	Gaussian wavelets
'mexh'	Mexican hat wavelet (also known as the Ricker wavelet)
'morl'	Morlet wavelet
'cgau'	Complex Gaussian wavelets
'shan'	Shannon wavelets
'fbsp'	Frequency B-Spline wavelets
'cmor'	Complex Morlet wavelets
'fk'	Fejer-Korovkin wavelets

Problem solution 2

Haar(step=3,6,9)

MATLAB

```
meth   = 'ezw';   % Method name
wname  = 'haar';  % Wavelet name

img = imread('lenna.tif');
[M,N] = size(img);
all_in = zeros(3*M,3*N);

for k=1:3
    [CR,BPP] = wcompress('c',X,'mask.wtc',meth,'maxloop', k*3,'wname',wname);
    img_rec = wcompress('u','mask.wtc');
    img_diff=imsubtract(img,uint8(img_rec));
    all_in((k-1)*M+1:(k-1)*M+M,1:N*3)=[img,img_rec,img_diff];
end

imshow(mat2gray(all_in));
```

Daubechies(step=3,6,9)

MATLAB

```matlab
meth   = 'ezw';    % Method name
wname  = 'db8';  % Wavelet name

img = imread('lenna.tif');
[M,N] = size(img);
all_in = zeros(3*M,3*N);

for k=1:3
    [CR,BPP] = wcompress('c',X,'mask.wtc',meth,'maxloop', k*3,'wname',wname);
    img_rec = wcompress('u','mask.wtc');
    img_diff=imsubtract(img,uint8(img_rec));
    all_in((k-1)*M+1:(k-1)*M+M,1:N*3)=[img,img_rec,img_diff];
end

imshow(mat2gray(all_in));
```

Symlet(step=3,6,9)

MATLAB

```
meth   = 'ezw';   % Method name
wname  = 'sym8';  % Wavelet name

img = imread('lenna.tif');
[M,N] = size(img);
all_in = zeros(3*M,3*N);

for k=1:3
    [CR,BPP] = wcompress('c',X,'mask.wtc',meth,'maxloop', k*3,'wname',wname);
    img_rec = wcompress('u','mask.wtc');
    img_diff=imsubtract(img,uint8(img_rec));
    all_in((k-1)*M+1:(k-1)*M+M,1:N*3)=[img,img_rec,img_diff];
end

imshow(mat2gray(all_in));
```

biorthogonal Cohen-Daubechies-Feauveau(step=3,6,9)

waveletcdf97.m

MATLAB

```matlab
function X = waveletcdf97(X, Level)
%WAVELETCDF97  Cohen-Daubechies-Feauveau 9/7 wavelet transform.
%   Y = WAVELETCDF97(X, L) decomposes X with L stages of the
%   Cohen-Daubechies-Feauveau (CDF) 9/7 wavelet.  For the
%   inverse transform, WAVELETCDF97(X, -L) inverts L stages.
%   Filter boundary handling is half-sample symmetric.
```

```matlab
%
%   X may be of any size; it need not have size divisible by 2^L.
%   For example, if X has length 9, one stage of decomposition
%   produces a lowpass subband of length 5 and a highpass subband
%   of length 4.  Transforms of any length have perfect
%   reconstruction (exact inversion).
%
%   If X is a matrix, WAVELETCDF97 performs a (tensor) 2D wavelet
%   transform.  If X has three dimensions, the 2D transform is
%   applied along the first two dimensions.
%
%   Example:
%   Y = waveletcdf97(X, 5);    % Transform image X using 5 stages
%   R = waveletcdf97(Y, -5);   % Reconstruct from Y

% Pascal Getreuer 2004-2006

if nargin < 2, error('Not enough input arguments.'); end
if ndims(X) > 3, error('Input must be a 2D or 3D array.'); end
if any(size(Level) ~= 1), error('Invalid transform level.'); end

N1 = size(X,1);
N2 = size(X,2);

% Lifting scheme filter coefficients for CDF 9/7
LiftFilter = [-1.5861343420693648,-0.0529801185718856,0.8829110755411875,0.44350685205
ScaleFactor = 1.1496043988602418;

S1 = LiftFilter(1);
S2 = LiftFilter(2);
S3 = LiftFilter(3);
ExtrapolateOdd = -2*[S1*S2*S3,S2*S3,S1+S3+3*S1*S2*S3]/(1+2*S2*S3);

LiftFilter = LiftFilter([1,1],:);

if Level >= 0   % Forward transform
   for k = 1:Level
      M1 = ceil(N1/2);
      M2 = ceil(N2/2);

      %%% Transform along columns %%%
      if N1 > 1
         RightShift = [2:M1,M1];
         X0 = X(1:2:N1,1:N2,:);

         % Apply lifting stages
         if rem(N1,2)
            X1 = [X(2:2:N1,1:N2,:);X0(M1-1,:,:)*ExtrapolateOdd(1)...
```

```
                  + X(N1-1,1:N2,:)*ExtrapolateOdd(2)...
                  + X0(M1,:,:)*ExtrapolateOdd(3)]...
               + filter(LiftFilter(:,1),1,X0(RightShift,:,:),...
               X0(1,:,:)*LiftFilter(1,1),1);
         else
            X1 = X(2:2:N1,1:N2,:) ...
               + filter(LiftFilter(:,1),1,X0(RightShift,:,:),...
               X0(1,:,:)*LiftFilter(1,1),1);
         end

         X0 = X0 + filter(LiftFilter(:,2),1,...
            X1,X1(1,:,:)*LiftFilter(1,2),1);
         X1 = X1 + filter(LiftFilter(:,3),1,...
            X0(RightShift,:,:),X0(1,:,:)*LiftFilter(1,3),1);
         X0 = X0 + filter(LiftFilter(:,4),1,...
            X1,X1(1,:,:)*LiftFilter(1,4),1);

         if rem(N1,2)
            X1(M1,:,:) = [];
         end

         X(1:N1,1:N2,:) = [X0*ScaleFactor;X1/ScaleFactor];
      end

      %%% Transform along rows %%%
      if N2 > 1
         RightShift = [2:M2,M2];
         X0 = permute(X(1:N1,1:2:N2,:),[2,1,3]);

         % Apply lifting stages
         if rem(N2,2)
            X1 = permute([X(1:N1,2:2:N2,:),X(1:N1,N2-2,:)*ExtrapolateOdd(1)...
                  + X(1:N1,N2-1,:)*ExtrapolateOdd(2) ...
                  + X(1:N1,N2,:)*ExtrapolateOdd(3)],[2,1,3])...
               + filter(LiftFilter(:,1),1,X0(RightShift,:,:),...
               X0(1,:,:)*LiftFilter(1,1),1);
         else
            X1 = permute(X(1:N1,2:2:N2,:),[2,1,3]) ...
               + filter(LiftFilter(:,1),1,X0(RightShift,:,:),...
               X0(1,:,:)*LiftFilter(1,1),1);
         end

         X0 = X0 + filter(LiftFilter(:,2),1,...
            X1,X1(1,:,:)*LiftFilter(1,2),1);
         X1 = X1 + filter(LiftFilter(:,3),1,...
            X0(RightShift,:,:),X0(1,:,:)*LiftFilter(1,3),1);
         X0 = X0 + filter(LiftFilter(:,4),1,...
            X1,X1(1,:,:)*LiftFilter(1,4),1);
```

```
            if rem(N2,2)
               X1(M2,:,:) = [];
            end

            X(1:N1,1:N2,:) = permute([X0*ScaleFactor;X1/ScaleFactor],[2,1,3]);
         end

         N1 = M1;
         N2 = M2;
      end
   else           % Inverse transform
      for k = 1+Level:0
         M1 = ceil(N1*pow2(k));
         M2 = ceil(N2*pow2(k));

         %%% Inverse transform along rows %%%
         if M2 > 1
            Q = ceil(M2/2);
            RightShift = [2:Q,Q];
            X1 = permute(X(1:M1,Q+1:M2,:)*ScaleFactor,[2,1,3]);

            if rem(M2,2)
               X1(Q,1,1) = 0;
            end

            % Undo lifting stages
            X0 = permute(X(1:M1,1:Q,:)/ScaleFactor,[2,1,3]) ...
               - filter(LiftFilter(:,4),1,X1,X1(1,:,:)*LiftFilter(1,4),1);
            X1 = X1 - filter(LiftFilter(:,3),1,X0(RightShift,:,:),...
               X0(1,:,:)*LiftFilter(1,3),1);
            X0 = X0 - filter(LiftFilter(:,2),1,X1,...
               X1(1,:,:)*LiftFilter(1,2),1);
            X1 = X1 - filter(LiftFilter(:,1),1,X0(RightShift,:,:),...
               X0(1,:,:)*LiftFilter(1,1),1);

            if rem(M2,2)
               X1(Q,:,:) = [];
            end

            X(1:M1,[1:2:M2,2:2:M2],:) = permute([X0;X1],[2,1,3]);
         end

         %%% Inverse transform along columns %%%
         if M1 > 1
            Q = ceil(M1/2);
            RightShift = [2:Q,Q];
            X1 = X(Q+1:M1,1:M2,:)*ScaleFactor;
```

```
        if rem(M1,2)
           X1(Q,1,1) = 0;
        end

        % Undo lifting stages
        X0 = X(1:Q,1:M2,:)/ScaleFactor ...
           - filter(LiftFilter(:,4),1,X1,X1(1,:,:)*LiftFilter(1,4),1);
        X1 = X1 - filter(LiftFilter(:,3),1,X0(RightShift,:,:),...
           X0(1,:,:)*LiftFilter(1,3),1);
        X0 = X0 - filter(LiftFilter(:,2),1,X1,...
           X1(1,:,:)*LiftFilter(1,2),1);
        X1 = X1 - filter(LiftFilter(:,1),1,X0(RightShift,:,:),...
           X0(1,:,:)*LiftFilter(1,1),1);

        if rem(M1,2)
           X1(Q,:,:) = [];
        end

        X([1:2:M1,2:2:M1],1:M2,:) = [X0;X1];
      end
   end
end
```

main.m

MATLAB

```
img = imread('lenna.tif');
[M,N] = size(img);
all_in = zeros(3*M,3*N);

for k=1:3
    tran_img = waveletcdf97(double(img),k*3);
    tran_img(tran_img<1/40);
    img_rec = waveletcdf97(tran_img(tran_img>1/40),-k*3);
    img_diff=imsubtract(img,uint8(img_rec));
    all_in((k-1)*M+1:(k-1)*M+M,1:N*3)=[img,img_rec,img_diff];
end

imshow(mat2gray(all_in));
```