EECE-5626 (IP&PR) : Homework-5 Solutions

Default Plot Parameters:

```
set(0,'defaultfigurepaperunits','points','defaultfigureunits','points');
set(0,'defaultaxesfontsize',10);
set(0,'defaultaxestitlefontsize',1.4,'defaultaxeslabelfontsize',1.2);
```

Table of Contents

Problem-5.1: Data and Image Quantization 2
(a) DIP4E Problem 8.3 (Page 688) 2
(b) Image Quantization 2
Problem-5.2: Huffman Coding 4
(a) Huffman Coding on Image Fragment 4
(i) Compression 4
(ii) Unique Codes 4
(iii) Compression Ratio 4
(b) Huffman Decoding 5
Problem-5.3: DIP4E MATLAB Project 8.3 (Page 691) 5
(a) MATLAB function compare4e 5
(b) RMSE between uncodede and Coded Images 5
Problem-5.4: DIP4E MATLAB Project 8.7 (Page 691) - Modified 6
(a) Compression of lena.tif Image 6
(b) Computation of Compression Ratio 6
(c) Part (c) of Project 8.7 6
Decoding of Compressed Image 8
(d) Part (d) of Project 8.7 8
Problem-5.5: DIPUM3E MATLAB Project 9.1 (Page 580) 9
(c) Function redundancy 9
(d) Computation of RDR 9
Problem-5.6: Erosion and Dilation 10
(a) DIP4E Problem 9.2 (b) only (Page-750) 10
(b) DIP4E Problem 9.6 (c) only (Page-750) 11
Problem-5.7: DIP4E 9.9 (b) only (Page 751) 12
Problem-5.8 Grayscale Erosion and Dilation 12
(a) Hand Calculations 12
(b) MATLAB Verification 13
Problem-5.9 DIP4E MATLAB Project 9.5 (a) & (c) only (Page 757) 13
(a) morphoBoundary4e Function 13
(c) Processing of testpattern512-binary.tif Image 14
Problem-5.10 DIPUM3E Project 10.8 (Page 632) 16
(b) Morphological Smoothing/Sharpening of cygnusloop.tif Image 16

Problem-5.1: Data and Image Quantization

(a) DIP4E Problem 8.3 (Page 688)

Consider an 8-pixel row of intensity data, $\{109, 140, 134, 245, 170, 175, 56, 100\}$. It is uniformly quantized (truncation and not rounding) with 3-bit accuracy. That is, discard the last five bits. Compute the rms (root-mean-squared) error and rms signal-to-noise ratio, in decibels, for the quantized data.

Solution: Let

$$f(n) = \{109, 140, 134, 245, 170, 175, 56, 100\}.$$

Since the maximum value in $f(n)$ is $245,$ we will need 8 bits to represent integer-valued data in binary form. This representation is

$$f_{\text{bin}}(n) = \begin{bmatrix} 01101101, & 10001100, & 10000110, & 11110101, \\ 10101010, & 10101111, & 00111000 & ,01100100 \end{bmatrix}.$$

After truncation to 3 bits we obtain the quantized representation as

$$f_{\text{bin\_quantized}}(n) = [011, 100, 100, 111, 101, 101.001, 011].$$

To obtain pixel decimal values after quantization, we first convert the 3-bit representation into decimal values and then multiply each value by 2^5 since five bits were discarded (the effect is to replace those discarded bits by zeros). The resulting values are

$$\hat{f}(n) = \{96, 128, 128, 224, 160, 160, 32, 96\}.$$

The quantization error values are

$$e(n) = \hat{f}(n) - f(n) = \{-13, -12, -6, -21, -10, -15, -24, -4\}.$$

The rms error is then given by

$$e_{\text{rms}} = \sqrt{\frac{1}{8}\sum_{n=0}^{7} e^2(n)} = 14.6074$$

or about 15 intensity levels. The rms signal-to-noise ratio, in decibels, is given by

$$\text{SNR}_{\text{rms}} = \sqrt{\frac{\sum_{n=0}^{7} f^2(n)}{\sum_{n=0}^{7} e^2(n)}} = 10.3292 \equiv 20\log_{10}(10.3292) = 20.2813 \text{ dB}$$

or about 20 dB.

(b) Image Quantization

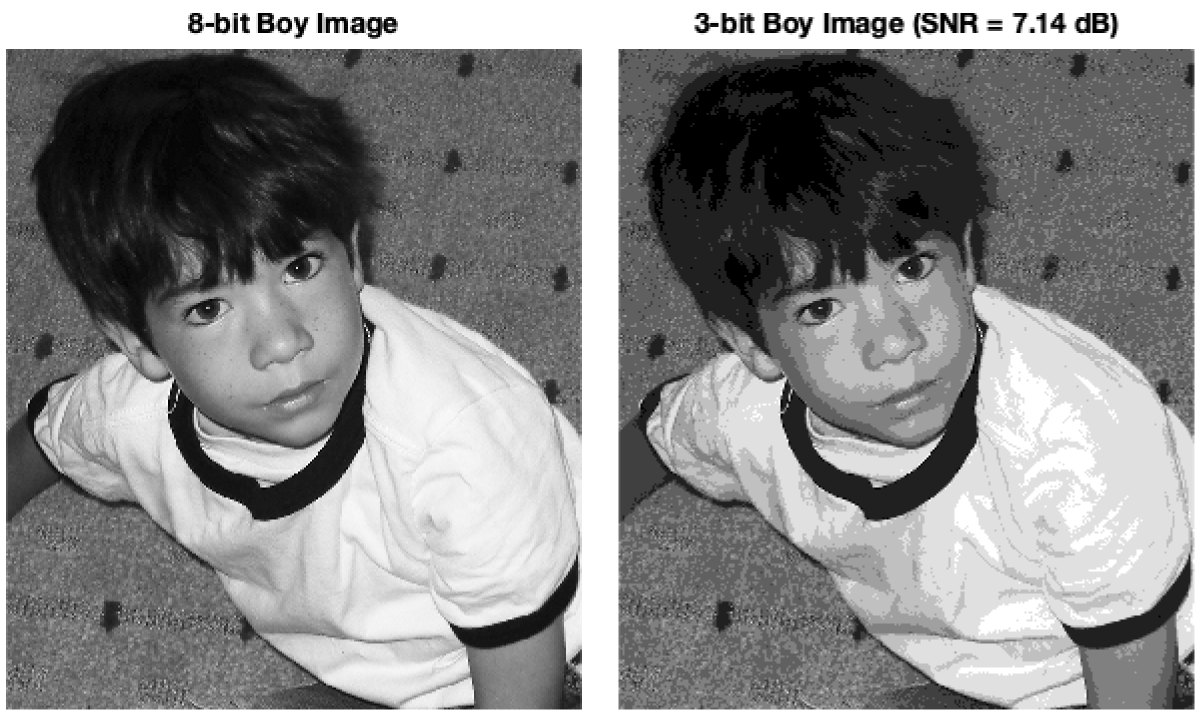
Repeat part (a) using the **boy.tif** image which is an 8-bit image. Display the original and quantized images side by side.

Solution:

```
clc; close all; clear;
f = imread('../artfiles/boy.tif'); [M,N] = size(f);
b = 8; % Original image bit depth
bQ = 3; % Qunatised image bit depth
qStep = 2^(b-bQ); % Quantization step (in integer value)
fQ = qStep*floor(f/qStep); % Quantized image
fe = fQ-f; fe = fe(:); % Error image
eRMS = sqrt(average(fe.^2)); % Root mean square value
SNRrms = round(20*log10(sqrt(average(f(:).^2))/eRMS),2);
fprintf('RMS Signal-to-Noise Ratio is %g dB.',SNRrms);
```

```
RMS Signal-to-Noise Ratio is 7.14 dB.
```

```
IW = 4; IH = (M/N)*IW; FW = 2*IW+0.25; FH = IH+0.25;
figure('Position',[0,0,FW,FH]*72);
subplot('Position',[0,0,4/FW,IH/FH]);
imshow(f); title('8-bit Boy Image');
subplot('Position',[4.25/FW,0,4/FW,IH/FH]); imshow(fQ);
Ttext = ['3-bit Boy Image (SNR = ',num2str(SNRrms),' dB)'];
title(Ttext);
```



Problem-5.2: Huffman Coding

This problem has two independent parts.

(a) Huffman Coding on Image Fragment

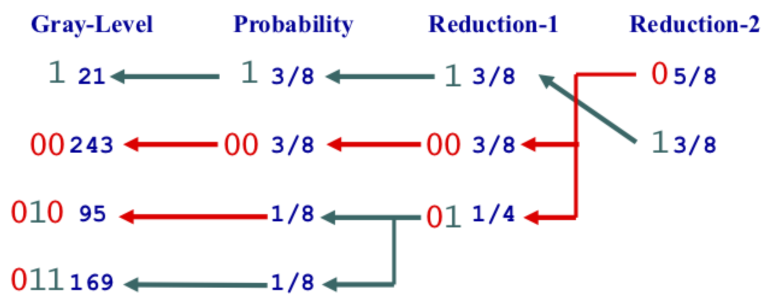
Consider the 4×8 size 8-bit image from Slide-13 of Topic-10 Powerpoint, which is given below.

$$\begin{matrix} 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \\ 21 & 21 & 21 & 95 & 169 & 243 & 243 & 243 \end{matrix}$$

(i) Compression

Compress the image using Huffman coding, that is, provide an encoded bit pattern (ignore that we may need end-of-line markers after every row). Provide either a row-wise bit arrangement or a column-wise bit arrangement.

Solution: The Huffman code for the set of symbols $\{21, 95, 169, 243\}$ with corresponding probabilities $\left\{\frac{3}{8}, \frac{3}{8}, \frac{1}{8}, \frac{1}{8}\right\}$ was developed in a class lecture. One possible codeword assignment is shown below.



The row-wise coded bit pattern is

```
111010011000000111010011000000111010011000000111010011000000
```

while the column-wise coded bit pattern is

```
111111111111010010010010011011011011000000000000000000000000.
```

(ii) Unique Codes

How many unique Huffman codes are possible for this four-symbol source?

Solution: The Huffman code assignment shown above is not unique for the given set of symbols and probabilities. We obtain a complementary assignment by switching `0` $\rightleftarrows$ `1`. Furthermore, we can switch assignments of symbols `21` and `243` since both have the same probability. Likewise, we can switch assignments of symbols `95` and `169`. Thus we have $2 \times 2 \times 2 = 8$ different codeword assignments. However, there are only two unique Huffman codes; one given above and the other with complementary assignments.

(iii) Compression Ratio

Compute the compression ratio achieved and discuss the effectiveness of the Huffman coding on this image.

Solution: The total number of bits assigned to the encoded image is $12(1)+12(2)+4(3)+4(3)=60.$ The uncoded image requires $8\times 32=256$ bits. Hence the compression ratio is $\mathrm{CR}=256/60=4.2667.$ The average bit-rate for Huffman coding is $R=60/32=1.875$ bits/pixel while the entropy of the source is $H=1.81$ bits/pixel. Thus Huffman code is very effective.

(b) Huffman Decoding

Using the Huffman code in Fig. 8.8 (in DIP4E), decode the encoded string

`010011000001010101011`.

Show all of your work to get full credit.

Solution: Using uniquely decodable code in Fig 8.8, we can parse the given string as follows:

```
0100 1 1 00 00 01010 1 01011 1.
```

Hence the decoded symbols are $\{a_4, a_2, a_2, a_6, a_6, a_3, a_2, a_5, a_2\}$.

Problem-5.3: DIP4E MATLAB Project 8.3 (Page 691)

(a) MATLAB function `compare4e`

Solution: Insert the code of your function below after the comments.

```
function rmse = compare4e(f1,f2)
%function rmse = compare4e(f1,f2)
%   RMSE = COMPARE4E(F, FA) returns the root-mean-square error
%   between inputs F and FA.

% Copyright 2017, R. C. Gonzalez & R. E. Woods

% Compute the root-mean-square error.
e = double(f1) - double(f2);
[m,n] = size(e);
rmse = sqrt(sum(e(:) .^ 2) / (m * n));

end
```

(b) RMSE between uncodede and Coded Images

Solution: The rms error between the images lena.tif and lena.jpg is computed using the following MATLAB script.

```
clc; close all; clear;
f1 = imread('../artfiles/lena.tif');
f2 = imread('../artfiles/lena.jpg');
RMSE = compare4e(f1,f2);
fprintf('RMSE between lena.tif and lena,jpg is %g.',round(RMSE,2));
```

```
RMSE between lena.tif and lena,jpg is 18.78.
```

Problem-5.4: DIP4E MATLAB Project 8.7 (Page 691) - Modified

JPEG Compression

The function **`c = im2jpeg4e(f,quality,bits)`**, to approximate the baseline JPEG image encoding process, is available as a solution to part (a) in Student MATLAB Project Solutions.

(a) Compression of `lena.tif` Image

Using this function with the quality factor equal to 2, compress the image **`lena.tif`**. From the resulting encoded structure, determine the number of bits in the encoded image.

Solution:

```
clc; close all; clear;
f = imread('../artfiles/lena.tif');
fjpg = im2jpeg4e(f,2,8);
```

Note that **`fjpg`** is a structure containing many fields. To obtain the number of bits in the encoded image let us look at the field **`fjpg.huffman`**:

```
fjpg.huffman
```

```
ans = struct with fields:
    size: [32125 1]
     min: 32743
    hist: [6 42 178 168 119 83 73 78 89 99 87 105 81 161 155 112 138 159 186 268 349 438 606 1274 4612 10558 4301 12
    code: [1×6991 uint16]
```

The field, **`fjpg.huffman.code`**, reports that there are 6991 **`uint16`** words. Since each **`uint16`** word is 16 bits, the total number of bits in the encoded image is $6991 \times 16 = 111856$.

(b) Computation of Compression Ratio

Compute the compression ratio using the **`compressionRatio4e`** function, which is also available as a solution to Problem 8.2 in Student MATLAB Project Solutions.

Solution:

```
CR = compressionRatio4e(f,fjpg)
```

```
CR = 18.5760
```

Thus the compression ratio is 18.5760.

(c) Part (c) of Project 8.7

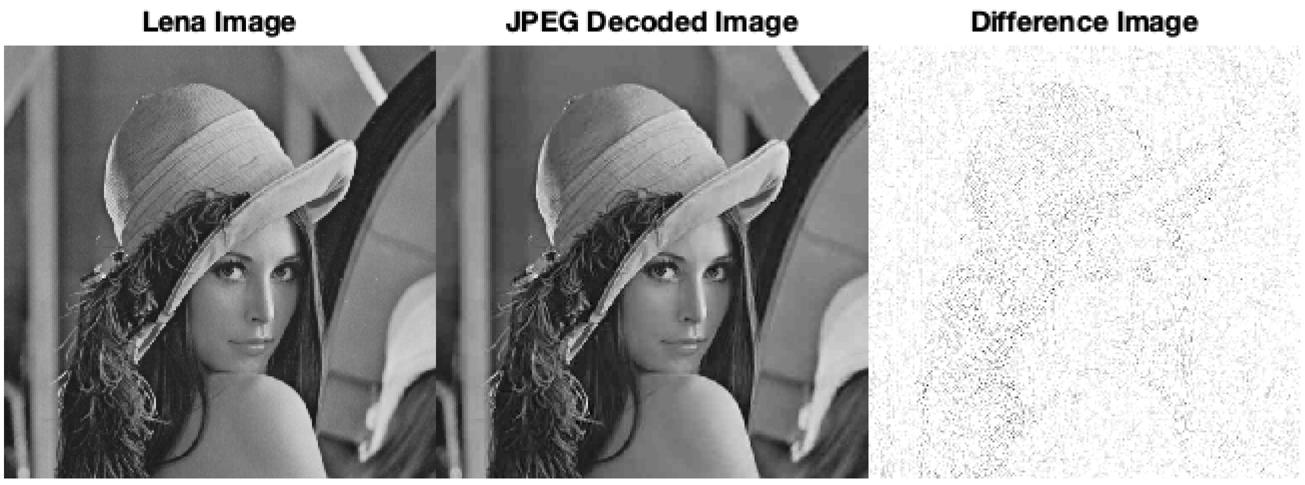
Solution: Insert the code of your **`jpeg2im4e`** function below after the comments.

```
function x = jpeg2im4e(y)
%JPEG2IM4E Decodes an IM2JPEG compressed image.
%   X = JPEG2IM4E(Y) decodes compressed image Y, generating
%   reconstructed approximation X.  Y is a structure generated by
%   IM2JPEG.
%
%   See also IM2JPEG.
%   Copyright 2002-2017 R. C. Gonzalez, R. E. Woods, and S. L. Eddins
%   From the book Digital Image Processing Using MATLAB, 2nd ed.,
%   Gatesmark Publishing, 2017.
%
%   Book web site: http://www.imageprocessingplace.com
%   Publisher web site: http://www.gatesmark.com/DIPUM2e.htm

error(nargchk(1, 1, nargin));             % Check input arguments
% JPEG normalizing array
m = [16 11  10  16  24  40  51  61
     12  12  14  19  26  58  60  55
     14  13  16  24  40  57  69  56
     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];
% Zig-zag reordering pattern
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];
% Compute inverse ordering
rev = order;
for k = 1:length(order)
   rev(k) = find(order == k);
end
% Get encoding quality
m = double(y.quality) / 100 * m;
% Get number of 8x8 blocks.
xb = double(y.numblocks);
% Get image size
sz = double(y.size);
xn = sz(2); % Get x columns.
xm = sz(1); % Get x rows.
% Obtain symbols using Huffman decode
x = huff2mat(y.huffman);
% Get end-of-block symbol
eob = max(x(:));
% Convert symbol vector into symbol blocks using EOB 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
% Restore order
z = z(rev, :);
% Convert symbol vectors into matrix blocks
x = col2im(z, [8 8], [xm xn], 'distinct');
% Unnormalize blocks and concatenate
```

```
        unnormjp = @(block_struct) (block_struct.data.*m);
        x = blockproc(x, [8 8], unnormjp);
        % Perform inverse 2D DCT and concatenate
        idct2jp = @(block_struct) idct2(block_struct.data);
        x = blockproc(x, [8 8], idct2jp);
        % Level shift blocks
        x = x + double(2^(y.bits - 1));
        % Convert to unint8 or uint16 using y.bits
        if y.bits <= 8
            x = uint8(x);
        else
            x = uint16(x);
        end

        end
```

Decoding of Compressed Image

```
fhat = jpeg2im4e(fjpg);
ferror = imsubtract(f,fhat);
figure('position',[1,1,9,3.25]*72);
subplot('position',[0,0,3/9,3/3.25]);
imshow(f); title('Lena Image');
subplot('position',[3/9,0,3/9,3/3.25]);
imshow(fhat); title('JPEG Decoded Image');
subplot('position',[6/9,0,3/9,3/3.25]);
imshow(imcomplement(ferror),[]); % for display
title('Difference Image');
```



(d) Part (d) of Project 8.7

Use function **`compare4e`** from Project 8.3 to compute the root-mean-squared error between image **`lena.tif`** and the decompressed image from part (c).

Solution: The \ml script is

```
RMSE = compare4e(f,fhat);
fprintf('RMS Error = %g\n',round(RMSE,2));
```

```
RMS Error = 4.93
```

Problem-5.5: DIPUM3E MATLAB Project 9.1 (Page 580)

Relative Data Redundancy (RDR)

(c) Function redundancy

Solution: Insert the code of your function below after the comments.

```
function [r] = redundancy(I1,I2)
%REDUNDANcY returns the relative data redundancy of I1 with respect to I2.
% Since R = (i1–i2)/i1, we can write R = (i1/i1) – (i2/i1) = 1 – i2/i1.
% Denoting cr = il/i2, we have R = 1 – 1/cr. Note that in these
% equations, i1 and i2 represent the number of bytes in input images I1
% and I2, respectively.

cr = imratio(I1,I2);
r = 1 - (1/cr);

end
```

(d) Computation of RDR

Use the function from (a) to find the relative data redundancy of image **`testpattern512.tif`** with respect to JPEG-encoded version of the image witha **`'quality'`** of 25.

Solution: MATLAB script.

```
clc; close all; clear;
% Read the input image and create a JPEG with a 'qua1ity' of 25, writing it
% to disk.

f = imread('../artfiles/testpattern512.tif');
imwrite(f,'testpattern512-25.jpg','quality',25);

% Use function 'redundancy' to compute the redundancy of the original 2-D
% image array with respect to the JPEG compressed image.
r = redundancy(imread('testpattern512-25.jpg'),'testpattern512-25.jpg')
```

```
r = 0.9015
```

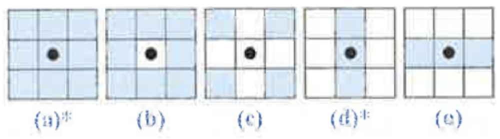
Thus, 90.15% of the data is redundant.

Problem-5.6: Erosion and Dilation

In the following parts, perform the stated operations by hand first and then verify using MATLAB.

(a) DIP4E Problem 9.2 (b) only (Page-750).

Sketch the results of eroding Fig. 9.3(a) with the structuring element (b) below. Assume a boundary of zeros.



Solution: Assuming a boundary of zeros, the eroded figure is shown below.

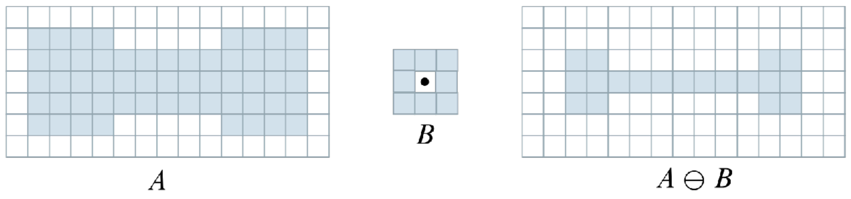


Figure 5.6(a)

MATLAB script:

```
clc; close all; clear;
%% Generate Figure 9.3(a) in the DIP4E Text
A = zeros(7,15); A(2:6,2:5) = 1; A(3:5,6:10) = 1; A(2:6,11:14) = 1;
disp('A = '); disp(A);
```

```
A =
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
     0     1     1     1     1     0     0     0     0     0     1     1     1     1     0
     0     1     1     1     1     1     1     1     1     1     1     1     1     1     0
     0     1     1     1     1     1     1     1     1     1     1     1     1     1     0
     0     1     1     1     1     1     1     1     1     1     1     1     1     1     0
     0     1     1     1     1     0     0     0     0     0     1     1     1     1     0
     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0
```

```
A = padarray(A,[1,1]); % Creates a boundary of 0s around A, 1 pixel wide
A = imbinarize(A); % Creates logical array
%% 9.2(b): Erode using B
B = strel('arbitrary',[1,1,1;1,0,1;1,1,1]);
disp('B = '); disp(B.Neighborhood);
```

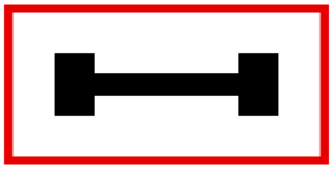
```
B =
   1   1   1
   1   0   1
   1   1   1
```

```
C = imerode(A,B); C = C(2:end-1,2:end-1); % Extract middle part of original size
disp('C = '); disp(C);
```

```
C =
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   1   1   0   0   0   0   0   0   0   1   1   0   0
```

```
    0    0    1    1    1    1    1    1    1    1    1    1    1    0    0
    0    0    1    1    0    0    0    0    0    0    0    1    1    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

```
figure('Position',[0,0,15/7+.1,1.1]*72,'color',[1,0,0]*0.9);
subplot('Position',[0.05/(15/7+0.1),0.05/1.1,(15/7)/(15/7+0.1),1/1.1]);
imshow(imcomplement(C));
```



Clearly, MATLAB result and the above image agrees with the Figure 5.6(a) above.

(b) DIP4E Problem 9.6 (c) only (Page-750).

Dilate Fig. 9.3(a) using the structuring element in figure (c) of Problem 9.2.

Solution: Assuming a boundary of zeros, the dilated figure is shown below. Assume a boundary of zeros.

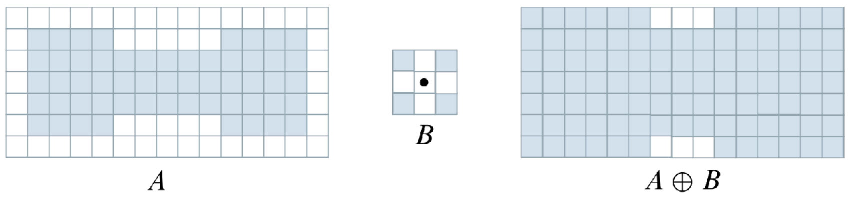


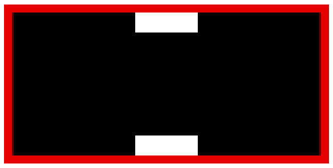
Figure 5.6(b)

MATLAB script:

```
B = strel('arbitrary',[1,0,1;0,0,0;1,0,1]);
C = imdilate(A,B); C = C(2:end-1,2:end-1); disp('C = '); disp((C));
```

```
C =
    1    1    1    1    1    1    0    0    0    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
    1    1    1    1    1    1    0    0    0    1    1    1    1    1    1
```

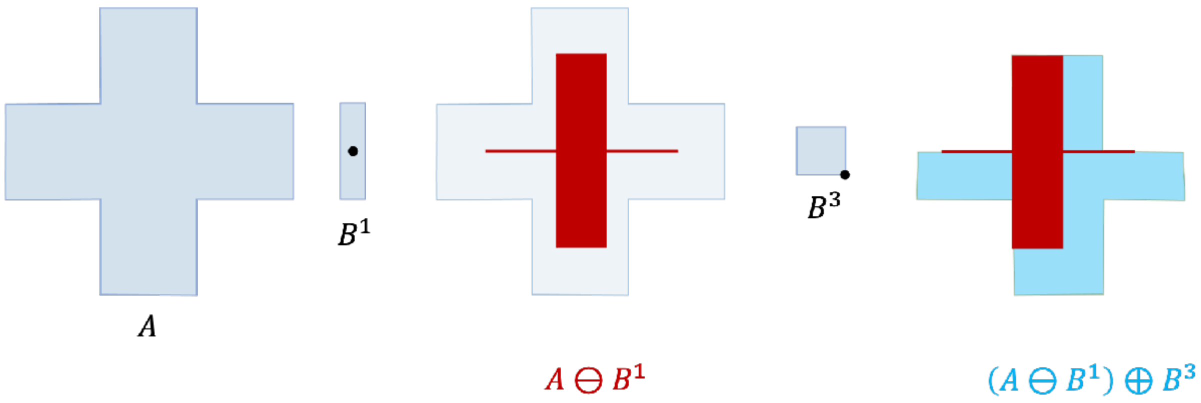
```
figure('Position',[0,0,15/7+.1,1.1]*72,'color',[1,0,0]*0.9);
subplot('Position',[0.05/(15/7+0.1),0.05/1.1,(15/7)/(15/7+0.1),1/1.1]);
imshow(imcomplement(C));
```



Clearly, MATLAB result and the above image agrees with the Figure 5.6(b) above.

Problem-5.7: DIP4E 9.9 (b) only (Page 751)

Solution: The result is shown below.



Problem-5.8 Grayscale Erosion and Dilation

Let A be the matrix

$$A = \begin{bmatrix} 35 & 1 & 6 & 26 & 19 & 24 \\ 3 & 32 & 7 & 21 & 23 & 25 \\ 31 & 9 & 2 & 22 & 27 & 20 \\ 8 & 28 & 33 & 17 & 10 & 15 \\ 30 & 5 & 34 & 12 & 14 & 16 \\ 4 & 36 & 29 & 13 & 18 & 11 \end{bmatrix}$$

(a) Hand Calculations

Using hand calculations, determine the grayscale erosion for $(A \ominus B)_{(1,1)}$ and $(A \ominus B)_{(2,2)}$ elements and dilation for $(A \oplus B)_{(3,4)}$ and $(A \oplus B)_{(5,2)}$ elements with the structuring element B given as

$$B = \begin{bmatrix} 5 & 20 & 5 \\ 20 & 5 & 20 \\ 5 & 20 & 5 \end{bmatrix}.$$

All details must be shown to get full credit.

Solution: The required hand calculations are given below:

$$(A \ominus B)_{(1,1)} = \min\left(\begin{bmatrix}35 & 1\\ 3 & 32\end{bmatrix} - \begin{bmatrix}5 & 20\\ 20 & 5\end{bmatrix}\right) = \min\left(\begin{bmatrix}30 & -19\\ -17 & 27\end{bmatrix}\right) = -19$$

$$(A \ominus B)_{(2,2)} = \min\left(\begin{bmatrix}35 & 1 & 6\\ 3 & 32 & 7\\ 31 & 9 & 2\end{bmatrix} - \begin{bmatrix}5 & 20 & 5\\ 20 & 5 & 20\\ 5 & 20 & 5\end{bmatrix}\right) = \min\left(\begin{bmatrix}30 & -19 & 1\\ -17 & 27 & -13\\ 26 & -11 & -3\end{bmatrix}\right) = -19$$

$$(A \oplus B)_{(3,4)} = \max\left(\begin{bmatrix}7 & 21 & 23\\ 2 & 22 & 27\\ 33 & 17 & 10\end{bmatrix} + \begin{bmatrix}5 & 20 & 5\\ 20 & 5 & 20\\ 5 & 20 & 5\end{bmatrix}\right) = \max\left(\begin{bmatrix}12 & 41 & 28\\ 22 & 27 & 47\\ 38 & 37 & 15\end{bmatrix}\right) = 47$$

$$(A \oplus B)_{(5,2)} = \max\left(\begin{bmatrix}8 & 28 & 33\\ 30 & 5 & 34\\ 4 & 36 & 29\end{bmatrix} + \begin{bmatrix}5 & 20 & 5\\ 20 & 5 & 20\\ 5 & 20 & 5\end{bmatrix}\right) = \max\left(\begin{bmatrix}13 & 48 & 38\\ 50 & 10 & 54\\ 9 & 56 & 34\end{bmatrix}\right) = = 56$$

(b) MATLAB Verification

Determine $A \ominus B$ and $A \oplus B$ using MATLAB and verify your calculations in (a).

```
clc; close all; clear;
A = magic(6); % The given matrix is a magic(6) matrix
B = strel('arbitrary',ones(3),[5,20,5;20,5,20;5,20,5]);
C = imerode(A,B); disp('C = '); disp(C);
```

```
C =
   -19   -14   -19   -14     3    -1
    -4   -19   -18   -13    -1     0
   -17   -18   -13   -18   -10    -5
     0   -15   -18   -10    -6   -10
   -16    -1   -15    -7   -10    -9
    -1   -16    -7    -8    -9    -4
```

```
D = imdilate(A,B); disp('D = '); disp(D);
```

```
D =
    40    55    46    41    46    45
    55    40    52    46    47    44
    37    52    53    47    43    47
    51    53    54    53    47    40
    41    56    53    54    38    35
    56    49    56    49    34    38
```

These results agree with our hand calculations for the respective elements in (a).

Problem-5.9 DIP4E MATLAB Project 9.5 (a) & (c) only (Page 757)

Boundary Extraction

(a) `morphoBoundary4e` Function

Write a function **BD = `morphoBoundary4e(I,B)`** that uses **`B`** to extract the boundary of the objects of foreground pixels in binary image **`I`**. If **`B`** is not included in the input argumet list, then it defaults to a 3×3 structuring elements of 1's.

Solution: Insert the code of your function below after the comments.

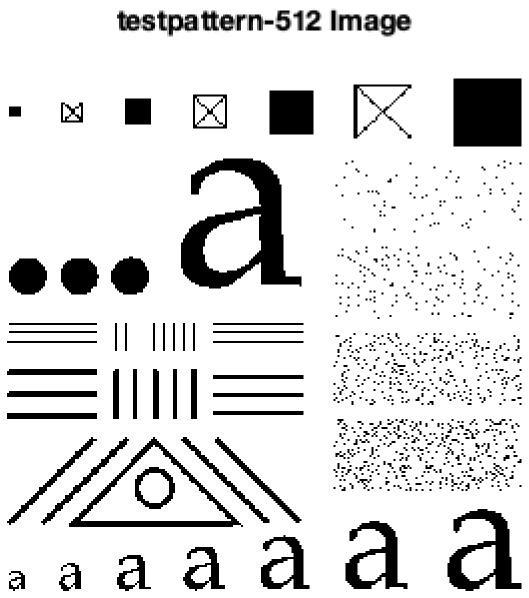
```
function BD = morphoBoundary4e(I,B)
%MORPHOBOUNDARY4E Boundary of objects in a binary image.
% BD = MORPHOBOUNDARY4E(I,B) computes the boundaries of all
% foreground pixels in binary image I using structuring element B.
% If B is not inc1uded in the input argument, it defauits to a
% 3-by—3 array of 1's. If it is, it must be a m—by-n array (m and n
% odd) of 1's.
%
% Copyright 2017, R. C. Gonza1ez & R. E. woods.

% Defauit.
if nargin == 1
    B = ones(3);
end

% We implement Eq. (9-18) in DIP4E, but using the set intersection
% equivaient of set differences(see Eq. (2-40) in DIP4E).

E = morphoErode4e(I,B);

BD = I & ~E;

end
```

(c) Processing of `testpattern512-binary.tif` Image

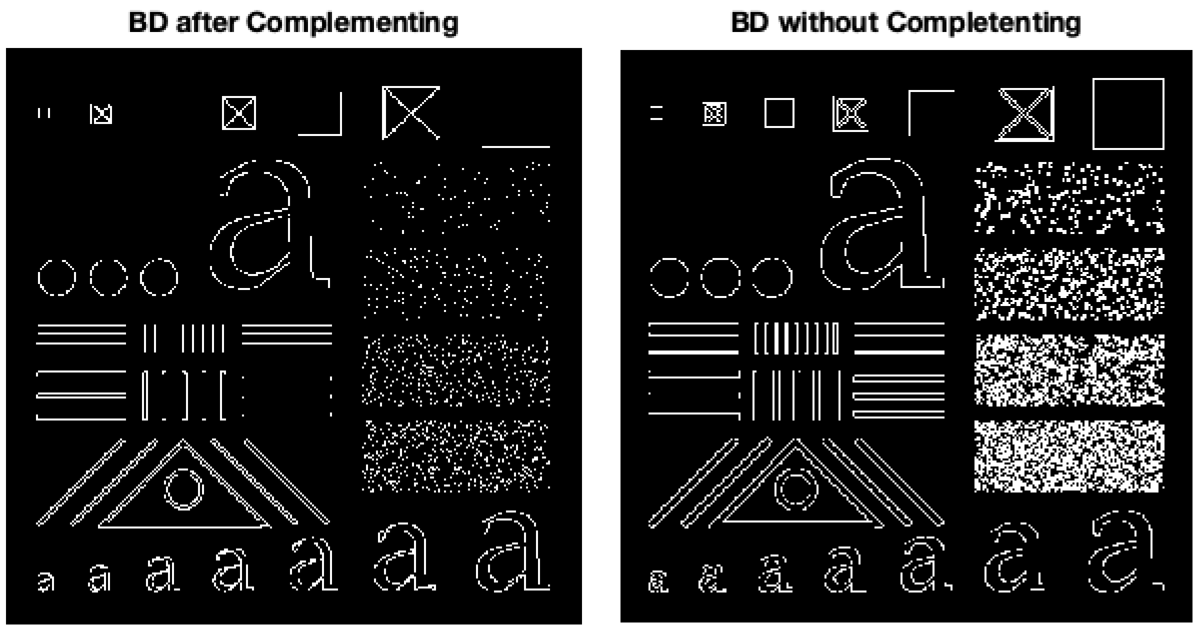
Read the image testpattern512-binary.tif whose objects are black. Use function **`morphoErode4e`** with its default setting and display the result. Comment on the difference between the result of (b) and the result in this part.

MATLAB script:

```
clc; close ALL; clear;
I = imread('../artfiles/testpattern512-binary.tif');
figure('position',[0,0,4,4.3]*72);
subplot('position',[0,0,1,4/4.3]);
imshow(I); title('testpattern-512 Image');
```



```
BD = morphoBoundary4e(I); % Boundary detection without complementing
BDc = morphoBoundary4e(not(I)); % Boundary detection after complementing
figure('position',[0,0,8.25,4.3]*72);
subplot('position',[0,0,4/8.25,4/4.3]);
imshow(BDc); title('BD after Complementing');
subplot('position',[4.25/8.25,0,4/8.25,4/4.3]);
imshow(BD); title('BD without Completenting');
```



Comment on the difference between the result of (b) and the result in this part.

Answer: From the two figures above, we observe that the boundaries in those two images are not the same, contrary to what we expected. For example, look at the double boundaries in the image on the right in the top squares containing the thin X's. The thickness of these X's is comparable to the size of **`B`**, so they should result in single, thin boundaries. The reason they don't is that the objects in **`I`** are black. our SEs are designed to work with objects composed of foreground (1—valued) pixels. So, because **B** is composed of foreground elements, what is happening here is that we are eroding the background (1s in this image) instead of the objects, and what is left as white are the areas in which the SE was not contained in the background, thus giving the resuls you see in the image on the right.

Problem-5.10 DIPUM3E Project 10.8 (Page 632)

(b) Morphological Smoothing/Sharpening of `cygnusloop.tif` Image

Solution: The following script provides images that match those shown in the book.

```
clc; close all; clear;
f = imread('../artfiles/cygnusloop.tif');
% Smooth the image
se = strel('disk',5); fo = imopen(f,se);
fsmooth = imclose(fo,se);
% Sharpen the image
ss = strel('disk',3);
fsharp = imdilate(fsmooth,ss) - imerode(fsmooth,ss);
% Fina result
final = fsmooth+fsharp;
% Display images
im(1) = {f}; im(2) = {fsmooth}; im(3) = {final};
figure('position',[0,0,9,3]*72); montage(im,'size',[1,3]);
```