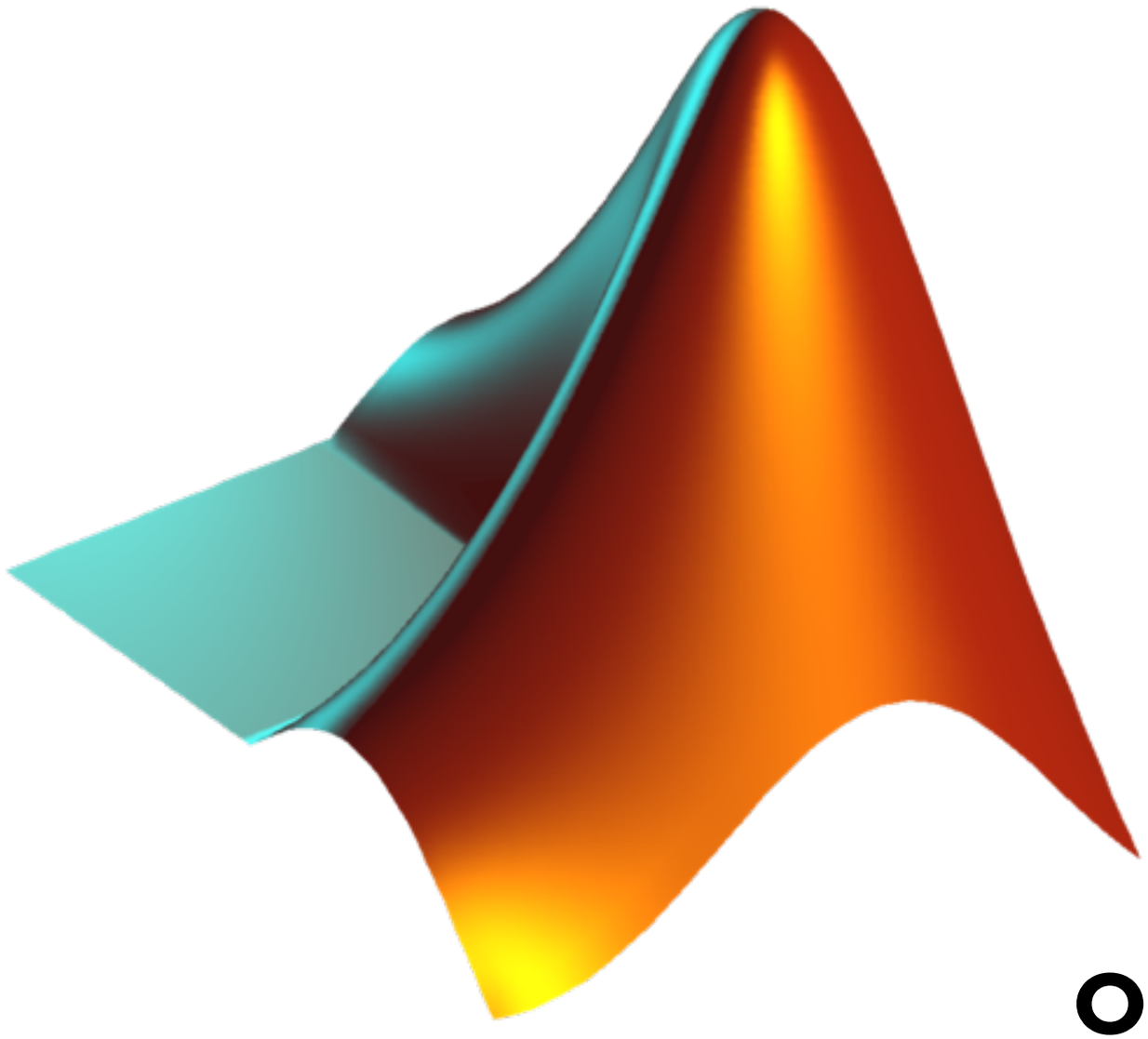


알기쉬운 MATLAB

DAY 2 **Lecture**
Vector Management
2015/1/5

고려대학교
KOREA UNIVERSITY

Outline

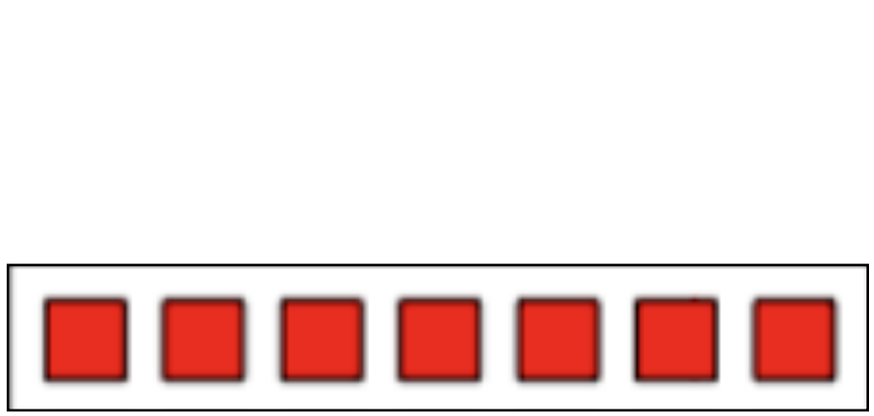
(1) What is Vector

(2) Why Vector is Good

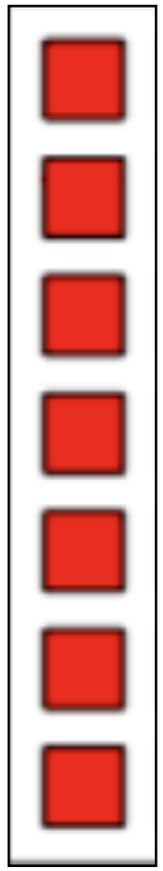
(3) Sound Processing

What is Vector

- An **array** of mathematical elements



1 x n

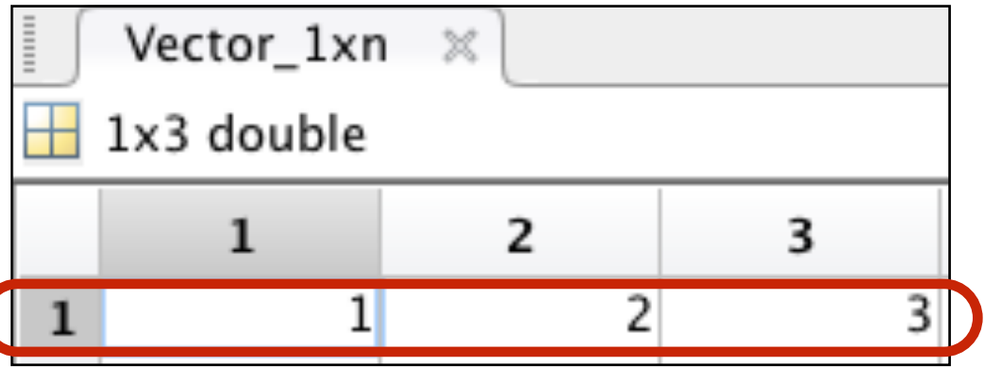


n x 1

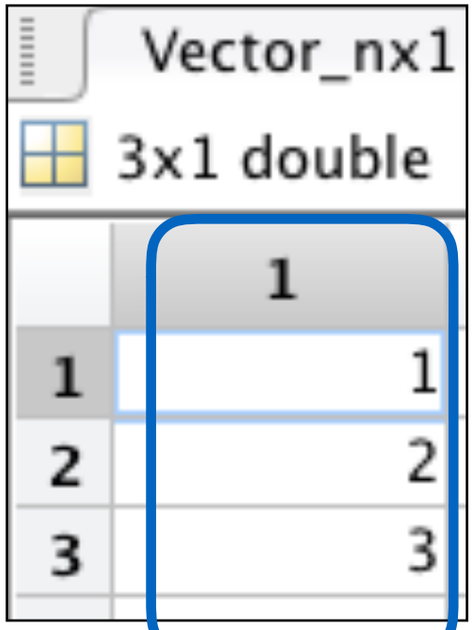
What is Vector

- A matrix with either one **row** or one **column**

row



1 x n



n x 1

column

Vector

- **Row** vector (1 x n)

```
Command Window
>> Vector_1xn = [1, 2, 3]

Vector_1xn =

     1     2     3
```

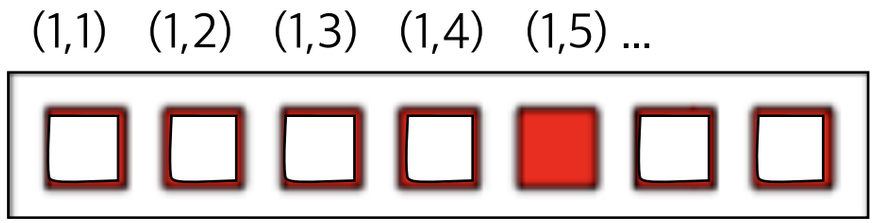
- **Column** vector (n x 1)

```
Command Window
>> Vector_nx1 = [1 ; 2 ; 3]

Vector_nx1 =

     1
     2
     3
```

Vector Access

- Access to a vector element

```
Command Window
  Vector_A =

       2     4     6     8    10

  >> Vector_A(1)

  ans =

       2

  >> Vector_A(1,1)

  ans =

       2

  >> Vector_A(1,5)

  ans =

      10
```

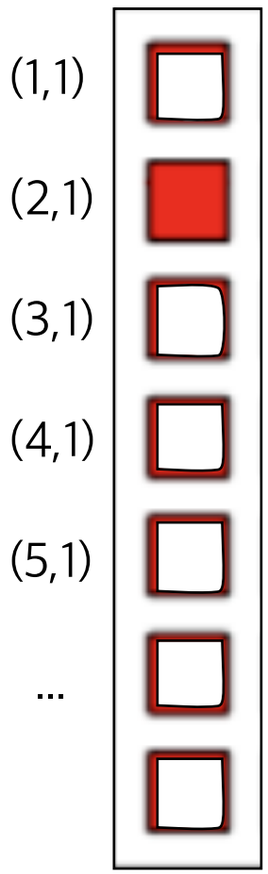


a = A(5)
= A(1,5)

Vector Access

- Access to a vector element

```
Command Window

Vector_B =

     1
     3
     5
     7
     9

>> Vector_B(2)

ans =

     3

>> Vector_B(4,1)

ans =

     7
```



a = A(2)
= A(2,1)

Vector Access

- Access to **multiple** vector elements
  - Use **colon (:)** to access to all the elements

A(:,n) entire **row** in column ‘n’

A(n,:) entire **column** in row ‘n’

A(:) entire **vector**

A(1 2 3 ; 4 5 6) 하고
A(1,[1 2])하면, A 1행의 1,2열이 Data Access
A(1,[1:3])도 가능함

Vector Access

- Access to **multiple** vector elements
  - Use **colon (:)** to access to all the elements

```
Command Window
  Vector_A =

       2     4     6     8    10
```

```
>> Vector_A(:,1)    <- entire rows in column 1

ans =

     2

>> Vector_A(1,:)    <- entire columns in row 1

ans =

     2     4     6     8    10

>> Vector_A(:)      <- entire vector
```

Vector Access

- Access to a **specific** vector element
  - Use '**end**' to access the last element

```
Command Window
>> Vector_A

Vector_A =

     2     4     6     8    10

>> Vector_A(end)

ans =

    10
```

end

Vector Generation

- Basic tools for generating vectors
  - **comma (,)** or space -> row
  - **semi-colon (;)** -> column

A = [1, 2, 3]

B = [1 2 3]

```
>> A = [1,2,3]

A =

     1     2     3

>> B = [1 2 3]

B =

     1     2     3
```

C = [1 ; 2 ; 3]

```
>> C = [1;2;3]

C =

     1
     2
     3
```

Vector Generation

- Vectors with **regular interval** (등차수열)
  - **colon (:)**

A = [n:m] `n, n+1, n+2, …, n+k (≤m)`

B = [n:p:m] `n, n+p, n+2p, …, n+kp (≤m)`

Vector Generation

- [n:m]

A = [1:5] 1, 2, 3, 4, 5

B = [2:4] 2, 3, 4

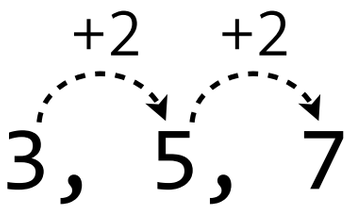
```
Command Window
>> A = [1:5]

A =

     1     2     3     4     5

>> B = [2:4]

B =

     2     3     4
```

Vector Generation

- [n:p:m]

C = [2:1:5]

D = [3:2:8]




```
Command Window
>> C = [2:1:5]

C =

     2     3     4     5

>> D = [3:2:8]

D =

     3     5     7
```

Vector Generation

- **Empty** vector
- Vector of **zeros** and **ones**

A = []

B = zeros(1, 4)

C = ones(3,1)

```
Command Window
>> A = []

A =

     []

>> B = zeros(1,4)

B =

     0     0     0     0

>> C = ones(3,1)

C =

     1
     1
     1
```

Vector Generation

- **Random** vector

A = rand(1, 3)

B = rand(5, 1)

C = rand(1,2,3) 하면
1,2 matrix의 rand가 3개 나옴
size(C) - matrix dimension 구하기
zeros(1,3,2)
size(ans)하면 1, 3, 2

```
Command Window
>> A = rand (1,3)

A =

   -0.2050   -0.1241    1.4897

>> B = rand (5,1)

B =

    1.4090
    1.4172
    0.6715
   -1.2075
    0.7172
```

Vector Concatenation

- **Concatenating** vectors
  - Dimensions must match

C = [A B]

```
Command Window
>> A = [1 2 3];
>> B = [4 5 6];
>> C = [A B]

C =

     1     2     3     4     5     6
```

D = [A ; B]

```
Command Window
>> D = [A ; B]

D =

     1     2     3
     4     5     6
```

Vector Concatenation

- **Concatenating** vectors
  - Rows and columns must match

```
Command Window
>> Vector_A

Vector_A =

     2     4     6

>> Vector_B

Vector_B =

     1
     3
     5
```

- Use **transpose**(') to match the dimensions

C = [A B']

```
Command Window
>> C = [Vector_A Vector_B]
Error using horzcat
Dimensions of matrices being concatenated are not consistent.

>> C = [Vector_A Vector_B']

C =

     2     4     6     1     3     5
```

Vector Deletion

- **Delete** vectors
  - Assign an empty matrix

A(4) = []

[] = 0,0의 Matrix로 봐야

```
Command Window
Vector_A =

     2     4     6     8    10

>> Vector_A(4) = []  <- delete 4th element

Vector_A =

     2     4     6    10
```

Vector Deletion

- **Delete** vectors
  - Assign an empty matrix

A(1,:) = []

A(:,1) = []

```
Command Window
Vector_A =

     2     4     6     8    10

>> Vector_A(:,1) = [] <- delete column 1

Vector_A =

     4     6     8    10

>> Vector_A(1,:) = [] <- delete row 1

Vector_A =

   Empty matrix: 0-by-4
```

Vector Deletion

- **Delete** vectors
  - Assign an empty matrix

A(:) = []

```
Command Window
>> Vector_A = [2 4 6 8 10]

Vector_A =

     2     4     6     8    10

>> Vector_A(:) = []
Vector_A =

     []
```

<- delete entire A

Vector Calculation

- Addition (+)

```
Command Window
>> Vector_A = [2 4 6 8 10]

Vector_A =

     2     4     6     8    10

>> Vector_A + 3

ans =

     5     7     9    11    13

>> Vector_A + ones(1,5)

ans =

     3     5     7     9    11
```

- Subtraction (-)

```
Command Window
>> Vector_A = [2 4 6 8 10]

Vector_A =

     2     4     6     8    10

>> Vector_A - 3

ans =

    -1     1     3     5     7

>> Vector_A - randi(5,1)

ans =

    -2     0     2     4     6
```

Vector Calculation

- Multiplication (*)

```
Command Window
  Vector_A =

       2     4     6     8    10

  >> Vector_A * 2

  ans =

       4     8    12    16    20

  >> Vector_A * Vector_A'

  ans =

     220
```

- **Element-wise** Multiplication (.*)

```
Command Window
  Vector_A =

       2     4     6     8    10

  >> Vector_A .* Vector_A

  ans =

       4    16    36    64   100
```

Outline

(1) What is Vector

(2) Why Vector is Good

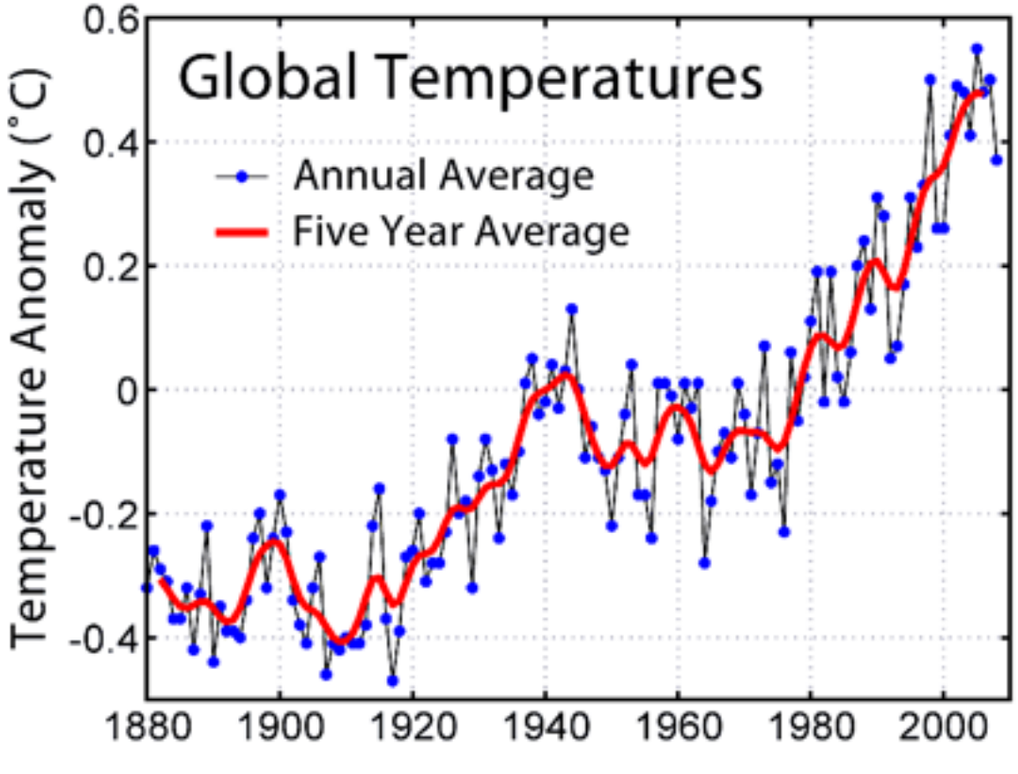
(3) Audio Processing

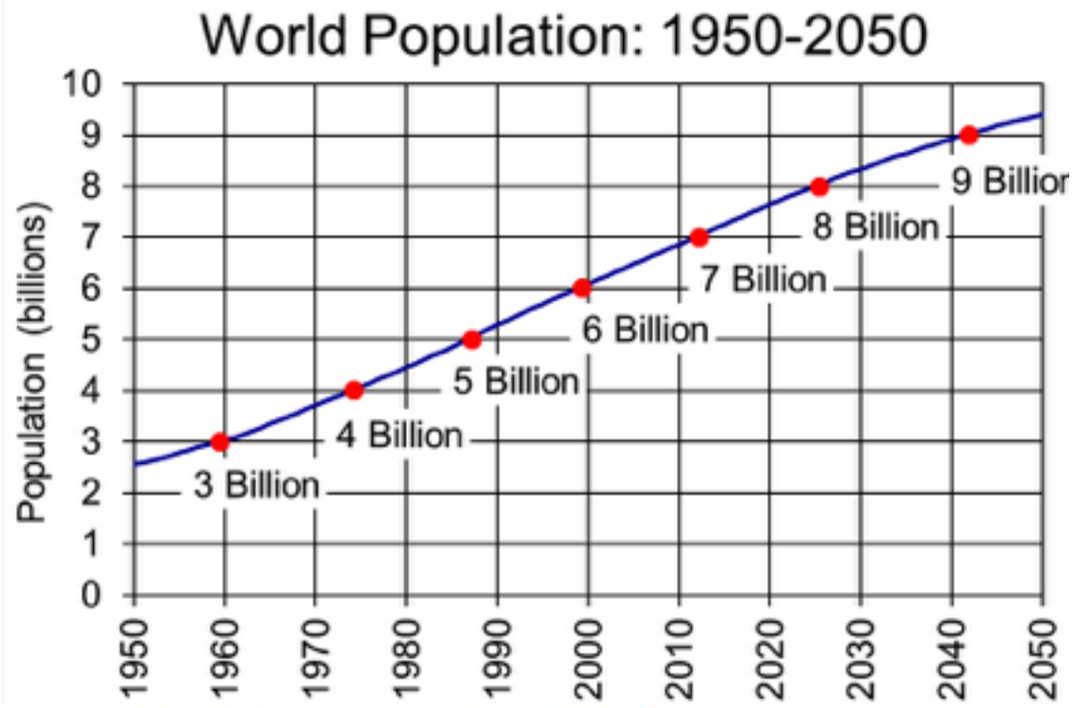
Why Vector is Good

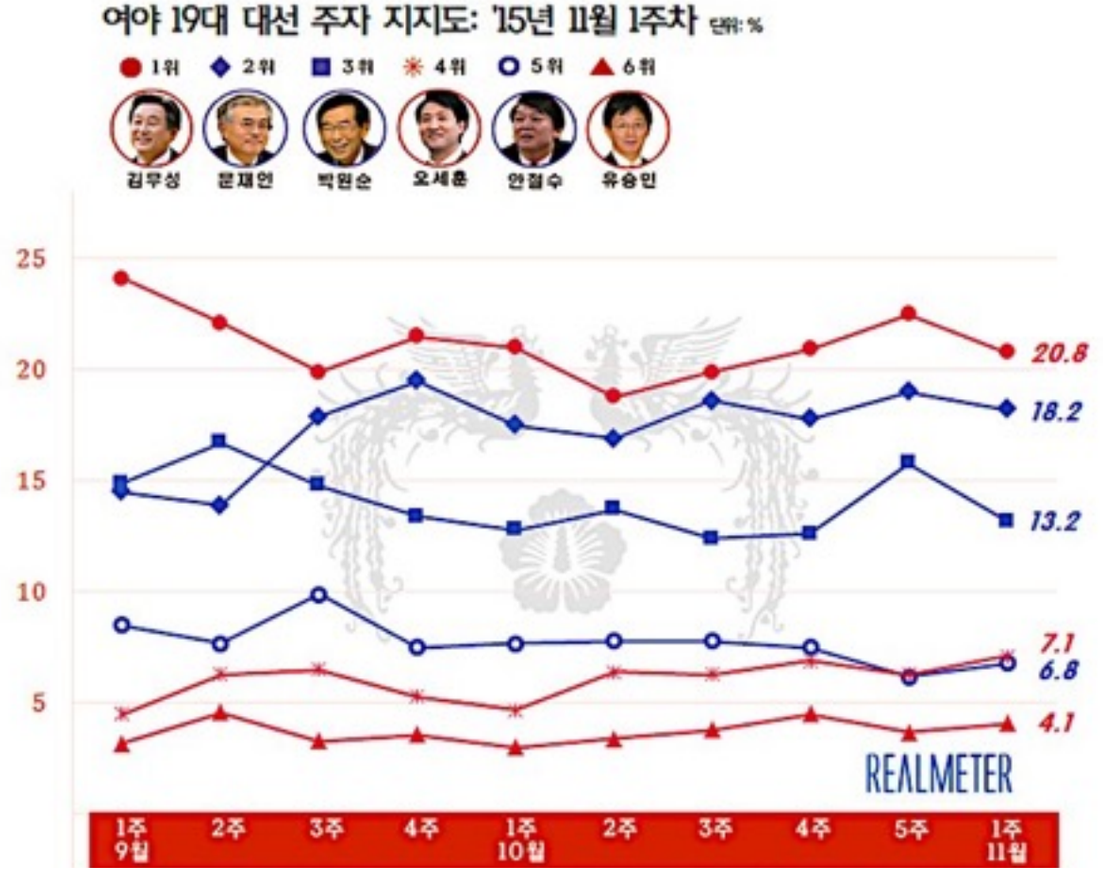
- Vector is all around us
  - What we call '**data**' is a group of **vectors**

= Vector

Vector is everywhere



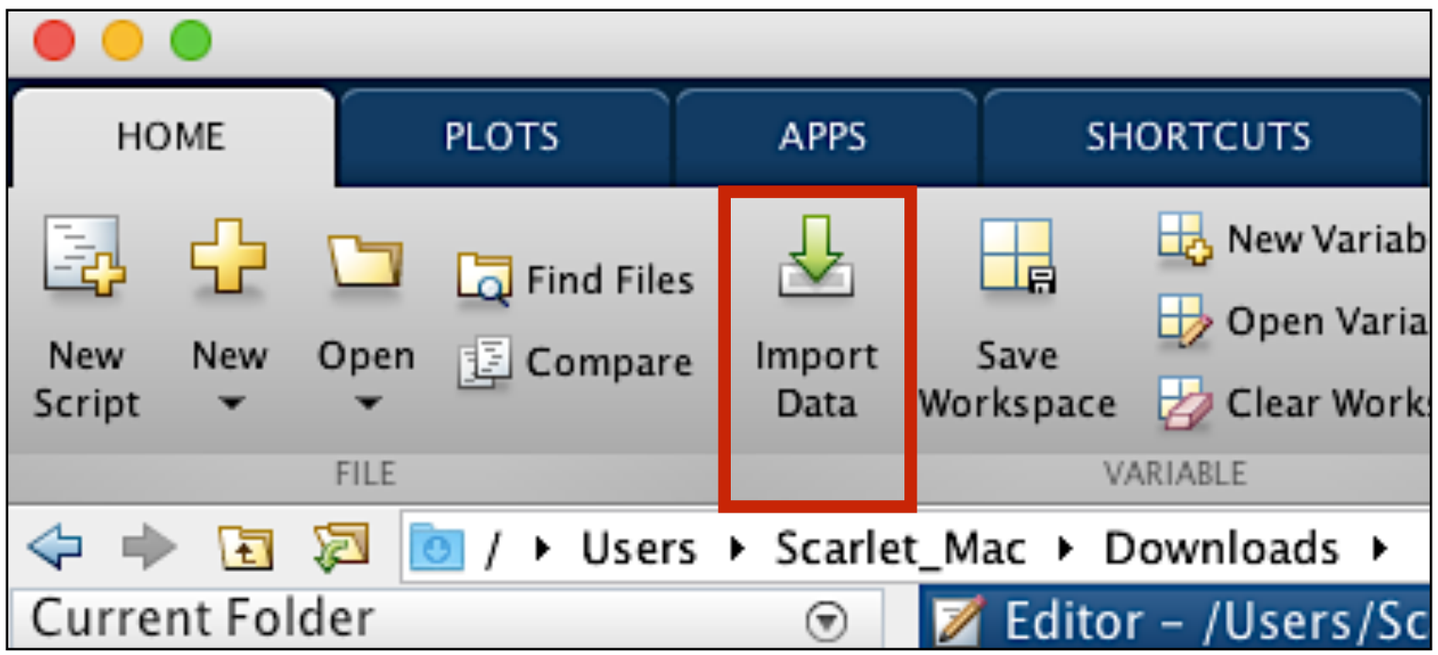




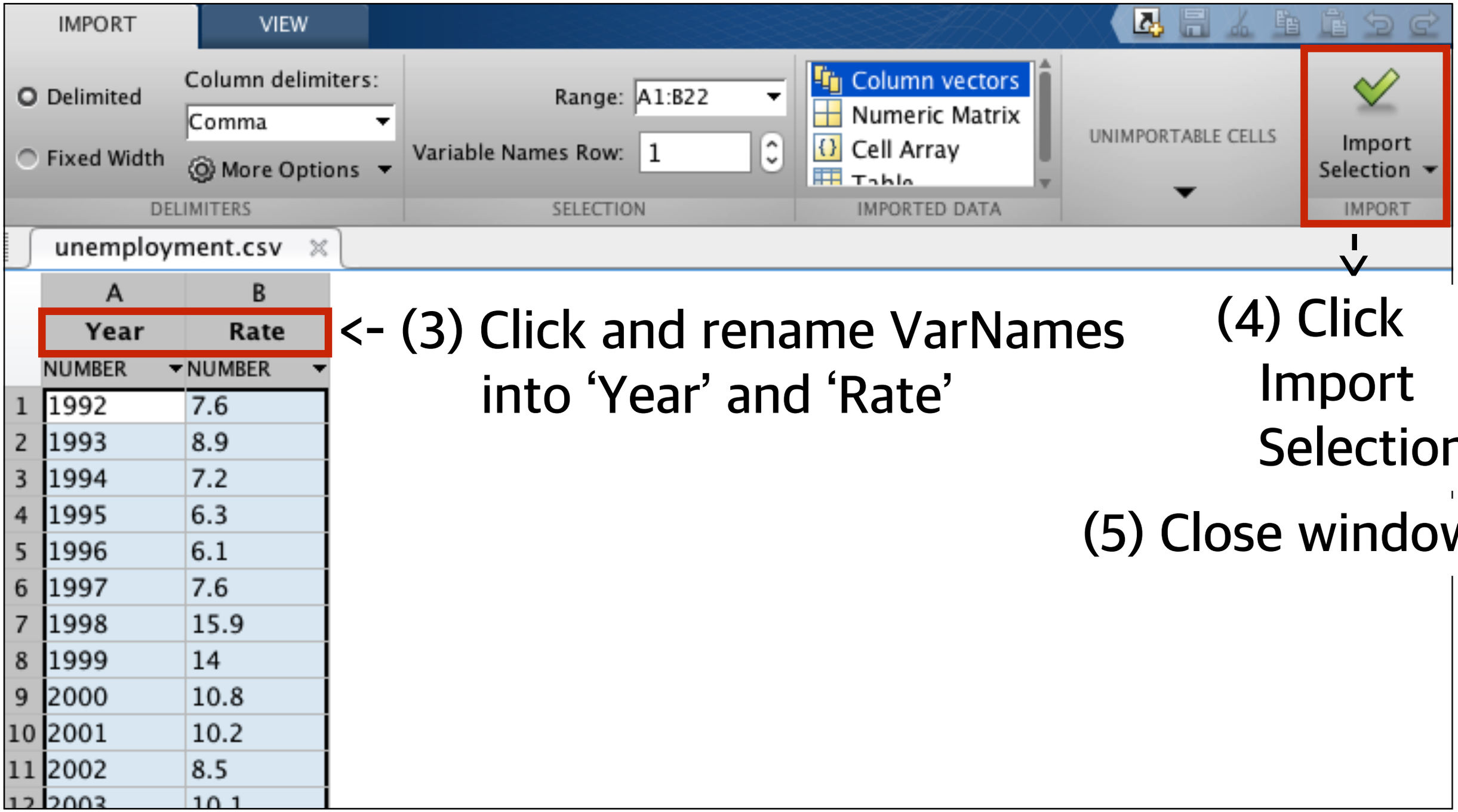
Vector is everywhere

- Vectors from real world
  - **Youth unemployment rate** in Korea (1992-2013)

(1) **Download** 'unemployment.csv' sent via email
(2) Click to **Import Data**



Vector is everywhere

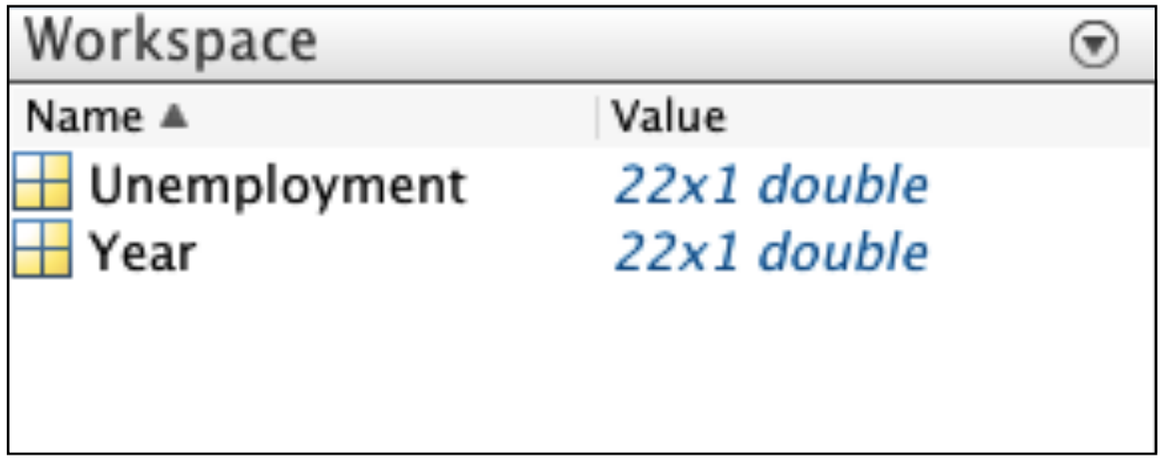


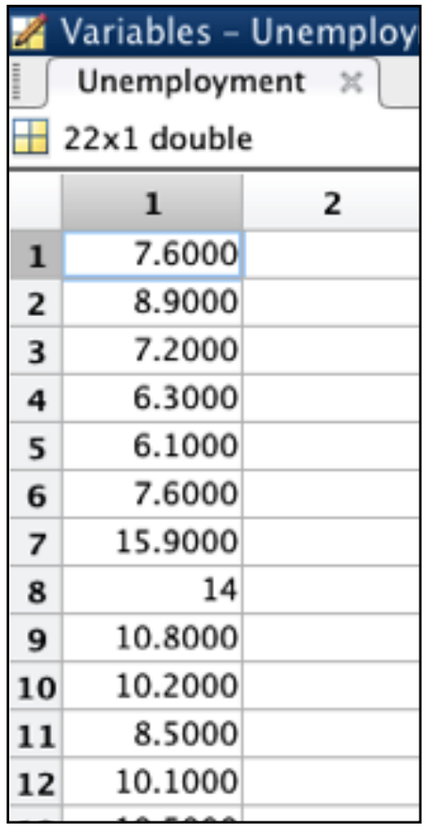
<- (3) Click and rename VarNames into ‘Year’ and ‘Rate’

(4) Click Import Selection

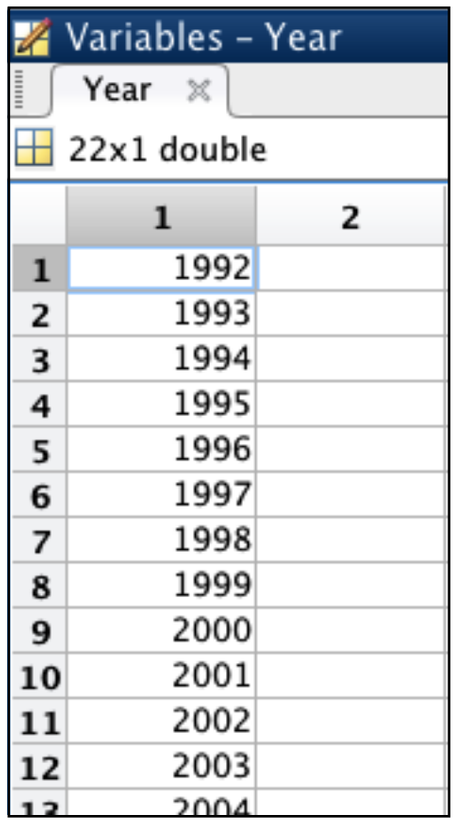
(5) Close window

Vector





	1	2
1	7.6000	
2	8.9000	
3	7.2000	
4	6.3000	
5	6.1000	
6	7.6000	
7	15.9000	
8	14	
9	10.8000	
10	10.2000	
11	8.5000	
12	10.1000	



	1	2
1	1992	
2	1993	
3	1994	
4	1995	
5	1996	
6	1997	
7	1998	
8	1999	
9	2000	
10	2001	
11	2002	
12	2003	
13	2004	

- Two **column vectors** (22x1)
  - Unemployment
  - Year

Vector

- **Descriptive statistics**
  - mean / mode / median
  - standard deviation
    표준편차 구하기

```
Command Window
>> std(Unemployment)

ans =

    2.2056
```

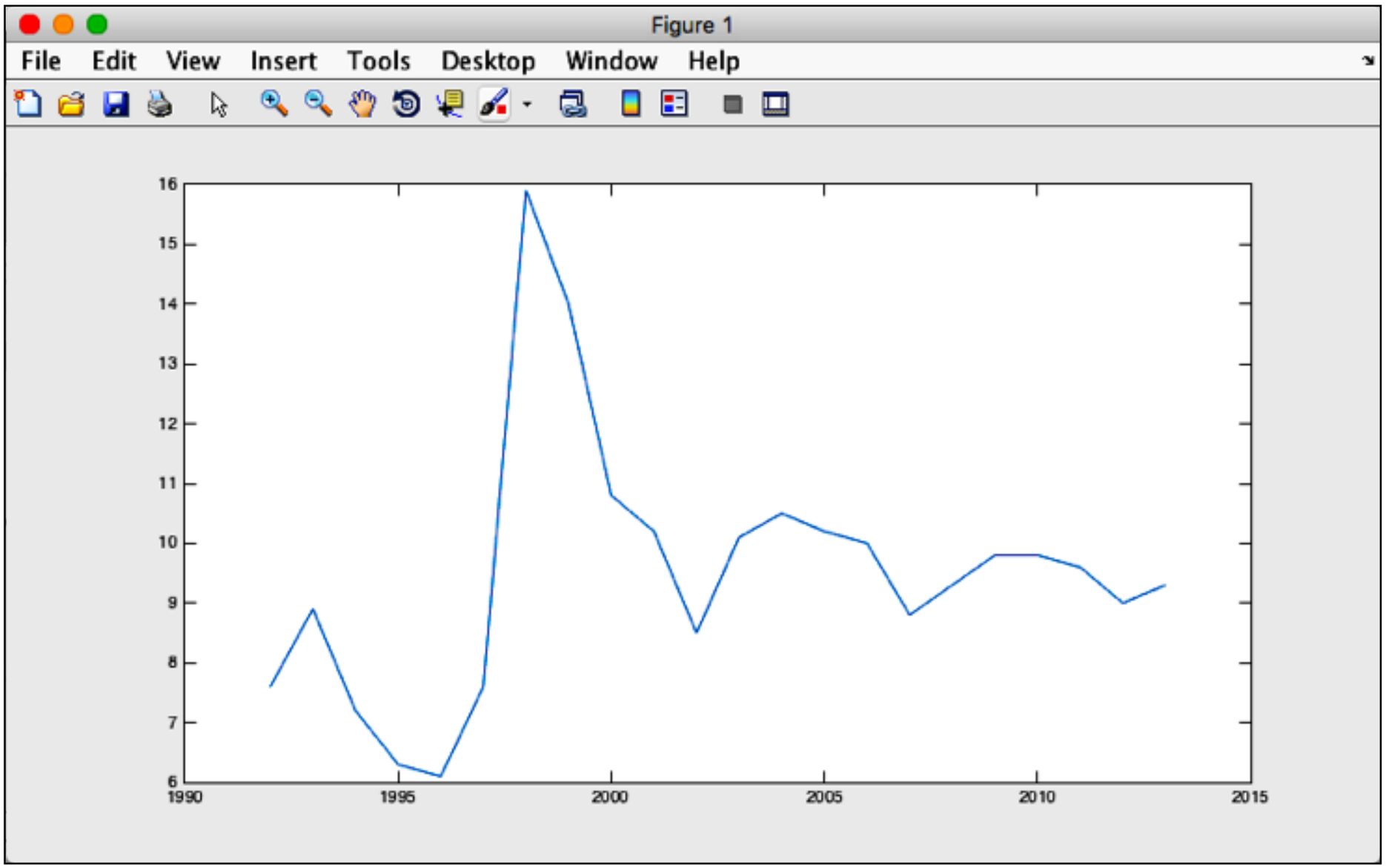
```
Command Window
>> mean(Unemployment)

ans =

    9.5227

>> median(Unemployment)

ans =

    9.4500

>> mode(Unemployment)

ans =

    7.6000
```

Plotting vectors

- Plotting vectors

```
>> plot(Year,Unemployment)
```

plot(x,y)

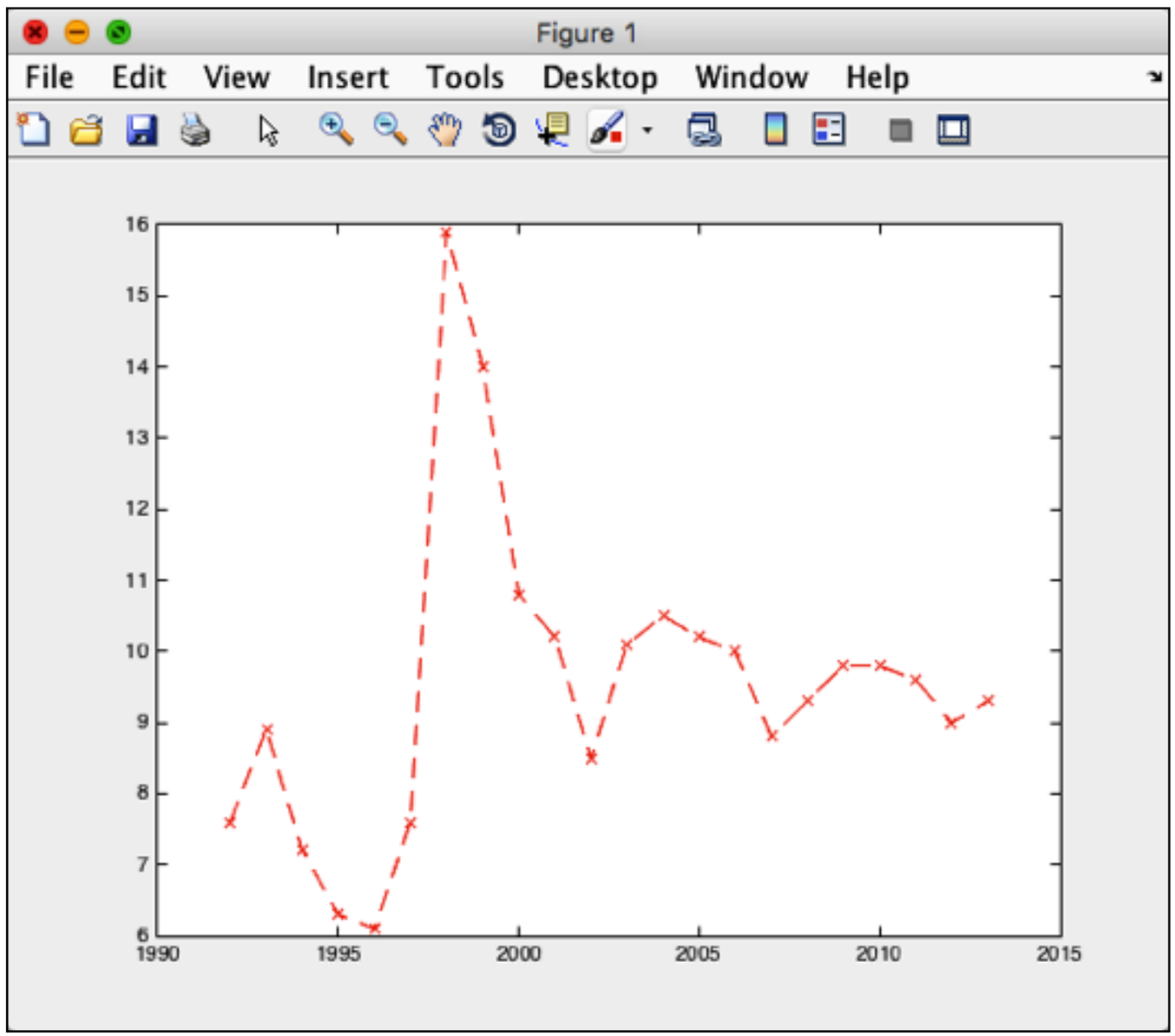


Plotting vectors

- Plot specification

```
>> plot(Year,Unemployment,'r--x')
```

plot(x,y, '_')



'**r--x**'

r (red)

-- (dashed line)

x (x markers)

':mx' (magenda dotted lines)

Plotting vectors

- Plot specification

Long Name	Short Name
'yellow'	'y'
'magenta'	'm'
'cyan'	'c'
'red'	'r'
'green'	'g'
'blue'	'b'
'white'	'w'
'black'	'k'

String	Line Style
'-'	Solid line
'--'	Dashed line
':'	Dotted line
'-.'	Dash-dotted line
'none'	No line

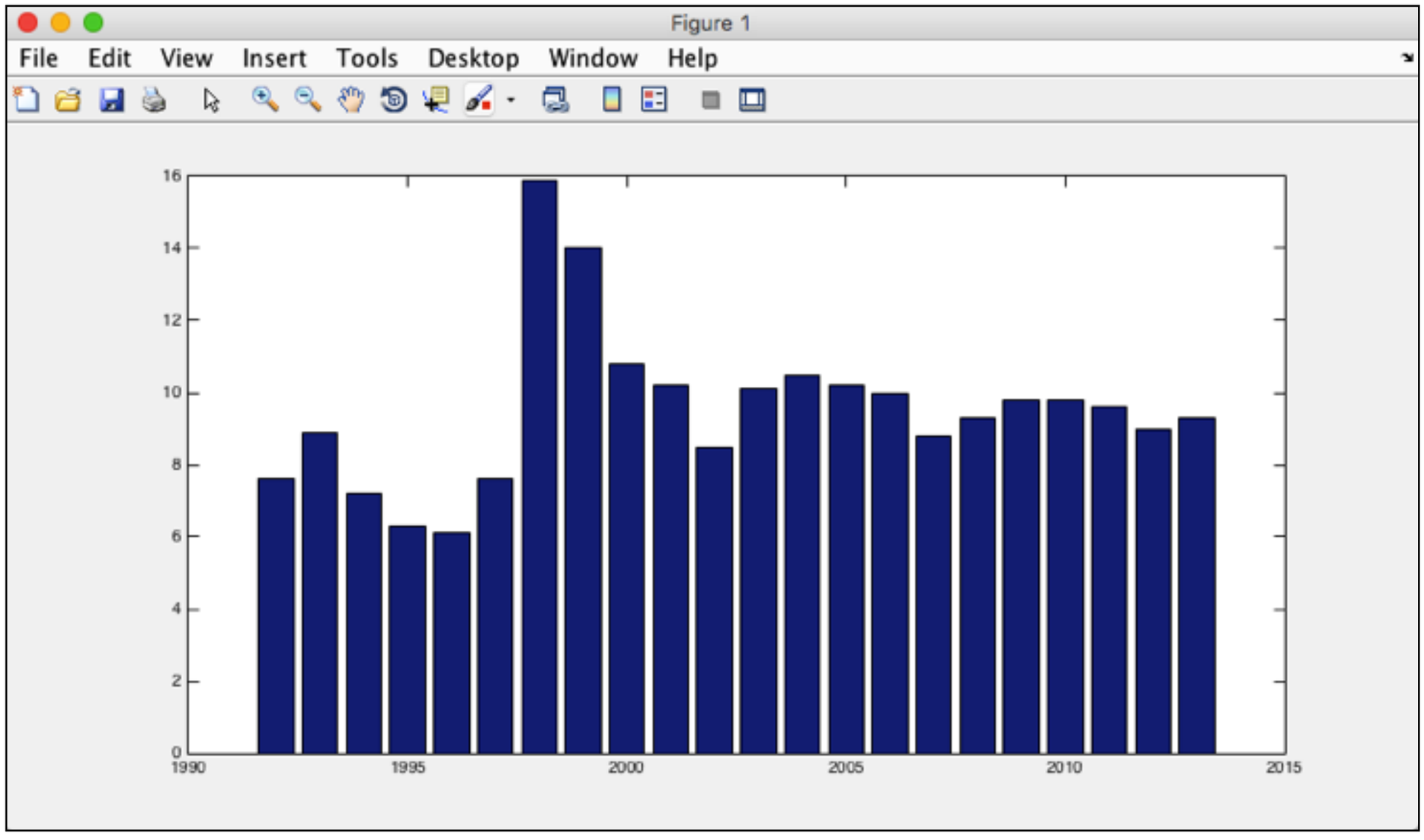
String	Marker Symbol
'o'	Circle
'+'	Plus sign
'*'	Asterisk
'.'	Point
'x'	Cross

Plotting vectors

- Plotting vectors

```
>> bar(Year,Unemployment)
```

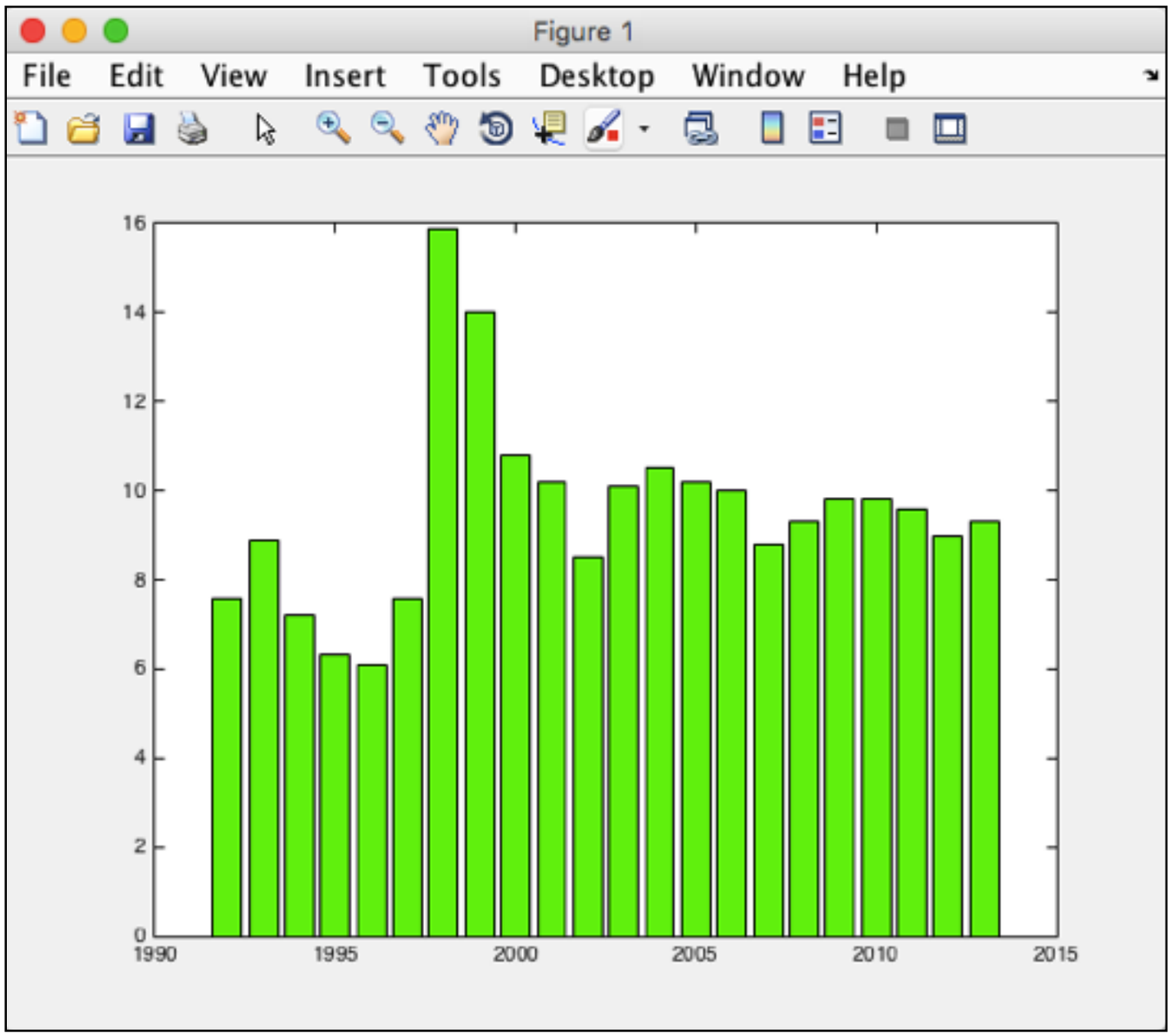
bar(x,y)



Plotting vectors

- Plot specification

```
>> bar(Year,Unemployment,'g')
```

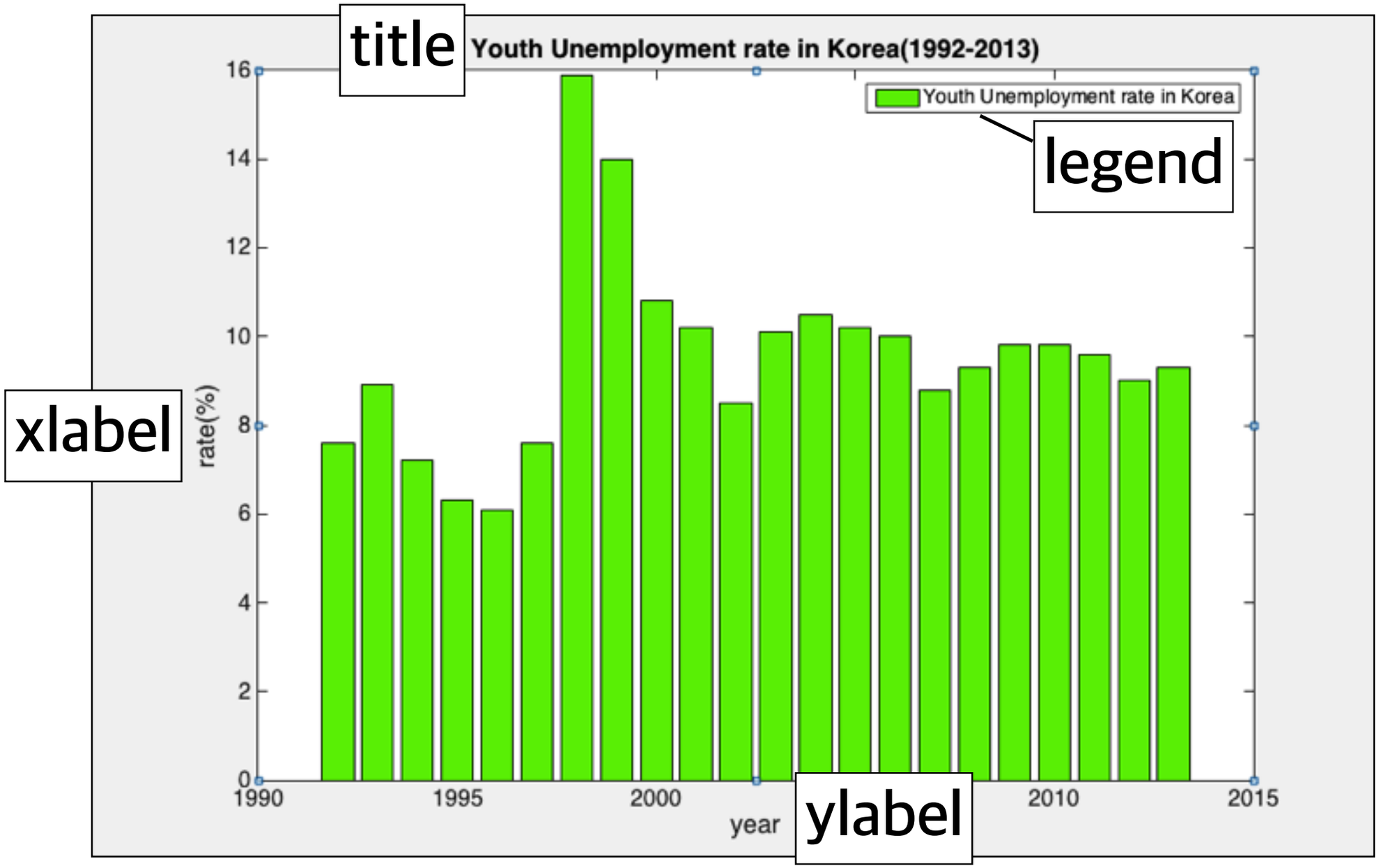


bar(x,y, '_')

'**g**' (green)

title / label / legend

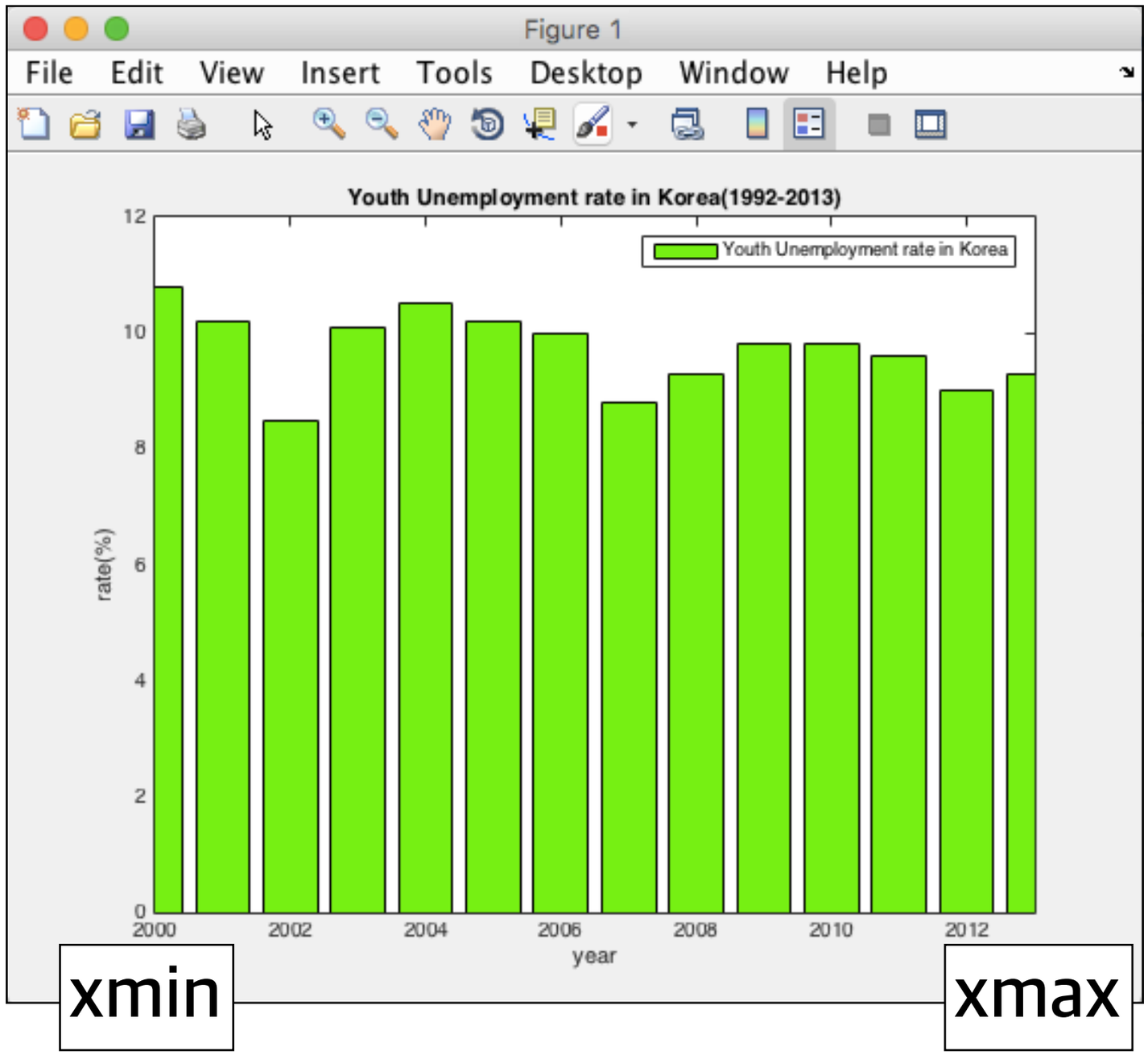
```
>> title('Youth Unemployment rate in Korea(1992-2013)')
>> xlabel('year')
>> ylabel('rate(%)')
>> legend('Youth Unemployment rate in Korea')
```



Axis range

```
Command Window
>> xlim([2000 2013])
```

xlim([xmin xmax])

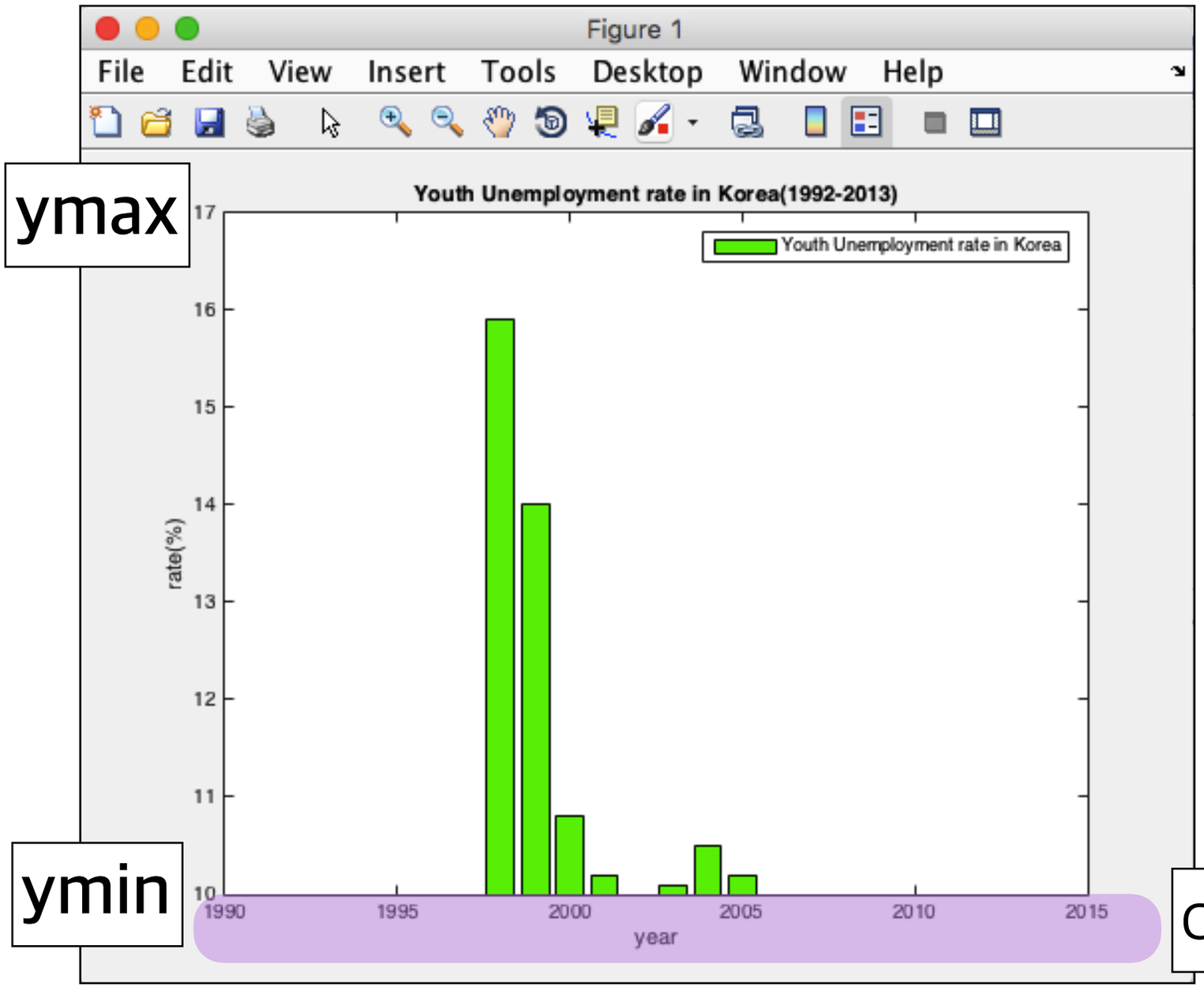


어디서부터
어디까지만
보고싶으면

Axis range

```
Command Window
>> xlim auto
>> ylim([10 17])
```

ylim([ymin ymax])



ymax

ymin

cf. xlim **auto**

Outline

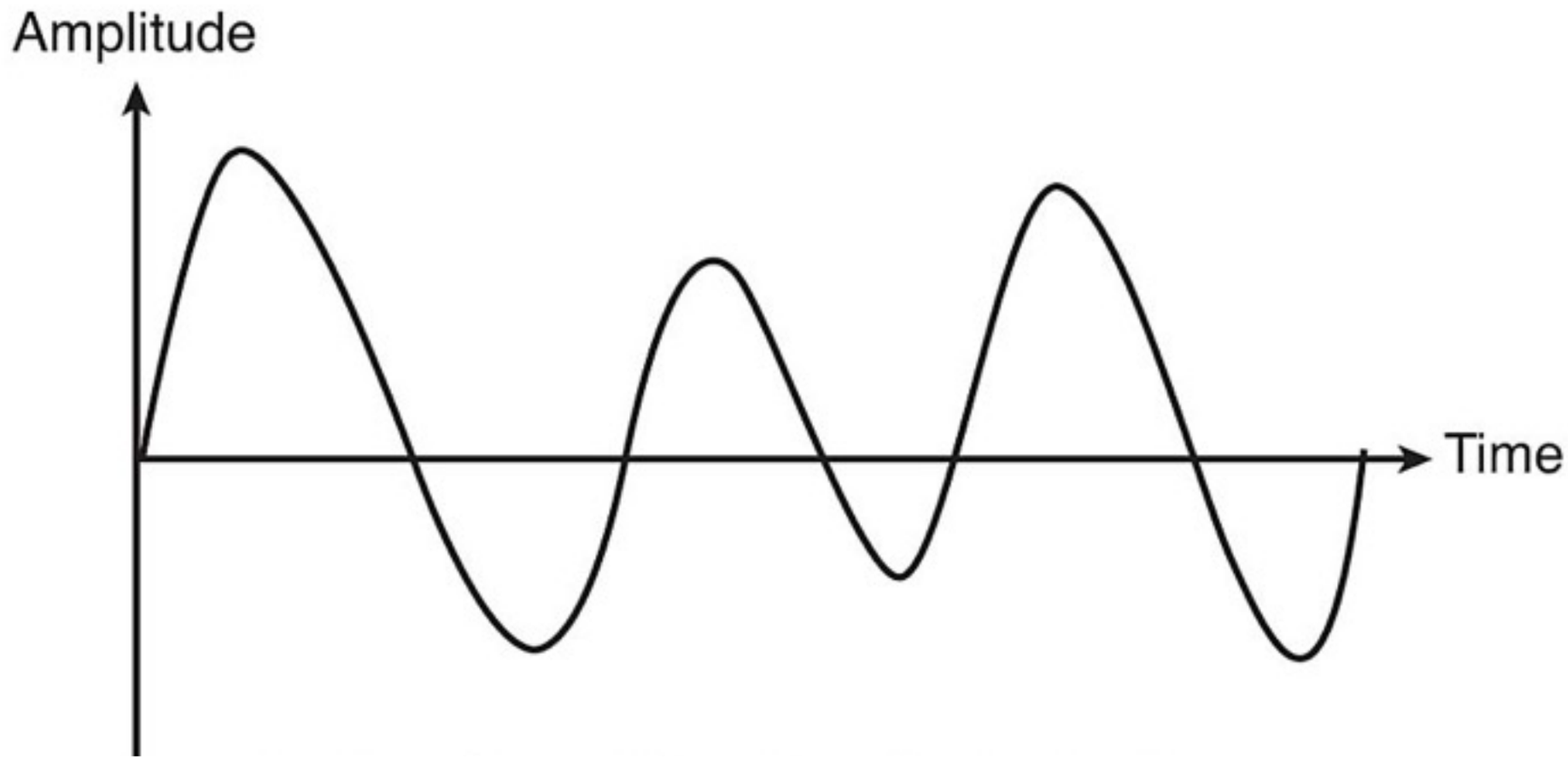
(1) What is Vector

(2) Why Vector is Good

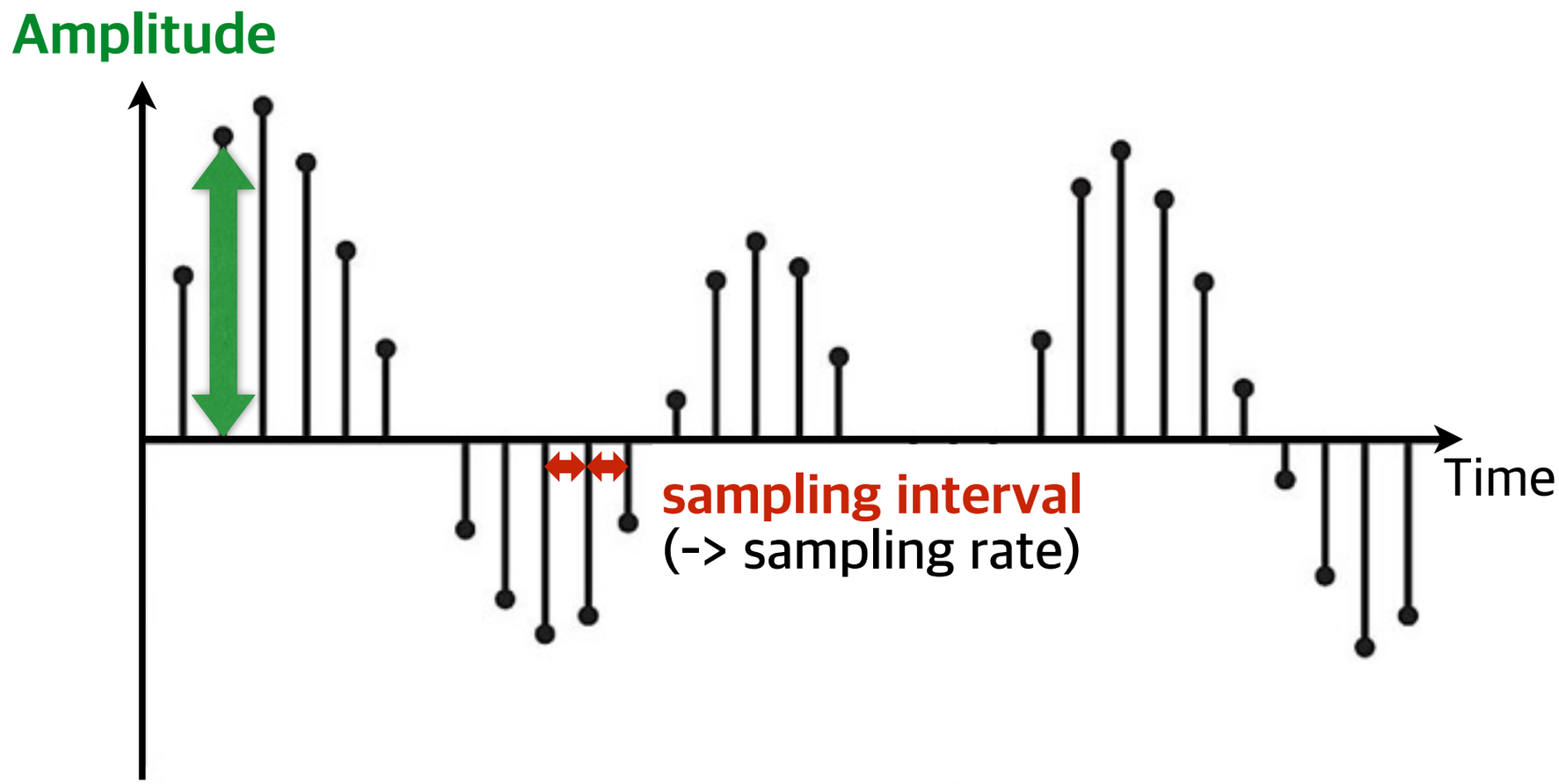
(3) Audio Processing

Sound as Vector

Analog signal



Digital signal



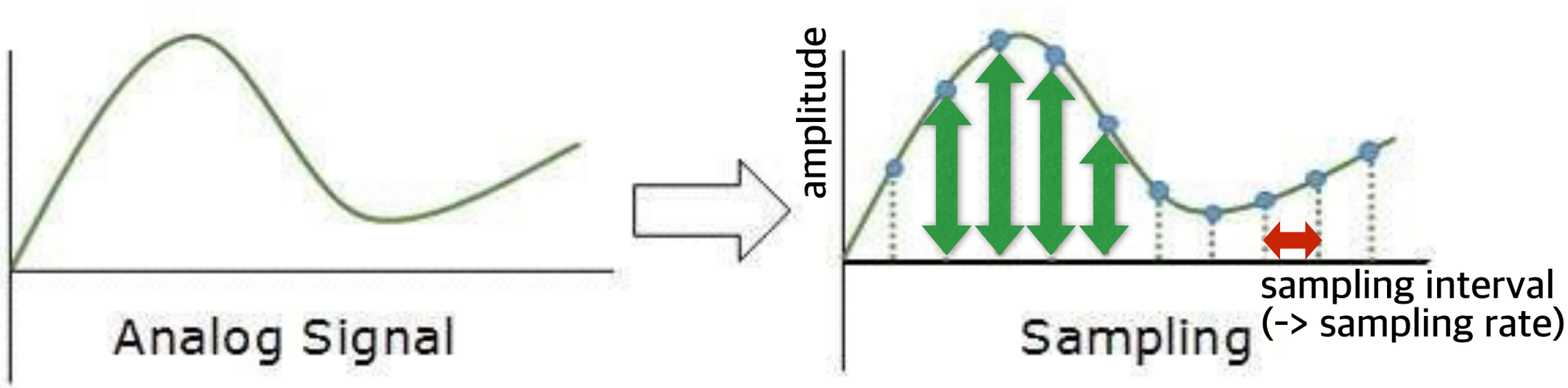
Why sample?

- Why can't use analog in computer?
  - can't deal with an infinite number of values
  (e.g. how many numbers between 0 and 0.1?)

- Needs to convert **analog to digital**
  - by chopping continuous signal into **discrete samples**
  - called '**sampling**' (or 'digitizing')

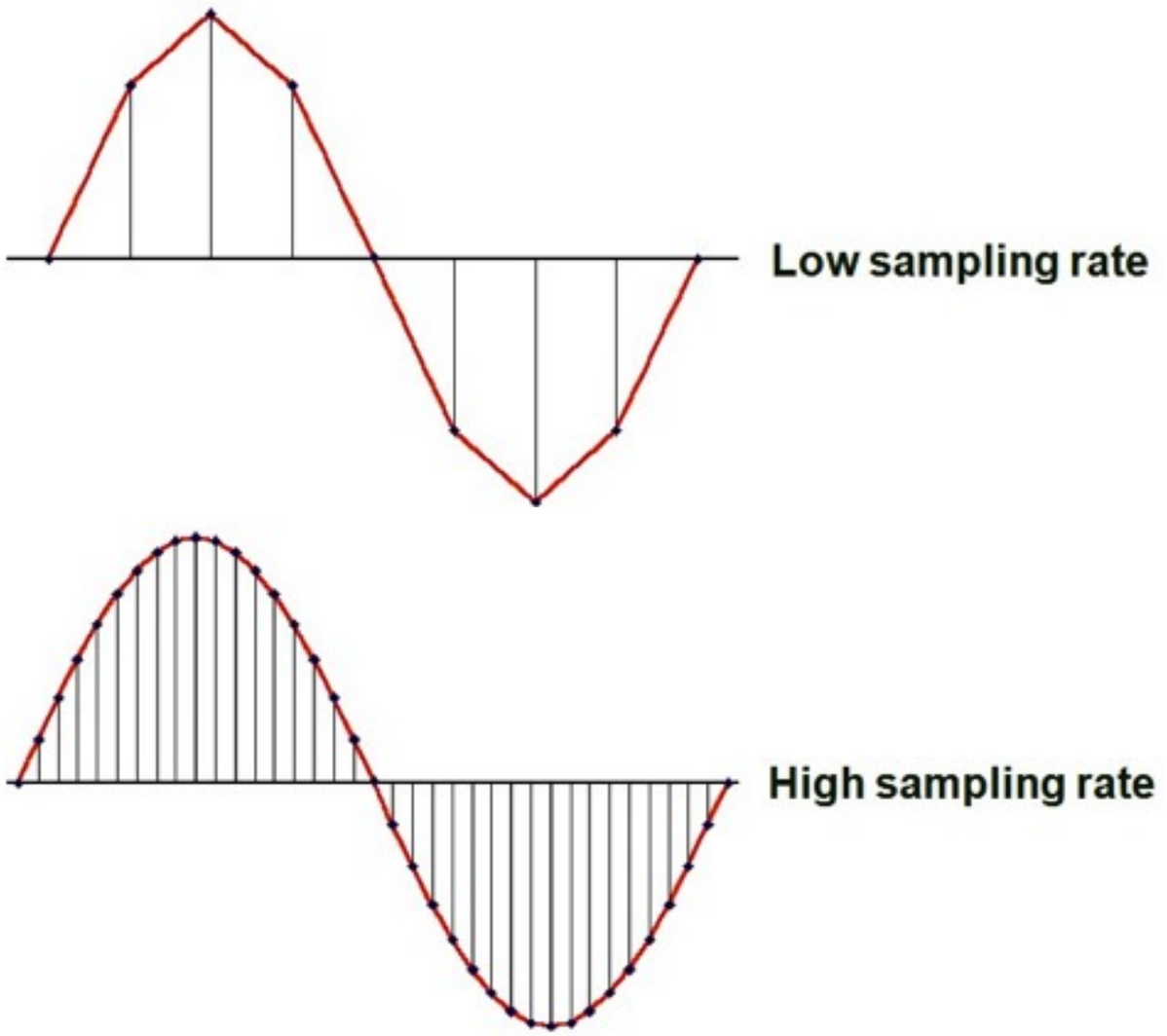
Sound as Vector

- Sound
  (1) **Amplitude** … y value
  (2) **Sampling rate** … x value (interval)



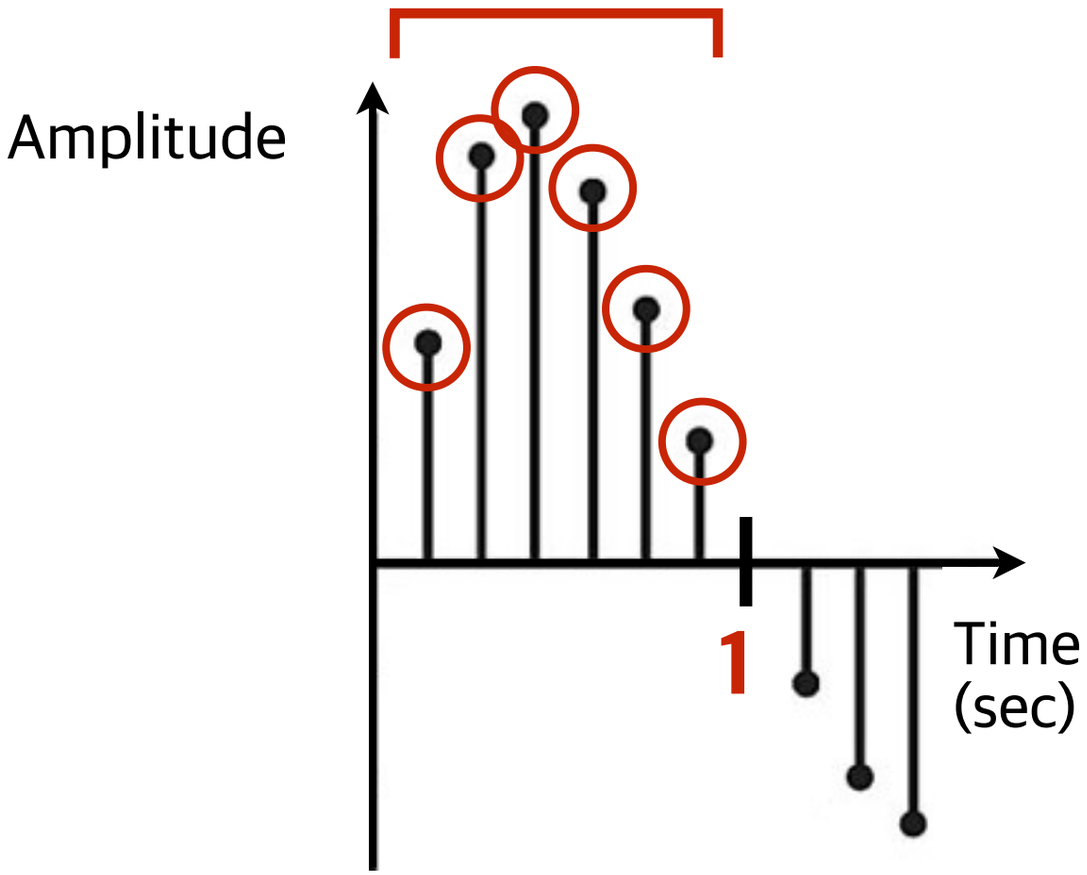
Sampling rate

- **Resolution** of sound signal (= ‘sampling frequency’)
- How many **samples per second**?



Sampling rate

- **Resolution** of sound signal (= ‘sampling frequency’)
- How many **samples per second**?

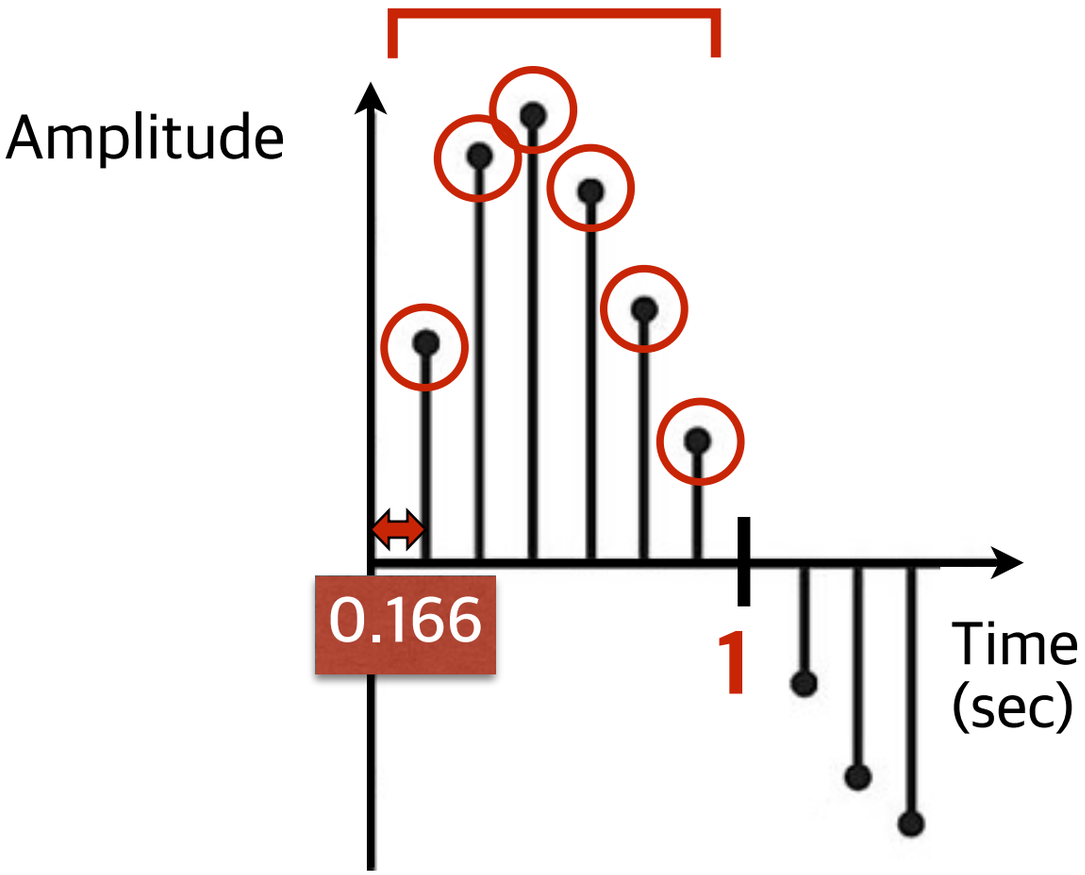


6 samples per second
= 6 Hz

cf. 44,100Hz

Sampling interval

- How to interpret **sampling rate** to **time (sec)?**
  - 1 / sampling rate = sampling interval

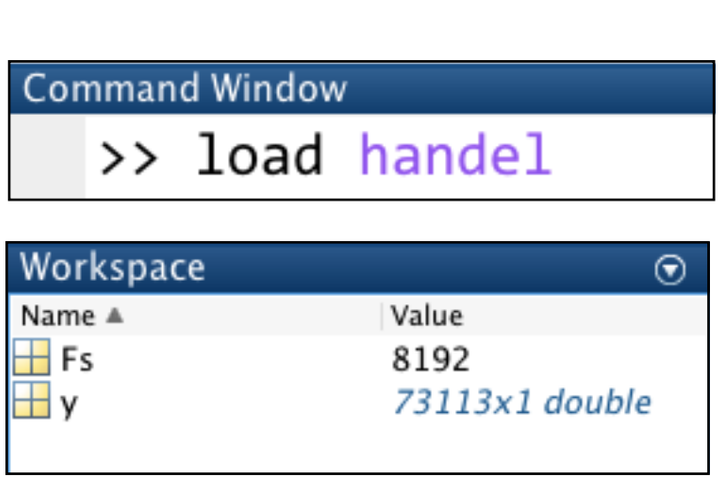


6 samples per second
= 6 Hz

1 sec / 6 samples
= 0.166 sec

Sound as Vector

- Sound data in Matlab
  Fs : sampling rate (‘sr’)
  y : amplitude data (‘sig’)



Variables – Fs

Fs

1x1 double

	1
1	8192
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	

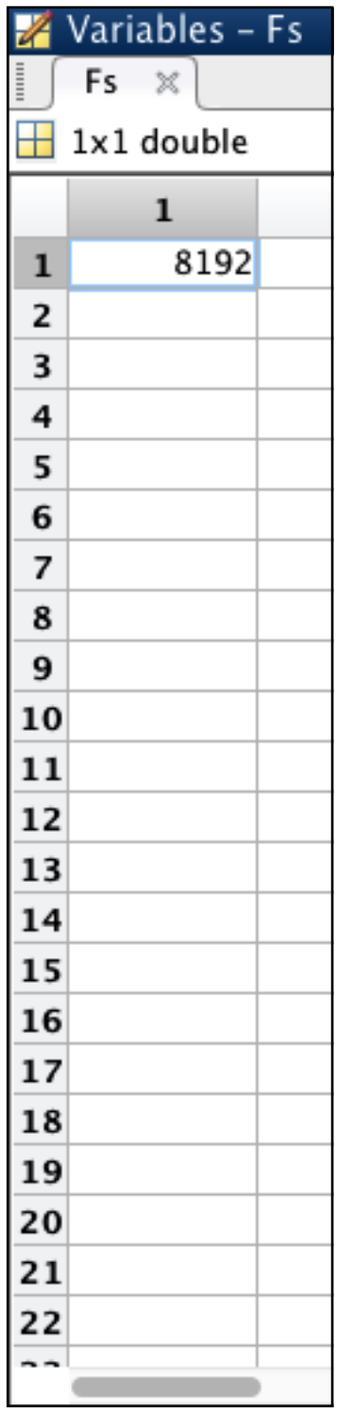
Variables – y

y

73113x1 double

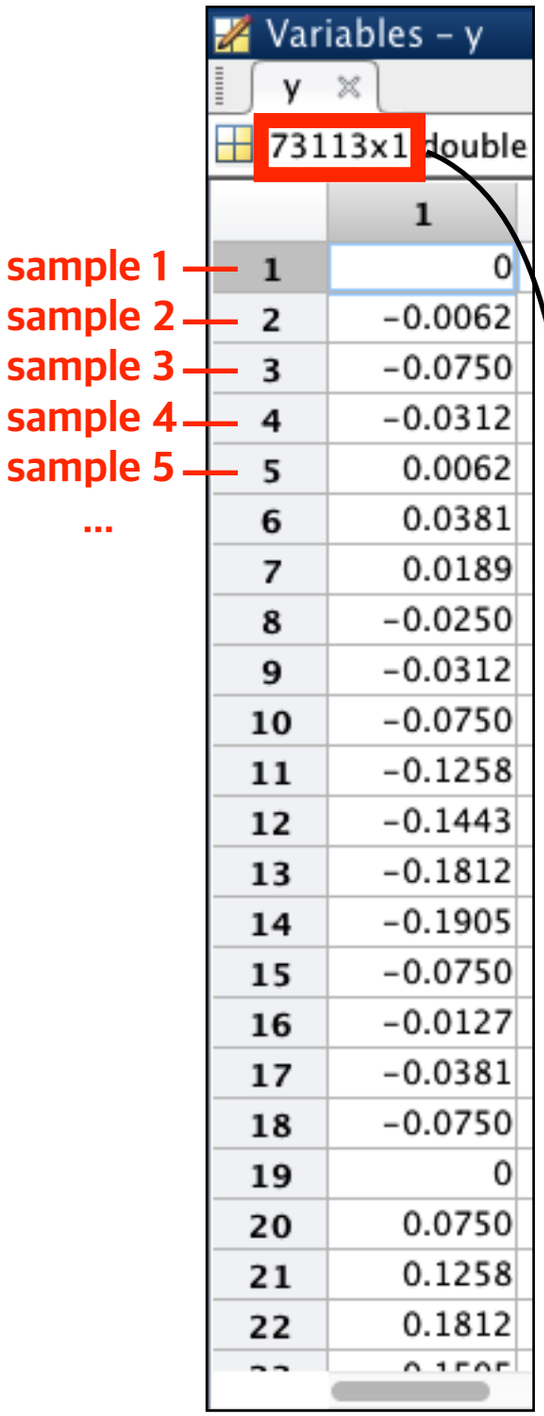
	1
1	0
2	-0.0062
3	-0.0750
4	-0.0312
5	0.0062
6	0.0381
7	0.0189
8	-0.0250
9	-0.0312
10	-0.0750
11	-0.1258
12	-0.1443
13	-0.1812
14	-0.1905
15	-0.0750
16	-0.0127
17	-0.0381
18	-0.0750
19	0
20	0.0750
21	0.1258
22	0.1812

Fs



- **Fs** (sampling rate)
  number of samples per second
  = 8192

y



- **y**

  amplitude data of each **sample**

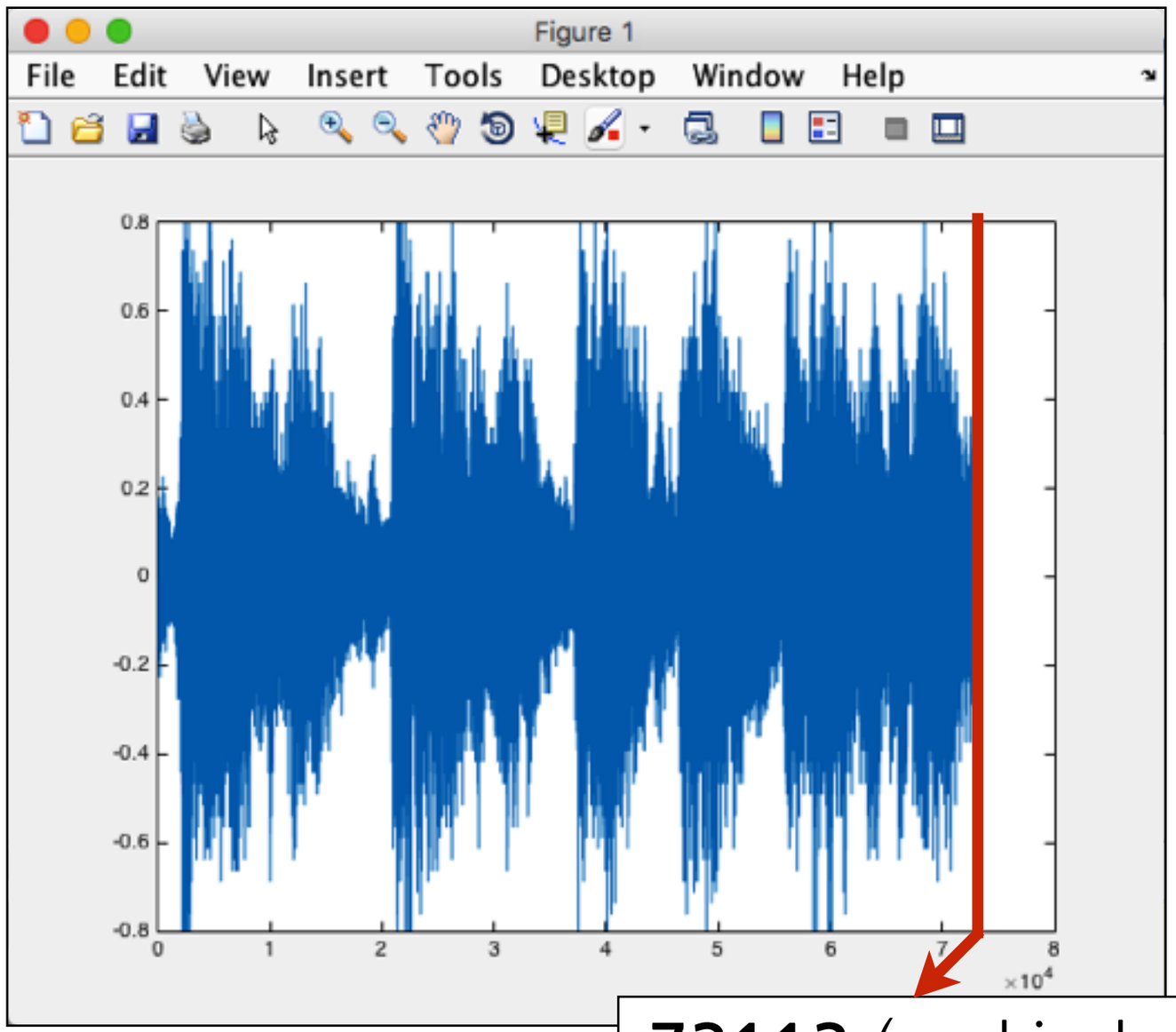
  (not of each **second**)

- 73113 = **number of samples**

Plotting sounds

- **plot (y)**
  x-axis: **sample**
  y-axis: amplitude

```
Command Window
>> plot(y)
```



73113 (end index)
= **number of samples**

time

Q: Temporal interval?

sample 1
sample 2
sample 3
sample 4
sample 5
...

Variables – y

y

73113x1 double

	1
1	0
2	-0.0062
3	-0.0750
4	-0.0312
5	0.0062
6	0.0381
7	0.0189
8	-0.0250
9	-0.0312

time

- **Sampling interval (sec)**
  = 1 / sampling frequency Fs 8192
  = 1 / 8192 (= 0.12 ms)

0.12 ms
0.12 ms
0.12 ms
0.12 ms
...

sample 1
sample 2
sample 3
sample 4
sample 5
...

y

73113x1 double

	1
1	0
2	-0.0062
3	-0.0750
4	-0.0312
5	0.0062
6	0.0381
7	0.0189
8	-0.0250
9	-0.0312

time

- Getting **timepoint** of each sample

0 ms — **sample 1**
0.12 ms — **sample 2**
0.24 ms — **sample 3**
0.36 ms — **sample 4**
0.48 ms — **sample 5**
... **...**

y

73113x1 double

	1
1	0
2	-0.0062
3	-0.0750
4	-0.0312
5	0.0062
6	0.0381
7	0.0189
8	-0.0250
9	-0.0312

time

- **Total duration** of sound?
  = (sampling interval) * (number of samples)

= 0.12207 ms * 73113 samples

= around 8.9 sec (8,924 ms)

time

- timevector = [0 : **sampling interval** : **total duration**]

= 1 / (sampling rate)

8192

= (sampling interval) * (number of samples)

= 1 / (sampling rate) * (number of samples)

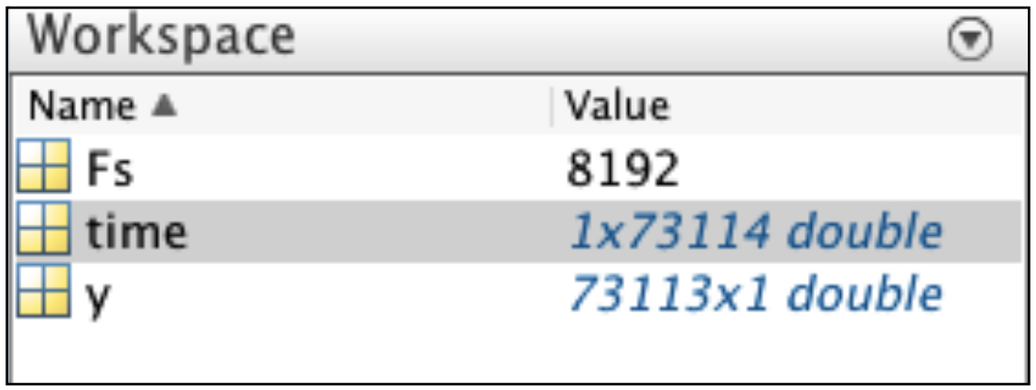
8192 73113

Command Window

```
>> time = [0 : (1/8192) : (1/8192)*73113];
```

time = [(1/Fs) : (1/Fs) : (1/Fs)*length(y)];
시간이 0초부터 존재할 수는 없음. 시간은 계속 가고는 있음!
사실상 시작인 1/Fs로 시작해야한다는 것!!

time



- Match dimension to plot

```
Command Window
>> plot(time(1:end-1),y)
```

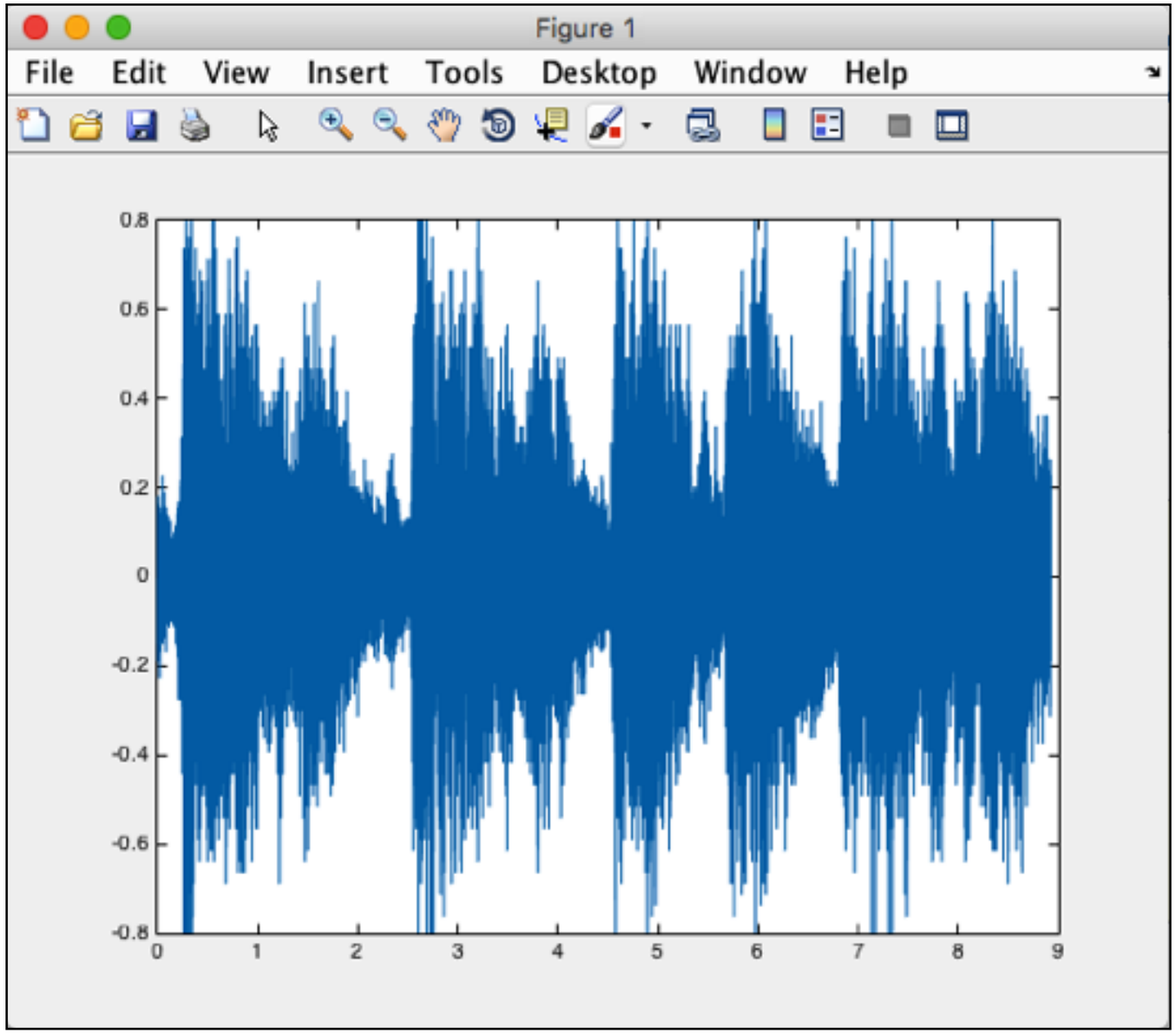
x-axis: **time** (sec)

y-axis: amplitude

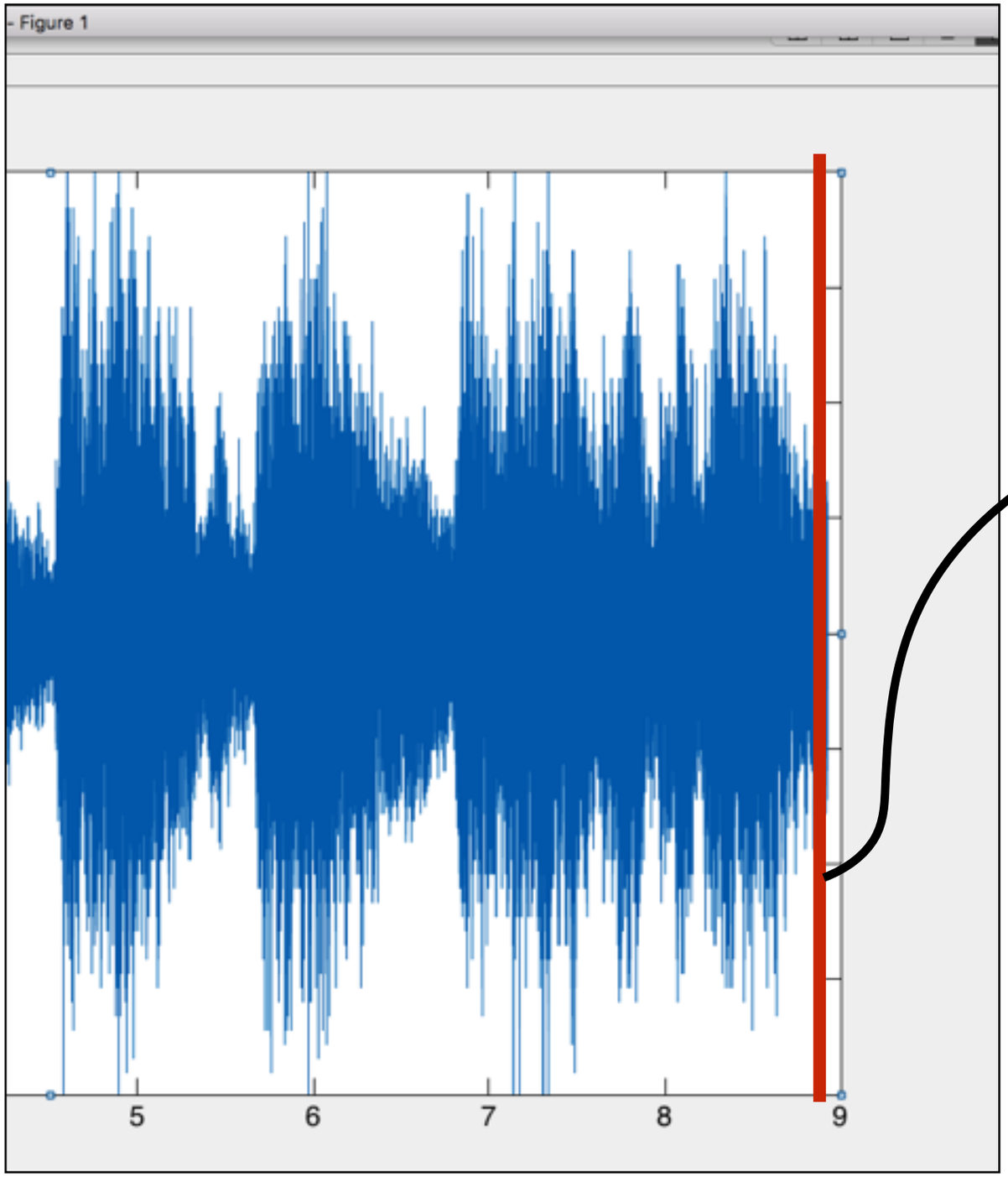
Plotting sounds

- **plot (time,y)**
  x-axis: **time** (sec)
  y-axis: amplitude

```
Command Window
>> plot(time(1:end-1),y)
```



Plotting sounds




```
Command Window
>> time(end-1)

ans =

    8.9248
```

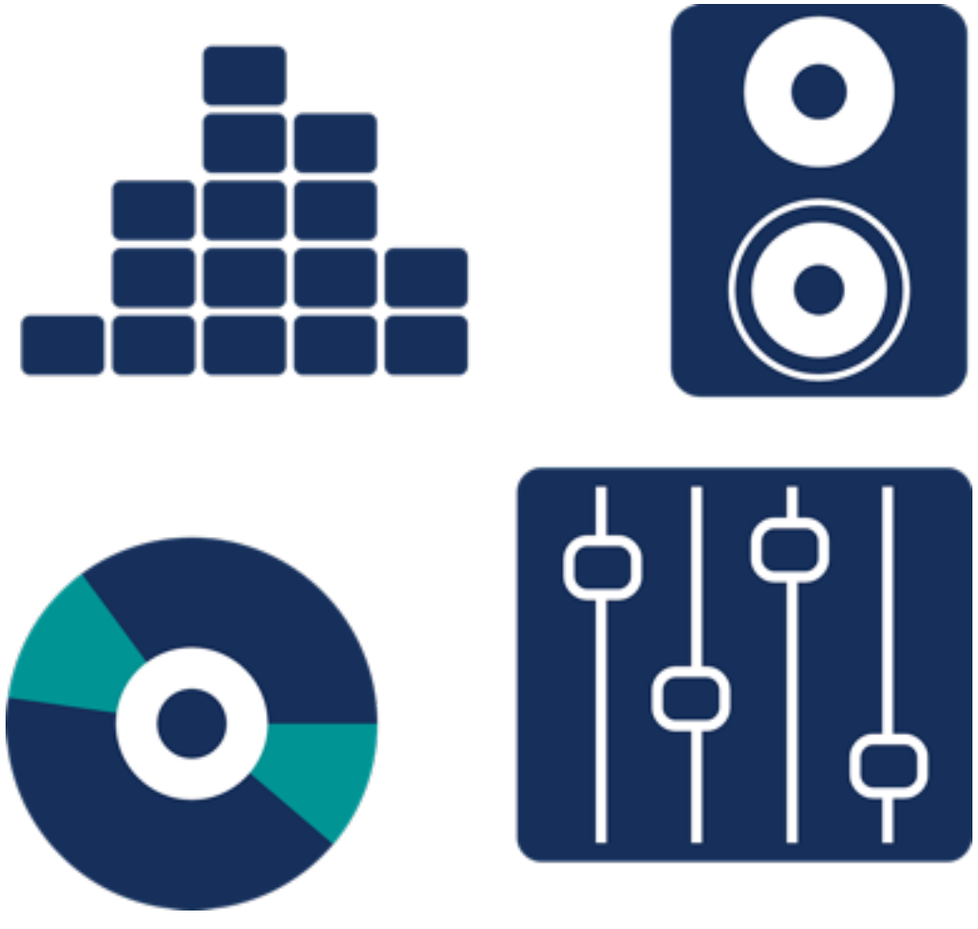
8.92 sec (end index)
= timepoint of
the **final sample**
= total **duration**

Sound Processing

(1) How to **import** and **play** sound?

(2) Which **processing** can be done with sound?

(3) How to **save** processed sound?



Import and Play

- **audioread**

  WAV, MP3, FLAC, OGG, MP4, etc…

Command Window

```
>> [y Fs] = audioread('a1.wav');
```

- **sound**: play sound
- **clear**: stop sound

Command Window

```
>> sound(y, Fs)
>> clear sound
```

cf. audioplayer

```
%% [tip] Audioplayer
player = audioplayer(y, Fs)
play(player); % start playing
pause(player); % pause
resume(player); % resume
stop(player); % stop
```

Import and Play

- **audiorecorder**
  - records audio using microphone connected to the computer

cf.
recordblocking (rec, recording duration)
play (recording)

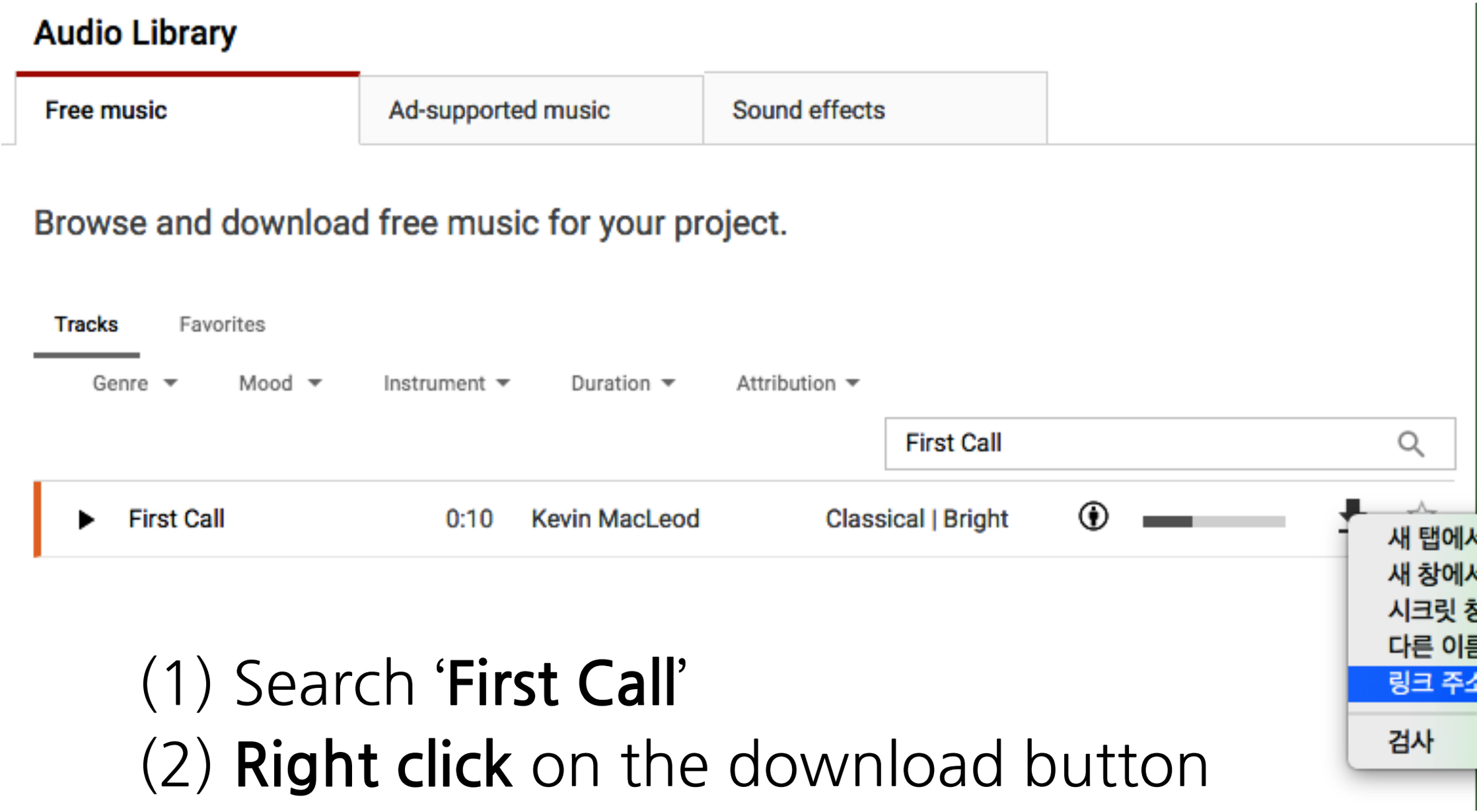
Command Window

```
>> rec = audiorecorder

rec =

  audiorecorder with properties:

         SampleRate: 8000
      BitsPerSample: 8
    NumberOfChannels: 1
           DeviceID: -1
      CurrentSample: 1
       TotalSamples: 0
            Running: 'off'
           StartFcn: []
            StopFcn: []
           TimerFcn: []
         TimerPeriod: 0.0500
                Tag: ''
           UserData: []
               Type: 'audiorecorder'

>> recordblocking(rec,3);
>> play(rec);
```

Import and Play

- **webread**
  - collects audio file from internet
  - Let's download an audiofile from **Youtube audio library**
    (https://www.youtube.com/audiolibrary/music)

Import and Play



(1) Search '**First Call**'
(2) **Right click** on the download button
(3) **Copy Link**

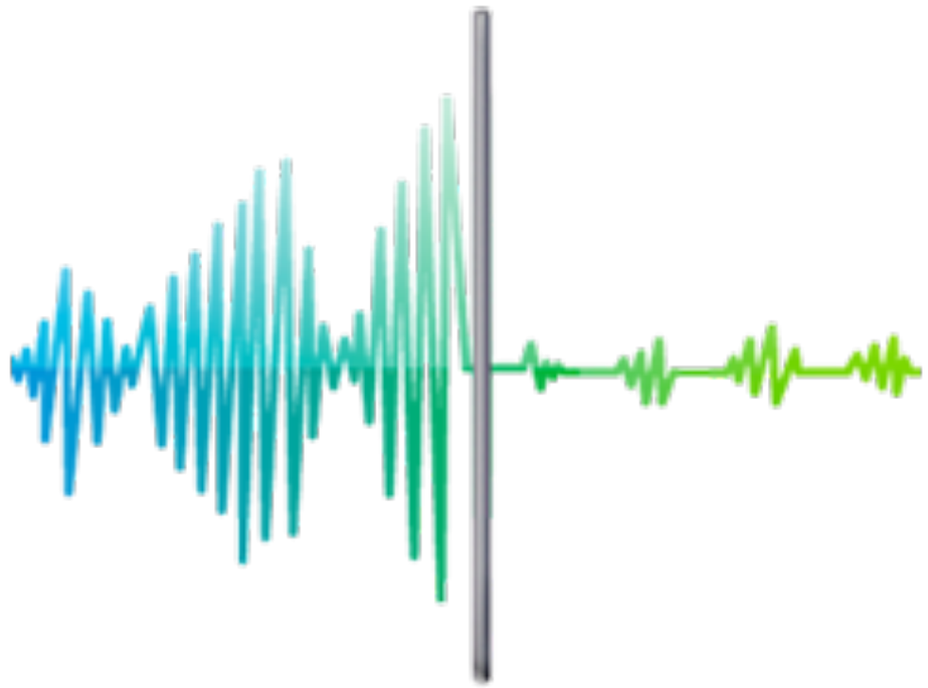
Import and Play

- webread

```
Command Window
>> url = 'https://www.youtube.com/audiolibrary_download?vid=1d91340e730d593b';
>> [y Fs] = webread(url);
>> sound(y, Fs)
```

Sound Processing

- What to do with the imported sound?
  - Cut & Combine
  - Volume control
  - Speed control
  - Inverse play
  - Overlap



Load Examples

- **Load** example files

```
%% Load audiofiles
clc;clear all;
[Rabbit, sr] = audioread('a1.wav');
[Carrot, sr] = audioread('a2.wav');
```

- **Play** (concatenate sounds)

```
%% Play
sound(Rabbit, sr)
sound(Carrot, sr)
```

Cut & Combine

- **Combine** (concatenate sounds)

```
%% Combine (Concatenate sounds)
all = [Rabbit ; Carrot];
sound(all,sr)
```

- Cut

```
%% Cut sounds
sound(Carrot, sr) % '당근을'
sound(Carrot(19000:end), sr) % '을'

cut = Carrot(1:19000) % '당근'
sound(cut, sr)
```

Volume control

- Volume **up**
  - **Multiply** a number larger than 1 to signal

```
%% (1) Volume up
sound(Carrot*5,sr)
```

- Volume **down**
  - **Divide** a signal by a number larger than 1

```
%% (2) Volume down
sound(Carrot/5,sr)
```

Volume control

- **Mute** specific intervals
  - Assign **zeros** to the signal

```
%% Mute out
Carrot(1:10000) = zeros(1,10000); % mute '당'
sound(Carrot, sr)
```

Volume control

- Fade-in
  - Multiply an **increasing** vector to signal

```
%% Fade-in
A = linspace(0.01,5,length(Carrot));
Carrot_In = Carrot;
for i = 1:length(A)
    Carrot_In(i) = Carrot(i)*A(i);
end
sound(Carrot_In, sr)
plot(Carrot_In)
```

linspace는 맨 처음 지정부터 그 다음 숫자의 등차간격까지의 Matrix를 만들어 준다. 위에 꺼는 0.01부터 5 간격으로 생김 length(Carrot)까지!

Volume control

- Fade-out
  - Multiply a **decreasing** vector to signal

```
%% Fade-out
B = linspace(5,0.1,length(Carrot));
Carrot_Out = Carrot;
for i = B(1):length(B)
    Carrot_Out(i) = Carrot(i)*B(i);
end
sound(Carrot_Out, sr)
plot(Carrot_Out)
```

Speed control

- Speed **up**
  - **Multiply** a number larger than 1 to **sampling rate**

```
%% Speed up (x2)
srFast = sr*2
sound(Carrot, srFast)
```

- Speed **down**
  - **Divide** a sampling rate by a number larger than 1

```
%% Speed down (/2)
srSlow = sr/2
sound(Carrot, srSlow)
```

Inverse play

- **Inverse** play
  - Flip a signal upside-down and play

```
%% Using flipud (up-down flip)
Carrot_inv = flipud(Carrot);
sound(Carrot_inv, sr)
```

```
%% Another easy way
sound(Carrot(end:-1:1),sr)
```

fliplr = flip left right (Row Vector)
flipud = flip upside down (Column Vector)

Overlap

- **Overlap** signals
  - **Add** signals

```
%% Overlap sounds
overlap = Carrot + Rabbit;
sound(overlap, sr)
```

Saving Audio

- audiowrite

```
%% Save audiofile
load handel
audiowrite('hallelujah.wav', y, Fs)
```

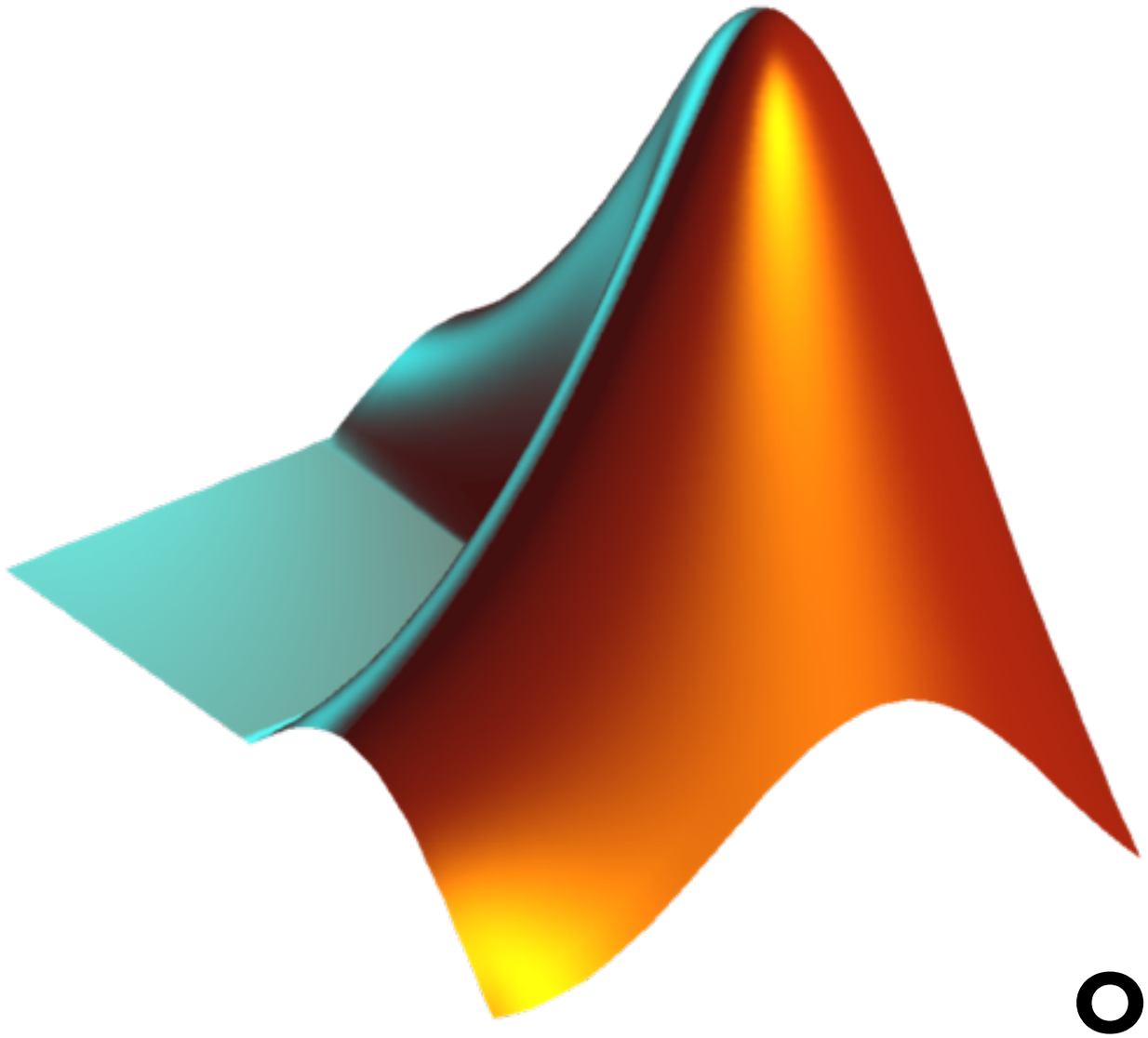
Summary

(1) Data = **vectors**

(2) **Sound data** is imported as **vectors** in Matlab

(3) Once sounds are imported, manipulating them is **easy**

- Cut & Combine
- Volume control
- Speed control
- Inverse play
- Overlap



알기쉬운 MATLAB

DAY 2 **Exercise**
Vector Management
2015/1/5

고려대학교
KOREA UNIVERSITY

Exercises: Vector

1.1 Assign an empty matrix to **A**

Hint: Use square brackets []

1.2 Assign a 1 by 3 row vector of zeros to **B**

Hint: Use **zeros**

1.3 Create a vector **C** by adding 3 to every element in B

1.4 Create a 3 by 1 column vector **D** by transposing C

Hint: Use **transpose** (')

1.5 Create a vector **E** by multiplying 0, 1, 2 respectively to the elements in vector D

Hint: Use element-wise multiplication (.*)

Exercises: Vector

1.6 Generate two random positive integers smaller than 100.

Assign the bigger integer to **k** and the smaller to **p**.

Hint: Use **randi**

```
A = rand(100,1,2)
p = min(A)
k = max(A)
```

1.7 Based on k and p (in 1.6),

(i) Assign **ones(k,1)** to variable **C**

(ii) **Delete** entire rows from **p** to **end** in C

```
C = ones(k,1)
C(p:end) = []
```

1.8 Given D = [-10:10], count the **number of elements** in D

Hint: Use **length** or **size**

1.9 Derive **mean** and **variance** of vector D

Hint: Use mean, std (or var)

randperm(20,20)

Exercises: Vector

1.10 Generate a 1 by 20 row vector **A** of random numbers from 1 to 20 Hint: Use randperm

1.11 Calculate **squared sum** of the vector A from 1.10

e.g. A = [5 17 8 ⋯]; newA = 5^2 + 17^2 + 8^2 ⋯

1.12 Generate following vectors using **colon(:)** or **'for' loop**

Hint: Think about any solutions other than typing out all elements. There can be **many different** answers to them.

i. [0 2 4 6 8]

ii. [15;14;12;9;5;0]

iii. [5 -4 3 -2 1]

```
a = 15;
for i= 1:5
    a(end+1)=a(end) - i
end
```

```
newA=A.*A
sum(newA)
A.^2도 가능함
```

1.13 Given B = [0:2:10], derive [1;3;5;9] by manipulating B

Hint: Use transpose ('), delete, index using 'end'

Exercises: Plotting

- Plotting **multiple** lines in a **single figure**

rand(1,20)

2.1 Generate a random vector **A** (size: 1 by 20)

2.2 Plot **A** using [red / solid line / circle marker]

2.3 Plot **2*A** using [black / dotted line / asterisk marker]

2.4 Plot **3*A** using [blue / dashed line / plus marker]

with a thicker 'LineWidth' (=5)

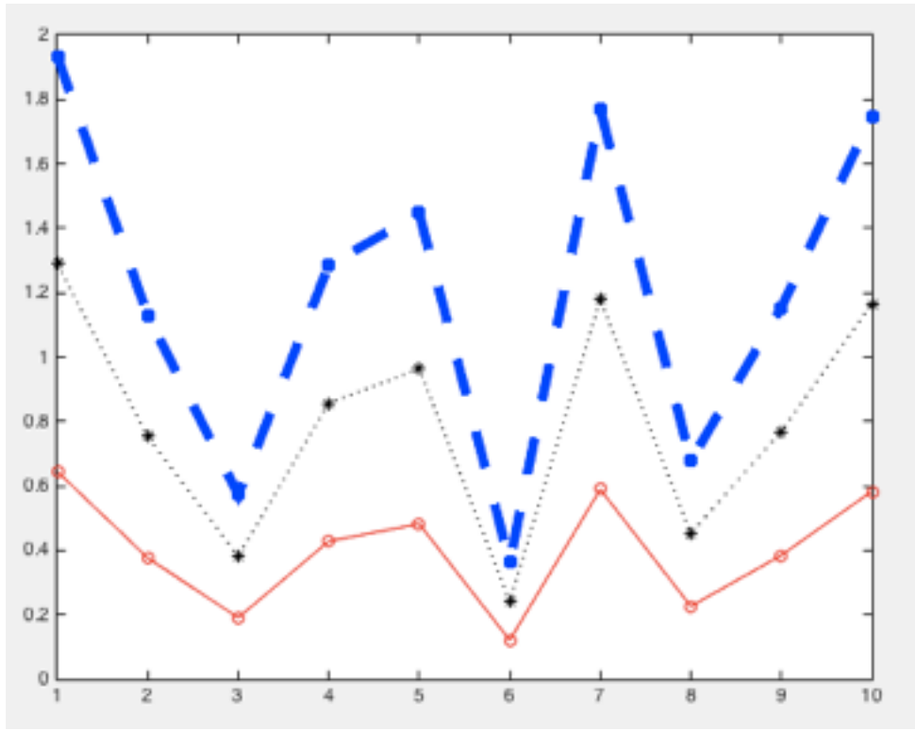
Hint: hold on, doc plot

hold on
plot(2*A)

hold on 한번하면
hold off까지 유지

cf. The expected figure:

(Your result figure may differ in shape, due to randomness in generating vector A)



Exercises: Sound

- Load **chirp** (built-in sound data in Matlab)

3.1 Sampling rate?

3.2 Total number of samples taken?

3.3 **Plot** signal by **sample**

3.4 Total duration?

3.5 **Plot** signal by **time** (seconds)

3.6 Assign **y** as 'sigBird' and **Fs** as 'srBird'

3.7 **Save** sigBird and srBird as '**bird.wav**'

Exercises: Sound

- Load **train** (built-in sound data in Matlab)

3.8 Assign **y** as 'sigTrain' and **Fs** as 'srTrain'

3.9 **Concatenate** sigTrain after sigBird (vertically),
and assign it to '**sigBirdtrain**' use semicolon to concatenate vertically

3.10 Assign any one of srBird or srTrain to '**srBirdtrain**'
(as they are the same value)

3.11 **Save** sigBirdtrain and srBirdtrain as '**birdtrain.wav**'

3.12 Compare length of sigTrain and sigBird.
Match length of sigTrain and sigBird

3.13 **Overlap** sigTrain and sigBird

3.14 **Inverse play** the overlapped signal (in 3.13)

Task1: MATLAB the Fortune-teller

- Believe it or not, Matlab knows

  (1) who is going to win the Lotto this week,

  (2) which set of numbers are going to be called,

  and (3) when to buy the lottery.

- Follow the instructions to find the information above.

students.Names(5)

students.Names(idx)

[Guess Who]

students.Names 하면 이름들나열
students.Gender 하면 성별 나열

Task1.1 Load **students.mat** (downloadable from email)

Task1.2 Create a **cell vector A** of randomly selected **ten** names from all students Hint: Generate a random vector '**idx**' of ten indices using **randi** first. Then, **extract** names **using the 'idx'** from students' names

idx = randperm(30, 10)

Task1: MATLAB the Fortune-teller

[Guess Who]

Task1.3 Matlab predicts the winner to be a **female**.

Create a **cell vector B** which extracts all females from A (the list of the randomly selected ten names)

Hint: Extract gender information using the '**idx**' (in 1.2).

Use **for** loop, **if**, and **isequal** to find females.

Task1.4 Finally, assign the person in the **end** of the vector **B** to a variable named '**winner**'

(), cell로는 == 판단이어려움
{} calibracket으로내용물을 보는것

```
B = [];
for idx = 1:length(idx)
    if students.Gender{idx} == 'F'
        B = [B ; students.Names(idx)];
    end
end
```

Task1: MATLAB the Fortune-teller

[Guess Which set of numbers]

Task1.5 Randomly derive a **vector** named '**num**' of six integers ranging from 1 to 50 for the lottery

Hint: Use **randperm** (cf. **randi** allows repetition)

[Guess When]

Task1.6 Matlab predicts the day of buying a winning lottery. Randomly select a day from a **cell vector** named '**days**'.

days = {'Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'}

Task2: Wordplay

Task2.1 Load '**I.wav**', '**love.wav**', '**MATLAB.wav**' (downloadable from email)

With the given records, create following sentences in red:

```
[a sr] = audioread('I.wav')
original = [a ; b ; c]
sound(original, sr)
```

Task2.2 'I love Matlab' (assign it to a variable '**original**')

Hint: Concatenate sounds

Task2.3 Plot the signal of 'I love Matlab'

Hint: Plot the signal not the sampling rate (sr)

Task2.3 'I love Mat' (assign it to a variable '**Mat**')

Hint: Mute out or cut out a specific range of sounds

Consider **matlab(1:21000)** as the range of 'Mat' in 'matlab'

Task2: Wordplay

Task2.4 ‘I lo——ve Mat’ (assign it to a variable ‘**Mat2**’)

Hint: Lengthen the vowel in ‘love’ by concatenating several ‘o’s. Consider **love(20000:25000)** as the range of the vowel in ‘**love**’

Task2.5 ‘I love Lab’ (assign it to a variable ‘**lab**’)

Hint: Cut out a specific range of sounds

Task2.6 ‘I love Lab mat’ (assign it to a variable ‘**labmat**’)

Hint: Concatenate sounds

Task2: Wordplay

Task2.7 'I love Matlab' in an increasing volume (assign it to a variable '**VolUp**')

Hint: Multiply increasing sequence of numbers that are larger than 1 to the sound signal vector

Hint: Use **linspace**(startN, endN, length(element))

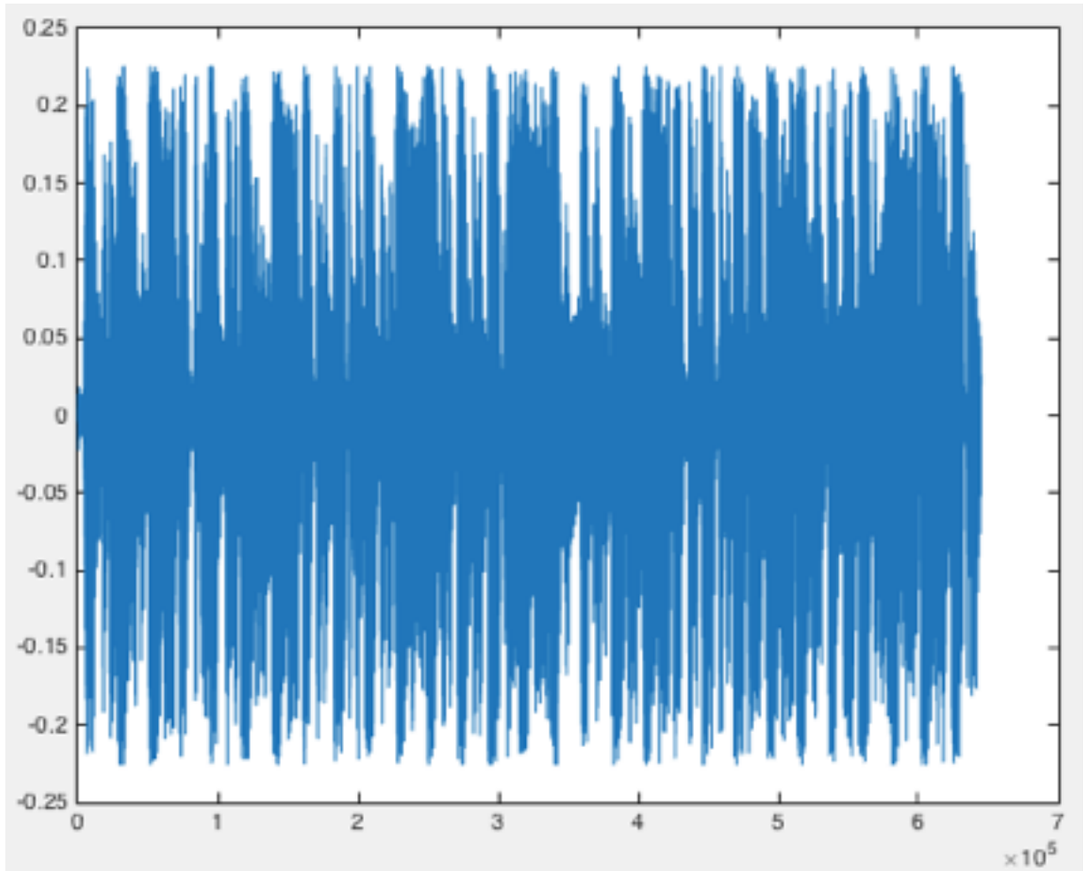
```
coeff = linspace(____,____,length(original))';
```

Assignments (Day2)

%% PSY **Dance Jockey**
%% Write your answers in a script file ('myAssignment.m')

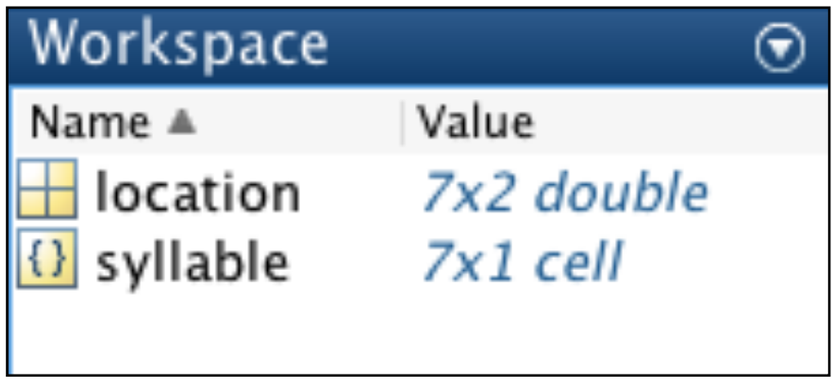
1.1 Load **psy.wav** in Matlab and plot signals

```
[y Fs] = audioread(          )
plot(     )
```



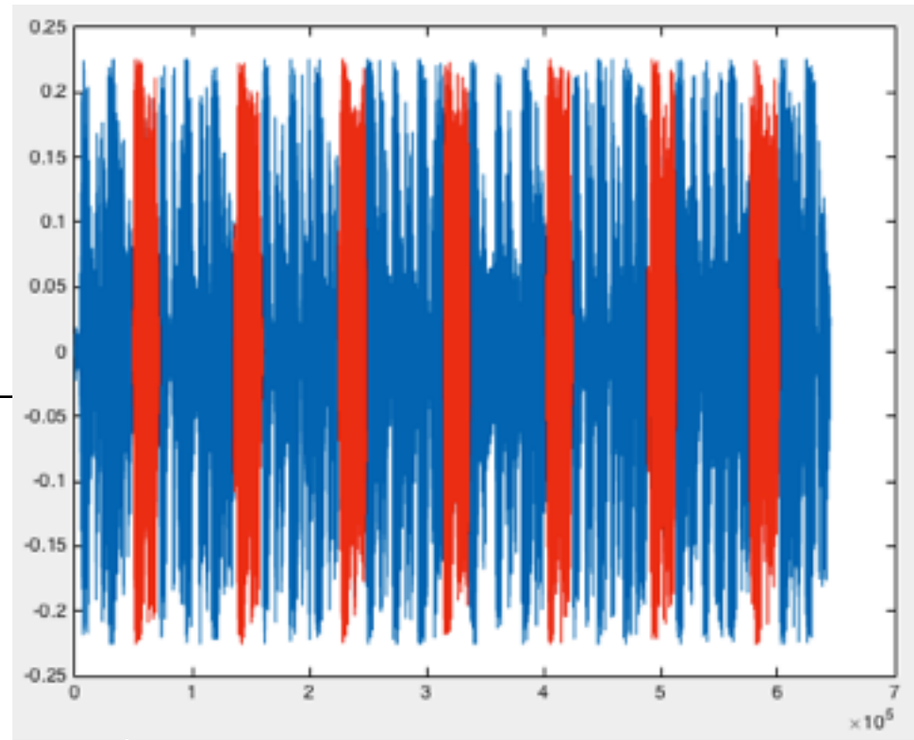
Assignments (Day2)

1.2 load **psy.mat** in Matlab and plot **syllable** using **location** variable overlapped on plot 1.1




```
[y Fs] = audioread(       ); % load sound
load(    )          % load psy.mat
plot(    )          % plot original sound
hold on             % hold on
for i = 1:length(location)
    % get x and y values
    xvalues = location(    ):location(    );
    yvalues = y(location(    ):location(    ));
    % plot
    plot(xvalues,yvalues,'r')
end
hold off            % hold off
```

가 나 다 라 마 바 사



Assignments (Day2)

1.3 Extract **'ga', 'na', 'da', 'ra', 'ma', 'ba', 'sa'**
Combine them all, and then play the sound!

(1) Load '**psy.wav**' and '**psy.mat**' in Matlab
(2) **Combine** 'ga', 'na', 'da', 'ra', 'ma', 'ba', 'sa'
(3) **Play** the sound

```
%% Combine parts
% 'ga' 'na' 'da' 'ra' 'ma' 'ba' 'sa'
allparts = [];              % make an empty variable
for                         % for loop
    part =                  % get part (look location)
    allparts = [allparts ; part]; % combine parts
end
plot(    )                  % plot allparts
soundsc(allparts, 44100) % play the sound!
```