CS371/AMATH242 Winter 2019: Assignment 4
Instructor: Maryam Ghasemi
Due Sunday, April 7, 11:59 pm via Crowdmark

---

1. [2 marks] Given that $W_N = e^{2\pi i/N}$, for any integers n and k, prove that

$$W_N^{-nk} + W_N^{-(N-n)k} = 2\cos\left(\frac{2\pi nk}{N}\right).$$

2. [20 marks] In this question, you will investigate a simple method of image compression that gives an idea of ways in which data can be compressed but still retain information.

Background
A picture in MATLAB can be represented in several ways. You will investigate using a 3D matrix to represent a colour image stored in jpeg/jpg format. Such an image can be loaded into MATLAB with the command `P = imread('picture.jpg');`

This defines a matrix P, of size $n \times m \times 3$. The image is $n \times m$. The entries `P(i,j,1), P(i,j,2) ,P(i,j,3)` give the intensities of Red, Green, and Blue (respectively) colours at the pixel location (i,j). The RGB combination determines the colour of each component pixel in the image. Each entry in P is a value between 0 and 255 (and is an unsigned integer, MATLAB type `u8int`). You can display this image in a MATLAB window with the command `image(P)`.

Overview of technique
You will divide each of the three colour matrices into 15x15 blocks, B, and apply the Fourier transform to each block in turn (using the `fft2` function). This will create a new 15x15 matrix, F$_B$, in the transformed (frequency) space. For each block, you will "drop" some of the smallest valued coefficients by replacing them with 0 (in a real compression algorithm, you would no longer store the dropped coefficients in order to reduce space requirements, but we will retain them in our simplified algorithm). After modifying the transformed 15x15 coefficient matrix in this way, you can go back to the image view by applying the inverse transformation (using the `ifft2` function) to it. Once all 15x15 blocks have been processed, the image can again be displayed (after a bit of "clean up", as noted below).

a) [5 marks] Write a MATLAB function `pad15` that consumes P, an $n \times m \times 3$ matrix, and produces a new matrix Q of size $n_p \times m_p \times 3$, where n_p is the smallest multiple of 15 satisfying $n_p \geq n$, and m_p is the smallest multiple of 15 satisfying $m_p \geq m$. If the number of rows or columns is already a multiple of 15, then no padding is required in that dimension. Start with the code given below and fill the empty spaces;

```
function [ P_padded ] = pad15( P )
%pad - produces a new matrix like P, except that the number of rows and
%columns are both multiples of 15.
% Note that P is n x m x 3.
% Rows and columns of 0 are added to the "end" of P, if needed.

% save original size of P
[rows,columns,colours] = ;

% determine the number of "extra" rows and columns in P
rm15 = ;
cm15 = ;

% Add 15-rm15 rows of zeros to each of the colour matrices
% The number of rows in each of P1, P2, P3 is now a multiple of 15.
if rm15 > 0
P_pad(:,:,1) = ;
P_pad(:,:,2) = ;
P_pad(:,:,3) = ;
else
P_pad = P;
end;

% Add 15-cm15 columns to the already enlarged matrices
% The number of columns in each of P1, P2, P3 is now a multiple of 15.
[rows,c1] = ;

if cm15 > 0
P_padded(:,:,1) = ;
P_padded(:,:,2) = ;
P_padded(:,:,3) = ;
else
P_padded = P_pad;
end
end
```

b) [5 marks] Write a MATLAB function `process_block` that consumes a 15x15 matrix, B, and a nonnegative value `tol`. The function will produce a new 15x15 matrix obtained as follows:

- Convert B to frequency space using `fft2`, creating a new matrix FB.
- Set all entries in FB(i,j) to 0 if FB(i,j) < `tol`.

- Convert the "compressed" FB back to the original (a matrix B1) domain using `ifft2`.
- In the process of transforming the values back and forth, it is now possible that a value is complex or lies outside the range [0,255]. For each value B1(i,j), convert back to an unsigned real-valued integer, using the `uint8` and `real` conversion operations.

The function `process_block` will also return the number of values in FB which are now 0. Start with the code given below and fill the empty spaces;

```
function [ newB, num_zeros ] = process_block( B, tol )
% Applies compression algorithm to a 15x15 block of integers
%   and produces a new image matrix and a count of zeros in
%   the compressed coefficient matrix.
%
% Consumed values:
%   B is a 15x15 image matrix
%   tol is a nonnegative number used in the compression of the
%     Fourier coefficients (only coefficients greater than tol are
%     maintained)

% Produced values:
%   newB is a 15x15 image matrix, resulting from applying ifft2 to our
%     compressed coefficient matrix (type uint8)
%   num_zeros is the number of coefficients that are 0 in the
%     compressed coefficient matrix (some of them may have been 0
%     originally)

% Apply 2-d fft to the 15x15 block
F = ;

% Discard ...
maxValue = ;
mask = ;
newF = ;
%F
% Apply the inverse operation to this "compressed" matrix
newB = ;
num_zeros = ;
end
```

c) [5 marks] Write a MATLAB function `compress_image` that consumes P, a matrix corresponding to an image (as described previously) and `tol`, a non-negative value, and produces a new matrix the same size as P, using the following process:

- Use `pad15` to create a padded version of P,

- Apply `process_block` to each 15x15 block of the padded version of P, to create a new matrix Q, which is the same size as the padded P, and whose 15x15 blocks were created by the `process_block` process. This must be done for each of the three colours.
- Trim any additional rows and columns from ”compressed” version of P, so that the produced matrix is the same size as P (i.e. go back to the original image size).
- You will need to ensure that the final image matrix contains `uint8` values (not `double` or ”standard” integer values). It will not display directly with `image` unless this is the case.

The function `compress_image` should also produce the percentage (as a value between 0 and 100) of the entries in the compressed Fourier transform matrices which were set to 0. Use the size of the padded matrices when calculating your percentages. And remember to combine the number of zeros in each colour matrix to get your final answer. This is your *compression rate*. Start with the code given below and fill the empty spaces;

```
function [ compressedP, compression_rate ] = compress_image( P, tol )
% Compresses an image setting small Fourier coefficients to 0
%
% Consumed values:
%   P is a n x m x 3 colour image matrix
%   tol is a nonnegative number used in the compression of the
%     Fourier coefficients (only coefficients greater than tol are
%     maintained)
%
% Produced values:
%   compressedP is the n x m x 3 colour image matrix, which is the same
%     size as P, but which is constructed from the compressed Fourier
%     coefficients of P
%   compression_rate is a number between 0 and 100, which gives the
%     percentage of zero Fourier coefficients set to 0 (i.e. whose
%     values were less than tol).
%     compression_rate is calculated as the total number of Fourier
%     coefficients set to 0 in the compression process divided by the
%     number of pixels in the padded image matrix of P.
%

% save the original size of P
[rows,columns,colours] = ;

% Pad matrix is assist with 15x15 block processing
paddedP = ;
padded_rows = ;
padded_columns = ;
```

```
% initialize compressedP to all uint8 zeros.
compressedP = ;
num_zeros = 0;

% for each colour
for colour = 1:3
% Divide paddedP(:,:,colour) into 15x15 blocks, processing each in term,
% and saving the result in compressedP.
rows_of_15 = ;
cols_of_15 = ;
for r = 0:rows_of_15 - 1
  for c = 0:cols_of_15 - 1
    block = ;
    [modified,nz] = ;
    num_zeros = ;
    compressedP(15*r+1:15*r+15,15*c+1:15*c+15,colour) = ;
  end
end
end

% Trim any extra rows and columns of P
compressedP = ;
compression_rate = ;
end
```

d) [5 marks] Pick some jpeg/jpg images (Note that the images must not be offensive. If you choose to submit images that are offensive to course staff it may be pursued as a violation of our academic integrity policy. Use your imagination but I trust you to act responsibly.) and for each one;

- Apply `compress_image` with various tolerances to try to find values corresponding to compression rates of 50%, 80%, and 95%.
- Plot the original image, and the three "compressed" images, on a single page (use MATLAB's subplot command). Clearly label the compression rate associated with each image on your plot.

Start with the provided function `plotImages` that should assist you with this process. Include your version of `plotImages` with your solution in addition to the plots.

```
function [ ] = plotImages( image_file)
% plotImages plots the image in image_file with various levels
%   of compression
%
% Consumed value:
% image_file is the name of a jpg/jpeg file in the current
%    folder that will be displayed with approximately 50, 80, and 95%
%    compression

orient landscape;

% Experiment with the values of tol50, tol80, tol95
% to achieve compressions rates close to 50%, 80%, and 95%

P = imread(image_file);
subplot(2,2,1);
image(P);
title('Original Image');
axis off;

subplot(2,2,2);
tol50 = 29; % You have to adjust this value
[cP50, comp50] = compress_image(P, tol50);
image(cP50);
title(strcat('Compression Rate: ',num2str(comp50)));
axis off;

subplot(2,2,3);
tol80 = 116; % You have to adjust this value
[cP80, comp80] = compress_image(P, tol80);
image(cP80);
title(strcat('Compression Rate: ',num2str(comp80)));
axis off;

subplot(2,2,4);
tol95 = 2000; % You have to adjust this value
[cP95, comp95] = compress_image(P, tol95);
image(cP95);
title(strcat('Compression Rate: ',num2str(comp95)));
axis off;
end
```

3. [8 mark] This question is about implementing various Newton-Cotes methods. Start with implementing;

    - Trapezoid Rule and
    - Simpson's Rule.

    These implementations will be building blocks for composite versions of same rules. Implement composite version of these rules using what you already implemented. Submit all the codes you wrote. Very important note; implement means you write the codes.

For example using already existing MATLAB command `integral(fun,xmin,xmax)` or `trapz(X,Y)`, or other existing MATLAB commands which implements these methods but I am not aware of, in your codes is NOT implementing these rules, it is called wrapping, it is useful but it is NOT what we are asking for. You will lose all the points in that case. Example headers;

```
function v=myTrapz(f,xmin,xmax)
% This function approximates the integral
%    \int_{xmin}^{xmax} f(x)dx
% using the trapezoidal rule.
%
% Inputs
%   f    - function handle to integrand
%   xmin - lower bound of integral
%   xmax - upper bound of integral
%
% Output
%   v    - value of the integral

function v=myCtrapz(f,xmin,xmax,n)
% This function approximates the integral
%    \int_{xmin}^{xmax} f(x)dx
% using the composite trapezoidal rule by dividing the interval
% into n subintervals.
%
% Inputs
%   f    - function handle to integrand
%   xmin - lower bound of integral
%   xmax - upper bound of integral
%   n    - number of subintervals
%
% Output
%   v    - value of the integral
```

Note that, you can easily vectorize composite versions. Try to do that as it is a huge performance improvement. Consider Runge's function $R(x) = \frac{1}{1+25x^2}$ and its n-th degree Lagrange interpolating polynomial $P_n(x)$ obtained from 3,5,7,9,11 and 15 equidistant points in the interval $I = [-1, 1]$. Using the composite Simpson's rule over 100 subintervals execute the integral below;

$$\int_{-1}^{1} |R(x) - P_n(x)| dx.$$

Summarize the results in a table.