

# HW #2 Solutions

## ECE 178 WINTER 2003

Prof: B.S. Manjunath  
TAs: Jelena Tešić, Marco Zuliani

### 2.19

The operator  $H$  is linear if  $H(aS_1 + bS_2) = aH(S_1) + bH(S_2)$  holds for  $\forall a, b \in \mathbb{R}$ . The median of a set of numbers is such that half the values in the set are below median value and the other half are above it. In this case  $H$  is the median operator. A simple example will suffice to show that linearity condition is violated by the median operator.

Let  $S_1 = \{1, -2, 3\}$ ,  $S_2 = \{4, 5, 6\}$ , and  $a = b = 1$ .  $H(S_1) = 1$ ,  $H(S_2) = 5$ . Therefore,  $H(S_1) + H(S_2) = 6$ . On the other hand,  $S_1 + S_2$  is the element-by-corresponding-element sum of  $S_1$  and  $S_2$ , and  $S_1 + S_2 = \{5, 3, 9\}$ . Therefore,  $H(S_1 + S_2) = 5$ . Since we found  $a, b, S_1$  and  $S_2$  for which linearity condition does not hold, i.e.  $H(aS_1 + bS_2) \neq aH(S_1) + bH(S_2)$ , it follows that the median is a nonlinear operator.

### Problem 4

To tile a plane,  $N$  corners of regular polygon with  $S$  sides must meet at a point. Since all of these corners have the same angle  $\alpha$ , it follows that:  $\alpha = \frac{360^\circ}{N}$ . The external angle of a regular polygon with  $S$  sides is  $\gamma = \frac{360^\circ}{S}$ . Then, the internal angle  $\beta$  is  $\beta = 180^\circ - \gamma$ . Therefore, the internal angle of a regular polygon with  $S$  sides is:  $\beta = 180^\circ - \frac{360^\circ}{S}$ . To tile a plane, we must have the internal angles of a polygon as the angles of the corners meeting at a point, i.e.  $\alpha = \beta$ :

$$\frac{360^\circ}{N} = 180^\circ - \frac{360^\circ}{S} \rightarrow \frac{2}{N} = \frac{S-2}{S} \rightarrow N = \frac{2S}{S-2}, S = \frac{2N}{N-2}$$

For a polygon,  $S \geq 3$ , and for  $N$  corners meeting at a point,  $N \geq 3$ . For  $S > 6 \rightarrow N < 3$ , so integer solutions exist only for  $S \leq 6$ :  $S = 3 \rightarrow N = 6$  (triangle),  $S = 4 \rightarrow N = 4$  (square) and  $S = 6 \rightarrow N = 3$  (hexagon). Therefore, only equilateral triangle, square and hexagon tile a plane.

### Problem 5

$$\text{a) } \begin{bmatrix} -1 & -2 & -2 & 2 & 3 \\ -4 & -5 & \underline{-2} & 5 & 6 \\ -3 & -2 & 2 & 2 & 1 \end{bmatrix} \quad \text{b) } [ 4 \quad 13 \quad \underline{28} \quad 27 \quad 18 ] \quad \text{c) } \begin{bmatrix} 0 & -1 & 0 & -1 & 0 \\ -1 & 2 & -7 & 1 & -1 \\ -2 & \underline{2} & 15 & 6 & -3 \\ 0 & -2 & -5 & -3 & 0 \end{bmatrix}$$

## MATLAB Problem

A few comments about the code below. First, note the convenience in using a function to carry out the halftoning procedure on the image. Second, note the use of multidimensional arrays for storing the halftoning patterns. This allows us to keep the code compact. You may also want to look at the method used to map the gray levels to the halftone: one line of code was enough to implement a basic quantizer.

### MATLAB test program

```
% NAME:      Hw2_sol.m
% DESC:      Test for the halftoning function for the HW #2 programming assignment
% AUTHOR:    Marco Zuliani
% DATE:      1-26-2003

close all clear all clc

% read the image
I = imread('image1.jpg');

% test image
% I = zeros(256);
% for ind = 1:256;
%     I(ind, :) = ind-1;
% end
% I = uint8(I);

% rescale it in order to have the halftoned image fitting in the screen
Ir = imresize(I, 0.5*size(I), 'bicubic');

% create the halftoned image
Iht = halftone(Ir);

% show results
imshow(Iht); title('Halftoned image')
```

## MATLAB halftoning function

```
function Iht = halftone(I)
% Iht = halftone(I)
%
% DESC:
% Returns the halftoned version of the input image I. Halftoned image dimensions
% are three times bigger than those of the original image.
%
% INPUT:
% I          = input image
%
% OUTPUT:
% Iht       = output image
%
% AUTHOR:
% Marco Zuliani

% Let's prepare the halftones: note the use of multidimensional arrays.
halftones = zeros(3, 3, 10);

% nice way to init the halftones preserving your fingertips...
halftones(1, 2, 2:10) = 255; halftones(3, 3, 3:10) = 255;
halftones(1, 1, 4:10) = 255; halftones(3, 1, 5:10) = 255;
halftones(1, 3, 6:10) = 255; halftones(2, 3, 7:10) = 255;
halftones(3, 2, 8:10) = 255; halftones(2, 1, 9:10) = 255;
halftones(2, 2, 10) = 255;

% quantize the original image. This formula allows us to span between 0 to 10
%(halftones indices)
%Note that I am dividing by 256 since we need to count also the value zero.
Iq = floor(double(I) / 256 * 10) + 1;

% re-map the quantized values using the halftones
% Note that when doing loops it is a good programming policy (read: more
% speed) to initialize the arrays out of the loop.
Iht = zeros(3*size(I)); for i = 1:size(I, 1)
    for j = 1:size(I, 2)
        ii = 3*i;
        jj = 3*j;
        Iht(ii:ii+2, jj:jj+2) = halftones(:, :, Iq(i, j));
    end
end

% enforce unsigned int
Iht = uint8(Iht);

return
```