ENEE408G Multimedia Signal Processing: Introduction to MATLAB Programming

Version 0608

1 Introduction

MATLAB is a powerful computing, simulation, and prototyping tool with a wide range of applications including multimedia signal processing. Documentation and up-to-date technical information can be found online at `http://www.mathworks.com/`. The computer classrooms in the UMD School of Engineering and ECE Department have the full version of MATLAB installed. Students interested in using MATLAB on their own computers may purchase a student version at the University Book Center or online at `http://www.mathworks.com/academia/student_version/`.

To begin a MATLAB session, click on the MATLAB icon from the Start Menu or the Desktop. Upon startup, the interface shown in Fig. 1 will appear[1].

1.1 MATLAB Features

Command Window. Use the Command Window to enter commands, and run functions and M-files.

Command History. The commands that you have entered in the Command Window are automatically logged in the Command History window. You can view and execute previously used functions.

Start Button and Launch Pad. The MATLAB Launch Pad provides easy access to tools, demos, and documentation.

Help Browser. Use the Help browser to search and view documentation for all MathWorks products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents. To open a Help browser, click the help button in the toolbar, or type `helpbrowser` in the Command Window.

Current Directory Browser. MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path. To search, view, open, or make changes to MATLAB-related directories and files, use the MATLAB Current Directory browser. Alternatively, you can use the commands `dir`, `cd`, and `delete` in the Command Window.

[1]For more details, please refer `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml`

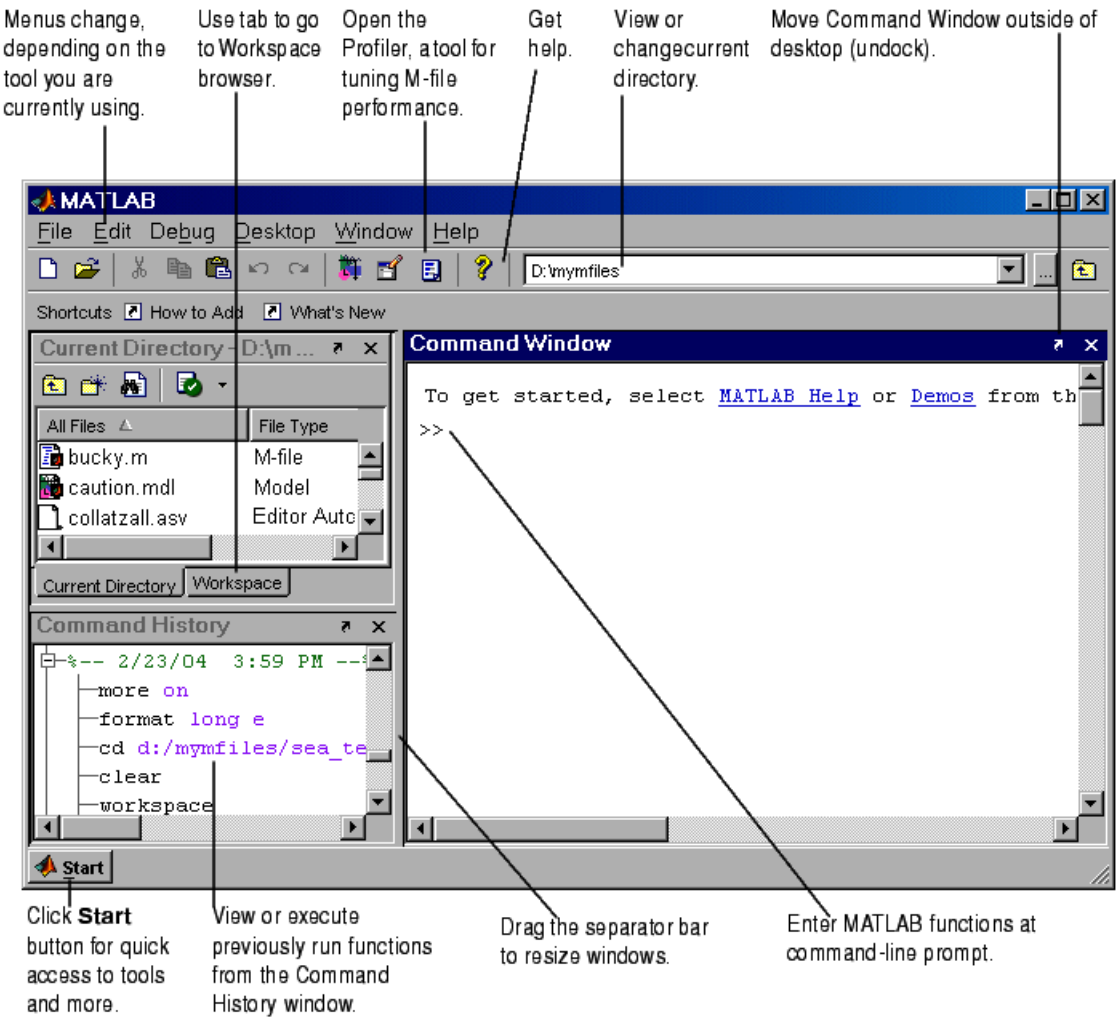


Figure 1: Default layout of the MATLAB desktop.

After starting MATLAB please change the current directory to your own working directory rather than staying in the default MATLAB directories. Otherwise, you could accidentally overwrite some important files!

Workspace Browser. The MATLAB workspace consists of the variables declared during a MATLAB session. Variables are added to the current workspace when declared in the Command Window, declared in a script, or loaded previously saved workspaces. The Workspace Browser also displays information about each variable. Alternatively, you may use the commands `who` and `whos`.

Array Editor. Double-click a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.

Editor/Debugger. Use the Editor/Debugger to create and debug M-files which contain MATLAB

functions or scripts. The Editor/Debugger provides a graphical user interface for basic text editing, as well as for M-file debugging.

You can use any other text editor to create M-files, such as Emacs and Notepad, and change the preferences (accessible from the MATLAB File menu under *Preferences*) to specify that editor as the default. If you choose to use other editors, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as `dbstop`, which sets a breakpoint.

If you just need to view the contents of an M-file, you can display it in the Command Window by using the `type` command.

2 Using MATLAB

Since most of you are already familiar with MATLAB in previous courses, this section provides only a brief review of a few important points of using MATLAB.

2.1 Vectors and Matrices

MATLAB has become a popular software tool of linear algebra, numerical analysis, and visualization. Much of its power lies in its highly optimized operations on vectors and matrices. In many cases, you should be able to eliminate the `for` loops you used to write in C code with MATLAB's simple and fast vectorized syntax[2]. Here is an example to compute the cosine of 10001 values ranging from 0 to 10.

```
i = 0;
for t = 0:.001:10
    i = i+1;
    y(i) = cos(t);
end
```

Here is a vectorized version of the same code.

```
t = 0:.001:10;
y = cos(t);
```

It is important for you to use vector operations whenever possible, since `for` loops are not optimized (in MATLAB 6.1 or lower) and could be very slow.

The second method to improve code execution time is to preallocate arrays that store output results. Here is an example to preallocate a row vector with 100 elements.

[2]If you are using MATLAB 6.5 or higher, before spending time to vectorize your code, please refer to `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_prog/ch7_per6.shtml#773530`. You may be able to speed up your program considerably by using the MATLAB JIT Accelerator.

```
y = zeros(1,100);
```

Preallocation makes it unnecessary for MATLAB to resize an array each time you enlarge it, and it also helps reduce memory fragmentation if you work with large matrices.

Another tip to speed up performance is to implement your code in a function rather than a script. Every time a script is used in MATLAB, it is loaded into memory and evaluated one line at a time. Functions, on the other hand, are compiled into pseudo-code and loaded into memory all together, so that repeated calls to the function can be faster.

2.2 Help for Using MATLAB

Type `help` *function_name* at the MATLAB Command Window to get help on a specific function. For a nicer interface to the help utility, click the question mark button on the desktop menu bar as explained above. Or, from the Help menu item, choose *Help Desk* to get to online HTML help.

From the Help Desk, you can search the online documentation to get help on how to accomplish tasks for which you may not know the specific function names. You can also try the `lookfor` command at MATLAB Command Window.

As a MATLAB programmer, the `help` command is your best friend. Use it often!

2.3 Saving and Loading Data

MATLAB data can be stored into `.mat` files on your disk. To save your whole workspace in a file called `filename.mat`, type `save filename`. To save one particular variable called `variable_name` into `filename.mat`, type `save filename variable_name`.

Saving will overwrite whatever filename you specify. To append instead of overwrite, use `save filename variable_name -append`.

Note that files are saved in a special binary format and are therefore unreadable by other applications. You can use `save ... -ascii` to save your workspace as a text file. See `help save` for more details.

To load a workspace from `filename.mat` file, we can type `load filename`. Again, you can load specific variables using `load filename variable_name`. See `help load` for more details.

2.4 Writing MATLAB Programs

Procedures that you call repeatedly can be stored either as function or script. Both can be created by writing M-files in your favorite text editor and storing them in your working directory.

Scripts do not accept input arguments or return output arguments. They operate on data in the

workspace, and they can create new data on which to operate. When you invoke a script, MATLAB simply executes the functions found in the file. Although scripts do not return output arguments, any variables that they create remain in the workspace and can be used in the subsequent computations.

Functions are M-files that can accept input arguments and return output arguments. The name of the M-file and of the function should be the same. Functions operate on variables within their own workspace, separate from the main workspace you access in the MATLAB Command Window. Here is an example function file, `MSE2PSNR.m`.

```
function PSNR=MSE2PSNR(MSE)
% Convert mean square error (MSE) into peak signal to noise ratio (PSNR)
% Input: MSE
% Output: PSNR
%
% Author: Guan-Ming Su
% Date: 8/31/02

A=255*255./MSE;
PSNR=10*log10(A);
```

A function consists of three parts. The first line of a function M-file starts with a keyword `function` and must use the following syntax:

```
function [out1, ..., outN] = function_name( input1, input2, ..., inputN )
```

The final value of variables `out1, ..., outN` will be automatically returned once the function execution is finished.

The next several lines which begin with `%`, up to the first blank or executable line, are comments that provide the help text. In other words, these lines are printed when you type `help MSE2PSNR` in the MATLAB Command Window. In addition, the first line of the help text is the H1 line, which MATLAB displays when you use the `lookfor` command or request help on a directory.

The rest of the file is the executable MATLAB code defining the function. The variable `A` is introduced in the body of the function, as well as the variables `MSE` and `PSNR` on the first line of the function. These variables are all local to the function; their scope is separate from any variables in the MATLAB workspace.

3 Multimedia Programming

3.1 Audio Representation and Playback

MATLAB supports multi-channel wave format with up to 16 bits per sample. To load a wave file, you can use `[Y, Fs, Nbits] = wavread(wave_filename)`, where `wave_filename` is the file name of the wave file, `Y` the sampled data with dimensions (number of samples)-by-(number of channels), `Fs` the sampling rate in Hz, and `Nbits` the number of bits per sample used to encode the data in the file. Amplitude values in `Y` vector are normalized to the range [-1, +1] according to the formula

$$Y = \frac{X}{2^{N_{bits}-1}} - 1$$

where X is a sample from the input wave file with N_{bits} bits per sample, and Y is the corresponding output sample. For instance, if $N_{bits} = 8$, then $X = 0$ yields $Y = -1$, and $X = 2^8 - 1$ yields $Y = 0.9921875$.

To generate a wave file and store it in the hard disk, you can use `wavwrite(Y, Fs, Nbits, wave_filename)`. To playback the signal in vector `Y`, use `sound(Y, Fs, Nbits)`.

3.2 Image Representation and Display

In MATLAB a grayscale image is simply a matrix with positive real-valued entries. Each entry represents one pixel. These entries may have `double` precision, i.e. 64-bit floating point, with a range of $[0, 1]$, or they may have `uint8` precision, i.e. 8-bit unsigned integer, with a range of $\{0, 1, ..., 255\}$. The `uint8` representation saves storage space and speeds up computation. We will use both representations in our lab assignment.

To display a grayscale image, use `imshow(image_name, [Low High])`, where the value `Low` (and any value less than `Low`) will be displayed as black, the value `High` (and any value greater than `High`) as white, and values in between as intermediate shades of gray. An image stored as `uint8` can be displayed directly using `imshow(image_name)` without any extra arguments. See `help imshow` for details.

If you wish to display a transformed image using the DCT or DFT, you may want to set the DC value to zero beforehand to make it easier to see the AC values that are usually a few orders of magnitude smaller than the DC. You could also try viewing the log of the DCT or brightening the color map with `brighten`.

4 Miscellaneous Tips

In addition to such basic commands as `for`, `end`, `if`, `while`, `;`, `==`, and `=`, you may find the following functions and concepts useful for this course.

`fft2` A two-dimensional Fast Fourier Transform routine. Make sure to use this command for

2-D images, not the one-dimensional transform `fft`! See the help text for these two commands to understand the difference.

Ctrl-C Stops execution of any command that went awry.

`clear all` Remove all variables, globals, functions and MEX links.

`close all` Close all the open figure windows.

`max` For vectors, `max(X)` is the largest element in `X`. For matrices, it gives a vector containing the maximum element from each column. To get the maximum element of the entire matrix `X`, try `max(X(:))`. Other functions such as min, mean, and median can be used in a similar manner.

`abs(X)` The absolute value of elements in `X`.

Indexing. Use the colon operator and the `end` keyword to get at the entries of a matrix easily. For example, to obtain every 5th element of a vector `a`, use `a(1:5:end)`. See `help colon` and `help end`.

`strcat(S1,S2,S3,...)` Concatenate the corresponding rows of the character arrays `S1`, `S2`, `S3` etc.

`num2str(X)` Convert a matrix `X` into a string representation. This is useful for labeling plots.

`flipud` Flip matrix vertically, i.e. reverse the order of elements in each column.

`fliplr` Flip matrix horizontally, i.e. reverse the order of elements in each row.

`tic`, `toc` `tic` starts a stopwatch timer and `toc` reads the stopwatch timer. Use these to monitor the executing time of your program.

`waitbar` This function can display the progress of your program when you use loops, such as a for loop.

`pause` This function causes a procedure to stop and wait for the user to strike any key before continuing.

`find` Find indices of nonzero elements.

`B = repmat(A,M,N)` Replicate and tile the matrix `A` to produce the `M`-by-`N` block matrix `B`. For example, let `A=[1 2 3]`. The result of `repmat(A,3,2)` is `[1 2 3 1 2 3; 1 2 3 1 2 3; 1 2 3 1 2 3]`.

Deleting rows and columns. You can delete a row or column by assigning it as a pair of square bracket. For example, if `A=[1 2 3;4 5 6;7 8 9]`, we can delete the second column using `A(:,2)=[];`. The resulting matrix `A` is `[1 3; 4 6; 7 9]`.

5 Examples on Digital Audio Processing

Let us first examine a few examples related to audio and image processing as warm-up exercises. Start MATLAB and type `Edit` in the command window. You will see a M-file Editor. We will use this editor to write M-files.

5.1 Read, Playback and Visualize an Audio Signal

Download the audio file `symphonic.wav` from the course web site. Make sure it is in your working directory.

Read an audio file into a matrix called `Music` using the function wavread.

```
[Music, Fs, Nbits] = wavread('symphonic.wav');
```

Obtain the dimension of this image.

```
[MusicLength, NumChannel] = size(Music);
```

The function `size` will return the number of samples and number of channels of this audio file into `MusicLength` and `NumChannel`.

Playback the audio vector. Make sure your speaker or earphone is on.

```
sound(Music, Fs, Nbits);
```

Visualize the waveform. You can adjust the display range by changing `Display_start` and `Display_end`.

```
Display_start = 1;
Display_end = MusicLength;
subplot(2,1,1); plot(Music(Display_start: Display_end,1));
title('First channel');
subplot(2,1,2); plot(Music(Display_start: Display_end,2));
title('Second channel');
```

5.2 Bit Manipulation

Convert the format of `Music` from double to unsigned integer with `Nbits` bits.

```
IntMusic = (Music+1)*power(2,Nbits-1);
```

Many music files use 16 bits to represent an audio sample. In this case, `Nbits` = 16.

Extract the lower byte from the first channel.

```
LowNbits = Nbits/2;
LowIntMusicCh1 = zeros(MusicLength,1);
FirstChannel = IntMusic(:,1); % extract the first channel
for ibit = 1:1:LowNbits
    LowIntMusicCh1 = LowIntMusicCh1 + bitget(FirstChannel, ibit)*power(2,ibit-1);
end
```

Convert unsigned integer to the normalized expression and listen to it.

```
LowRecMusicCh1 = LowIntMusicCh1/power(2,LowNbits-1) - 1;
sound(LowRecMusicCh1, Fs, LowNbits);
```

Repeat the procedure for the second channel and store the final result in `LowRecMusicCh2`. What do you hear?

5.3 Window-based Audio Processing

Many audio/speech processing techniques divide audio signals into segments before further processing, as depicted in Fig. 2.

As we can see, a sliding window is used to extract a set of data from the original data stream and the adjacent windows have some overlap with each other. For the example in Fig. 2, the window size is 6 and the overlap size is 2. The first six samples are extracted as the first segment, and the start point of the second segment is the 5th sample.

In this example, we segment the lower bytes of the second channel, `LowRecMusicCh2`, with window size 2^{15} and zero overlapping. Then re-order samples in each window using `flipud`.

```
LenWindow = power(2,15);                % Length of window
LenOverlap = 0;                         % Length of overlapping
LenNonOverlap = LenWindow - LenOverlap;
NumSegCh2 = floor((length(LowRecMusicCh2)-LenWindow)/(LenNonOverlap)) + 1;
for iseg=1:1:NumSegCh2
    seg_start = (iseg-1)*LenNonOverlap+1;    % start point of current window
    seg_end = seg_start + LenWindow - 1;     % end point of current window
    LowRecMusicCh2(seg_start:seg_end) = flipud(LowRecMusicCh2(seg_start:seg_end));
```

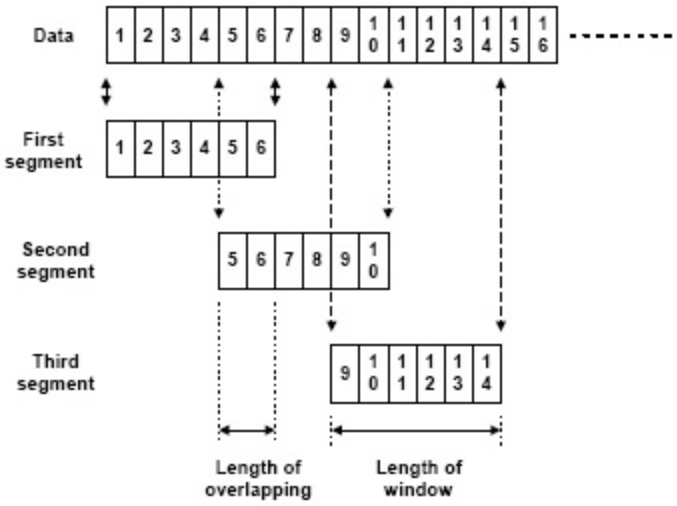


Figure 2: Segmentation of an audio signal.

```
end
sound(LowRecMusicCh2, Fs, LowNbits);
```

Can you hear the "secrets"?

6 Examples on Digital Image Processing

6.1 Read and Display an Image File

Download the `CuteBaboon.bmp` image file from the course web site. Make sure it is in your working directory. You can read an image file into a matrix `Im` using `imread`.

```
Im = imread('CuteBaboon.bmp');
```

Obtain the dimension of this image. The function `size` will return the dimension of this 2-D matrix, `Im`, into `height` and `width`.

```
[height, width] = size(Im);
```

Display the image.

```
imshow(Im, [0 255]);
```

Since each pixel in this image has 8 bits, the gray level range is between 0 and 255. We specify this range as the second argument in the `imshow` function.

6.2 Bit Planes

As we have mentioned, each pixel is represented using 8 bits. We can put together the most significant bit (MSB) of each pixel and form the first bit plane, as shown in Fig. 3. Similarly, we can extract the second MSB of each pixel to form the second bit plane, and so on.

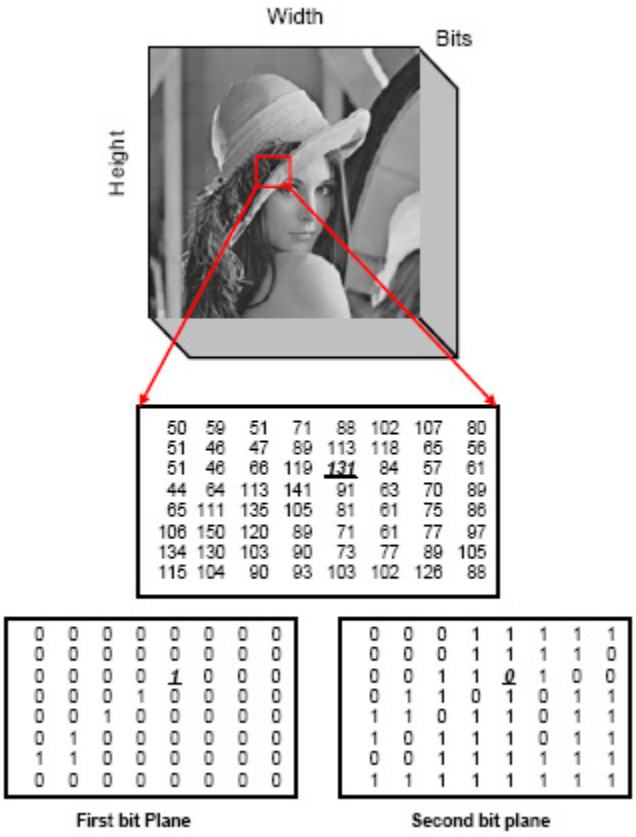


Figure 3: Two most significant bit planes of the image *Lena*, i.e. $131(10) = 10000011(2)$.

Extract MSB plane using MATLAB's library function `bitget`.

```
msbIm = bitget(Im, 8);
```

Display the bit plane.

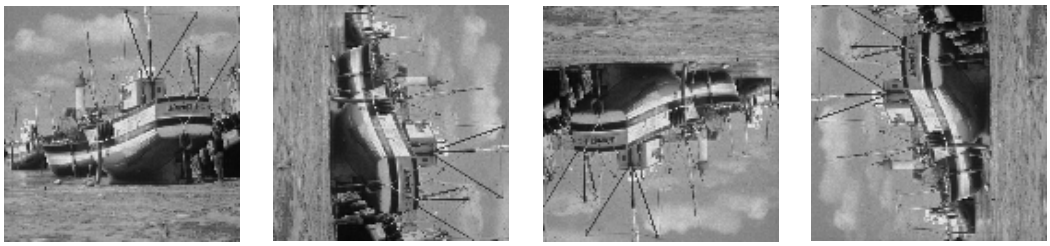
```
imshow(msbIm, [0 1]);
```

Since each bit is represented by 0 or 1, the value range of a bit plane is $\{0, 1\}$.

Use your own MATLAB code to observe other bit planes. As you proceed from MSB to LSB, you will find that the bit plane becomes more noise-like.

6.3 Image Rotation

We can rotate images by using MATLAB's built-in matrix operations. Fig 4 shows the relation between MATLAB functions and image rotations.



(a) `Im` (b) `fliplr(Im')` (c) `flipud(fliplr(Im))` (d) `flipud(Im')`

Figure 4: Image rotations and the associated commands.

Let's try image rotation on the 2nd LSB bit plane of `Im`.

Extract 2nd least significant bit plane of `Im`.

```
lsb2Im = bitget(Im, 2);
imshow(lsb2Im, [0 1]);
```

Rotate each 16-by-16 block.

```
bs = 16;
for i=1:1:height/bs
    for j=1:1:width/bs
        lsb2Im((i-1)*bs+1:1:i*bs,(j-1)*bs+1:1:j*bs) = ...
            flipud(lsb2Im((i-1)*bs+1:1:i*bs,(j-1)*bs+1:1:j*bs)');
    end
end
```

Display the result and see what "secret" message you can discover.

```
imshow(lsb2Im, [0 1]);
```

6.4 Image Down-Sampling

There are many ways to down-sample an image to reduce its size. The basic idea is to use one pixel to represent a small area that is usually in square shape.

Method 1: Use the top-left pixel of each square area to represent that area.

```
sIm = Im(1:8:end,1:8:end);
imshow(sIm, [0 255]);
```

Method 2: Take the average of each square area. Here we use a function call.

```
meansIm = mean_subfunction(Im);
```

To create this subfunction, select *New* from the File menu, type the following MATLABcode, and save it as `mean_subfunction.m`.

```
function Y=mean_subfunction(X)

[row col] = size(X);
Y = zeros(row/8, col/8);
for i=1:1:row/8
    for j=1:1:col/8
        Y(i,j) = mean2(X((i-1)*8+1:1:i*8, (j-1)*8+1:1:j*8));
    end
end
```

Observe this downsampled image.

```
imshow(meansIm, [0 255]);
```

Compare the result of methods 1 and 2. Which is a better way to do down-sampling?

6.5 Histogram

A histogram of a grayscale image shows the luminance distribution of this image. It tells us the statistics of how many pixels (or what percentage of pixels) an image has at each gray level. Although the MATLAB Image Processing Toolbox already provides histogram functions, we will write our own version here.

The `find` function is a very powerful function and can make your MATLAB program elegant if used properly. `I = find(X)` returns the indices of the vector `X` that are non-zero. The `length` function will return the number of elements in a vector.

We can use `find` and `length` together to obtain the number of pixels with a specific luminance value `igray`.

```
histogram = zeros(1,256);
for igray=0:1:255
    histogram(igray+1) = length(find(Im==igray));
end
plot(histogram);
axis([0 255 min(histogram) max(histogram)]);
title('Histogram');
xlabel('Luminance value');
ylabel('Number of pixels');
```

7 Exercises

7.1 Audio Steganography

The word "steganography" comes from Greek and means "secret writing". Here we apply steganography to audio to hide a secret and inaudible message in a music file.

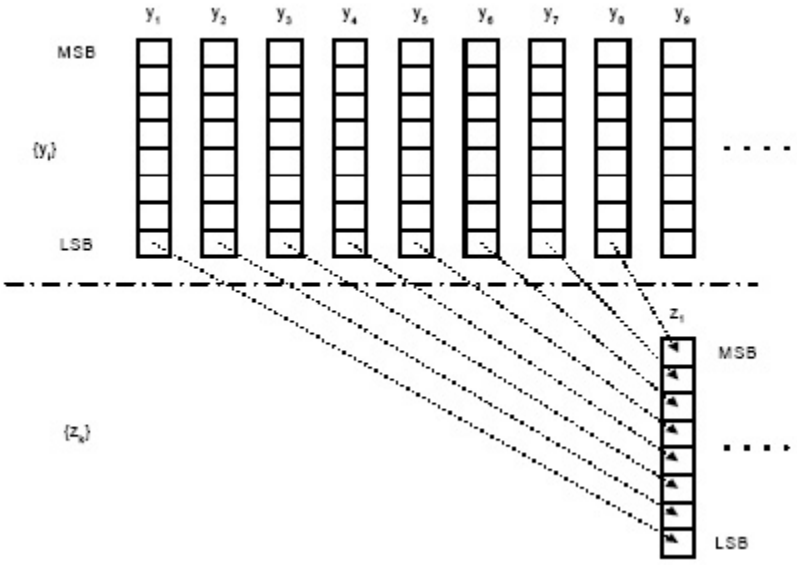


Figure 5: Extraction of the LSBs from an audio signal.

As illustrated in Fig. 5, a secret audio signal $z[n]$ is placed in the least significant bits of samples

from a host audio signal to produce a new audio signal, $y[n]$, which contains the secret signal. We can extract the hidden message $z[n]$ from $y[n]$ according to the following formula:

$$z[k] = \sum_{i=8(k-1)+1}^{8k} LSB(y[i]) \cdot 2^{i-8(k-1)-1} \qquad \text{for } k = 1, ..., N/8$$

where $LSB(y[n])$ represents the least significant bit of sample $y[n]$ and N is the total number of samples in signal $y[n]$.

Download `guitar.wav` from course web site and write a simple M-file to extract the hidden message $z[n]$.

7.2 Image Up-Sampling (Enlargement)

The simplest way to implement image upsampling is replication. To upsample an M-by-N image to $2M$-by-$2N$, each original pixel is expanded into four pixels of the same value. These four pixels are arranged in a 2-by-2 manner. Here is an example where A_2 is an upsampled version of A_1:

$$A_1 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix}$$

Download `Girl.bmp` image file from course web site and write a simple M-file to enlarge it without using any `for` loops.

Hint. You may find MATLAB's built-in Kronecker Product function `kron` useful. If A is an M-by-N matrix and B is any another matrix, then their Kronecker product is defined as

$$A \otimes B \triangleq \{a(m,n)B\} = \begin{bmatrix} a(1,1)B & \cdots & a(1,N)B \\ \vdots & & \vdots \\ a(M,1)B & \cdots & a(M,N)B \end{bmatrix}$$

7.3 Erroneously Indexed Image

In the image `Noise.bmp`, the indexing of bits in each pixel is wrong. We want to rearrange the bits in each pixel to obtain the correct image. Thankfully, we know the relation between indices of the correct image and that of the erroneously indexed image, `Noise.bmp`. If we represent the correct luminance values in binary form as

$$L_{orig} = a_7 2^7 + a_6 2^6 + a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$$

then the luminance values of the erroneously indexed image are expressed as the following:

$$L_{err} = a_0 2^7 + a_1 2^6 + a_2 2^5 + a_3 2^4 + a_4 2^3 + a_5 2^2 + a_6 2^1 + a_7 2^0$$

Download the image file `Noise.bmp` and recover the original image.