Rutgers University
School of Engineering

Fall 2011

14:440:127 - Introduction to Computers for Engineers

Sophocles J. Orfanidis
ECE Department
orfanidi@ece.rutgers.edu

week 1

> “The purpose of computing is insight, not numbers”
>
> Richard Hamming

> “I hear and I forget,
> I see and I remember,
> I do and I understand.”
>
> Confucious

This course is an introduction to MATLAB, a powerful programming language and development environment for engineers and scientists.

Syllabus and other course materials can be found in:

https://sakai.rutgers.edu

exam dates,
lecture notes,
homeworks, etc.

MATLAB = Matrix Laboratory (Cleve Moler)

MATLAB ® is a registered trademark of The Mathworks Inc., http://www.mathworks.com

MATLAB & Simulink Student Version

Main Features of MATLAB

- Easy and efficient programming in a high-level language, with an interactive interface for rapid development.

- Vectorized computations for efficient programming, and automatic memory allocation.

- Built-in support for state-of-the-art numerical computing methods.

- Has variety of modern data structures and data types, including complex numbers.

- High-quality graphics and visualization.

- Symbolic math toolbox for algebraic and calculus operations, and solutions of differential equations.
- Simulation capability with SIMULINK.
- Portable program files across platforms.
- Large number of add-on toolboxes for applications and simulations.
- Huge database of user-contributed files & toolboxes, including a large number of available tutorials & demos.
- Allows extensions based on other languages, such as C/C++, supports Java and object-oriented programming.

MATLAB Toolbox Application Areas

- Parallel Computing (2)
- Math, Statistics, and Optimization (8)
- Control System Design and Analysis (6)
- Signal Processing and Communications (7)
- Image Processing and Computer Vision (4)
- Test and Measurement, Data Acquisition (5)
- Computational Finance, Datafeeds (5)
- Computational Biology (2)
- Code Generation and Application Deployment (7)
- Database Connectivity (2)

(48 toolboxes)

SIMULINK Applications

- Fixed-Point and Event-Based Modeling
- Physical Modeling (mechanics, driveline, hydraulics, RF, electronics, power systems, biology)
- Control Systems (design, optimization, aerospace)
- Signal & Image Processing and Computer Vision
- Communication Systems (digital, analog, wireless)
- Code Generation (for embedded systems, DSP chips and FPGAs)
- more

Web Resources

- Getting Started with MATLAB (HTML)
- Getting Started with MATLAB (PDF)
- MATLAB Examples
- MATLAB Online Tutorials and Videos
- MATLAB Interactive Tutorials
- MATLAB Toolbox Reference Manuals
- MATLAB Interactive CD
- Newsletters

- MATLAB User Community
- Other MATLAB Online Resources
- comp.soft-sys.matlab newsgroup

- Octave – a free look-alike version of MATLAB

- NIST – Digital Library of Mathematical Functions
- NIST – Physical Constants

Weekly Topics

Week 1 - Basics – variables, arrays, matrices, plotting (ch. 2 & 3)
Week 2 - Basics – operators, functions, program flow (ch. 2 & 3)
Week 3 - Matrices (ch. 4)
Week 4 - Plotting – 2D and 3D plots (ch. 5)
Week 5 - User-defined functions (ch. 6)
Week 6 - Input-output formatting – fprintf, sprintf (ch. 7)
Week 7 - Program flow control & relational operators (ch. 8)
Week 8 - Matrix algebra – solving linear equations (ch. 9)
Week 9 - Structures & cell arrays (ch. 10)
Week 10 - Symbolic math (ch. 11)
Week 11 - Numerical methods – data fitting (ch. 12)
Week 12 – Selected topics

Textbook: H. Moore, *MATLAB for Engineers*, 2nd ed., Prentice Hall, 2009

MATLAB Basics

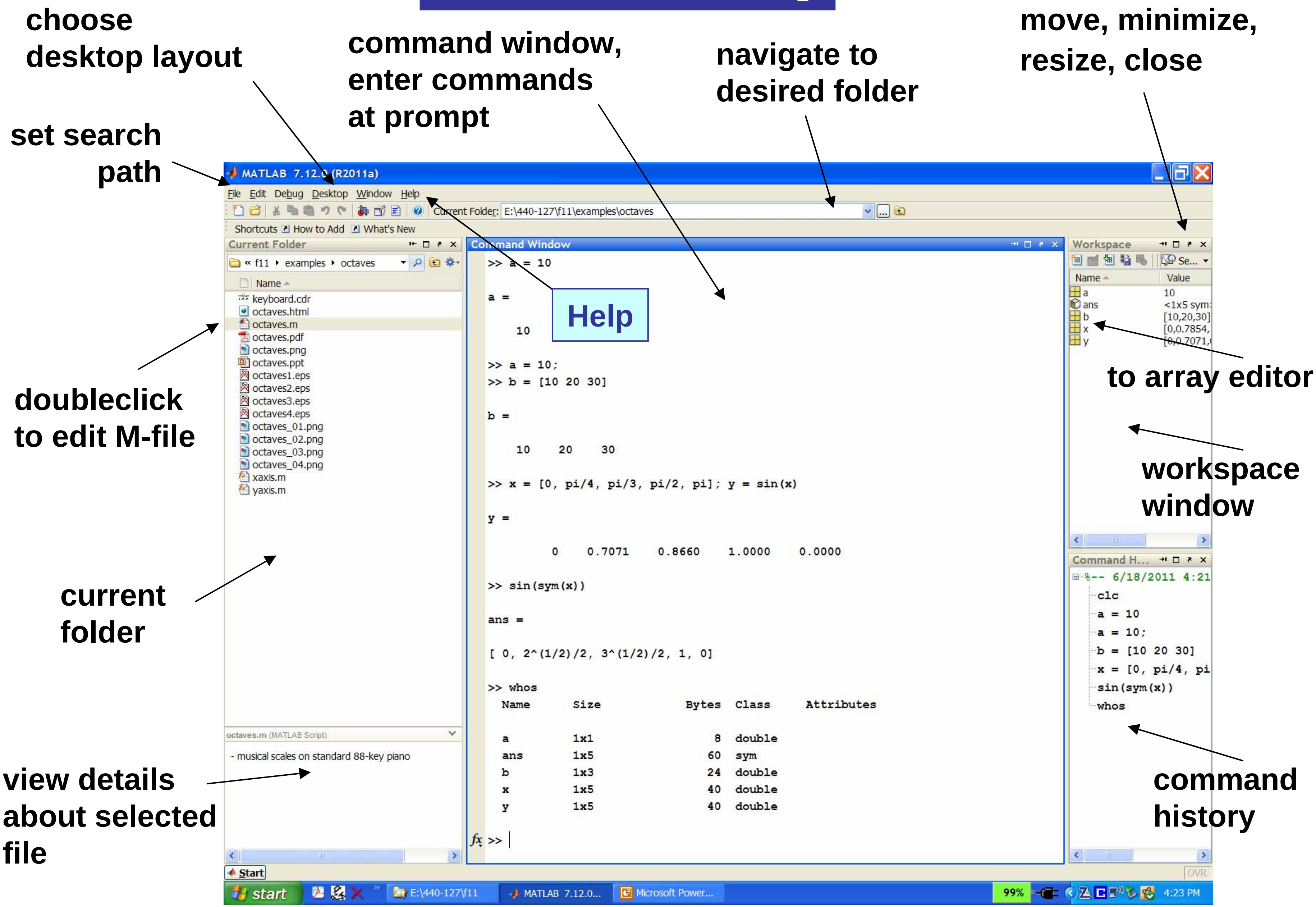
1. MATLAB desktop
2. MATLAB editor
3. Getting help
4. Variables, built-in constants, keywords
5. Numbers and formats
6. Arrays and matrices
7. Operators and expressions
8. Functions
9. Basic plotting
10. Function maxima and minima
11. Relational and logical operators
12. Program flow control
13. Matrix algebra and linear equations

week 1 (items 1–6)

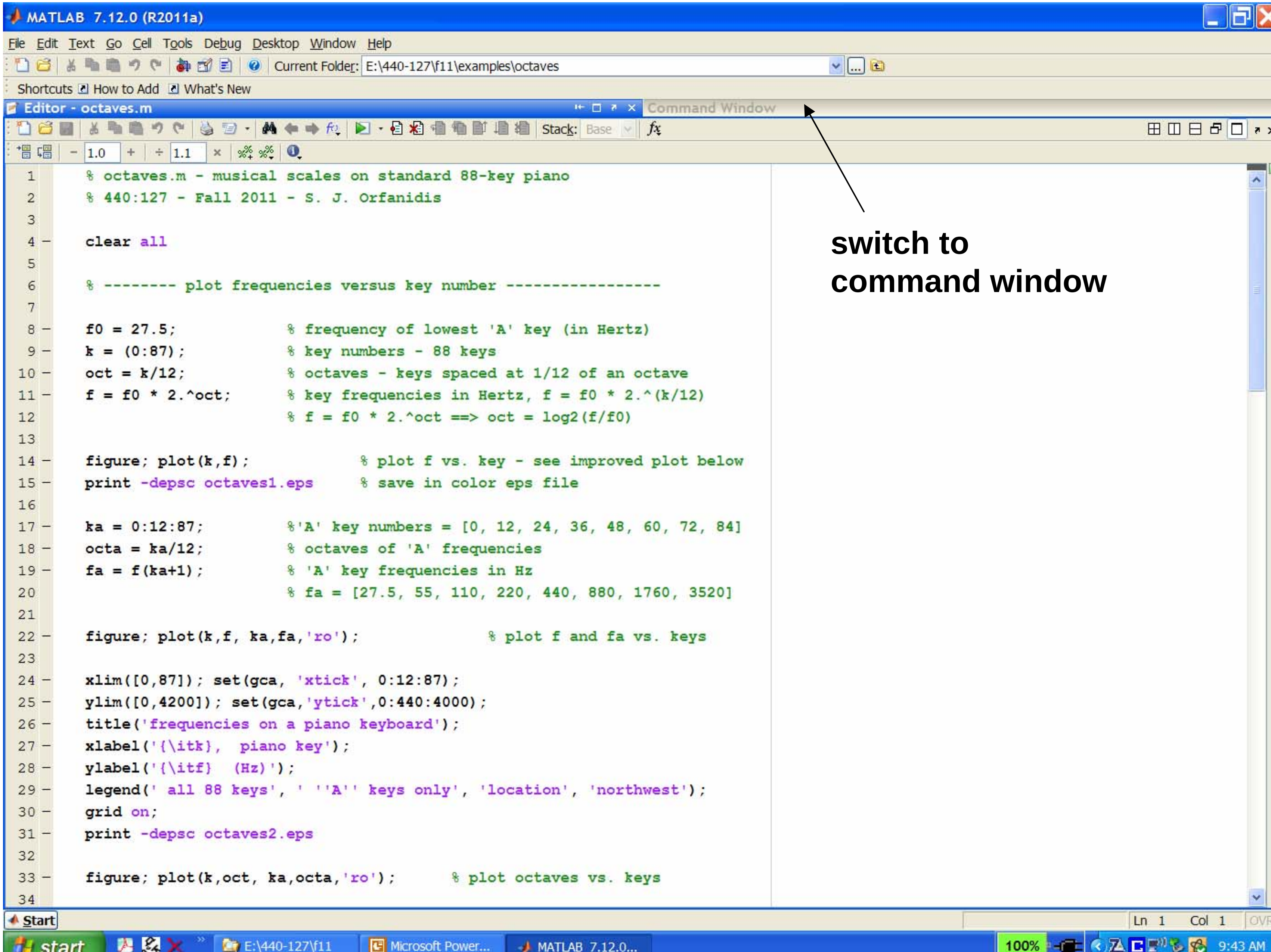
week 2 (items 7–13)

These should be enough to get you started. We will explore them further, as well as other topics, in the rest of the course.

1. MATLAB Desktop



2. MATLAB Editor



switch to command window

3. Getting Help

Several ways of getting help:

1) help menu item on MATLAB desktop opens up searchable help browser window

2) from the following commands:

comments begin with %

```
>> helpdesk             % open help browser
>> help topic           % e.g., help log10
>> doc topic            % e.g., doc plot
>> help                 % get list of all help topics
>> help dir             % get help on entire directory
>> help syntax          % get help on MATLAB syntax
>> help /               % operators & special characters
>> docsearch text       % search HTML browser for ‘text’
>> lookfor topic        % e.g., lookfor acos
```

4. Variables, Constants, Keywords

Variables require no special declarations of type or storage. Examples:

```
>> x = 3;                        % simple scalar
>> y = [4, 5, 6];                % row vector of length 3
>> z = [4; 5; 6];                % column vector of length 3
>> A = [1,2,3; 4,5,6];           % 2x3 matrix
>> s = 'abcd efg';               % string
>> s = {'abc', 'defg', '123-456'};  % cell array
```

$$y = [4,5,6]\,, \quad z = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, \quad A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

math notation

```
>> x = 3
x =
     3

>> y = [4, 5, 6]
y =
     4     5     6

>> z = [4; 5; 6]      % note, z = y'
z =
     4
     5
     6

>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
```

What are your variables? How to clear them? Use workspace window, or the commands:

who, whos, clear, clc, close

```
>> who
Your variables are:
A  y  z

>> whos
  Name         Size        Bytes  Class       Attributes
  A            2x3            48  double
  y            1x3            24  double
  z            3x1            24  double

>> clear all      % clear all variables from memory
>> clc            % clear command window
>> close all      % close all open figures
```

Operating system commands:

```
>> path                   % display search path
>> pathtool               % modify search path
>> addpath dir            % add directory to path

>> cd dir                 % change directory
>> pwd                    % print working directory

>> dir                    % list all files in current dir
>> what                   % list MATLAB files only
>> which file             % display location of file

>> edit file              % invoke MATLAB editor

>> quit                   % quit MATLAB
>> exit                   % quit MATLAB
```

Special built-in math constants that should not (though they can) be re-defined as variables:

```
eps          % machine epsilon - floating-point accuracy
i,j          % imaginary unit, i.e., sqrt(-1)
Inf,inf      % infinity
intmax       % largest value of specified integer type
intmin       % smallest value of specified integer type
NaN,nan      % not-a-number, e.g., 0/0, inf/inf
pi           % pi
realmax      % largest positive floating-point number
realmin      % smallest positive floating-point number
```

Note: **i,j** are commonly used for array and matrix indices. If you're dealing with complex-valued data, avoid redefining both **i,j**.

Values of special constants:

```
>> eps                  % equal to 2^(-52)
ans =
  2.2204e-016            % MATLAB's floating-point accuracy
                         % i.e., 2.2204 * 10^(-16)
>> intmax                % 2^(31)-1 for 32-bit integers
ans =
  2147483647

>> intmin               % equal to -2^(31)
ans =
 -2147483648

>> realmax              % equal to (2-eps)*2^(1023)
ans =
  1.7977e+308            % i.e., 1.7977 * 10^(308)

>> realmin              % 2^(-1022) = 2.2251 * 10^(-308)
ans =
  2.2251e-308
```

Special keywords that cannot be used as variable names:

```
>> iskeyword

ans =
    'break'          'function'
    'case'           'global'
    'catch'          'if'
    'classdef'       'otherwise'
    'continue'       'parfor'
    'else'           'persistent'
    'elseif'         'return'
    'end'            'switch'
    'for'            'try'
                     'while'

'true' , 'false'
```

5. Numbers and Formats

MATLAB by default uses double-precision (64-bit) floating-point numbers following the IEEE floating-point standard. You may find more information on this standard in:

Representation of Floating-Point Numbers

C. Moler, "Floating Points," MATLAB News and Notes, Fall, 1996 (PDF file)

x = (-1)^s * (1+f) * 2^(e-1023)

1 bit sign | 52 bits mantissa | 11 bits exponent

1 <= e <= 2046, e=0, e=2047

0 <= f < 1

f_min = eps = 2^(-52)

machine epsilon

MATLAB can also use single-precision (32-bit) floating point numbers if so desired.

There are also several integer data types that are useful in certain applications, such as image processing or programming DSP chips. The integer data types have 8, 16, 32, or 64 bits and are signed or unsigned:

```
    int8,   int16,   int32,   int64
   uint8,  uint16,  uint32,  uint64
```

For more information do:

```
>> help datatypes
>> help class          % determine datatype
```

Complex Numbers

By default, MATLAB treats all numbers and expressions as complex (even if they are real).

No special declarations are needed to handle complex-number operations. Examples:

```
>> z = 3+4i;                % or, 3+4j, 3+4*i, 3+4*j
>> x = real(z);             % real part of z
>> y = imag(z);             % inaginary part of z
>> R = abs(z);              % absolute value of z
>> theta = angle(z);        % phase angle of z in radians
>> w = conj(z);             % complex conjugate, w=3-4i
>> isreal(z);               % test if z is real or complex
```

$$z = x + j\,y = R\,e^{j\theta}\,, \quad R = |z| = \sqrt{x^2 + y^2}\,, \quad \theta = \arctan\frac{y}{x}$$

cartesian & polar forms

math notation: θ = Arg(z)

```
>> z = 3+4j
z =
   3.0000 + 4.0000i

>> x = real(z)
x =
     3
>> y = imag(z)
y =
     4
>> R = abs(z)
R =
     5
>> theta = angle(z)                % in radians
theta =
    0.9273

>> abs(z - R*exp(j*theta)) + abs(z-x-j*y)   % test
ans =
  6.2804e-016
```

equivalent definitions:

```
z = 3+4*j
z = 3+4i
z = 3+4*i
z = complex(3,4)
```

Display Formats

```
>> format                % default - 4 decimal places
>> format short          % same as the default
>> format long           % 15 decimal places
>> format short e        % 4 decimal – exponential format
>> format short g        % 4 decimals – exponential or fixed
>> format long e         % 15 decimals - exponential
>> format long g         % exponential or fixed
>> format shorteng       % 4 decimals, engineering
>> format longeng        % 15 decimals, engineering
>> format hex            % hexadecimal
>> format rat            % rational approximation
>> format compact        % conserve vertical spacing
>> format loose          % default vertical spacing

>> vpa(x,digits)         % variable-precision-arithmetic
```

These affect only the display format – internally all computations are done with full (double) precision

Example - displayed value of **10*pi** in different formats:

```
31.4159                      % format, or format short
31.415926535897931           % format long
3.1416e+001                  % format short e
31.416                       % format short g
3.141592653589793e+001       % format long e
31.4159265358979             % format long g
31.4159e+000                 % format shorteng
31.4159265358979e+000        % format longeng

>> vpa(10*pi)                % symbolic toolbox
ans =
31.415926535897932384626433832795

>> vpa(10*pi,20)             % specify number of digits
ans =
31.415926535897932385
```

```
>> help format
>> help vpa
>> help digits
```

input/output functions: **disp**, **input**

```
>> x = 10; disp('the value of x is:'); disp(x);
the value of x is:
    10

>> x = input('enter x: ')        % numerical input
enter x: 100                      % 100 entered by user
x =
   100

>> y = input('enter string: ', 's');  % string input
enter string: abcd efg
>> y = input('enter string: ')
enter string: 'abcd efg'
y =
abcd efg
```

prompt string in single quotes

string entered with no quotes
string entered in quotes

```
>> help fprintf
>> help sprintf
```

```
>> help disp
>> help input
>> help menu
```

6. Arrays and Matrices

arrays and matrices are the most important data objects in MATLAB

We discuss briefly:

a) row and column vectors
b) transposition operator, '
c) colon operator, **:**
d) equally-spaced elements, linspace
e) accessing array elements
f) dynamic allocation & de-allocation
g) pre-allocation

The key to efficient MATLAB programming can be summarized in three words:

vectorize, vectorize, vectorize

and avoid all loops

Compare the two alternative computations:

```
x = [2,-3,4,1,5,8];
y = zeros(size(x));
for n = 1:length(x)
   y(n) = x(n)^2;
end
```

```
x = [2,-3,4,1,5,8];
y = x.^2;
```

element-wise exponentiation **.^**

ordinary exponentiation **^**

answer: y = [4,9,16,1,25,64]

```
>> x = [0 1 2 3 4 5]                 % row vector
x =
     0     1     2     3     4     5

>> x = 0:5                           % row vector
x =
     0     1     2     3     4     5

>> x = [0 1 2 3 4 5]'                % column vector, (0:5)'
x =
     0
     1
     2
     3
     4
     5
```

the prime operator, ', or transpose, turns row vectors into column vectors, and vice versa

caveat: ' is actually conjugate transpose, use dot-prime, .' , for transpose w/o conjugation

```
>> z = [i; 1+2i; 1-i]        % column vector
z =
        0 + 1.0000i
   1.0000 + 2.0000i
   1.0000 - 1.0000i

>> z.'                   % transpose without conjugation
ans =
        0 + 1.0000i   1.0000 + 2.0000i   1.0000 - 1.0000i

>> z'                    % transpose with conjugation
ans =
        0 - 1.0000i   1.0000 - 2.0000i   1.0000 + 1.0000i

>> (z.')'                % same as (z').' , or, conj(z)
ans =
        0 - 1.0000i
   1.0000 - 2.0000i
   1.0000 + 1.0000i
```

about linspace:

```
x = linspace(a,b,N+1);
```

is equivalent to:

```
x = a : (b-a)/N : b;
```

i.e., N+1 equally-spaced points in the interval [a,b] or, dividing [a,b] into N equal sub-intervals

$$x(n) = a + \left(\frac{b-a}{N}\right)(n-1)\,, \quad n = 1, 2, \ldots, N+1$$

step increment

```
>> x = 0 : 0.2 : 1           % in general, x = a:s:b
>> x = linspace(0,1,6)       % see also logspace
x =
     0   0.2000   0.4000   0.6000   0.8000   1.0000
```

6 points, 5 subintervals

```
>> x = 0 : 0.3 : 1
x =
    0    0.3    0.6    0.9
```

```
>> x = 0 : 0.4 : 1
x =
    0    0.4    0.8
```

```
>> x = 0 : 0.7 : 1
x =
    0    0.7
```

step increment

x = a : s : b;

the number of subintervals within [a,b] is obtained by rounding **(b-a)/s**, down to the nearest integer,

N = floor((b-a)/s);

length(x) is equal to **N+1**

x(n) = a + s*(n-1),
n = 1,2,...,N+1

```
% before rounding, (b-a)/s was in the three cases:
% 1/0.3 = 3.3333,  1/0.4 = 2.5,  1/0.7 = 1.4286
```

Note: MATLAB array indices always start with 1 and may not be 0 or negative

exception: logical indexing, discussed later

```
>> x = [ 2,    5,   -6,   10,    3,    4 ];
         ↑     ↑     ↑     ↑     ↑     ↑
       x(1), x(2), x(3), x(4), x(5), x(6)
```

Other languages, such as C/C++ and Fortran, allow indices to start at 0. For example, the same array would be declared/defined in C as follows:

```
double x[6] = { 2,    5,   -6,   10,    3,    4 };
                ↑     ↑     ↑     ↑     ↑     ↑
              x[0], x[1], x[2], x[3], x[4], x[5]
```

rule of thumb: M = C + 1

accessing array entries:

```
>> x = [2, 5, -6, 10, 3, 4]
x =
     2     5    -6    10     3     4

>> length(x)      % length of x, see also size(x)
ans =
     6

>> x(1)           % first entry
ans =
     2

>> x(3)           % third entry
ans =
    -6

>> x(end)         % last entry – need not know length
ans =
     4
```

accessing array entries:

```
>> x(end-3:end)              % x = [2, 5, -6, 10, 3, 4]
ans =
     -6   10      3     4          % last four

>> x(3:5)                    % list third-to-fifth entries
ans =
    -6    10      3

>> x(1:3:end)                % every third entry
ans =
     2    10

>> x(1:2:end)                % every second entry
ans =
     2    -6      3
```

accessing array entries:

```
>> x = [2, 5, -6, 10, 3, 4];

>> x(end:-1:1)     % list backwards, same as fliplr(x)
ans =
     4   3   10   -6    5    2

>> x([3,1,5])          % list [x(3),x(1),x(5)]
ans =
     -6     2     3

>> x(end+3) = 8
x =
    2     5    -6    10     3     4     0     0     8
```

automatic memory re-allocation

automatic memory allocation and de-allocation:

```
>> clear x

>> x(3) = -6
x =
     0     0    -6

>> x(6) = 4
x =
     0     0    -6     0     0     4

>> x(end) = []                        % delete last entry
x =
     0     0    -6     0     0

>> x = [2, 5, -6, 10, 3, 4];
>> x(3)=[]                            % delete third entry
x =
     2     5    10     3     4
```

pre-allocation

```
>> clear x
>> x = zeros(1,6)              % 1x6 array of zeros
x =
     0     0     0     0     0     0

>> x = zeros(6,1)              % 6x1 array of zeros
x =
     0
     0
     0
     0
     0
     0
```

Pre-allocation is useful for very large arrays, e.g., **`length > 10^4`**, for example, in dealing with audio or image files, or finite-element methods.

See, for example, the program **`echoes.m`**, which reads an audio file and adds reverberation effects to it, as described in echoes.pdf, and discussed also in week-2 lectures.

```
>> help zeros
>> help ones
```

illustrating dynamic allocation & pre-allocation

```
clear x;
for k=[3,7,10]              % k runs successively through
   x(k) = 3 + 0.1*k;        % the values of [3,7,10]
   disp(x);                 % diplay current vector x
end
```

```
    0.0   0.0   3.3
    0.0   0.0   3.3   0.0   0.0   0.0   3.7
    0.0   0.0   3.3   0.0   0.0   0.0   3.7   0.0   0.0   4.0
```

```
x = zeros(1,10);            % pre-allocate x to length 10
for k=[3,7,10]
   x(k) = 3 + 0.1*k;
   disp(x);
end
```

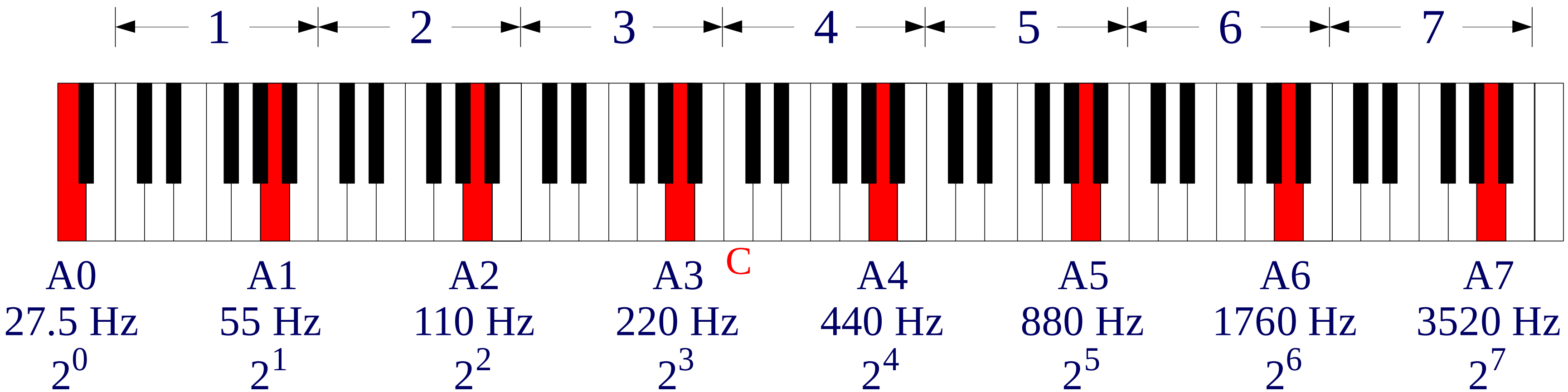
```
    0.0   0.0   3.3   0.0   0.0   0.0   0.0   0.0   0.0   0.0
    0.0   0.0   3.3   0.0   0.0   0.0   3.7   0.0   0.0   0.0
    0.0   0.0   3.3   0.0   0.0   0.0   3.7   0.0   0.0   4.0
```

Example: Octave Frequency Scales

- Tasks:
- calculate and plot the 88 frequencies of a standard 88-key piano keyboard
- introduce the concept of octave frequency scales
- generate and print the major notes (do, re, mi, fa, sol, la, si, do) of the middle (4th) octave, and play them forward & backward on the PC's soundcard (need earphones in the DSV lab)

Complete MATLAB program is in the M-file, **octaves.m**.
Please, see also the handout, **octaves.pdf**, for more details.

standard 88-key piano keyboard



$$\text{octaves} = \log_2\left(\frac{f}{f_0}\right) \quad \Rightarrow \quad f = f_0 \cdot 2^{\text{octaves}}$$

$$y(t) = \sin(2\pi f t) \longrightarrow$$

generate tone and send it to MATLAB's **sound( )** function

Running the program, **`octaves.m`**, in command window:

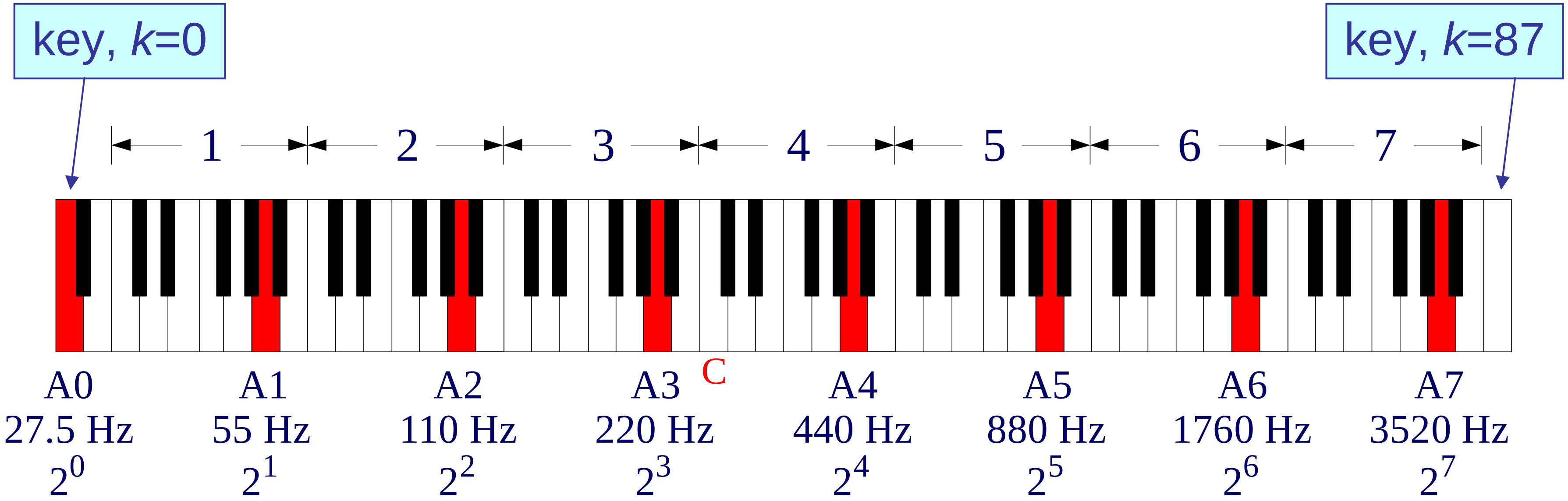
```
>> octaves;                                 % run the program

>> close all;                               % close figure windows

>> publish('octaves', 'html');              % export to HTML
```

publishing your programs to HTML is a good way to print and submit your homework problems. However, you must re-save it as a single file web archive (i.e., MHT file) that incorporates all the generated graphs (for Firefox you must install the Mozilla-Archive-Format add-on).

Alternatively, you can use any word processor and insert into it your MATLAB code and your graphs in WMF or EPS formats (this allows you to submit a single file, and preferably convert it into a PDF)



$$\text{octaves} = \log_2\left(\frac{f}{f_0}\right) \Rightarrow f = f_0 \cdot 2^{\text{octaves}}$$

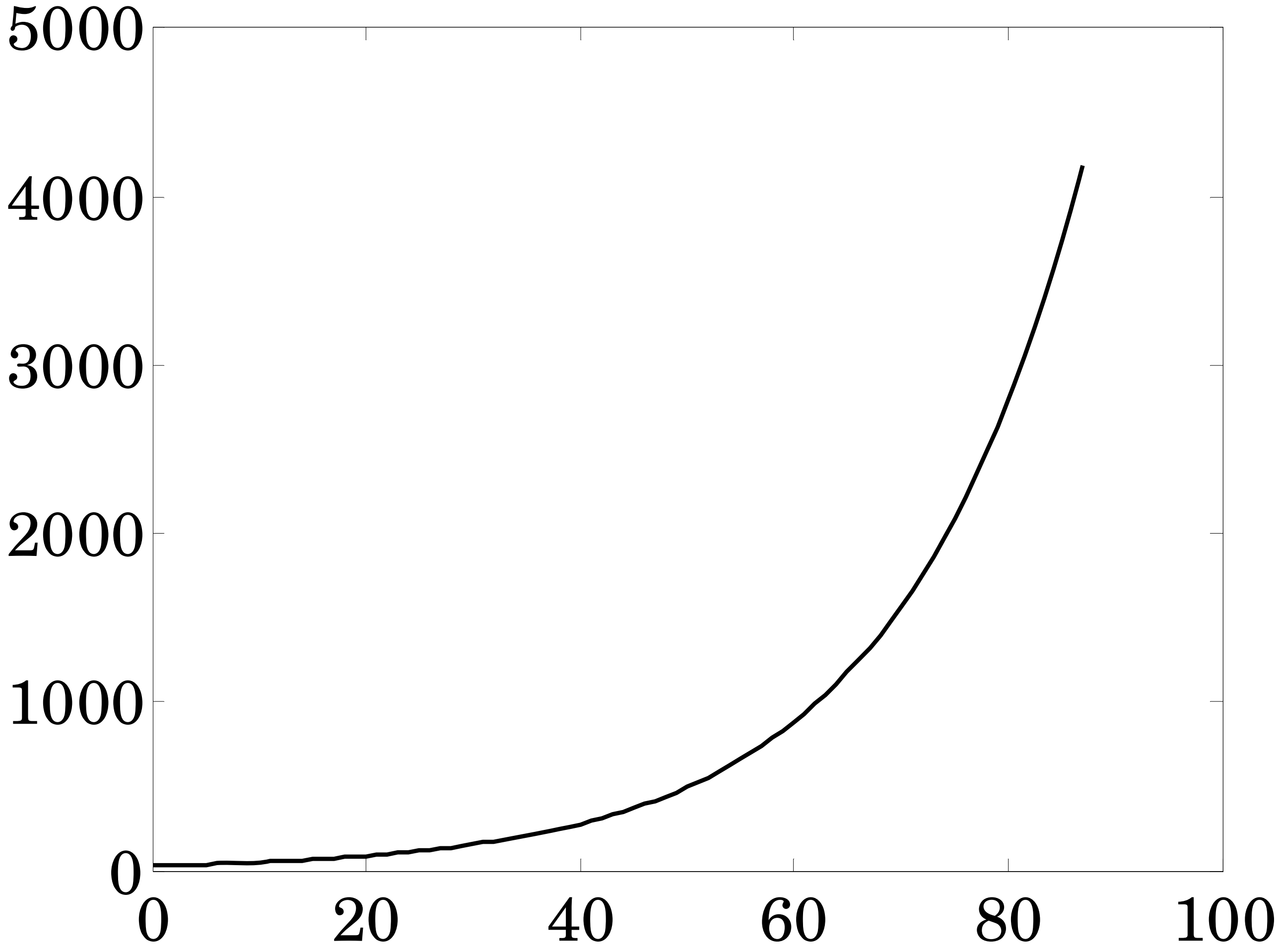
keys, $k = [0, 1, 2, \ldots, 87]$

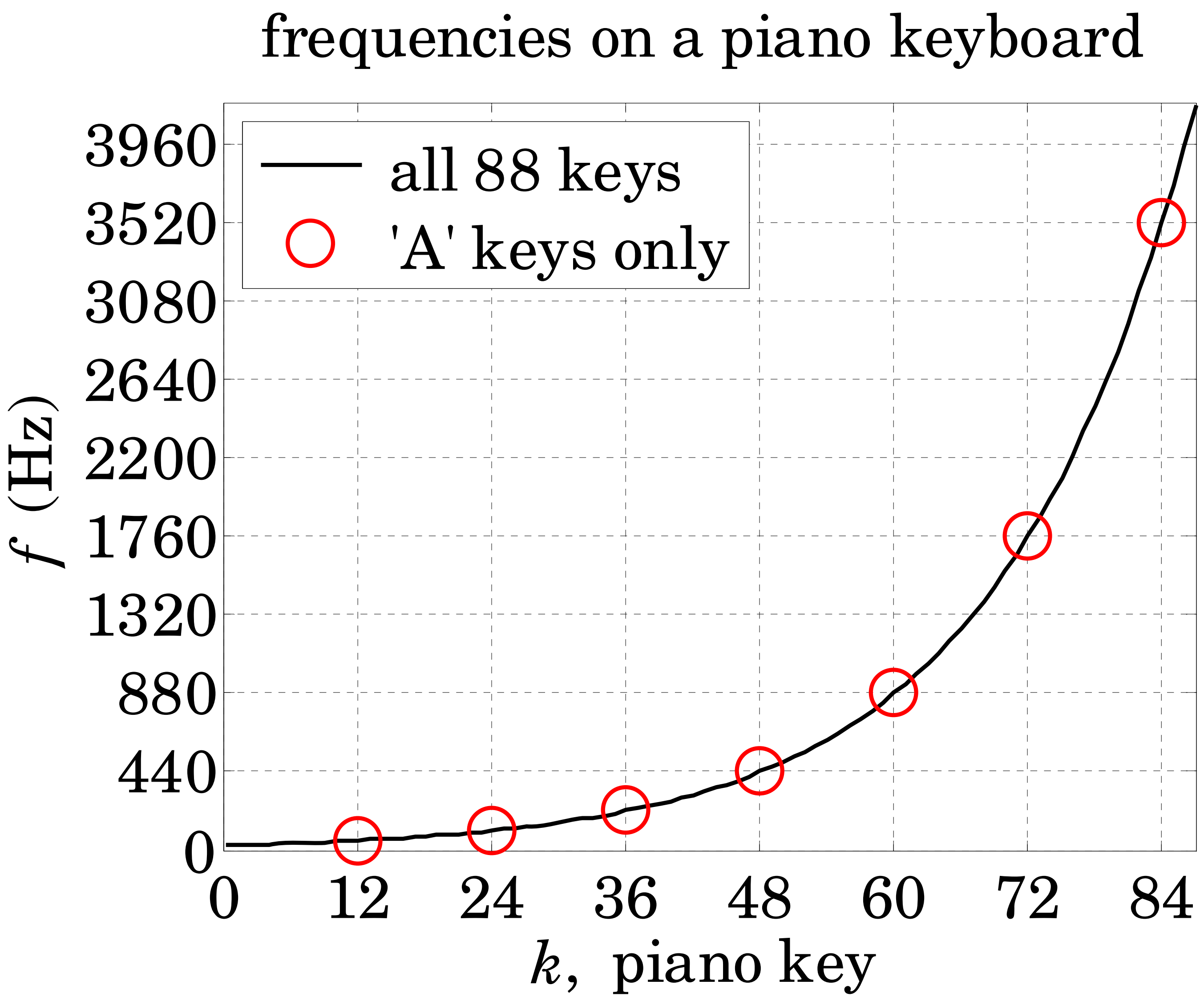
keys are separated by 1/12 of an octave

```
f0 = 27.5;
k = 0:87;
f = f0 * 2.^(k/12);
figure; plot(k,f);
```

MATLAB code from file **octaves.m**

plain vanilla plot with system default choices for axes limits, tick marks, and no labels




frequencies on a piano keyboard
all 88 keys
'A' keys only
f (Hz)
3960
3520
3080
2640
2200
1760
1320
880
440
0
0
12
24
36
48
60
72
84
k, piano key

```
ka = 0:12:87;
fa = f(ka+1);
figure; plot(k,f, ka,fa,'ro');
```

red color, **o**pen circles

plot **fa** vs. **ka**, i.e., ‘A’ keys

```
% ka = [0, 12, 24, 36, 48, 60, 72, 84]
% fa = [27.5, 55, 110, 220, 440, 880, 1760, 3520]
```

Next, we add commands to annotate the graph with axis labels, axis limits, tick marks, grid, title, and legends

```
>> help plot
```

note: we defined **fa** as a subset of **f**, but we could have defined it directly as,

```
fa = f0 * 2.^(ka/12);
```

```
ka = 0:12:87;
fa = f(ka+1);
figure; plot(k,f, ka,fa,'ro');
```

red color, open circles

plot **fa** vs. **ka**, i.e., 'A' keys

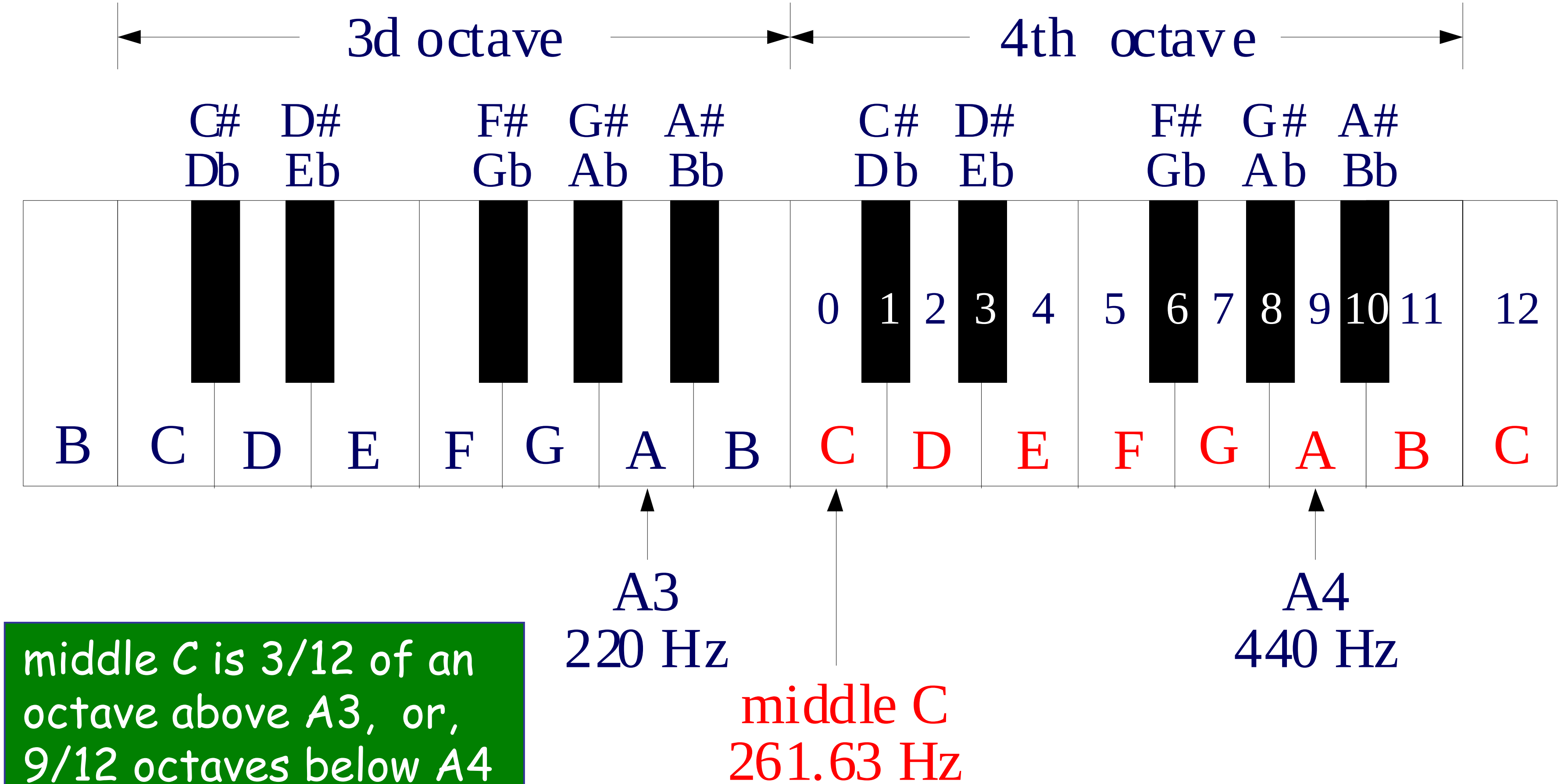
```
% ka = [0, 12, 24, 36, 48, 60, 72, 84]
% fa = [27.5, 55, 110, 220, 440, 880, 1760, 3520]

xlim([0,87]); set(gca,'xtick', 0:12:87);
ylim([0,4200]); set(gca,'ytick', 0:440:4000);
title('frequencies on a piano keyboard');
xlabel('{\itk},  piano key');
ylabel('{\itf}  (Hz)');
legend(' all 88 keys', ' ''A'' keys only', 'location', 'nw');
grid on;
print -depsc octave2.eps      % save plot in color EPS file
print -dmeta octave2.wmf      % save plot in windows metafile
```

set axis limits and tick marks

note: we defined **fa** as a subset of **f**, but we could have defined it directly as,

```
fa = f0 * 2.^(ka/12);
```



middle C is 3/12 of an octave above A3, or, 9/12 octaves below A4

$$261.63 = 220 \cdot 2^{3/12} = 440 \cdot 2^{-9/12}$$

4[th] octave keys, k = 0:12, MATLAB index = k+1 = 1:13

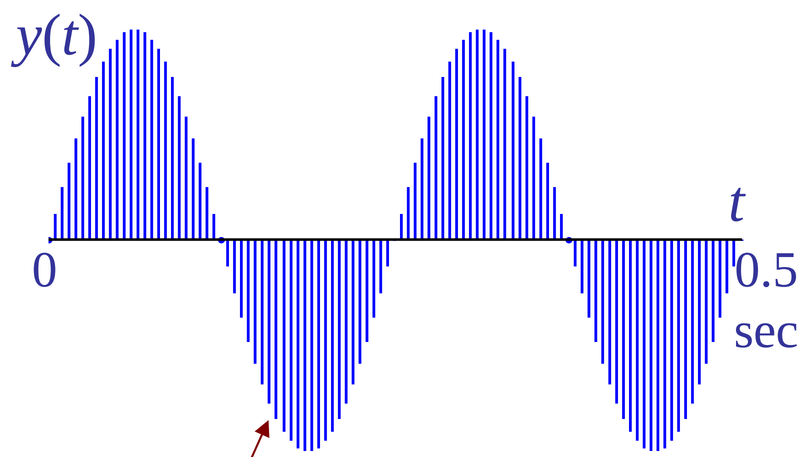
major keys are a subset of k, m = [0, 2, 4, 5, 7, 9, 11, 12] calculate frequencies in 4th octave

```
fc = 220 * 2^(3/12);        % middle C frequency
k = 0:12;                   % keys in 4th octave only
f = fc * 2.^(k/12);         % frequencies of 4th octave

% fc = 440 * 2^(-9/12); % alternative calculation
```

Next, for each *f* of the major keys, we generate a tone of half-second duration & play it on the PC's sound card (at the card's default sampling rate, fs = 8192 samples/sec):

$$y(t) = \sin(2\pi f t), \quad 0 \le t \le 0.5 \text{ sec}$$



time samples are spaced at the default sampling interval T = 1/fs = 0.122 msec

i.e., $t = 0 : T : 0.5$

generate & play major notes in 4th octave

```
fs = 8192; T = 1/fs;       % default sampling rate
Tmax = 0.5;                % half-second duration for notes
t = 0:T:Tmax;              % length(t) = 4097 points
                           % steps of T = 1/fs = 0.1221 msec

m = [0 2 4 5 7 9 11 12];  % major keys in 4th octave
                           % CDEFGABC = do re mi fa sol la si do

for i=m+1,                     % m+1 = [1 3 5 6 8 10 12 13]
  y = sin(2*pi*f(i)*t);        % y has half-second duration
  sound(y,fs);                 % play y at rate fs
end

pause;                       % pause until a key is depressed

for i=fliplr(m+1),           % fliplr(m+1)=[13 12 10 8 6 5 3 1]
   y = sin(2*pi*f(i)*t);
   sound(y,fs);              % play them in reverse order
end
```

formatted printing of frequencies and key names

```
 k      oct=k/12     f=fc*2^(k/12)      keys
----------------------------------------------
 0      0.0000          261.63         C    do
 1      0.0833          277.18         C#
 2      0.1667          293.66         D    re
 3      0.2500          311.13         D#
 4      0.3333          329.63         E    mi
 5      0.4167          349.23         F    fa
 6      0.5000          369.99         F#
 7      0.5833          392.00         G    sol
 8      0.6667          415.30         G#
 9      0.7500          440.00         A    la
10      0.8333          466.16         A#
11      0.9167          493.88         B    si
12      1.0000          523.25         C    do
```

cell arrays

```
% formatted printing of frequencies and key names
% define cell arrays of key names to facilitate printing

keys = {'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#',...
        'A', 'A#', 'B', 'C'};

doremi = {'do', '', 're', '', 'mi', 'fa', '', 'sol',...
                '', 'la', '', 'si', 'do'};

fprintf('\n');
fprintf(' k    oct=k/12   f=fc*2^(k/12)    keys\n');
fprintf('-----------------------------------------\n');
for i=k+1,
  fprintf('%2d    %1.4f        %3.2f          %s   %s\n', ...
          i-1, k(i)/12, f(i), keys{i}, doremi{i});
end
```

empty string

cell arrays use {…}

i-th entry of cell arrays

ellipsis continues to next line

```
>> help fprintf  % formatted printing
>> doc fprintf
```