Homework on Fractal Coding

Problem 3.1 (20 points)

Figures 1 show attractors of iterated function systems (IFS) of the form \( \{ \mathbb{R}^2; w_i, i = 1,2, \ldots, N \} \) (with no condensation transformations) where each \( w_i \) is an affine transformation. For each figure,

1. Find the IFS by determining \( N \) and the transformations \( w_i \). Write \( w_i \)'s in matrix form (estimate any necessary parameters not given in the figures).
2. Find the contractivity factor of the IFS you find.
3. Find the fractal dimension of the attractor.
4. Is the IFS found unique? Explain your answers in one or two lines.

Figure 1.
Problem 3.2 (100 points)

In this computer assignment, you are asked to write one main MATLAB program, and two MATLAB functions, which will be called in the main program. These functions are described below.

1 Encoder

The first MATLAB function must have the name `encode`, and has the following specifications:

1. The MATLAB header of this function has the form:

   ```matlab
   function [row, column, a, b] = encode(inputImage)
   ```

   This function takes a grayscale image (the `inputImage`), applies a basic form of fractal coding to it, and generates a set of fractal code parameters (`row`, `column`, `a`, `b`) as its output.

   For every range block in the image, this function searches through all domain blocks, shrinks them, and finds the one that best approximates the range block using a multiplicative factor and an additive factor. For all the range blocks, the address of the best domain blocks (the row and column numbers of the top-left corner of the selected domain blocks), the multiplicative factor and the additive factors, make up the code for the image and are stored in the returned arrays `row`, `column`, `a` and `b`, respectively.

2. This function uses the following settings:

   - The `inputImage` is a 128 × 128 array. Each element of this array can have a value from 0 to 255.
   - Range blocks are of size 4×4. As range blocks do not overlap, there are \((128/4) \times (128/4) = 32 \times 32 = 1024\) range blocks in the image.
   - Domain blocks are of size 8×8. Domain blocks, in general, may overlap. However, to reduce computation time in this assignment, we fetch domain blocks in step size of 8, both horizontally and vertically. Therefore, there are \((128/8) \times (128/8) = 16 \times 16 = 256\) candidate domain blocks in the image.
   - In search for the best domain block for each range block, sweep the image from the top-left corner of the image, to the bottom right of the image, column by column, using a step size of 8 (both horizontally and vertically).
   - For shrinking domain blocks, use simple averaging of every four neighboring pixels in the domain block.
   - As MATLAB saves two-dimensional arrays in its memory in a column-by-column format, in order to reduce computation time, and facilitate computations, in this homework, wherever it is necessary to save or scan a
two-dimensional block or image into a one-dimensional array, do the sweep in a column-by-column form.

3. The function **encode** uses the following local arrays:

   - **range**: is a 4×4 array that can contain one range block at a time.
   - **rangke1D**: is **range(;)**
   - **domain**: is a 8×8 array that contains one range block at a time.
   - **shrunkDomain**: is a 4×4 array that can contain one shrunk domain block at a time.
   - **pool**: is a 16×256 array. Every column of this array will contain one shrunk domain block of the image. Note that using a step size of 8 for fetching domain blocks, there are 16×16 = 256 domain blocks in the image, and there are 16 pixels in each shrunk domain block. This array is constructed in the **encode** once, before range blocks are fetched and processed.
   - **prow**: is a 1×256 array, which contains the row numbers of the pixel at the top-left corner of the blocks whose shrunk versions are put in **pool**. This array does not change as range blocks are fetched.
   - **pcolumn**: is a 1×256 array, which contains the column numbers of the pixel at the top-left corner of the domain blocks whose shrunk versions are put in **pool**. This array does not change as range blocks are fetched.
   - **pa**: is a 1×256 array, which for each range block will contain the multiplicative factors of columns of **pool**. This array should be updated for each range block.
   - **pb**: is a 1×256 array, which for each range block will contain the additive factors of columns of **pool**. This array should be updated for each range block.
   - **perror**: is a 1×256 array, which for each range block will contain the error resulting from approximating each range block using columns of **pool**. This array should be updated for each range block.

4. **encode** proceeds in the following way:

   (a) The **pool**, **prow**, and **pcolumn** are filled.
      - **encode** fetches domain blocks and puts them in the **domain** array, one at a time.
      - The domain block in **domain** is shrunk by a factor of 2 in each direction and the results are put in **shrunkDomain**.
      - The **shrunkDomain** is scanned column by column, and is put in the corresponding column of **pool**. For example, for the i-th domain block, you may write:
        
        ```
        pool(:, i)=shrunkDomain(:);
        ```
      - The row and column number of the top-left pixel of the domain blocks is put in the corresponding element of arrays **prow** and **pcolumn**.
(b) Range blocks are fetched from the image one by one, and their code is found:
   i. Range blocks are fetched one at a time and put in range.
   ii. For the fetched range block, the best $a$, $b$, and RMS error is found for each column of the pool, and stored in elements of the arrays $pa$, $pb$, and $perror$.
      You are allowed to choose $a$ from the set $\{0, 0.2, 0.4, 0.6, 0.8\}$, and for each $a$, you may find the best possible $b$ and the error from the following functions:

      \[
      \text{function} \ [\text{error}, b] = \text{find\_error}(\text{range1D}, p, a) \\
      b = \text{mean}(\text{range1D}) - a * \text{mean}(p); \\
      e = \text{range1D} - a * p - b; \\
      \text{error} = e' * e; \\
      \]
      where $p$ is one column of pool.
   iii. For each range block, by looking at the values of $perror$, see which column of the pool has the lowest value, and save its corresponding values from $prow$, $pcolumn$, $pa$, and $pb$ in arrays row, column, $a$, and $b$.

2 Decoder

The second MATLAB function has the name decode and has the following specifications:

1. The MATLAB header of this function has the form:

   \[
   \text{function} \ [\text{decodedImage}] = \text{decode}(\text{row, column, a, b}) \\
   \]
   This function takes the parameters generated by the encode and generates the decoded image. This is done by taking an initial all-zero image, and iteratively applying the transformation found by the function encode on it. It must continue its iterations until the RMS error between images obtained this way at two consecutive iterations become less than 1.

2 This function has the following variables:
   - **oldImage**: is a $128 \times 128$ array containing the image resulting from the last iteration. Initially, set all its value to zero (although it can be set to any arbitrary initial image).
   - **newImage**: is a $128 \times 128$ array that must be constructed using the encoding parameters and the oldImage at each iteration. After completion, it is copied to oldImage.
   - **domain**: is a $8 \times 8$ array to contain domain blocks one at a time.
• **shrunkDomain**: is a $4 \times 4$ array to contain the shrunk domain block.

3 The main program

The main MATLAB program reads and displays the binary grayscale image **lena.bin** and saves it in the array **inputImage** using the following MATLAB commands:

```matlab
fid=fopen('lena.bin', 'r');  %Open the lena.bin file
inputImage=fread(fid, [128, 128], 'uchar');  % Read the file to inputImage
figure(1);  %Open a window for display
colormap(gray(256));  % Use gray levels as colors
image(inputImage);  % Display the inputImage
title('inputImage');  % Generate title for the image
```

The binary file **lena.bin** for this problem will be posted on the course web page of the class, which you can download to your computer electronically.

The main program then calls the **encode** function to generate the fractal coding parameters. These parameters are then given to the function **decode** for decoding. The main program then writes and displays the decoded image using the following MATLAB commands:

```matlab
fid=fopen('decodedLena.bin', 'w');  %Open the decodedLena.bin file
fwrite(fid, decodedImage, 'uchar');  %Write the file to decodedImage
figure(2);  %Open a window for display
colormap(gray(256));  %Use gray levels as colors
image(decodedImage);  %Display the decodedImage
```

At the end, the main program should compute the rms error and the PSNR of the decoded image, compared to the **inputImage**.

4 What to hand in

For grading, you should hand in a printout of your MATLAB files, the decoded image, and its rms error and PSNR, and a concise report.

You should also send your MATLAB files to the lecturer via email qitian@cs.utsa.edu by the due date.

**Late homework will NOT be accepted.**