ECE 3640 - Discrete-Time Signals and Systems
Matlab: Basic Knowledge and Skills

Jake Gunther

Spring 2015

UtahStateUniversity

Department of Electrical & Computer Engineering

outline

- signal input and output using Matlab's media read and write functions
  - audio: wavread/wavwrite and audioread/audiowrite
  - images: imread/imwrite
  - videos: videoReader/videoWriter
- multidimensional arrays in Matlab
- Matlab low-level file I/O: fopen, fclose, fread, fwrite (interface to C)
- `plot` function
  - plotting and signal (one independent variable)
  - embellishments
  - handle graphics
- `image` and `imagesc` function
  - plotting and image (two independent variables)
  - embellishments
  - handle graphics
- Fourier transforms via `fft` and spectral plots
- `spectrogram` function and time-frequency plots

`help` in Matlab

getting help

- `help` `function` - brings up information about the function
- `lookfor` `keyword` - searches for that keyword throughout the help system
- `doc` - brings up the documentation system
- `helpdesk`
- `www.mathworks.com`
- ask a human
- ask the Internet (i.e. Google)

example

```
>> help relop
 Relational operators.
  < > Relational operators.
      The six relational operators are <, <=, >, >=, ==, and ~=.
      A < B does element by element comparisons between A and B ...

  &   Element-wise Logical AND.
      A & B is a matrix whose elements are logical 1 (TRUE) where both A
      and B have non-zero elements, and logical 0 (FALSE) where either has ...

  &&  Short-Circuit Logical AND.
      A && B is a scalar value that is the logical AND of scalar A and B.
      This is a "short-circuit" operation in that MATLAB evaluates B only ...

  |   Element-wise Logical OR.
      A | B is a matrix whose elements are logical 1 (TRUE) where either
      A or B has a non-zero element, and logical 0 (FALSE) where both have ...

  ||  Short-Circuit Logical OR.
      A || B is a scalar value that is the logical OR of scalar A and B.
      This is a "short-circuit" operation in that MATLAB evaluates B only ...

  ~   Logical complement (NOT).
      ~A is a matrix whose elements are logical 1 (TRUE) where A has zero
      elements, and logical 0 (FALSE) where A has non-zero elements.

  xor Exclusive OR.
      xor(A,B) is logical 1 (TRUE) where either A or B, but not both, is
      non-zero.  See XOR.
```

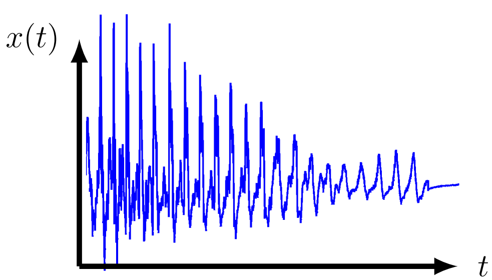
useful Matlab commands

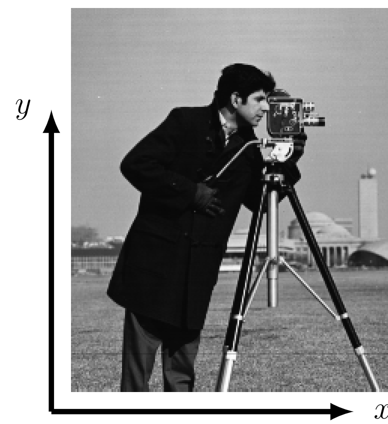
- `pwd` - show the present working directory
- `dir` and `ls` - lists all files in the current directory
- `cd ./bob` and `chdir ./bob` - changes to the `bob` subdirectory
- `what` - lists all m-files in the current directory
- `which test` display path to `test.m`
- `type test` - display `text.m` in command window
- `delete test` - deletes file `text.m`
- `who` and `whos` - displays all variables in the workspace
- `save` - saves all the variables in the workspace
- `load` - loads variables into the workspace
- `clear all` - clears all variables from the workspace
- `clear x` - clears only the variable `x` from the workspace

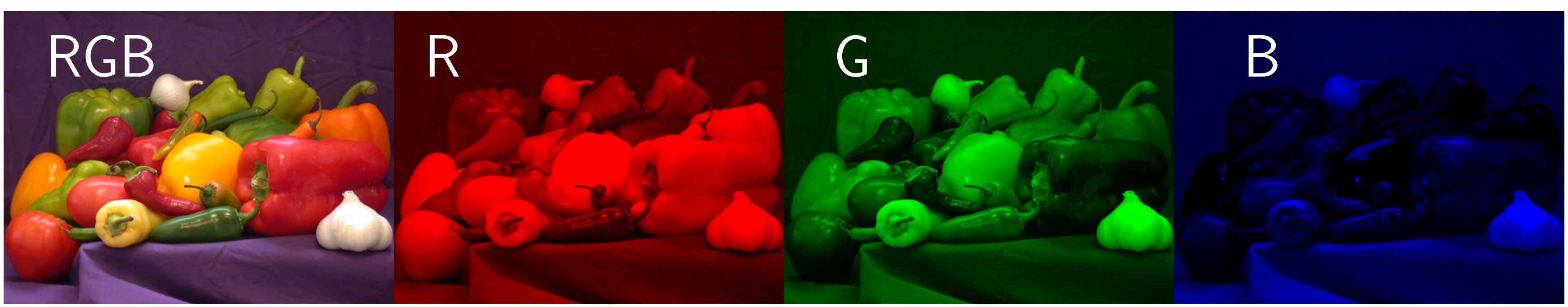
signal input and output in Matlab

different kinds of signals

signal	dimension	number channels
audio (monaural)	1	1
audio (stereo)	1	2
image (grayscale)	2	1
image (RGB color)	2	3
video (grayscale)	3	1
video (RGB color)	3	3



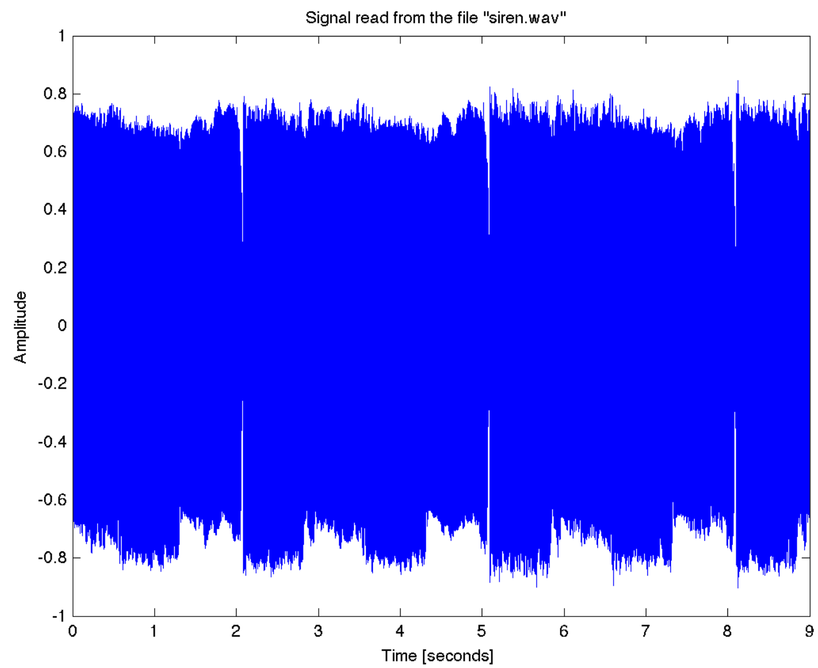




media: read audio signals

```
ai = audioinfo('../siren.wav');
disp(ai); % display audio header structure
[x,fs] = audioread('../siren.wav',[1 10]*ai.SampleRate); % read audio file
soundsc(x,fs); % play sound to speaker
t = [0:length(x)-1]/fs;
plot(t,x);
xlabel('Time [seconds]');
ylabel('Amplitude');
title('Signal read from the file "siren.wav"');
print -dpng siren.png
```

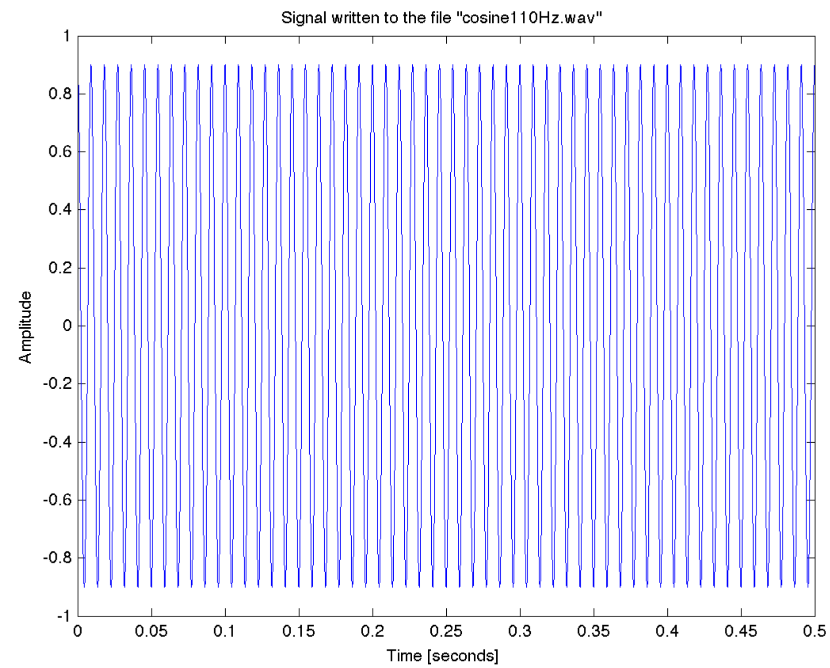
```
           Filename: 'siren.wav'
  CompressionMethod: 'Uncompressed'
        NumChannels: 1
         SampleRate: 44100
       TotalSamples: 2182685
           Duration: 49.4940
              Title: []
            Comment: []
             Artist: []
      BitsPerSample: 16
```



media: write audio signals

```
fs = 8000; % sample rate [samples/second]
t = [0:fs*0.5-1]/fs;
x = 0.9*cos(2*pi*110*t);
audiowrite('cosine110Hz.wav',x,fs);
ai = audioinfo('cosine110Hz.wav');
disp(ai);
soundsc(x,fs); % play sound to speaker
plot(t,x);
xlabel('Time [seconds]');
ylabel('Amplitude');
title('Signal written to the file "cosine110Hz.wav"');
print -dpng cosine110Hz.png
```

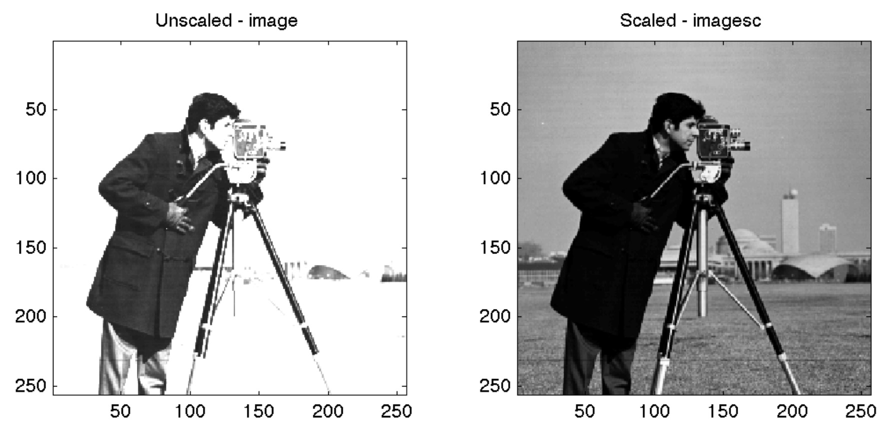
```
           Filename: 'cosine110Hz.wav'
  CompressionMethod: 'Uncompressed'
        NumChannels: 1
         SampleRate: 8000
       TotalSamples: 4000
           Duration: 0.5000
              Title: []
            Comment: []
             Artist: []
      BitsPerSample: 16
```



media: read images

```
clear all; clf;
%imfinfo('cameraman.tif')
x = imread('cameraman.tif');
subplot(121);
image(x);    % show the image
axis image; % make the pixels square
title('Unscaled - image');
subplot(122);
imagesc(x); % show scaled image
axis image; % make the pixels square
title('Scaled - imagesc');
colormap gray; % this is a grayscale image
print -dpng cameraman_scaling.png
[nrows,ncols] = size(x);
fprintf('This image is %d X %d pixels.\n\n',nrows,ncols);
whos
```

```
This image is 256 X 256 pixels.

  Name         Size            Bytes  Class     Attributes

  ncols        1x1                 8  double
  nrows        1x1                 8  double
  x          256x256           65536  uint8
```

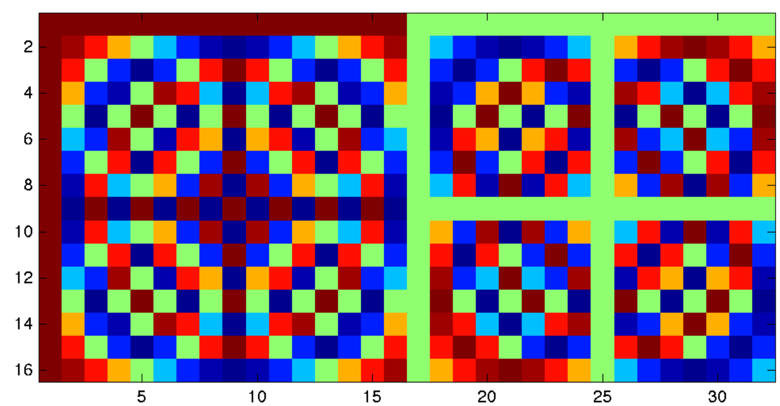


media: write images

```
clear all; clf;
x = dftmtx(16);
x = [real(x) imag(x)];
imwrite(x,'dftmtx.png','PNG');
imagesc(x);
axis image;
colormap(jet);
print -dpng dftmtx_matlab.png
[nrows,ncols] = size(x);
fprintf('This image is %d X %d pixels.\n\n',nrows,ncols);
whos
```

```
This image is 16 X 32 pixels.

  Name          Size            Bytes  Class     Attributes

  ncols         1x1                 8  double
  nrows         1x1                 8  double
  x            16x32             4096  double
```

(in the image file) ⟶ 

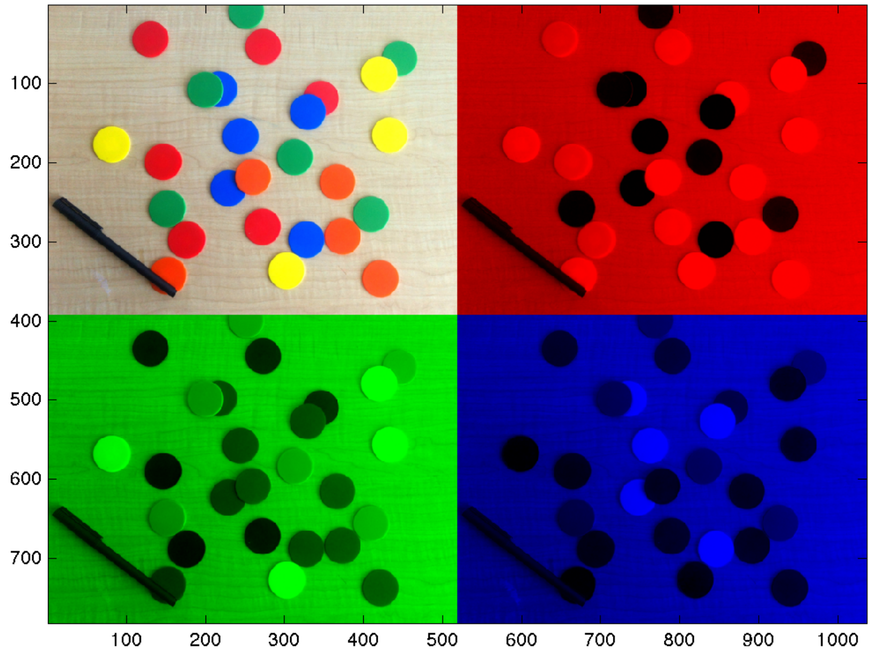


(in the Matlab figure window)

media: read color images

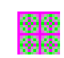
```
clear all; clf;
x = imread('coloredChips.png');
r = x; r(:,:,[2,3]) = 0;
g = x; g(:,:,[1,3]) = 0;
b = x; b(:,:,[1,2]) = 0;
image([x r; g b]);
axis image; % make the pixels square
print -dpng coloredChips_color_planes.png;
[nrows,ncols,nrgb] = size(x);
fprintf('This image is %d X %d pixels X %d color planes.\n\n',...
        nrows,ncols,nrgb);
whos
```

```
This image is 391 X 518 pixels X 3 color planes.

  Name        Size              Bytes  Class     Attributes

  b         391x518x3          607614  uint8
  g         391x518x3          607614  uint8
  ncols       1x1                   8  double
  nrgb        1x1                   8  double
  nrows       1x1                   8  double
  r         391x518x3          607614  uint8
  x         391x518x3          607614  uint8
```

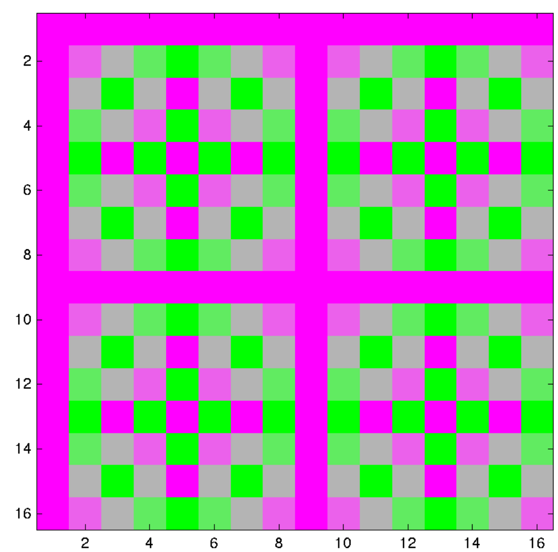


media: write color images

```
clear all; clf;
x = dftmtx(16);
r = abs(real(x));
g = abs(imag(x));
b = abs(real(x));
x = cat(3,r,g,b); % concatenate along the third dimension
imwrite(x,'dftmtx_color.png','PNG');
image(x);
axis image; % make the pixels square
print -dpng dftmtx_color_matlab.png;
[nrows,ncols,nrgb] = size(x);
fprintf('This image is %d X %d pixels X %d color planes.\n\n',...
         nrows,ncols,nrgb);
whos
```

(in the image file) ⟶ 

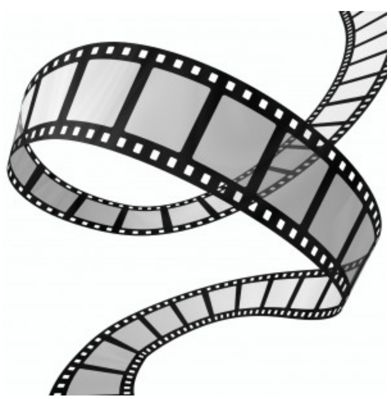
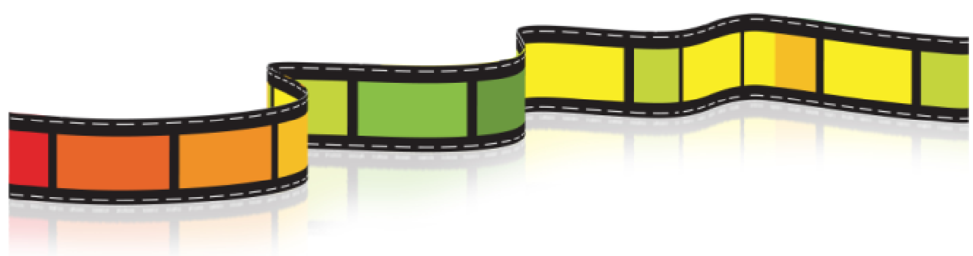
```
his image is 16 X 16 pixels X 3 color planes.

  Name         Size              Bytes  Class     Attributes

  b           16x16               2048  double
  g           16x16               2048  double
  ncols        1x1                   8  double
  nrgb         1x1                   8  double
  nrows        1x1                   8  double
  r           16x16               2048  double
  x           16x16x3             6144  double
```



(in the Matlab figure window)

media: videos

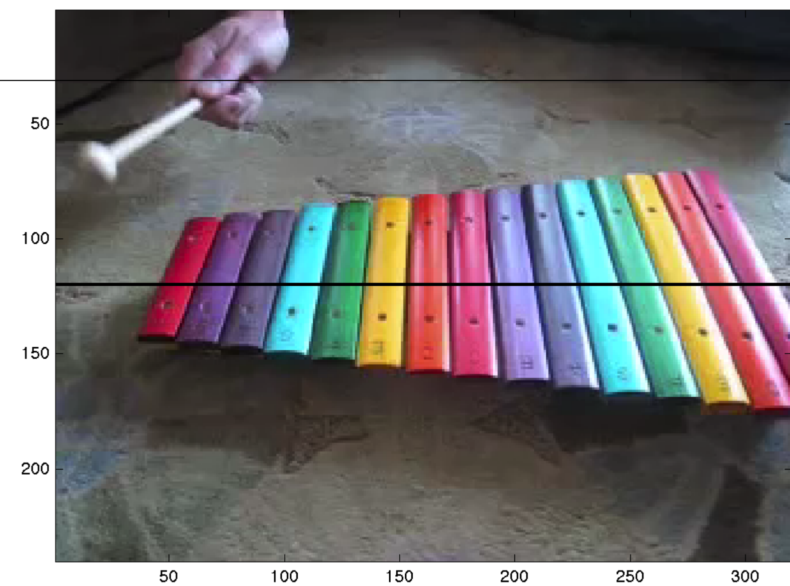
- grayscale videos are 3-dimensional: (row,col,frame)
- color videos are 4-dimensional: (row,col,rgb,frame)
- video files can be very large
- read and write one frame at a time

media: read videos

```
clear all; clf;
obj = VideoReader('xylophone.mp4', 'tag', 'myreader1');
x = read(obj); % read in all the frames
fprintf('video x: '); disp(size(x));
y = read(obj,20); % read in the i-th frame
fprintf('frame y: '); disp(size(y));
% read in the frames one at a time
nframes = obj.NumberOfFrames;
for i=1:nframes
  y = read(obj,i);
  if(i==1)
    imhan = image(y); axis image; drawnow;
  else
    set(imhan,'CData',y); drawnow;
  end
end
```

```
video x:    240   320     3   141

frame y:    240   320     3
```



(first frame of the video)

media: write videos

```
clear all; clf;
obj = VideoWriter('wave.mp4','MPEG-4'); % create video file
open(obj); % open video file
fs = 30; % sample rate [frames/second]
t = [0:60]/fs; y = [0:31]/50; x = [0:63]/50; [Y,X]=meshgrid(y,x);
for i=1:length(t)
  Zc = cos(2*pi*(t(i)*1 - X*1 - Y*0.75)).^2;
  Zs = sin(2*pi*(t(i)*1 - X*1 - Y*0.75)).^2;
  Z = cat(3,Zc,Zs,fliplr(Zc)); % make a video frame = RGB image
  writeVideo(obj,Z); % write the video frame
  if(i==1)
    imhan = image(Z); drawnow;
  else
    set(imhan,'CData',Z); drawnow;
  end
end
close(obj); % close video file
% See also "getframe" to make a video of the Matlab figure window.
```

- watch the video

multidimensional arrays in Matlab

MD arrays: terminology

- 1D array can be a row vector or a column vector

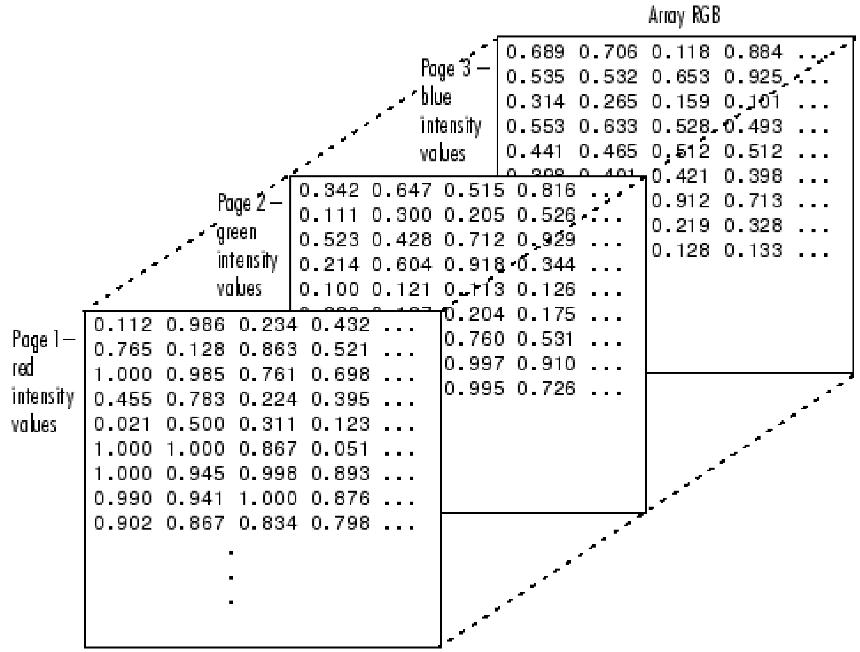
```
x = zeros(1,4)
y = ones(3,1)
```

```
x =
     0     0     0     0
y =
     1
     1
     1
```

- 2D array has both rows and columns

```
x = rand(2,3)
```

```
x =
    0.2621    0.7549    0.4424
    0.0445    0.2428    0.6878
```



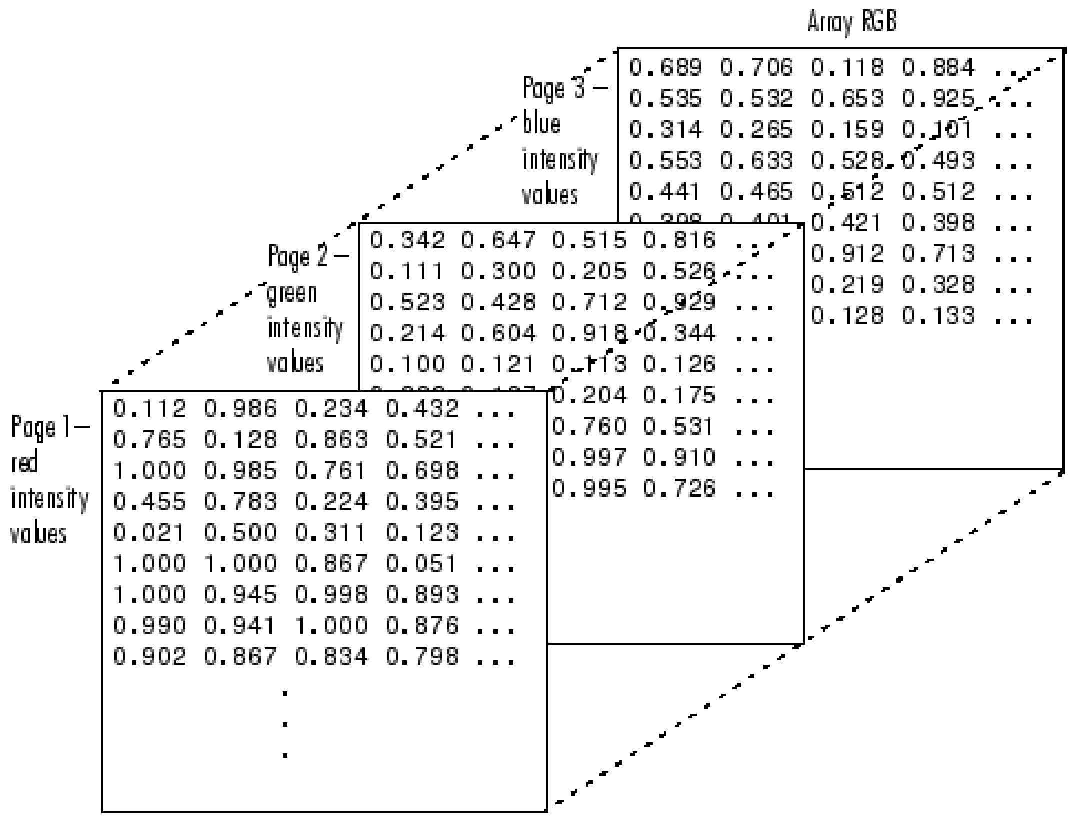
(3D array representing a color image)

- 3D arrays have rows, columns and pages/slabs (see picture)
- 4D arrays have rows, columns, pages, and books
- 5D arrays have rows, columns, pages, books and shelves
- 6D arrays have rows, columns, pages, books, shelves and aisles

creating MD arrays: built-in functions

```
x = floor(10*rand(2,5,6))   % create a random 4D array
n = ndims(x)                % return the number of dimensions
s = size(x)                 % return the size of x in each dimension
```

```
x(:,:,1) =
     9     7     1     4     6
     2     2     6     4     7
x(:,:,2) =
     3     4     8     6     5
     6     8     2     5     8
x(:,:,3) =
     2     1     6     6     6
     3     9     4     5     5
x(:,:,4) =
     7     9     1     0     4
     5     2     1     4     3
x(:,:,5) =
     7     7     9     1     0
     6     9     1     6     5
x(:,:,6) =
     5     4     6     5     1
     8     3     7     3     5
n =
     3
s =
     2     5     6
```

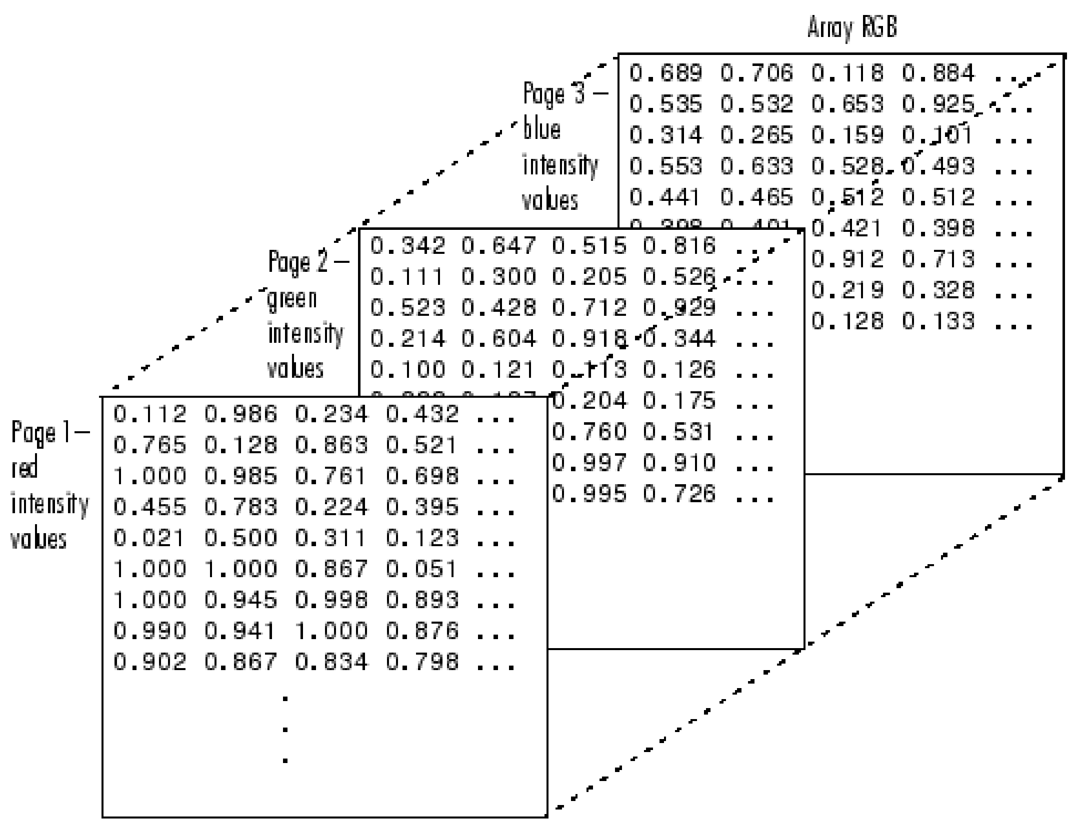


- functions `rand`, `randn`, `ones`, `zeros` produce MD arrays
- can do 4D `ones([2 3 4 5])` and 5D `zeros([4 2 20 75 8])` and so on

creating MD arrays: concatenation

```
a = floor(10*rand(3,4));
b = ones(3,4);
c = zeros(3,4);
d = round(10*randn(3,4));
x = cat(3,a,b,c,d)
n = ndims(x)          % return the number of dimensions
s = size(x)           % return the size of x in each dimension
```

```
x(:,:,1) =
     0     4     6     2
     1     1     1     9
     7     3     7     2
x(:,:,2) =
     1     1     1     1
     1     1     1     1
     1     1     1     1
x(:,:,3) =
     0     0     0     0
     0     0     0     0
     0     0     0     0
x(:,:,4) =
     2    -6     3     8
   -15     4    -4     1
    -7     9     8     6
n =
     3
s =
     3     4     4
```



creating MD arrays: replication

```
x = repmat(eye(3),[1,1,4])
n = ndims(x)            % return the number of dimensions
s = size(x)             % return the size of x in each dimension
```

```
x(:,:,1) =
     1     0     0
     0     1     0
     0     0     1
x(:,:,2) =
     1     0     0
     0     1     0
     0     0     1
x(:,:,3) =
     1     0     0
     0     1     0
     0     0     1
x(:,:,4) =
     1     0     0
     0     1     0
     0     0     1
n =
     3
s =
     3     3     4
```

- can do 4D `repmat(eye(3),[1 1 4 6])`
- can do 5D `repmat(eye(3),[1 1 4 100 35])` and so on

creating MD arrays: reshaping

```
x = reshape([1:48],[2 3 2 4])
n = ndims(x)           % return the number of dimensions
s = size(x)            % return the size of x in each dimension
```

```
x(:,:,1,1) =
     1     3     5
     2     4     6
x(:,:,2,1) =
     7     9    11
     8    10    12
x(:,:,1,2) =
    13    15    17
    14    16    18
x(:,:,2,2) =
    19    21    23
    20    22    24
x(:,:,1,3) =
    25    27    29
    26    28    30
x(:,:,2,3) =
    31    33    35
    32    34    36
x(:,:,1,4) =
    37    39    41
    38    40    42
x(:,:,2,4) =
    43    45    47
    44    46    48
n =
     4
s =
     2     3     2     4
```

- this example illustrates how Matlab's MD arrays are laid out in memory: the leftmost subscript varies fastest as elements are accessed in memory

- this is opposite to the way that MD arrays are laid out in C

creating MD arrays: reading in media

```matlab
[v,fs] = audioread('axe-head.wav'); % mono  wav file
sv = size(v)
[w,fs] = audioread('music.mp3'); % stereo wav file
sw = size(w)
x = imread('cameraman.tif'); % grayscale image
sx = size(x)
y = imread('coloredChips.png'); % color image
sy = size(y)
obj = VideoReader('xylophone.mp4','tag','myreader1');
z = read(obj); % color video
sz = size(z)
```

```
sv =
      187165           1     <= mono audio
sw =
     2087424           2     <= stereo audio
sx =
   256   256                 <= grayscale image
sy =
   391   518     3           <= color image
sz =
   240   320     3   141     <= color video
```

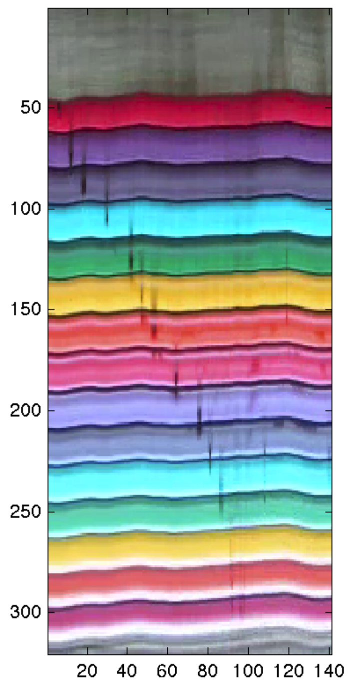
accessing MD array elements

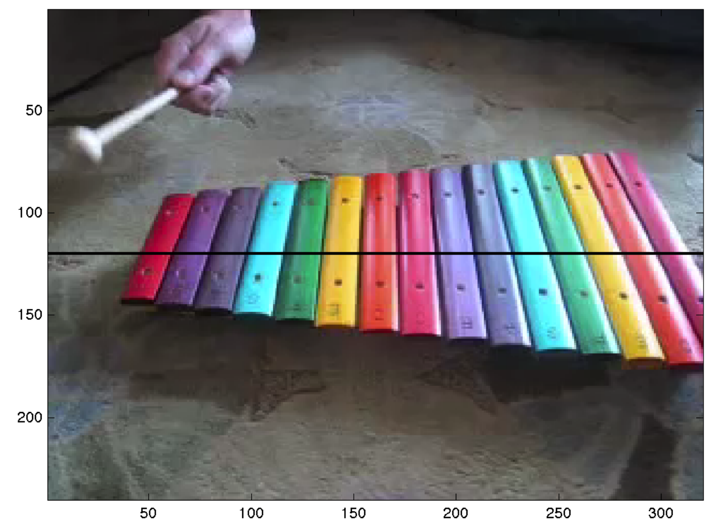
- get and set elements of MD arrays by reference to specific element
  - 1D: `x(30000)`
  - 2D: `x(30000,2)` or `x(100,200)`
  - 3D: `x(100,200,2)`
  - 4D: `x(100,200,2,75)` or `x(i,j,k,l)`
- slicing using colon `(:)` to get and set
  - 1D: `x(1:14000)` or `x(14000:end)` or `x(1:2:end)`
  - 2D: `x(30:35,70:75)` or `x([30,23,99],[10:2:20,70:5:end])`
  - 3D: `x(:,:,3)` accesses the blue plane in an RGB color image
  - 4D: `x(:,:,:,121)` accesses the 121st frame in a color video, returns a color image
  - 4D: `x(100,200,3,:)` accesses the (100,200)th pixel, blue color plane over all the frames in a video
- `x(:)` vectorizes the object (returns a column vector)

squeezing and permuting by example

```
obj = VideoReader('xylophone.mp4','tag','myreader1');
z = read(obj); % color video
sz = size(z)
r = round(sz(1)/2)
y = z(r,:,:,:);
sy = size(y)
x = squeeze(y); % squeeze removes singleton dimensions
sx = size(x)
w = permute(x,[1 3 2]); % permutes the dimensions, need RGB in 3rd
                          % dimension of image (see also ipermute)
sw = size(w)
imagesc(w); axis image;
```

```
sz =
   240   320     3   141
r =
   120
 sy =
     1   320     3   141
sx =
   320     3   141
sw =
   320   141     3
```





operations on MD arrays

- arithmetic: `+`, `−`, `.*`, `./`, `.^`, etc.
- functions that do not change dimension: `round`, `mod`, `log`, `exp`, `abs`, `cos`, `tanh`, etc.
- functions that reduce dimension by 1: `sum`, `mean`, `std`, `var`, `min`, `max`, etc.
- example: `max(max(max(x)))` finds the largest element in a 3D array
- example: `mean(mean(y))` finds the average R, G and B values in a color image

low-level file I/O in Matlab

example file I/O

```
fid = fopen('data.bin','rb');
if(fid == -1) fprintf('ERROR:  Could not open file.\n'); end
[x,cnt] = fread(fid,  5,'int');    % read in 5 integers
[y,cnt] = fread(fid, 10,'float'); % read in 10 floats
[z,cnt] = fread(fid,inf,'short'); % read to the end of the file
fclose(fid);

y = rand(10,1); % 10 double-precision uniform [0,1] random numbers
fid = fopen('data.bin','wb');
cnt = fwrite(fid,y,'float'); % write values in y array as floats
fclose(fid);
```

- you have to know the file format to read it correctly
- use `fclose all` to close all open files
- other functions available: fseek, frewind, ftell, feof, ferror, fscanf, fgetl, fgets, etc.

the `plot` command

using `plot`

- `plot` plots points connected by line segments
- points are provided to `plot` using equal length vectors:
  - `plot(x,y)` - plots `y` versus `x` by connecting the points with line segments
  - `plot(y)` - plots `y` versus the index `[1:length(y)]`
- plotting multiple data sets on the same axis done in two ways
  - `plot(x1,y1,x2,y2,x3,y3)` plots 3 lines using different colors for each
  - add lines to an existing plot using `hold on` and `hold off`

```
plot(x1,y1,'Color',[0 0 1]);   % create a plot
hold on;  % add subsequent plots to this axis
plot(x2,y2,'Color',[0 1 0]);   % add a plot
plot(x3,y3,'Color',[1 0 0]);   % add a plot
hold off; % stop adding plots to this axis
```

  does the same thing as

```
plot(x1,y1,'b'); hold on;
plot(x2,y2,'g');
plot(x3,y3,'r'); hold off;
```

predefined colors, markers and line types

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

- `plot(x,y,'m*--')` plots magenta line, “*” marker, and dashed line

- `plot(x,y,'k',x,y,'r+')` (what does this do?)

handle graphics

- a plot has many different attributes that can be examined/changed using `get`/`set`

```
x = [0:8]*pi/8; y = sin(x);
han = plot(x,y)
get(han);
```

```
          Color: [0 0 1]
      LineStyle: '-'
      LineWidth: 0.5000
         Marker: 'none'
     MarkerSize: 6
MarkerEdgeColor: 'auto'
MarkerFaceColor: 'none'
          XData: [1x21 double]
          YData: [1x21 double]
           Type: 'line'
        Visible: 'on'
         Parent: 173.0143
```

- to see the attributes of a graphics handle use `get(han)`
- to see the attributes of a graphics handle that the user can set use `set(han)`
- to set the attributes use `set(han,'attribute',value)`

- then do

```
set(han,'LineWidth',2,'Marker','^','MarkerSize',12);
set(han,'MarkerFaceColor',[1 0 1]);
```

- each plot line has its own attributes and can abe set independently

figures

- plot lines are children of an axes objects
- axes are children of figure objects
- figures and axes have many properties that can be set by the user
- `gcf` returns a handle to the current figure
- `get(gcf)` lists the attributes of a figure
- below is only a partial list

```
Color = [0.8 0.8 0.8]                    PaperSize = [8.5 11]
CurrentAxes = [173.014]                  PaperType = usletter
PaperUnits = inches                      Position = [1201 440 560 420]
PaperOrientation = portrait              Children = [173.014]
PaperPosition = [0.25 2.5 8 6]           Parent = [0]
PaperPositionMode = manual               Visible = on
```

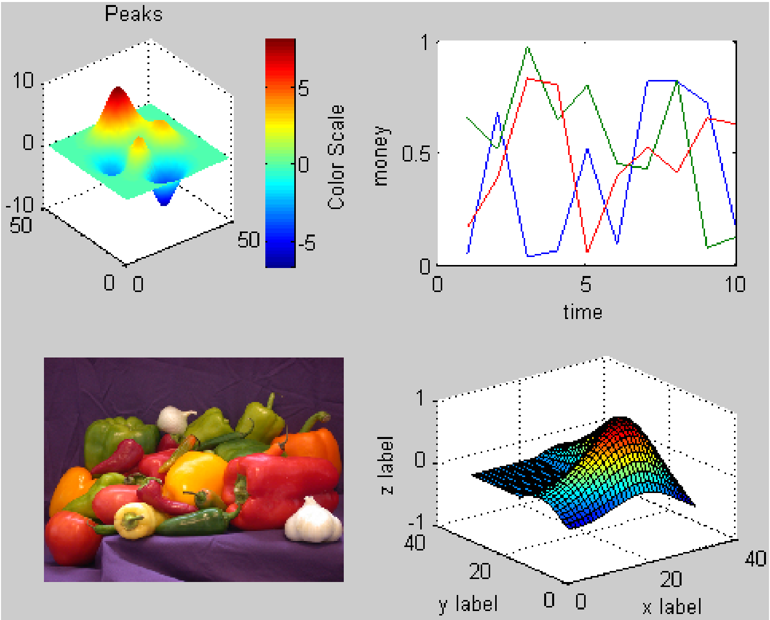
- very commonly used functions that access or modify figure properties: figure (creates a new figure), clf, shg, gcf, tightfig

axes

- plot lines are children of an axes objects
- axes are children of figure objects
- figures and axes have many properties that can be set by the user
- `gca` returns a handle to the current axes
- `get(gca)` lists the attributes of an axes
- there are too many attributes to list here
- can set: fonts, font size, axis limits, grids, tick marks, tick labels, x and y labels, title, colors, viewing angle, size and position in the figure window, callback functions for user interaction, etc.
- very commonly used functions that access or modify axes properties: axis (on, off, square, image, equal, etc.), xlim, ylim, grid on, grid off, cla, gca, xlabel, ylabel, title, hold on, hold off, tight

subplots

- `subplot(m,n,i)` creates an axis at position `i` in a `m` by `n` array of plots
- this is useful to place more than one plot in a figure window and on a printed page
- see also `tightfig`, `subplot_tight`, `spaceplots`, `subplotplus`, `subplot1` and `tight_subplot` (some of these available from Matlab Central)



printing

- `print -device -options filename` prints the contents of the figure window to an image file that can be printed or included in a report

- a few of the possibilities for the `-device` flag are

```
-dps       % PostScript for black and white printers
-deps      % Encapsulated PostScript
-djpeg<nn> % JPEG image, quality level of nn (figures only)
             E.g., -djpeg90 gives a quality level of 90.
             Quality level defaults to 75 if nn is omitted.
-dtiff     % TIFF with packbits (lossless run-length encoding)
             compression (figures only)
-dtiffnocompression % TIFF without compression (figures only)
-dpng      % Portable Network Graphic 24-bit truecolor image
-dpdf      % PDF document
```

- `orient` sets the paper orientation for printing (call before `print`)

```
orient landscape;
orient rotated;
orient portrait;
orient tall;
```

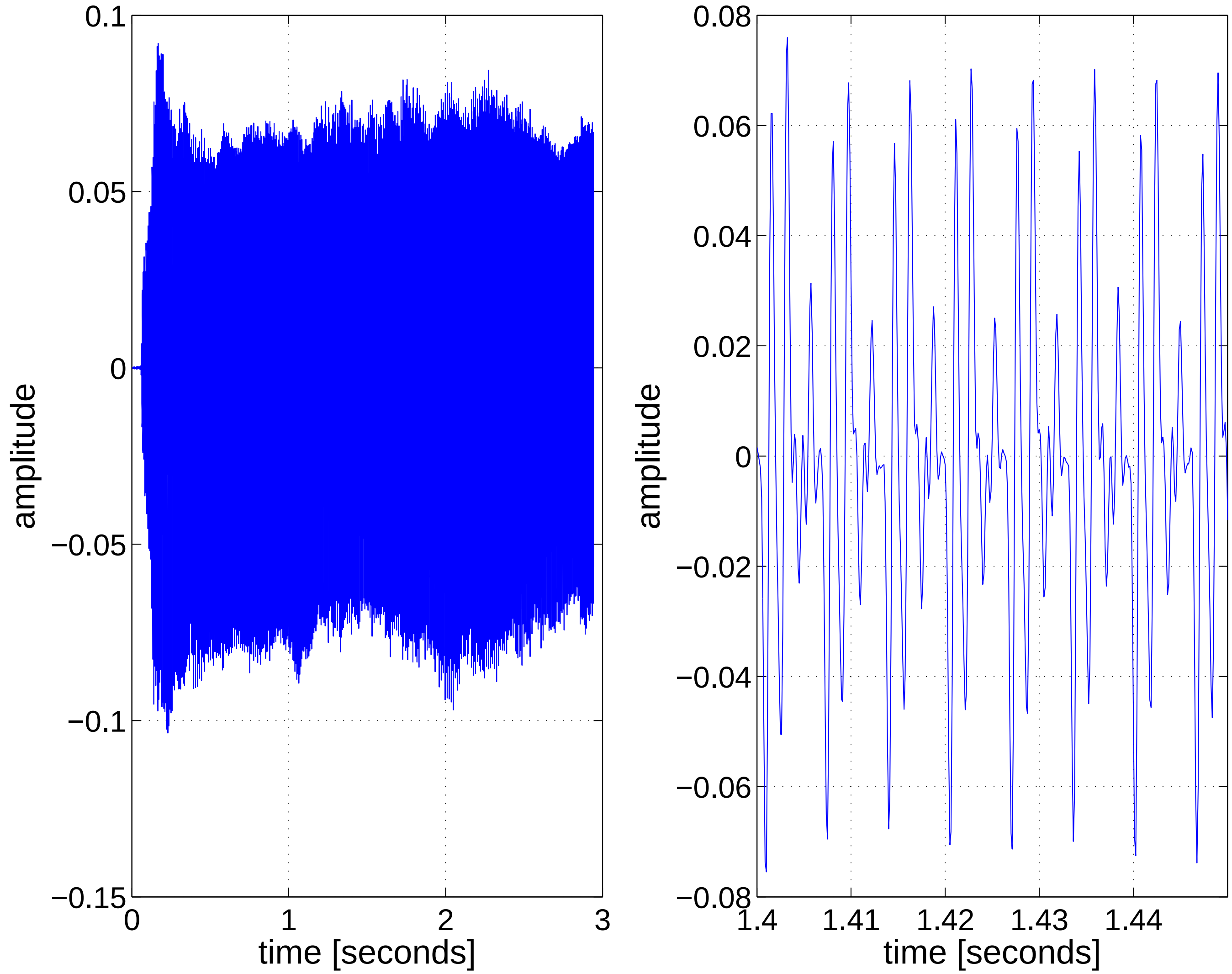
example

I used a microphone and recorded myself whistling a tune for about ten seconds and saved the data into the wave file “whistle.wav”.

time domain plot

```
[x,sample_rate,nbits]=wavread('whistle.wav');
time = [0:length(x)-1]/sample_rate; % time [seconds]

subplot(1,2,1)
plot(time,x); % plot with accurately scaled time axis
xlabel('time [seconds]','FontSize',18);
ylabel('amplitude','FontSize',18);
set(gca,'FontSize',16)
grid on;

subplot(1,2,2);
plot(time,x); % plot with accurately scaled time axis
% adjust the view
t1 = 1.40; % seconds
t2 = 1.45; % seconds
xlim([t1 t2]);  % view from 4.4 to 4.6 seconds
xlabel('time [seconds]','FontSize',18);
ylabel('amplitude','FontSize',18);
set(gca,'FontSize',16)
grid on;

orient landscape;
print -dpdf plotting_time.pdf
```

time domain plot



oscilloscope

- sometimes it is useful to visualize a signal over time

```
[x,fs]=audioread('../../../dsp_labs/whistle.wav');
t = [0:length(x)-1]/fs; % time [seconds]
%plot(t,x); % this plots whole waveform, gives big picture,
            % but cannot see detail

win_sec = 0.05; % window length [seconds]
win_sam = round(win_sec*fs); % window length [samples]
step_sec = 0.001; % step length [seconds]
step_sam = round(step_sec*fs); % step length [samples]

han = plot(t(1:win_sam),x(1:win_sam)); drawnow;
ylim(0.1*[-1 1]);
for i=win_sam:step_sam:length(x)
  ind = [i-win_sam+1:i];
  set(han,'XData',t(ind),'YData',x(ind));
  xlim(t(ind([1,end])));
  drawnow;
  %pause; % if this runs too fast, include a pause
end
```

`image` and `imagesc`

using `image`

- for 2D arrays: `image(C)` displays an image using a colormap
  - `colorbar` adds a colorbar to an image
  - `colormap(gray)` changes to a grayscale colormap
  - `colormap(jet)` is the default colormap
  - built-in colormaps: `parula`, `hsv`, `hot`, `cool`, and many more
  - can also construct custom colormaps

- for 3D arrays: `image(C)` uses `C(:,:,1)` as the red intensity, `C(:,:,2)` as the green intensity, `C(:,:,3)` as the blue intensity

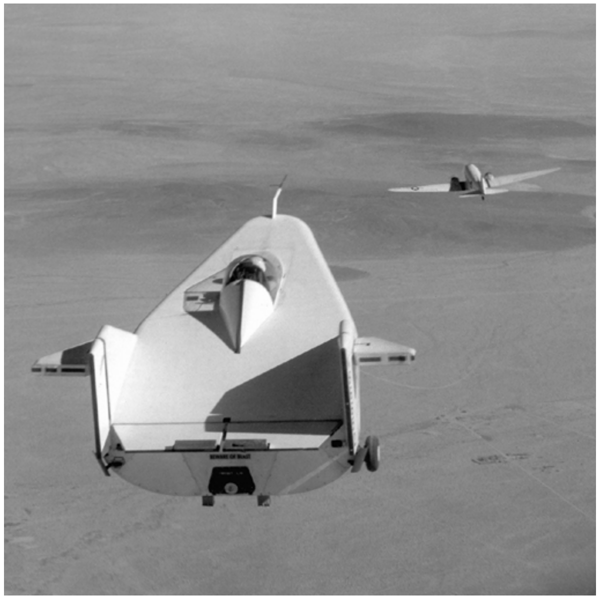
- if `C` is `double`, then the elements must be in the range [0,1]

- if `C` is `uint8`, then the elements must be in the range [0,255]

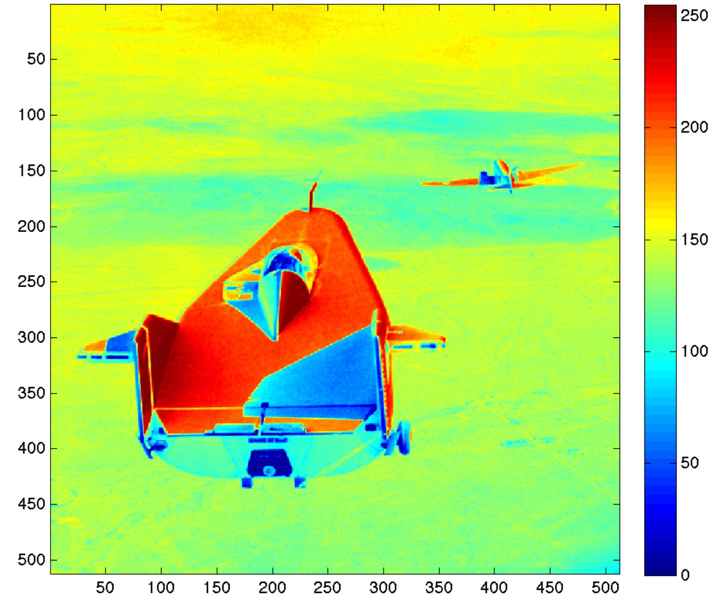
- `image(X,Y,C)` uses the values in vectors `X` and `Y` to specify the location of the pixel centers

example grayscale image

```
x = imread('liftingbody.png');
h = image(x); axis image;
set(h,'CDataMapping','scaled'); % I should not have to do this.
colormap(jet); % set the colormap
colorbar;       % show the colorbar
```



(image file)

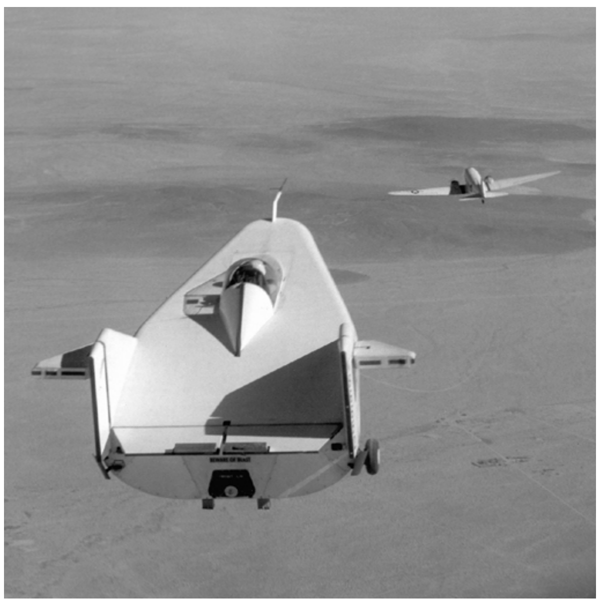


(Matlab figure window)

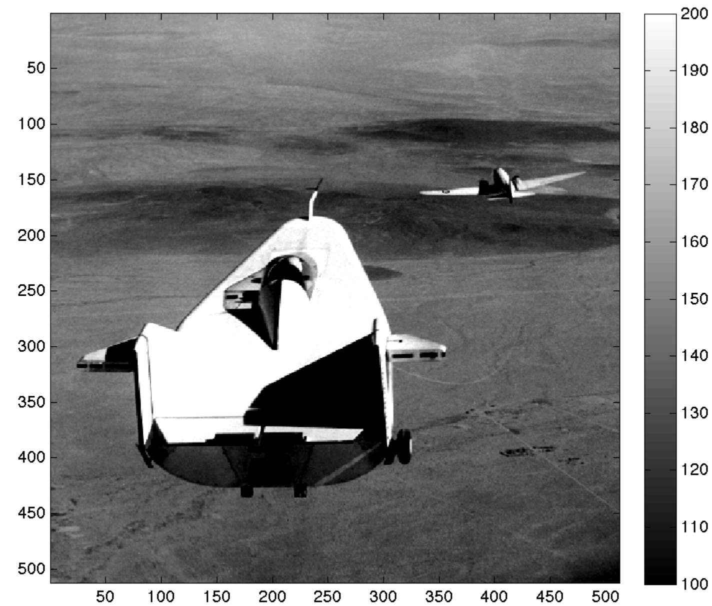
using `imagesc`

- `imagesc` is much easier to use – you don't have to worry about the input data range
- I almost always use `imagesc`
- `imagesc(C,[clo chi])` lets you clip the colormap

```
x = imread('liftingbody.png');
imagesc(x,[100 200]); axis image;
colormap(gray); % set the colormap
colorbar;       % show the colorbar
```



(image file)



(Matlab figure window)

`imagesc` with handle graphics

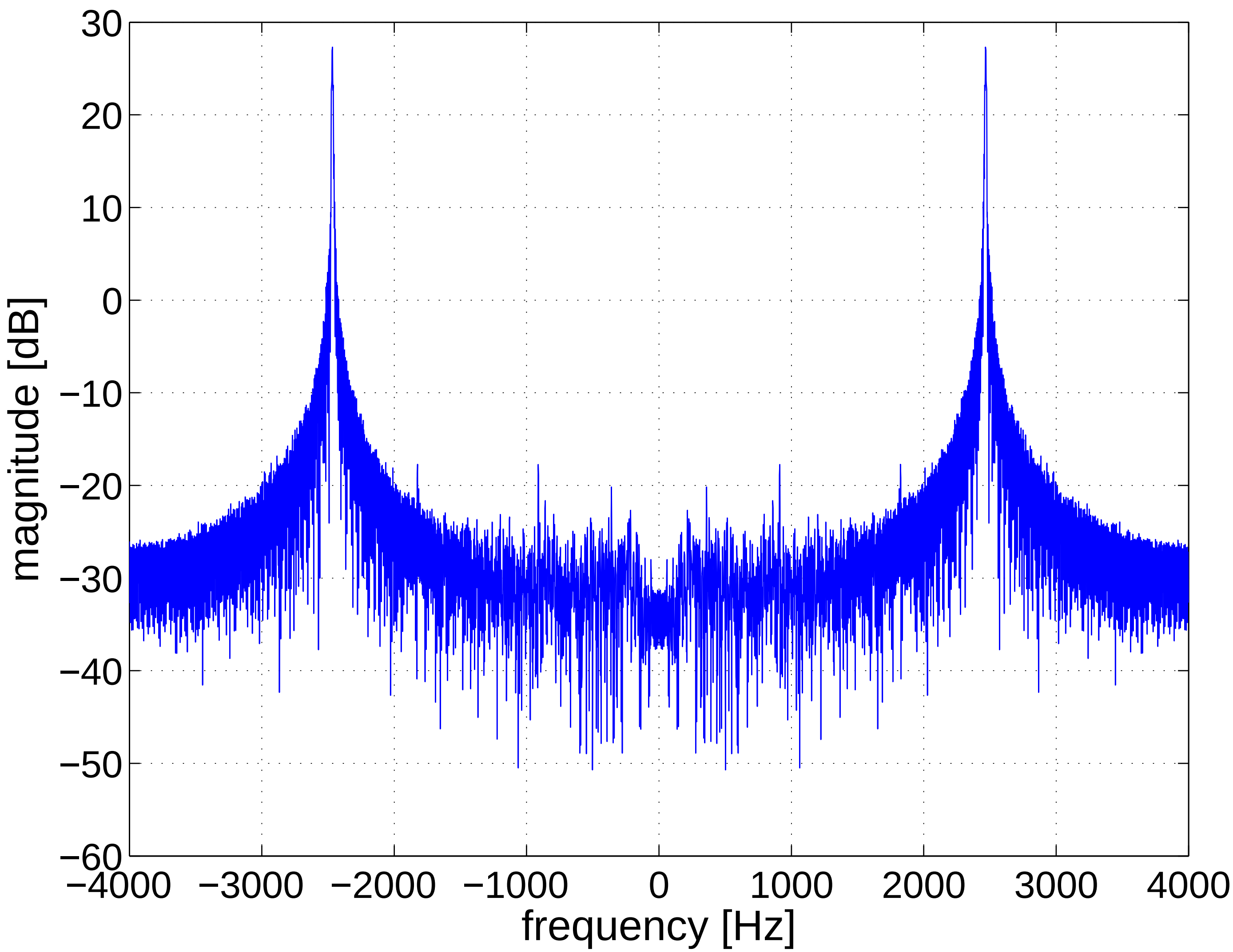
```
% load the frames of a video one at a time
% and display each one as an image

obj = VideoReader('xylophone.mp4','tag','myreader1');

nframes = obj.NumberOfFrames;
for i=1:nframes
  y = read(obj,i); % read the next video frame
  if(i==1)
    han = imagesc(y); axis image; drawnow;
  else
    set(han,'CData',y); drawnow;
  end
end
```

FFT and spectral plots

frequency domain plot

```
[x,sample_rate,nbits]=wavread('whistle.wav');
t1 = 1.40; % seconds
t2 = 1.45; % seconds
i1 = round(t1*sample_rate); % convert time to index
i2 = round(t2*sample_rate); % convert time to index

nfft = 2^12; % FFT size
freq = ([0:nfft-1]/nfft - 0.5)*sample_rate; % frequency [Hz]
X = fft(x(i1:i2),nfft); % compute the discrete-Fourier transform
plot(freq,20*log10(abs(fftshift(X))));
% plot with accurately scaled frequency axis

xlabel('frequency [Hz]','FontSize',18);
ylabel('magnitude [dB]','FontSize',18);
set(gca,'FontSize',16)
grid on;
print -dpdf plotting_freq.pdf
```

frequency domain plot



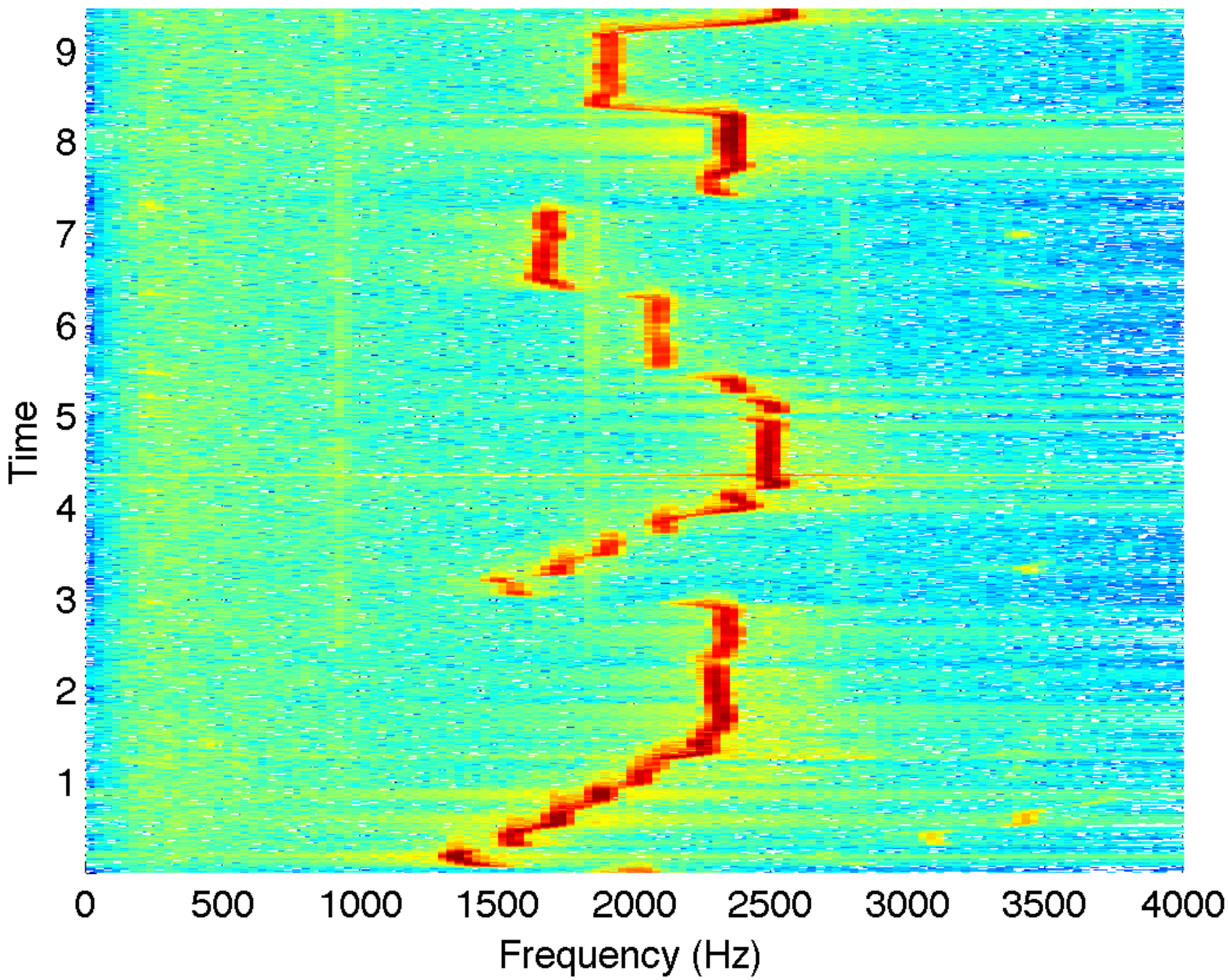
- whistling can produce very pure tonal/sinusoidal signals
- what would this plot look like if it was a pure tone/sinusoid?

spectrograms and time-frequency plots

spectrogram plot

```
[x,sample_rate,nbits]=wavread('whistle.wav');

nfft = 2^8; % FFT size
overlap = round(0.8*nfft);
window = hamming(nfft);
spectrogram(x,window,overlap,nfft,sample_rate);

set(gca,'FontSize',16);
grid on;
print -dpdf plotting_spectrogram.pdf
```

spectrogram (time-frequency) plot



- spectrogram shows where energy is concentrated in time and frequency

assignment

file format

- this assignment involves reading various types of signals into Matlab and writing them out to a raw binary file
- raw binary files have a common file format regardless of the type of media
- raw binary files have two parts: a 20 byte header followed by data (in that order)
- the first 20 bytes of the file should be five integers written in the following order

  1. `ndim` - number of dimensions in the signal
  2. `nchan` - number of channels in the signal
  3. `dim0` - the size of the first dimension
  4. `dim1` - the size of the second dimension
  5. `dim2` - the size of the third dimension

  (all of these integers must appear in this order regardless of the number of dimensions or channels in the signal)

- the rest of the file consists of the signal samples written as float data type
- the order in which the samples are written is important
- the order may not be the same as Matlab's natural memory layout
- some data permutations may be required

audio file format

- audio signals are written in sample index order with samples from multiple channels interleaved

- this format enlarges the file size, but it makes processing in C easier

- for audio files, dim1 and dim2 are not used

- let dim1 be the sample rate (Fs)

ndim = 1 (int)
nchan = 2 (int)
dim0 = L (int)
dim1 = Fs (int)
dim2 = 0 (int)
sample 0, chan 0 (float)
sample 0, chan 1 (float)
sample 1, chan 0 (float)
sample 1, chan 1 (float)
sample 2, chan 0 (float)
sample 2, chan 1 (float)
sample 3, chan 0 (float)
sample 3, chan 1 (float)
sample 4, chan 0 (float)
sample 4, chan 1 (float)
⋮
sample L-1, chan 0 (float)
sample L-1, chan 1 (float)

image file format

- image signals are written in row-major order with (R,G,B) interleaved
- this format enlarges the file size, but it makes processing in C easier

	ndim = 2 (int)				
	nchan = 3 (int)				
	dim0 = M (int)				
	dim1 = N (int)				
	dim2 = 0 (int)				
row 0	pixel (0,0)	R	G	B	(float × 3)
	pixel (0,1)	R	G	B	(float × 3)
	⋮				
	pixel (0,N-1)	R	G	B	(float × 3)
row 1	pixel (1,0)	R	G	B	(float × 3)
	pixel (1,1)	R	G	B	(float × 3)
	⋮				
	pixel (1,N-1)	R	G	B	(float × 3)
⋮	⋮				
row M-1	pixel (M-1,0)	R	G	B	(float × 3)
	pixel (M-1,1)	R	G	B	(float × 3)
	⋮				
	pixel (M-1,N-1)	R	G	B	(float × 3)

video file format

- the video file format follows the image format with each frame of video written out in sequence: frame 0, frame 1, ..., frame T-1, where each frame is an RGB or grayscale image

- this enlarges the file size, but it makes processing in C easier

assignment: do the following

1. audio (use `flute22.wav` and `music.mp3`)
   (a) write a Matlab function named `audio2bin` that reads signal samples from an audio file (such as .wav or .mp3) and writes the signal samples to a raw binary file
   (b) write a Matlab function named `bin2audio` that reads signal samples from a raw binary file and writes the signal samples to an audio file (such as .wav or .mp3)
   (c) use the `sound` or `soundsc` Matlab commands to play an audio signal to the speaker
   (d) make a subplot plot of the audio signal using a true time-scaled x-axis (set the x-axis limits to [1.0 1.01] seconds)
   (e) make a subplot plot of the spectrum (FFT) of the audio signal data from 1 to 1.01 seconds and use a true frequency-scaled x-axis (what is the frequency of the highest peak?)
   (f) in a subplot show the spectrogram of the signal
   (g) write an oscilloscope script to visualize the audio signal over a 10 msec window (handle graphics)

2. image (use `liftingbody.png` and `coloredChips.png`)
   (a) write a Matlab function named `image2bin` that reads signal samples from an image file and writes the signal samples to a raw binary file

(b) write a Matlab function named `bin2image` that reads signal samples from a raw binary file and writes the signal samples to an image file
(c) make a plot of the image (pixels should be square)

3. video (use `xylophone.mp4`)

   (a) write a Matlab function named `video2bin` that reads signal samples from a video file and writes the signal samples to a raw binary file (only one video frame in memory at a time)
   (b) write a Matlab function named `bin2video` that reads signal samples from a raw binary file and writes the signal samples to a video file (only one video frame in memory at a time)
   (c) write a movie player script to visualize the frames of video (handle graphics)