COMPSCI 527 Homework 1

Cody Lieu

September 10, 2015

Problem 1(a)

$r_{Mf} = (r_A + r_H - 1)(c_A + c_H - 1)$
$c_{Mf} = r_A(c_A)$

Problem 1(b)

$$Mf = \begin{bmatrix}
1 & & & & & & & & & & & \\
2 & 1 & & & & & & & & & & \\
 & 2 & 1 & & & & & & & & & \\
 & & 2 & & & & & & & & & \\
3 & & & 1 & & & & & & & & \\
4 & 3 & & 2 & 1 & & & & & & & \\
 & 4 & 3 & & 2 & 1 & & & & & & \\
 & & 4 & & & 2 & & & & & & \\
5 & & & 3 & & & 1 & & & & & \\
6 & 5 & & 4 & 3 & & 2 & 1 & & & & \\
 & 6 & 5 & & 4 & 3 & & 2 & 1 & & & \\
 & & 6 & & & 4 & & & 2 & & & \\
 & & & 5 & & & 3 & & & 1 & & \\
 & & & 6 & 5 & & 4 & 3 & & 2 & 1 & \\
 & & & & 6 & 5 & & 4 & 3 & & 2 & 1 \\
 & & & & & 6 & & & 4 & & & 2 \\
 & & & & & & 5 & & & 3 & & \\
 & & & & & & 6 & 5 & & 4 & 3 & \\
 & & & & & & & 6 & 5 & & 4 & 3 \\
 & & & & & & & & 6 & & & 4 \\
 & & & & & & & & & 5 & & \\
 & & & & & & & & & 6 & 5 & \\
 & & & & & & & & & & 6 & 5 \\
 & & & & & & & & & & & 6
\end{bmatrix}$$

Problem 1(c)

$r_{Ms} = r_A(c_A)$
$c_{Ms} = r_A(c_A)$

Problem 1(d)

$$Ms = \begin{bmatrix}
4 & 3 & & 2 & 1 & & & & & & & \\
 & 4 & 3 & & 2 & 1 & & & & & & \\
 & & 4 & & & 2 & & & & & & \\
6 & 5 & & 4 & 3 & & 2 & 1 & & & & \\
 & 6 & 5 & & 4 & 3 & & 2 & 1 & & & \\
 & & 6 & & & 4 & & & 2 & & & \\
 & & & 6 & 5 & & 4 & 3 & & 2 & 1 & \\
 & & & & 6 & 5 & & 4 & 3 & & 2 & 1 \\
 & & & & & 6 & & & 4 & & & 2 \\
 & & & & & & 6 & 5 & & 4 & 3 & \\
 & & & & & & & 6 & 5 & & 4 & 3 \\
 & & & & & & & & 6 & & & 4
\end{bmatrix}$$

Problem 1(e)

```
function M = convMatrix(H, sA, type)

rotH = rot90(H, 2);

rA = sA(1);
cA = sA(2);

rH = size(rotH, 1);
cH = size(rotH, 2);

rB = rA + rH - 1;
cB = cA + cH - 1;

cMf = rA*cA;

Mf = [];

for c = 1:cB
    for r = 1:rB
        tempArr = zeros(1, cMf);
        for rBound = 1:rH
            for cBound = 1:cH
                if r+rBound-1 >= rH  && r+rBound <= rA+rH
                    if c+cBound-1>=cH && c+cBound <= cA+cH
                        origR = r+rBound-rH;
                        origC = c+cBound-cH;
                        y = (origC-1)*rA+origR;
                        tempArr(y) = rotH(rBound, cBound);
                    end
                end
            end
        end
        Mf = [Mf;tempArr];
    end
end

if strcmp(type, 'full') == 1
```

```
    M = Mf;
else
    Ms = [];

    leftPaddingSame = floor((cH-1)/2);
    rightPaddingSame = cH-leftPaddingSame-1;
    topPaddingSame = floor((rH-1)/2);
    bottomPaddingSame = rH-topPaddingSame-1;

    leftPaddingFull = cH-1;
    rightPaddingFull = cH-1;
    topPaddingFull = rH-1;
    bottomPaddingFull = rH-1;

    leftPaddingDiff = leftPaddingFull - leftPaddingSame;
    rightPaddingDiff = rightPaddingFull - rightPaddingSame;
    topPaddingDiff = topPaddingFull - topPaddingSame;
    bottomPaddingDiff = bottomPaddingFull - bottomPaddingSame;

    % take out rows lost by the left and right padding difference
    MsTemp = [];
    MsTemp = Mf((leftPaddingDiff*rB+1:end),:);

    MsSize = size(MsTemp);
    MsRow = MsSize(1);

    MsTemp = MsTemp((1:MsRow-(rightPaddingDiff*rB)),:);

    % then take out the rows lost by the top and bottom padding difference

    MsSize = size(MsTemp);
    MsRow = MsSize(1);
    for index = 1:rB:MsRow
        Ms = [Ms;MsTemp(index+topPaddingDiff:index+rB-1-bottomPaddingDiff,:)];
    end

    M = Ms;
end

end
```

Problem 1(f)

$$M1 = \begin{bmatrix} 1 & & & & & & \\ & 1 & & & & & \\ & & 1 & & & & \\ & & & 1 & & & \\ & & & & 1 & & \\ & & & & & 1 & \end{bmatrix} \quad M2 = \begin{bmatrix} 2 & & 1 & & & & \\ & 2 & & 1 & & & \\ & & 2 & & 1 & & \\ & & & 2 & & 1 & \\ & & & & 2 & & \\ & & & & & 2 & \end{bmatrix}$$

$$M3 = \begin{bmatrix} 2 & 1 & & & & \\ & 2 & & & & \\ & & 2 & 1 & & \\ & & & 2 & & \\ & & & & 2 & 1 \\ & & & & & 2 \end{bmatrix} \quad M4 = \begin{bmatrix} 4 & 3 & 2 & 1 & & \\ & 4 & & 2 & & \\ & & 4 & 3 & 2 & 1 \\ & & & 4 & & 2 \\ & & & & 4 & 3 \\ & & & & & 4 \end{bmatrix}$$

$$M5 = \begin{bmatrix} 5 & 4 & 2 & 1 & & \\ 6 & 5 & 3 & 2 & & \\ & & 5 & 4 & 2 & 1 \\ & & 6 & 5 & 3 & 2 \\ & & & & 5 & 4 \\ & & & & 6 & 5 \end{bmatrix} \quad M6 = \begin{bmatrix} 5 & 4 & 2 & 1 & & \\ 6 & 5 & 3 & 2 & & \\ 8 & 7 & 5 & 4 & 2 & 1 \\ 9 & 8 & 6 & 5 & 3 & 2 \\ & & 8 & 7 & 5 & 4 \\ & & 9 & 8 & 6 & 5 \end{bmatrix}$$

Problem 1(g)

reshape(convMatrix(H, size(A), 'same')*A(:), size(A));

Problem 1(h)

H = reshape(1:6, [2 3]);
A = reshape(1:12, [3 4]);

S = conv2(A, H, 'same');
S2 = reshape(convMatrix(H, size(A), 'same')*A(:), size(A));

$$S = \begin{bmatrix} 23 & 69 & 132 & 155 \\ 33 & 90 & 153 & 173 \\ 24 & 60 & 96 & 102 \end{bmatrix} \quad S2 = \begin{bmatrix} 23 & 69 & 132 & 155 \\ 33 & 90 & 153 & 173 \\ 24 & 60 & 96 & 102 \end{bmatrix}$$

Problem 2(a)

$$K = \frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

A two-dimensional filter kernel is separable if it can be expressed as the outer product of two vectors.

$$K = \left(\frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\right)\left(\frac{1}{2}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}\right)$$

$$K_1 = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} K_2 = \frac{1}{2}\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

The matrix product of a column vector and row vector is equivalent to the two-dimensional convolution of the two vectors. Therefore:

$K = K_1 K_2 = K_1 * K_2$

Problem 2(b)

The Sobel operator approximates a horizontal gradient and a vertical gradient. K can be decomposed into the product of K1 (a 3x1 matrix) and K2 (a 1x3 matrix). Convolving with K1 smoothes/averages the image perpendicular to the derivative direction. K2 is the differentiating operator and convolving with K2 approximates the differentiation of the underlying continuous intensity function of the image. The result of convolving an image with K2 and then K1 is a gradient vector G for each discrete image point, which can be used to calculate the magnitude and direction of the vector.

Problem 2(c)

$$4MK1 = \begin{bmatrix} 2 & 1 & & & & & & & \\ 1 & 2 & 1 & & & & & & \\ & 1 & 2 & & & & & & \\ & & & 2 & 1 & & & & \\ & & & 1 & 2 & 1 & & & \\ & & & & 1 & 2 & & & \\ & & & & & & 2 & 1 & \\ & & & & & & 1 & 2 & 1 \\ & & & & & & & 1 & 2 \end{bmatrix}$$

$$2MK2 = \begin{bmatrix} & & & 1 & & & & & \\ & & & & 1 & & & & \\ & & & & & 1 & & & \\ -1 & & & & & & 1 & & \\ & -1 & & & & & & 1 & \\ & & -1 & & & & & & 1 \\ & & & -1 & & & & & \\ & & & & -1 & & & & \\ & & & & & -1 & & & \end{bmatrix}$$

Problem 2(d)

$M_k = M_{k1} * M_{k2} = M_{k2} * M_{k1}$

Problem 2(e)

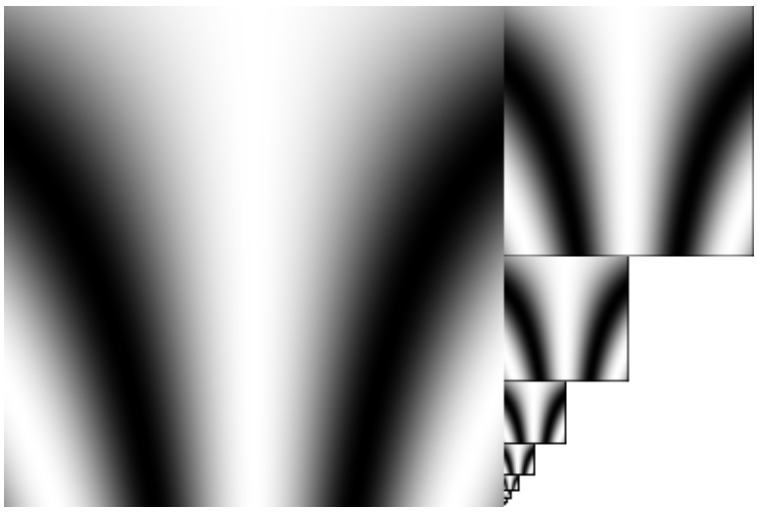
The square matrices M_s have the special property that they are separable.

Problem 3(a)

```
function p = blockPyramid(img)

% error checking

if ~ismatrix(img)
    error('Input must be a black-and-white image (two-dimensional array)');
end

imgSize = size(img);

if imgSize(1) ~= imgSize(2)
    error('Input must be a square matrix');
end

if (floor(log2(imgSize(1))) ~= log2(imgSize(1))) || (floor(log2(imgSize(1))) ~= log2(imgSize(2)))
    error('Each dimension must be a power of 2');
end

% main code

B = (1/4) .* [1, 1;1, 1];
rotB = rot90(B, 2);
rB = size(rotB, 1);
cB = size(rotB, 2);

p = {};
p{1} = img;

count = 2;
s = size(p{count-1});

leftPaddingSame = floor((cB-1)/2);
rightPaddingSame = cB-leftPaddingSame-1;
topPaddingSame = floor((rB-1)/2);
bottomPaddingSame = rB-topPaddingSame-1;

while s(1) > 1
    lastImgPadded = p{count-1};
    lastImgPadded = [zeros(size(lastImgPadded,1),leftPaddingSame) lastImgPadded];
    lastImgPadded = [lastImgPadded zeros(size(lastImgPadded,1),rightPaddingSame)];
    lastImgPadded = [zeros(topPaddingSame, size(lastImgPadded, 2));lastImgPadded];
    lastImgPadded = [lastImgPadded; zeros(bottomPaddingSame, size(lastImgPadded, 2))];

    rPadded = size(lastImgPadded, 1);
    cPadded = size(lastImgPadded, 2);

    r = 2;
    while r < rPadded
        c = 2;
        while c < cPadded
            zTemp = rotB.*lastImgPadded(r:r+rB-1, c:c+cB-1);
            Z(r/2,c/2) = sum(zTemp(:));
            c = c + 2;
        end
```

```
        r = r + 2;
    end

    p{count} = Z;
    Z = [];
    s = size(p{count});
    count = count + 1;
end

end
```

Problem 3(b)



Problem 3(c)

```
function gp = gMag(p)
    % K not necessary but wanted to show it
    K = (1/8) .* [1, 0, -1;
                  2, 0, -2;
                  1, 0, -1];
    K1 = (1/4) .* [1; 2; 1];
    K2 = (1/2) .* [1, 0, -1];

    gp = {};
    for i = 1:size(p, 2)
        gpTemp = imfilter(p{i}, K2, 'replicate');
        gp{i} = imfilter(gpTemp, K1, 'replicate');
    end
end
```

Problem 3(d)