

Issued: [February 14, 2006](#)

Problem Set 3 Solutions

Due: February 17, 2006

Solution to **Problem 1: Is it Over-Compressed or is it Modern Art?**

The following MATLAB code solves parts a to e³:

```
% Perform initialization as indicated in the problem set.
load indemos vertigo;
vertigo=double(vertigo);
colormap('gray');

% Since many figures will be produced by this script, we use meaningful labels.
set(gcf,'NumberTitle','off','Name','Vertigo'); imshow(vertigo,[0 255]);

% Implement the compression scheme detailed in the problem set.
encoded=blkproc(vertigo,[8 8],'dct2');
encoded(abs(encoded)<10)=0;
decoded=round(blkproc(encoded,[8 8],'idct2'));

% Provide the error value to check against the expected value from the set.
sprintf('With cutoff=10, the mean squared error is %.4f', ...
mean2((vertigo - decoded).^2))
```

The mean squared error you should have obtained is 10.2970. The next piece of code produces the graph of file size versus mean squared error.

```
% Initialize the vectors that will store the data for the graph.
x=[];
y=[];

% We need only encode the image once. After that, since we will be steadily
% increasing the threshold, we need to reconvert again more because we will be
% simply zeroing-out more elements with each iteration through the for loop
% (there is no reason to recover all the original elements and start from scratch
% each time through the loop; we can progressively drop more and more data).
encoded=blkproc(vertigo,[8 8],'dct2');

% Now we begin to collect data for the graph.
for cutoff=0:4:100,
    encoded(abs(encoded)<cutoff)=0;
    decoded=round(blkproc(encoded,[8 8],'idct2'));
```

³MATLAB 6.5 was used to compose these solutions. MATLAB 7 has changed slightly the image vertigo; therefore, if you used MATLAB 7, your numbers will be different. The trend though should be the same

```

% We will simply append to the vectors each time through this loop.
x=[x,nnz(encoded)];
y=[y,mean2((vertigo - decoded).^2)];

% The next three lines can be commented out if they are not desired. They
% will produce a new window, label it, and print a representation of the
% newly decoded image for each cutoff threshold. This is for comparison
% with the original image to answer the question in the set that asks at
% which point the difference between the original image and the compressed
% image becomes perceptible to the human eye.
figure;
set(gcf,'NumberTitle','off','Name',sprintf('cutoff=%d',cutoff));
imshow(decoded,[0 255]);
end

% Now, plot the graph with a smooth curve and boxes around all the actual data points.
figure;
set(gcf,'NumberTitle','off','Name','Graph for Problem 1');
plot(x,y,'s-')
title('Comparison of File Size and Image Error');
xlabel('Non-zero matrix values (number of bytes to store)');
ylabel('Mean squared error');

```

You should have gotten something remotely resembling the graph in Figure 3-1. As you can see, there is a point where the MSE increases exponentially giving a quantitative value to the degradation of the reconstructed picture. Medical applications such as in X-rays tend to discourage the use of JPEG or similar lossy compression algorithms for saving images due to chances of distortion leading to an incorrect diagnosis.

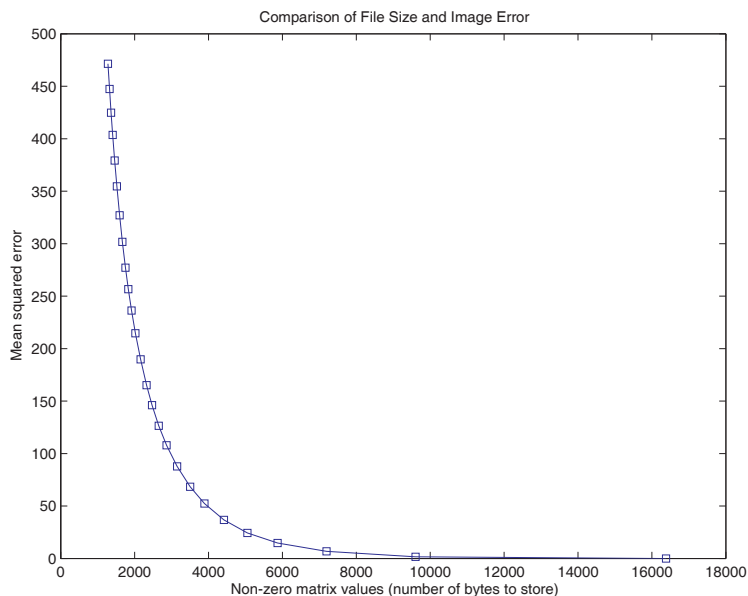


Figure 3-1: Comparison of File Size and Image Error

Solution to Problem 2: Compression is Fun and Easy

Solution to Problem 2, part a.

Table 3-1 represents the LZW analysis of the phrase “yubba dubba dubba dubba dubba doo.” The resulting data stream is:

```
2 79 75 62 62 61 20 64 81 83 85 87 84 86 82 91 90 85 6F 6F 3
```

This is 21 bytes long, whereas the original message was 35 bytes long (including the start and stop control characters), which amounts to 60% compression. If we count bits, the original message could have been sent in 7 bits per character (total 245 bits) whereas the LZW code requires 8 bits per character (total 168 bits) so the compression is 68.6%. Apparently, this is a very short example exaggeratedly contrived with too many repetitions to make the dictionary grow quickly. For a sizeable selection of average English text, LZW typically yields 50% compression.

Input	New dictionary entry	Transmission	New dictionary entry	Output
- -	- -	02 (start)	- -	-
79 y	- -	- -	- -	-
75 u	80 yu	79 y	- -	y
62 b	81 ub	75 u	80 yu	u
62 b	82 bb	62 b	81 ub	b
61 a	83 ba	62 b	82 bb	b
20	84 a(space)	61 a	83 ba	a
64 d	85 (space)d	20 (space)	84 a(space)	(space)
75 u	86 du	64 d	85 (space)d	d
62 b	- -	- -	- -	-
62 b	87 ubb	81 ub	86 du	ub
61 a	- -	- -	- -	-
20	88 ba(space)	83 ba	87 ubb	ba
64 d	- -	- -	- -	-
75 u	89 (space)du	85 (space)d	88 ba(space)	(space)d
62 b	- -	- -	- -	du
62 b	- -	- -	- -	-
61 a	90 ubba	87 ubb	89 (space)du	ubb
20	- -	- -	- -	-
64 d	91 a(space)d	84 a(space)	90 ubba	a(space)
75 u	- -	- -	- -	-
62 b	92 dub	86 du	91 a(space)d	du
62 b	- -	- -	- -	-
61 a	93 bba	82 bb	92 dub	bb
20	- -	- -	- -	-
64 d	- -	- -	- -	-
75 u	94 a(space)du	91 a(space)d	93 bba	a(space)d
62 b	- -	- -	- -	-
62 b	- -	- -	- -	-
61 a	- -	- -	- -	-
20	95 ubba(space)	90 ubba	94 a(space)du	ubba
64 d	- -	- -	- -	-
6F o	96 (space)do	85 (space)d	95 ubba(space)	(space)d
6F o	97 oo	6F o	96 (space)do	o
- -	- -	6F o	97 oo	o -
-	-	03 (stop)	- -	-

Table 3-1: Solution to Problem 2, part a

Solution to Problem 2, part b.

The following two m-files implement LZW encoder and decoder.

Beginning of file LZWencoder.m

```
function [transmissionHEX]=LZWencoder(message);
%usage [Transmission]=LZWencoder(message)
%Function to encode a message with LZW, the message is expected as a matrix
%of strings, each row representing a character, or command (256, 257)
if size(message,1)<2
    %wrong input type
    disp('Error the input is expected to be a string array');
end
%loop through input character by character.
%setup initial dictionary
idictionary=cellstr(char(0:127)');
%matlab does not properly convert spaces with the char, so we shall correct
%character 32 (which will appear in position 33 because matlab starts to
%count with 1)
idictionary{33}=' ';
new_entry=[]; %to store the cumulative string
last_code=[]; %to store the last code found
transmission={'02'}; %transmit start of message.
for i=1:size(message,1)
    new_character=deblank(message(i,:));
    %to account for matlab padding we need to deblank the message
    %this will eventually produce aproblem with spaces,
    if isempty(new_character)
        new_character=' ';
    end;
    %Accumulate string
    new_entry=[new_entry,new_character];
    %check if the new entry exists in the dictionary
    match_Q=MatchDictionaryEntry(new_entry,idictionary);
    if isempty(match_Q)
        %if it does not exist, we add it
        idictionary{end+1}=new_entry;
        %we transmit the code for the last matched string
        last_match=MatchDictionaryEntry(new_entry(1:(end-1)),idictionary);
        %transmit the last_match
        transmission{end+1}=num2str(last_match-1);
        %set the new_entry to the last character read
        new_entry=new_character;
    else
        %set transmission to nil (this is mostly for readability)
        transmission{end+1}='-';
    end
end
%flush the content of the new_entry variable
last_match=MatchDictionaryEntry(new_entry,idictionary);
transmission{end+1}=num2str(last_match-1);
%transmit end of message
```

```

transmission{end+1}='03';idicexpanded{end+1,1}=NaN;idicexpanded{end,2}='-';
End of file LZWencoder.m
Beginning of file LZWdecoder.m

function [output]=LZWdecoder(transmission);
%usage [output]=LZWdecoder(transmission)
%Function to decode a message compressed with LZWencoder
if ~iscellstr(transmission)
    %wrong input type
    disp('Error the input is expected to be cell string');
end
i=1;
output=[];last_output=[];
tr=char(transmission);
nothing=strmatch('-',tr);
transmission(nothing)={'0'};
%figure out whether input is hexadecimal, if it is, then convert to decimal numbers
if any(ismember('ABCDEF',char(transmission)))
    %convert to decimal
    trans=hex2dec(char(transmission));
else
    %convert to numbers
    trans=str2num(char(transmission));
end
trans(nothing)=NaN;
%loop through transmission until end of message character
while trans(i)~=3;
    next_code=trans(i);
    if ~isnan(next_code(1)) %characters '-' were added for readability during encoding
        %next_code=str2num(next_code);
        if next_code==2
            %create dictionary
            odictionary=cellstr(char(0:127)');
            odictionary{33}=' ';
            output{1}='-';
        else
            if next_code>(length(odictionary)-1)
                %sometimes the code does not exist. then we need to acknowledge
                %that this can only happen when the new codeword is in fact the
                %previous one with the first character of the old codeword
                %appended
                this_output=[last_output,last_output(1)];
                %update the dictionary
                odictionary{end+1}=this_output;
            else
                this_output=odictionary{next_code+1};
                %update the dictionary
                if ~isempty(last_output)
                    odictionary{end+1}=[last_output,this_output(1)];
                end
            end
        end
        %now just output the expected code(beware of the fact that matlab

```

```
        %arrays start with one and our code with zero)
        output{end+1}=this_output;
        last_output=output{end};
    end
else
    %there was no input output '-'
    output{end+1}='-';
end
i=i+1;
end
output{end+1}='-';

End of file LZWdecoder.m
```