Problem 3 Overview D: The "one-round simple raster" method to compress 2D photos using Haar wavelets

1) How to read in photos in matlab

Suppose you were given a jpg or tiff that contained 16 x 16 pixels (ie. $2^n \times 2^n$ photo sizes):

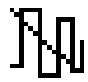


Figure 1: A 16 x 16 pixel photo name "squiggle.tif." This is an 8-bit black + white image

Since the image is 8-bit black + white, the grayscale for each pixel are initially "0" for black and "255" for white. In this exercise, we would like to invert the tone:

$$\begin{cases} Black = 0 = 2^0 - 1 \\ White = 255 = 2^8 - 1 \end{cases} \xrightarrow{We\ will\ change\ this\ to:} \begin{cases} Black = 255 \\ White = 0 \end{cases}$$

To read this image into matlab, you can use the *imread* function. Assuming "squiggle.tif" was already in your working matlab directory, you would type the following to load the file, change the precision of our data for processing, and then, invert the black / white tone:

```
my_filename = {'squiggle'
                              };

number_of_bits = 8;   % -- Definition !


% -- Initialize variable
A = [ ];

% -- Load image
A = imread(char(my_filename(1)), 'tiff');

%  -- Since the direct read from a TIFF file is a 16-bit unsigned integer
%  (uint16), you need to convert to floating decimals

A = single(A);

% \\\\\\\\\\\\\\\\\\\\\\\\ Un-normalized bits option \\\\\\\\\\\\\\\\\\
% -- Was:            Black = 0,   White = 255
%    we want:        Black = 255, White = 0;... and that's it !

A = abs(A - (2^number_of_bits -1)) ;
```

If you echo matrix A, you'll get a bodacious, 80's-style ASCII art that's in the form of numbers !!   =)

$$A = \begin{bmatrix}
255 & & 255 & 255 & 255 & 255 & 255 & & & & & & & & & \\
 & 255 & 255 & & & & 255 & & & & & & & & & \\
 & & 255 & & & & 255 & & & & 255 & 255 & 255 & 255 & & \\
 & & 255 & 255 & & & 255 & & & & 255 & & & 255 & & \\
 & & 255 & & 255 & & 255 & & & & 255 & & & 255 & & \\
 & & 255 & & & 255 & 255 & & & & 255 & & & 255 & & \\
 & & 255 & & & & 255 & & & & 255 & & & 255 & & \\
 & & 255 & & & & 255 & 255 & & & 255 & & & 255 & & 255 \\
 & & 255 & & & & 255 & & 255 & & 255 & & & 255 & & 255 \\
 & & 255 & & & & 255 & & & 255 & 255 & & & 255 & & 255 \\
 & & 255 & & & & 255 & & & & 255 & 255 & & 255 & & 255 \\
 & & 255 & & & & 255 & & & & 255 & & 255 & 255 & & 255 \\
255 & 255 & 255 & & & & 255 & & & & 255 & & & 255 & & 255 \\
 & & & & & & 255 & & & & 255 & & & 255 & 255 & 255 \\
 & & & & & & 255 & 255 & 255 & 255 & 255 & & & & & \\
 & & & & & & & & & & & & & & &
\end{bmatrix}$$

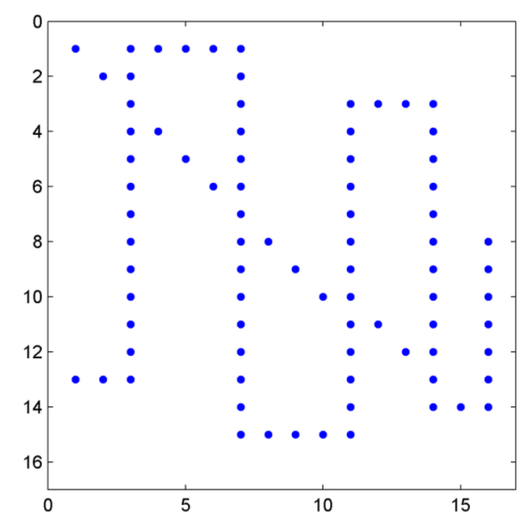


Figure 2: If you typed *spy(A)* in matlab, you'll Get a nice representation of your data too !

2) The game plan: One round "simple" raster in the vertical / horizontal axes

To perform *one round (J = 4 to 3)* of image compression for a 16×16 photo array called "squiggle," we will:

a) Column rastering (taking successive avg and ½ differences from each column vector), then
b) Row rastering of the resulting data matrix

Result: We will have compressed our 16 x 16 squiggle down to a smaller, 8 x 8 squiggle on the top-right corner

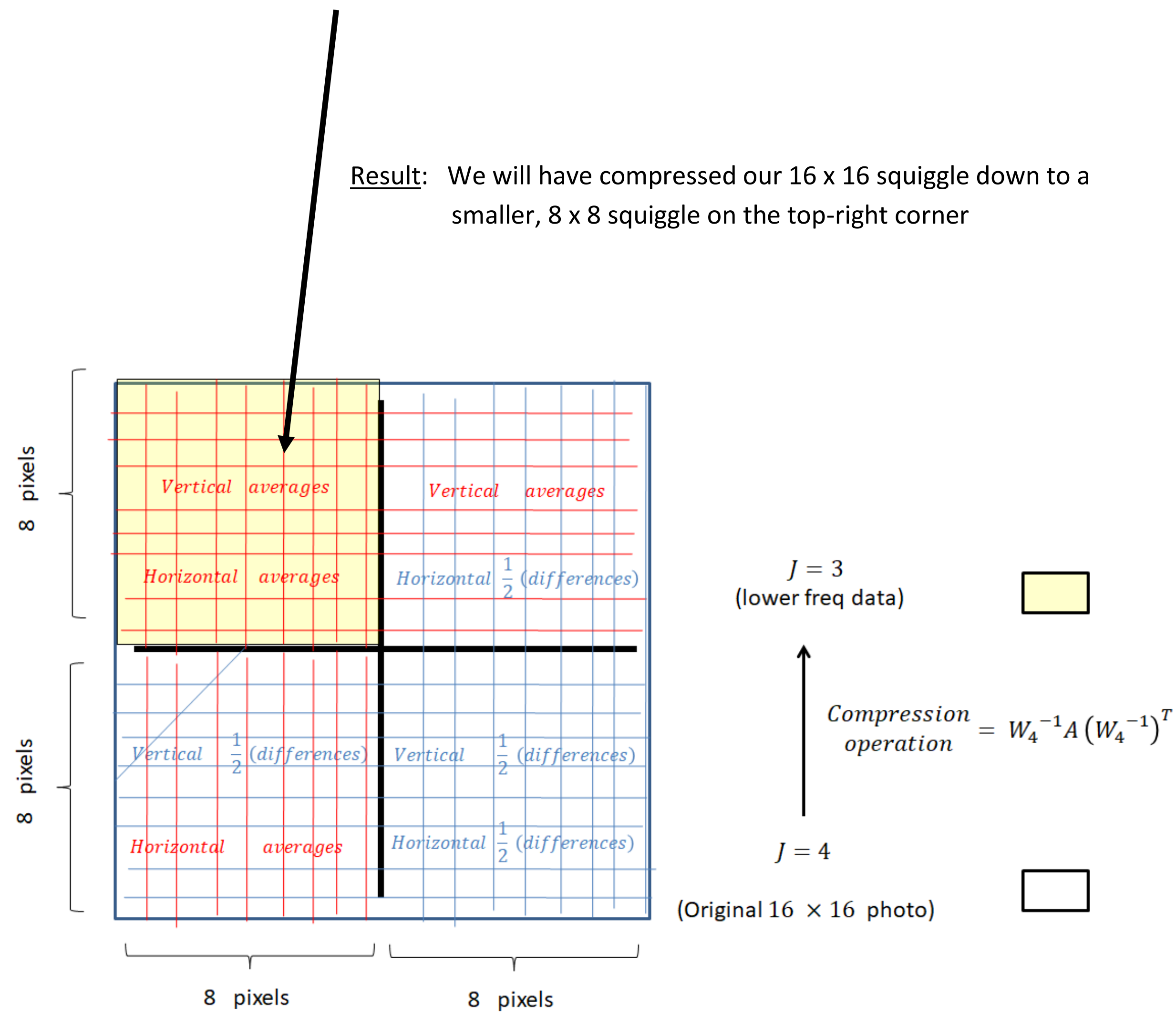
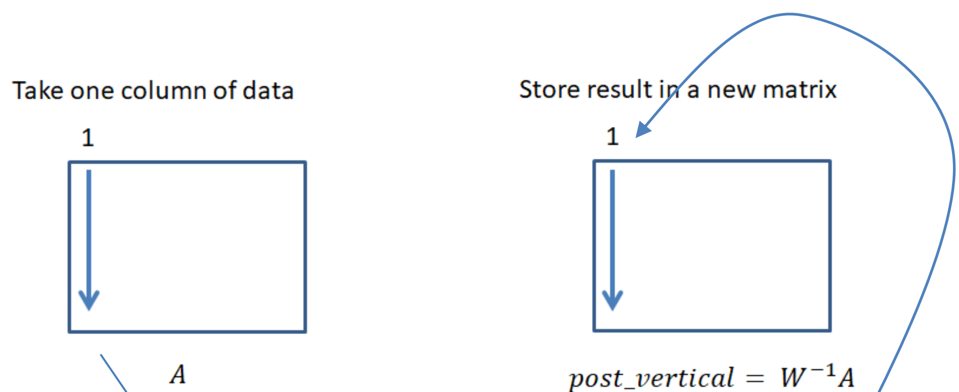


Figure 1: The 2D wavelet transform: First, we transform the columns of A, and then, the *rows* of the resulting matrix experiences another transform. The final result is that all *non-upper-left* quadrants in each $J - scale$ are the 2D wavelet coefficients.

2A) "One simple round" of column rastering of our 2D image data

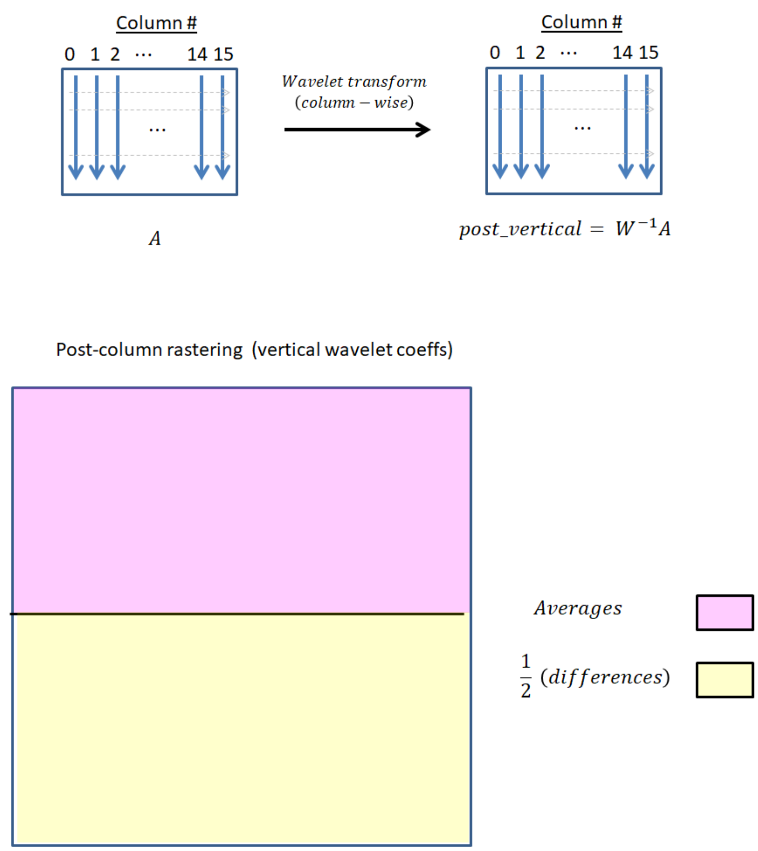
Let's apply the butterfly diagram to column #1 of our photo matrix A. For visual aid, we'll color-code the 4 groups of wavelet coefficients:



$$\begin{bmatrix} 255 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 255 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{Single round of top / bottom bunk only !!}} \begin{bmatrix} 127.5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 127.5 \\ 0 \\ \hline 127.5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -127.5 \\ 0 \end{bmatrix}$$

$$column \ \#1 \ of \ A \qquad\qquad ultra-fine \ (J_y = 4)$$

You would repeat the same operation for all columns ! When you're done, you'll get a set of "vertically-decomposed" wavelet coefficient matrix that's denoted by $W_4^{-1}A$.



$$Post\_vertical = W_4^{-1}A$$

Figure 2: For a "one round raster" process, we first column-raster all columns of data…. but when we do, we will restrict ourselves to performing *one* top-bunk / bottom-bunk operation !

Single vertical raster details

Starting with the 16 x 16 matrix A, we can perform top & bottom-bunk operations on the *columns* of A:

$$\begin{matrix} Transforming \\ columns\ of\ A \end{matrix} = {W_4}^{-1} A = \begin{bmatrix} Perform\ your\ high-school \\ averge\ and\ difference \\ transformations\ on\ each \\ columns\ of\ A \end{bmatrix}$$

After column-rastering your photo, you'll get:

Vertical averages

Vertical ½ (diff)

$$W^{-1}A = \begin{bmatrix}
127.5 & 127.5 & 255 & 127.5 & 127.5 & 127.5 & 255 & & & & & & & & & \\
 & & 255 & 127.5 & & & 255 & & & & 255 & 127.5 & 127.5 & 255 & & \\
 & & 255 & & 127.5 & 127.5 & 255 & & & & 255 & & & 255 & & \\
 & & 255 & & & & 255 & 127.5 & & & 255 & & & 255 & & 127.5 \\
 & & 255 & & & & 255 & & 127.5 & 127.5 & 255 & & & 255 & & 255 \\
 & & 255 & & & & 255 & & & & 255 & 127.5 & 127.5 & 255 & & 255 \\
127.5 & 127.5 & 127.5 & & & & 255 & & & & 255 & & & 255 & -127.5 & 255 \\
 & & & & & & 127.5 & 127.5 & 127.5 & 127.5 & 127.5 & & & & & \\
127.5 & -127.5 & & 127.5 & 127.5 & 127.5 & & & & & & & & & & \\
 & & & -127.5 & & & & & & & & 127.5 & 127.5 & & & \\
 & & & & 127.5 & -127.5 & & & & & & & & & & \\
 & & & & & & & -127.5 & & & & & & & & -127.5 \\
 & & & & & & & & 127.5 & -127.5 & & & & & & \\
 & & & & & & & & & & & 127.5 & -127.5 & & & \\
127.5 & 127.5 & 127.5 & & & & & & & & & & & & -127.5 & \\
 & & & & & & 127.5 & 127.5 & 127.5 & 127.5 & 127.5 & & & & &
\end{bmatrix}$$

2B) "One simple round" of row rastering of our 2D image data

Next, we would take the previous result and do 1 round of row-rastering on it. From the last overview file, you know that this is equivalent to:

1) Take the previous matrix and take the *transpose* of it
2) Then, perform 1 round of column rastering again (which is equivalent to row rastering !)
3) Then, transpose the final results back to obtain the final product !

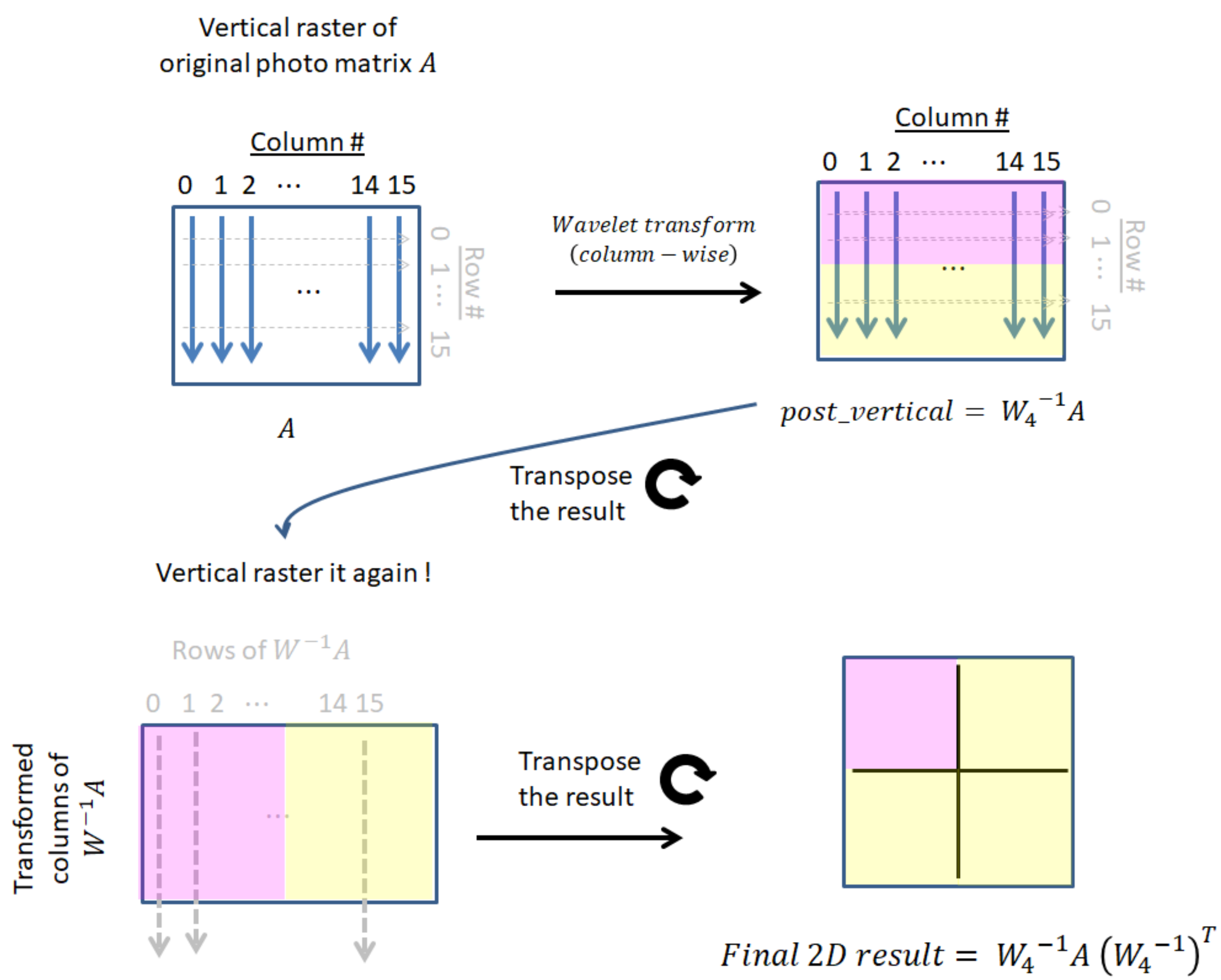


Figure 2: The complete algorithm for *one round* of vertical + horizontal rastering of a 16×16 squiggle phot. The end result will be a 8×8 smaller squiggle photo on the upper-left hand corner of the matrix, and this corresponds to one complete round of wavelet compression from $J = 4$ to $J = 3$.

Single horizontal raster details (after the vertical raster step)

Starting with the post-vertically-rastered photo, we can perform top & bottom-bunk operations on the *__rows__* of A. Mathematically, this is equivalent to a right-hand matrix multiplication by $\left(W_4{}^{-1}\right)^T$:

$$\begin{matrix} Transforming \\ rows\ of\ previous\ result \end{matrix} \quad = \quad B \quad = \quad W_4{}^{-1}\, A\ \left(W_4{}^{-1}\right)^T$$

Equivalent !!

Again, to do this, we shall

1) Transpose the previous matrix
2) Perform the same "vertical raster" operation
3) Transpose the result again…. this will return our image to the original orientation !

Hey…. this means we could use our algorithm and rewrite an equivalent matrix operation as:

$$\begin{matrix} Transforming \\ rows\ of\ previous\ result \end{matrix} \quad = \quad B \quad = \left(\ W_4{}^{-1}\left(W_4{}^{-1}\, A\right)^T\right)^T$$

In the end, here's what you should get ! =)

Vertical & horizontal averages

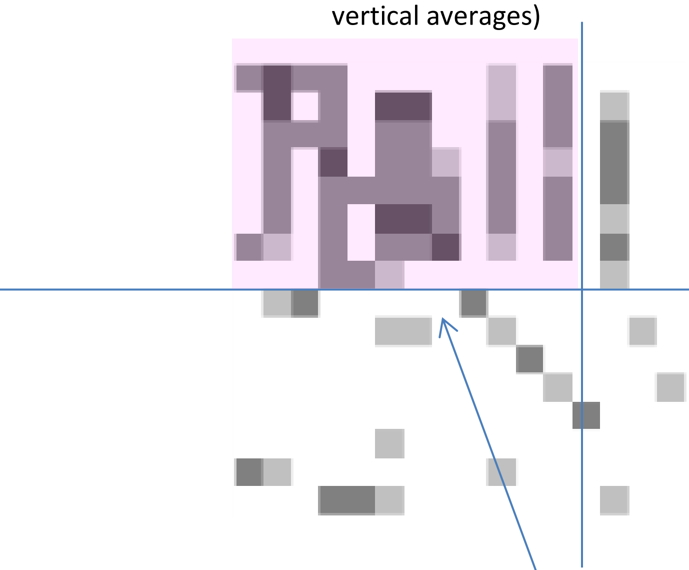
Vertica avg horizontal diff

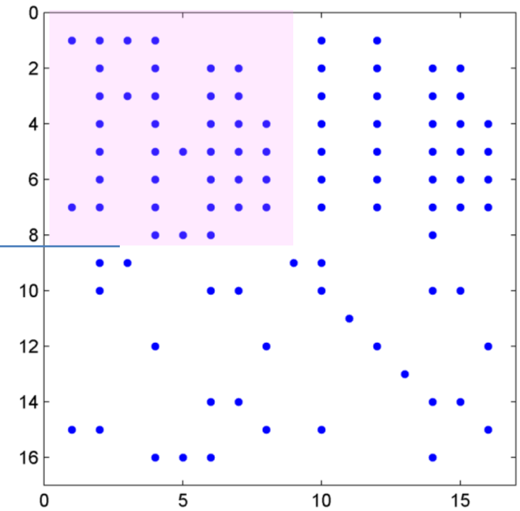
$$\begin{bmatrix}
127.5 & 191.25 & 127.5 & 127.5 & & & & & & 63.75 & & 127.5 & & & & \\
 & 191.25 & & 127.5 & & 191.25 & 191.25 & & & 63.75 & & 127.5 & & 63.75 & -63.75 & \\
 & 127.5 & 127.5 & 127.5 & & 127.5 & 127.5 & & & 127.5 & & 127.5 & & 127.5 & -127.5 & \\
 & 127.5 & & 191.25 & & 127.5 & 127.5 & 63.75 & & 127.5 & & 63.75 & & 127.5 & -127.5 & -63.75 \\
 & 127.5 & & 127.5 & 127.5 & 127.5 & 127.5 & 127.5 & & 127.5 & & 127.5 & & 127.5 & -127.5 & -127.5 \\
 & 127.5 & & 127.5 & & 191.25 & 191.25 & 127.5 & & 127.5 & & 127.5 & & 63.75 & -63.75 & -127.5 \\
127.5 & 63.75 & & 127.5 & & 127.5 & 127.5 & 191.25 & & 63.75 & & 127.5 & & 127.5 & -127.5 & -63.75 \\
 & & & 127.5 & 127.5 & 63.75 & & & & & & & & 63.75 & & \\
 & 63.75 & 127.5 & & & & & & 127.5 & -63.75 & & & & & & \\
 & -63.75 & & & & 63.75 & 63.75 & & & 63.75 & & & & -63.75 & 63.75 & \\
 & & & & & & & 63.75 & & & 127.5 & & & & & \\
 & & & -63.75 & & & & & & & & 63.75 & & & & 63.75 \\
 & & & & & & & & & & & & 127.5 & & & \\
 & & & & & 63.75 & -63.75 & & & & & & & -63.75 & -63.75 & \\
127.5 & 63.75 & & & & & & -63.75 & & 63.75 & & & & & & -63.75 \\
 & & & 127.5 & 127.5 & 63.75 & & & & & & & & 63.75 & &
\end{bmatrix}$$

Vertical differences
Horizontal averages

Vertical and horizontal differences

J = 3 picture
(horizontal and
vertical averages)





J = 4
vertical diff
horizontal avg

J = 4
vertical & horizontal
differences

Figure 3: The compressed data can be found in the $J = 3$ low-frequency quadrant that's shaded in magenta. The left plot is the *imshow* plot of the resulting $B = \left(W_4^{-1} \left(W_4^{-1} A\right)^T \right)^T$ matrix, whereas the right plot is the spy plot of the same thing.

Data Compression:

Our low-frequency J = 3 data sort of looks like our original squiggle, albeit with:

- Loss of details, but…
- It's only ¼ of the size of the original squiggle !

You can convert matrix $B = \left(W_4^{-1} \left(W_4^{-1} A\right)^T \right)^T$ as a black & white and plot the result with these commands:

```
my_black = 255;
my_white = 0;

J4grayscale = mat2gray(B, [my_black, my_white]);

figure
J4Horizontalgrayscale_plothandle = imshow(J4grayscale);
title('Post column and row-rastered squiggle (J = 3 on top left corner)')
```