MATLAB 기초

Computational Design Laboratory
Department of Automotive Engineering
Hanyang University, Seoul, Korea

CONTENTS

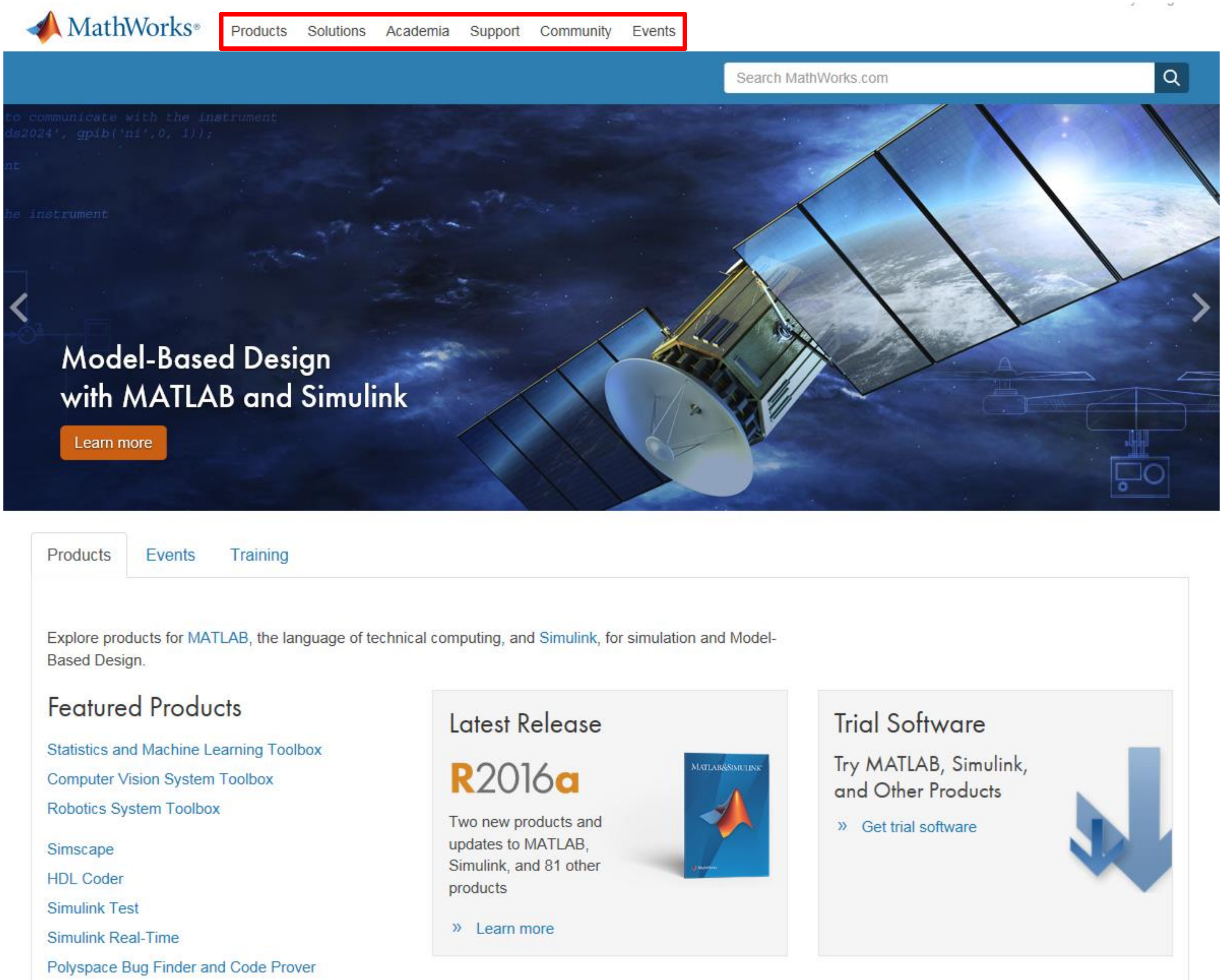
- Overview
- MATALB Environment
- Mathematical operations
- Built-in functions
- Graphics
- Case study
- Assignment

OVERVIEW

- The Language of Technical Computing
  - To analyze and design the systems and products transforming real world
  - Machine learning, signal processing, image processing, computer vision, communications, computational finance, control design, robotics, and much more
- Features
  - Matrix-based language
  - Vast library of prebuilt toolboxes
  - Integrated with other languages
  - Video : MATLAB Overview

HOMEPAGE

http://www.mathworks.com



- **The MATLAB Environment**

  ✓ **Interface**

  ✓ **Command Window / Workspace**

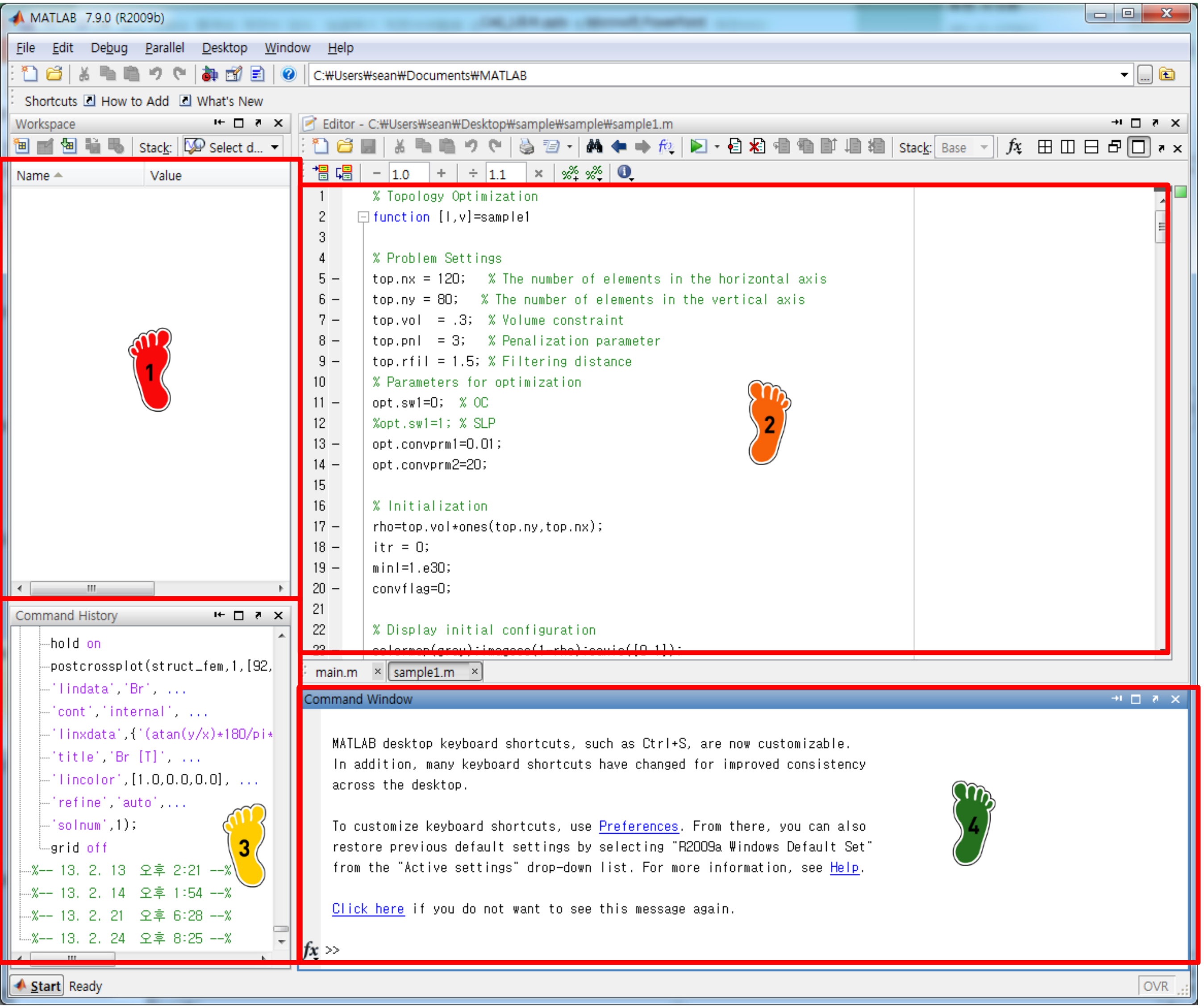
  ✓ **Scalars**

  ✓ **Arrays, Vectors and Matrices**

  ✓ **The Colon Operator**

  ✓ **Character Strings**

INTERFACE



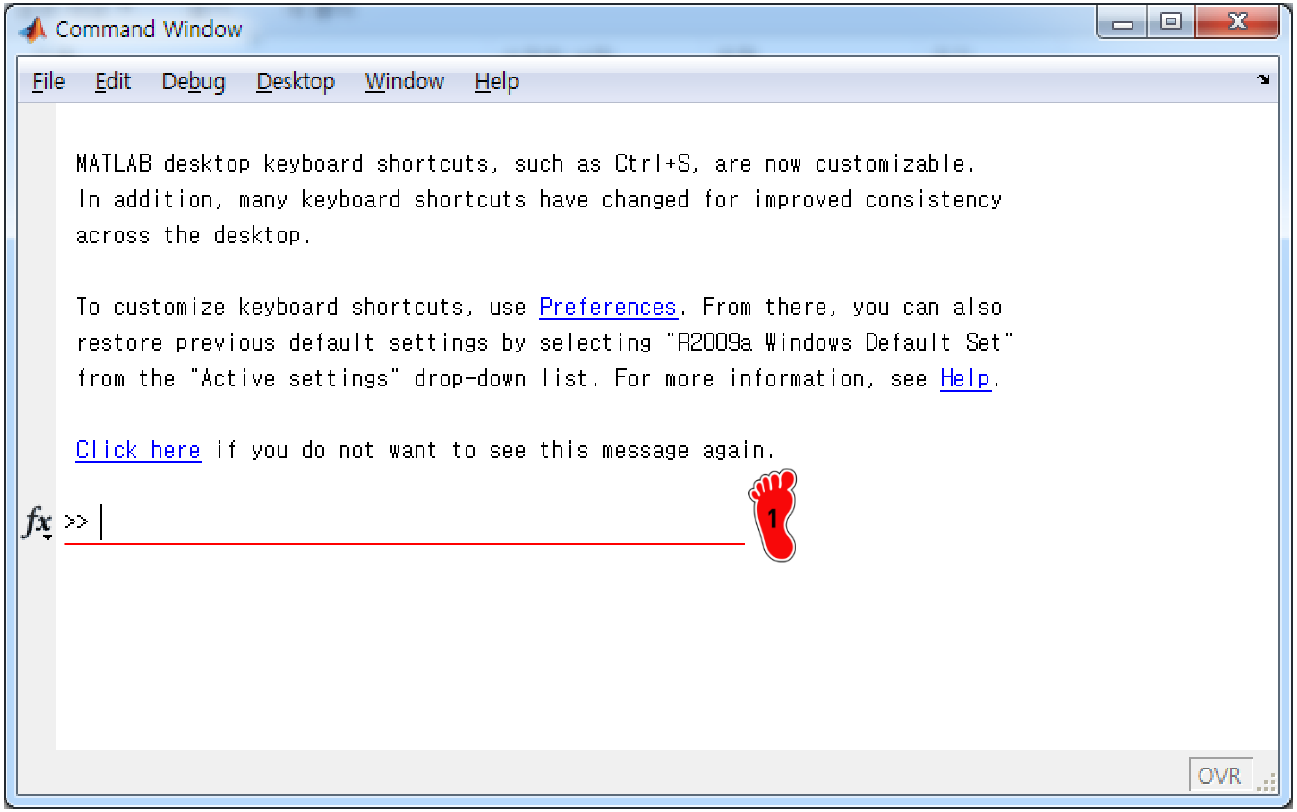
1 Workspace: 저장된 데이터를 보여주는 창

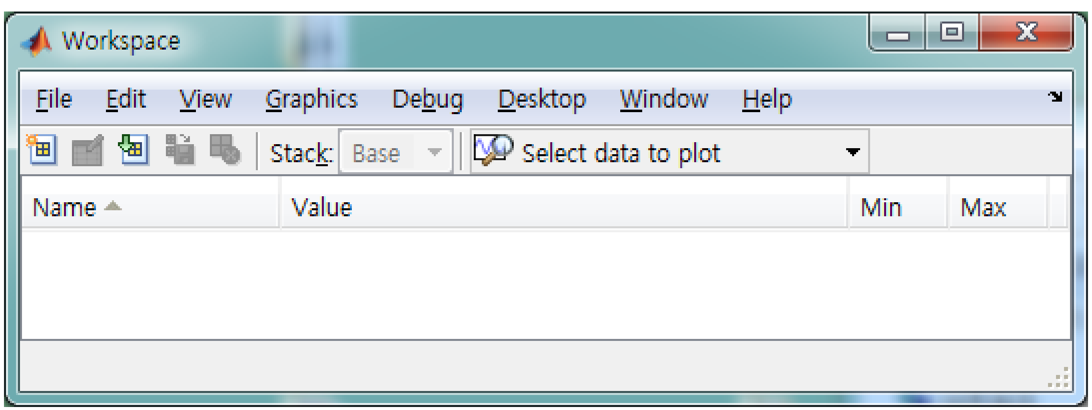
2 Editor: m-file 프로그래밍

3 Command History: 지난 명령어 이력

4 Command Window: 매틀랩 명령어를 입력하는 곳

COMMAND/WORKSPACE

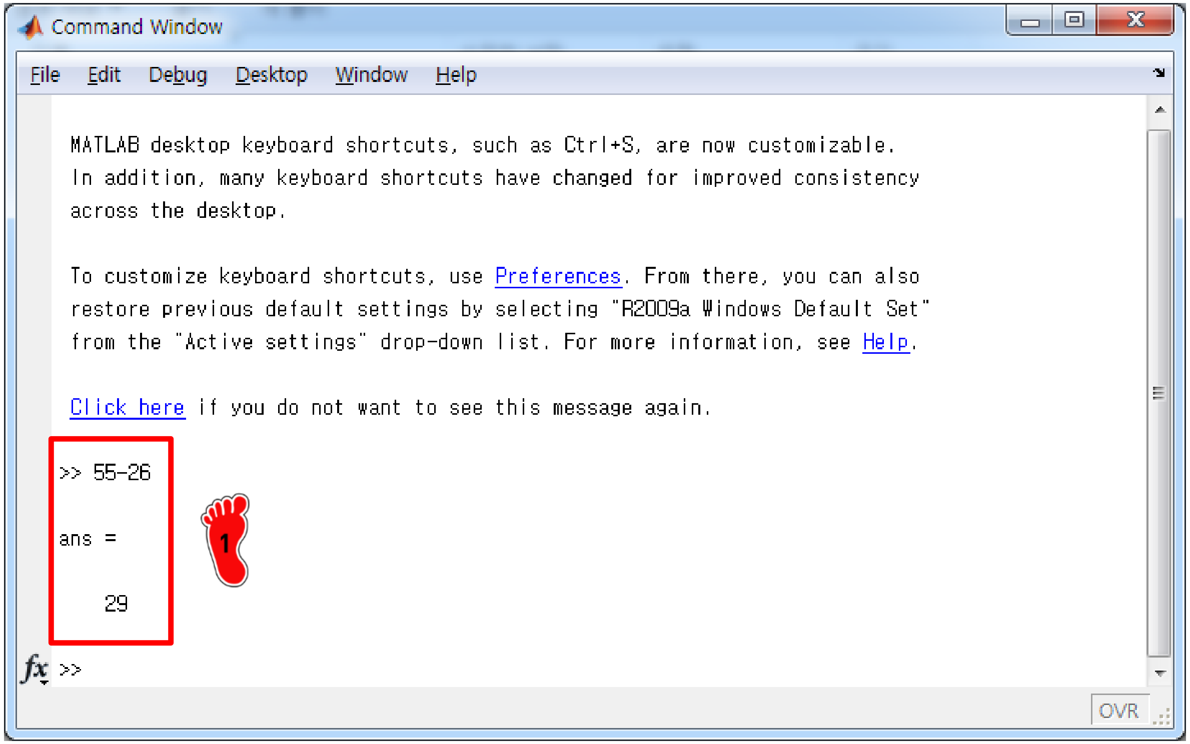


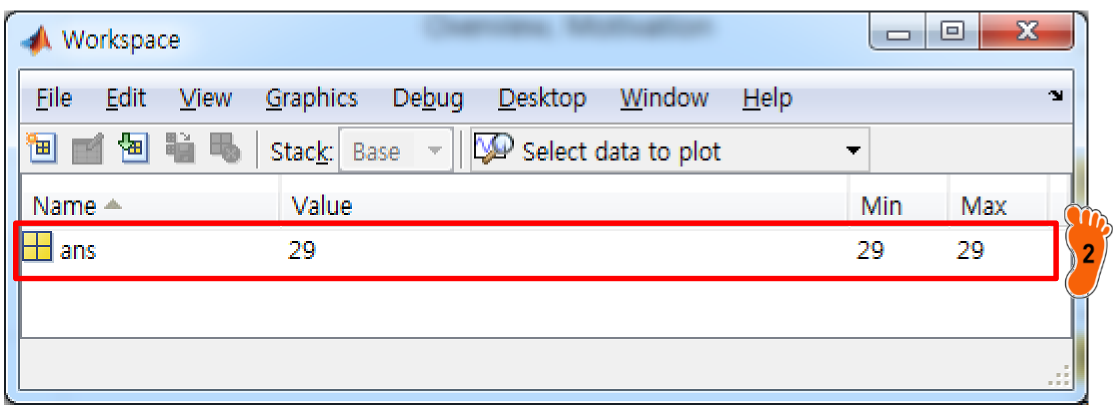


1 매틀랩 명령어를 입력하는 command line

> >

COMMAND/WORKSPACE



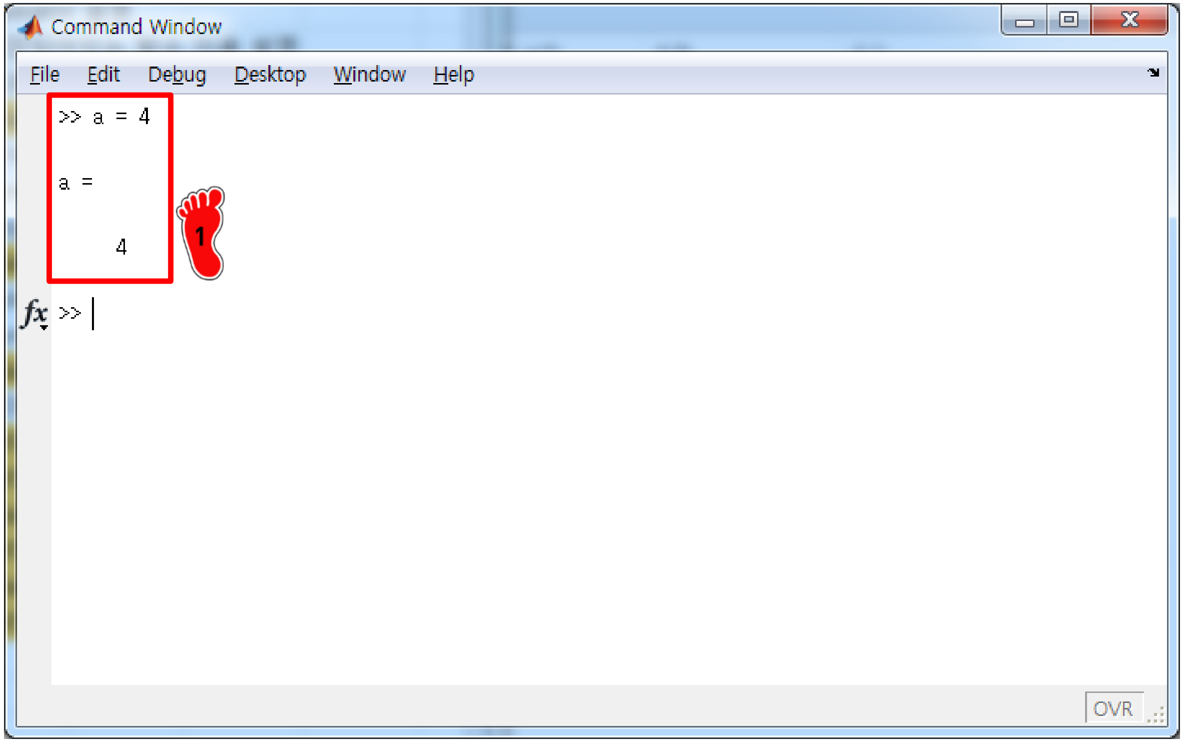


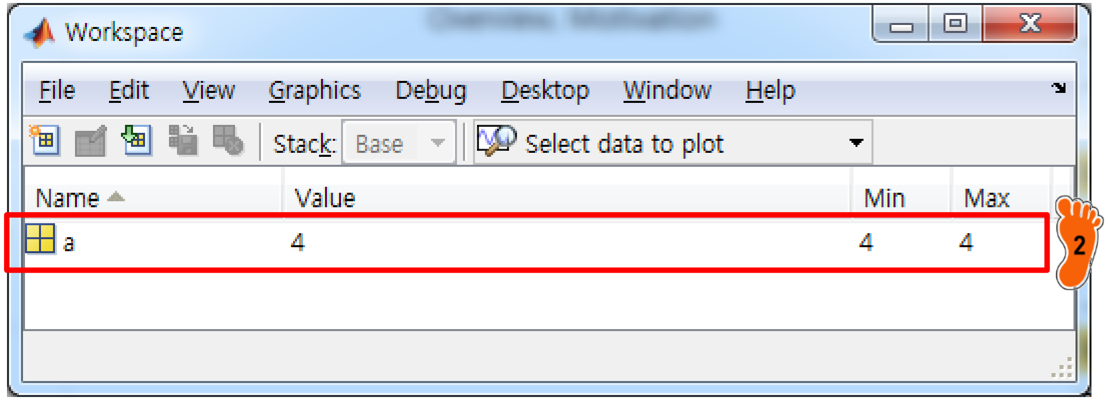
1 55 – 26 엔터를 입력하면 ans = 29 라는 결과를 확인

ans 는 answer 의 약자

2 workspace 에 ans 라는 변수가 저장됨

SCALARS

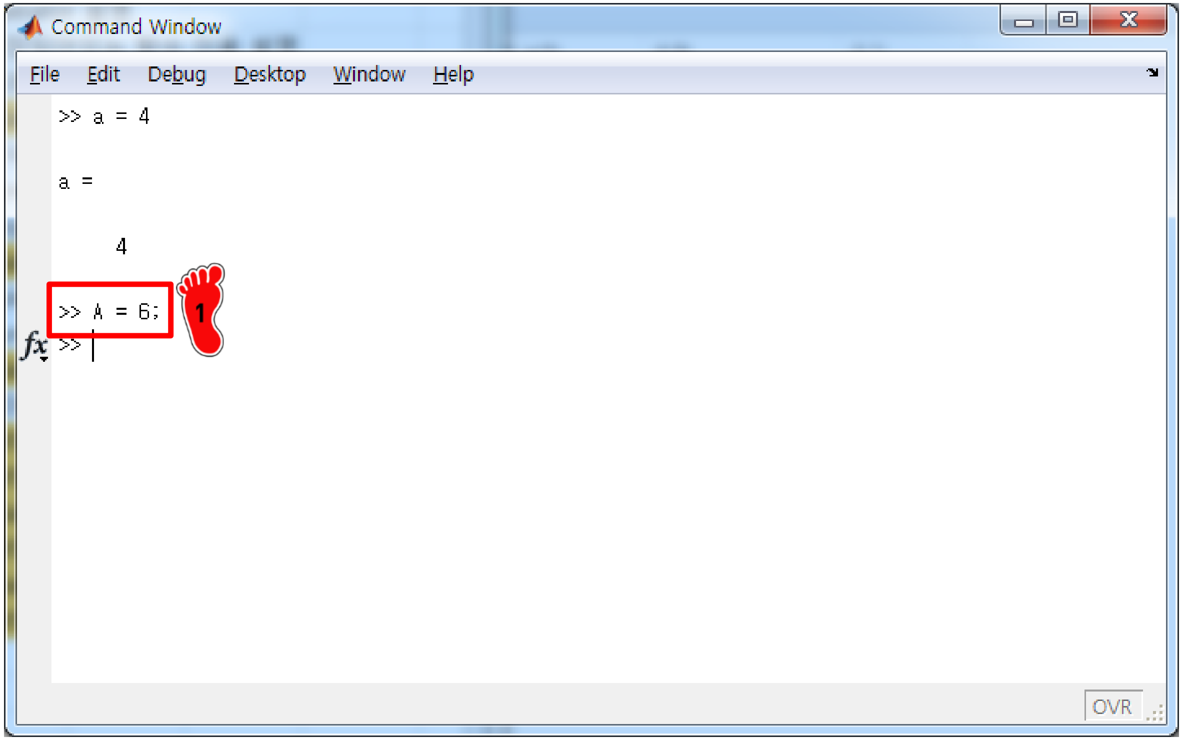


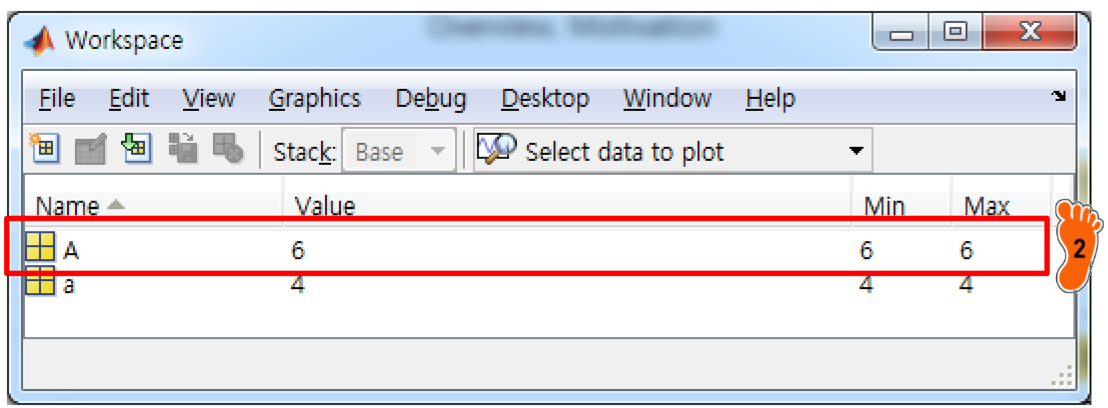


1 a = 4 Enter를 입력하면

2 workspace에 상수 a 가 저장됨

SCALARS

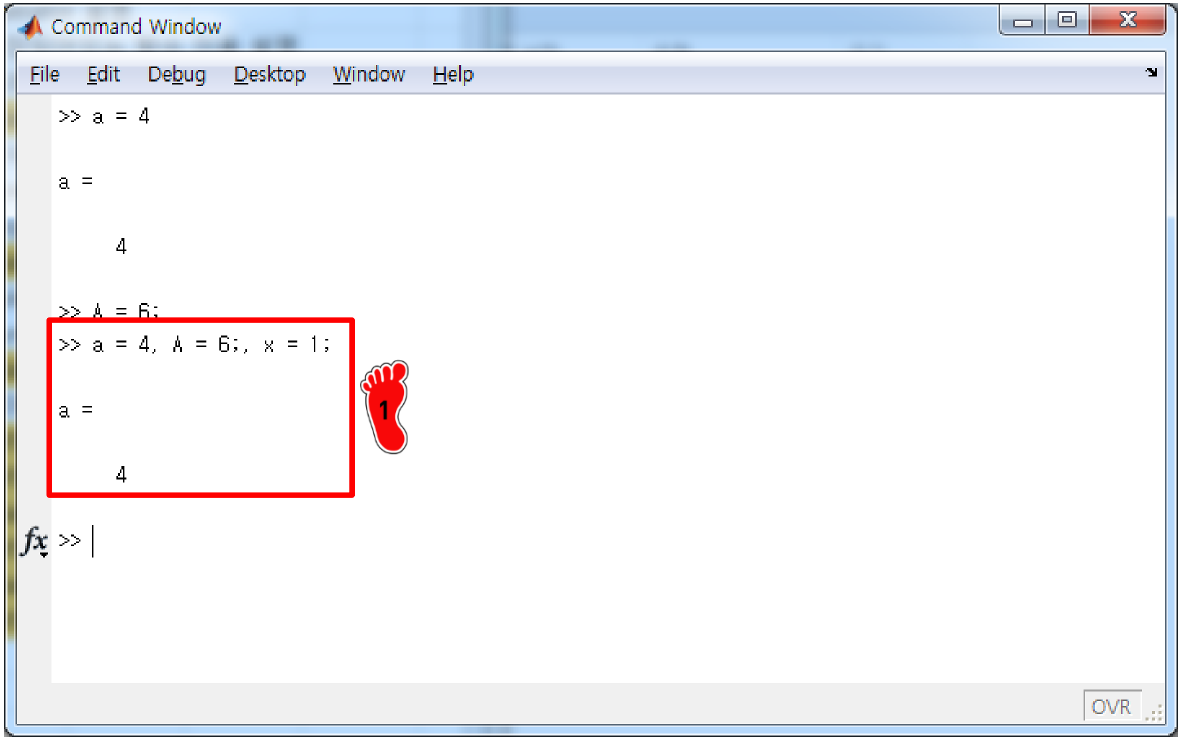


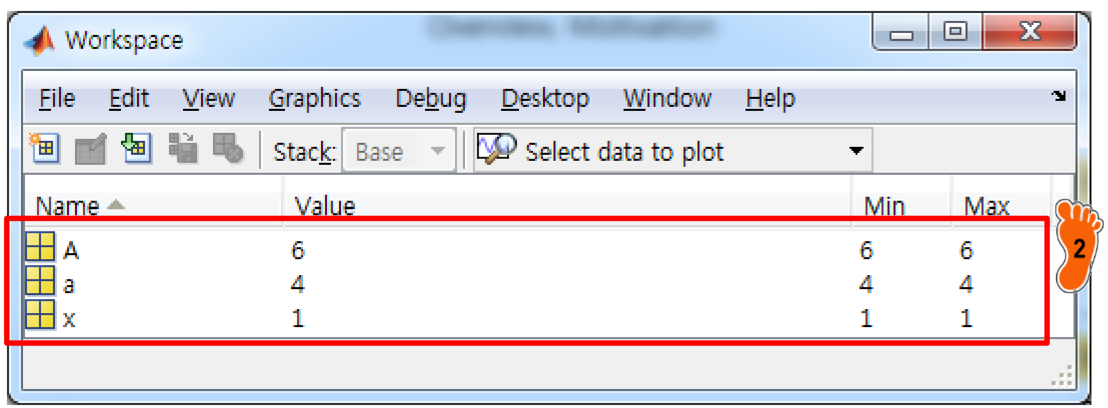


1 상수를 입력할 때 뒤에 세미콜론(;) 을 붙이면 결과값이 창에 뜨지 않음

2 하지만 workspace 에 상수 A 가 저장됨

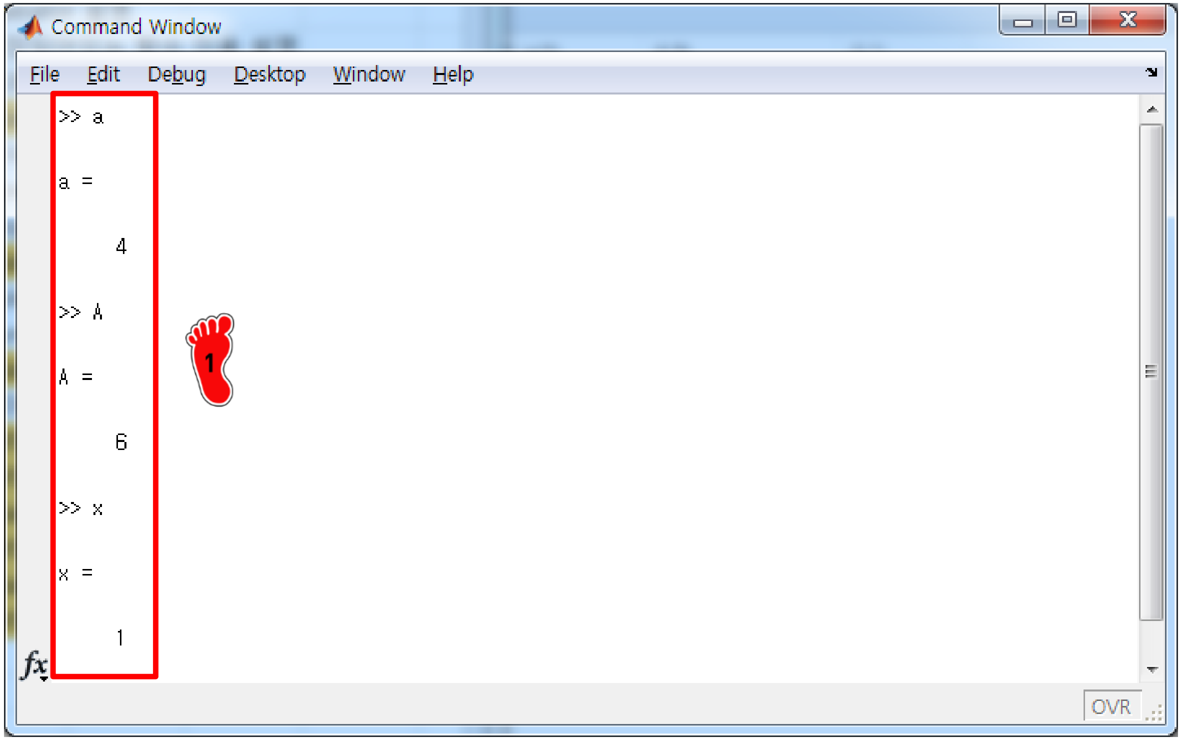
SCALARS

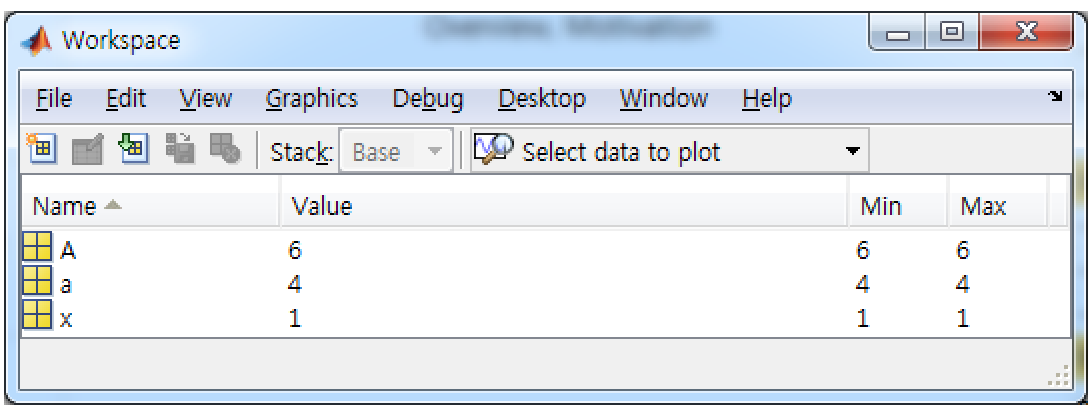




1. 쉼표(,) 를 이용하여 여러가지 상수를 한꺼번에 입력할 수 있음
2. workspace에 저장된 값 확인

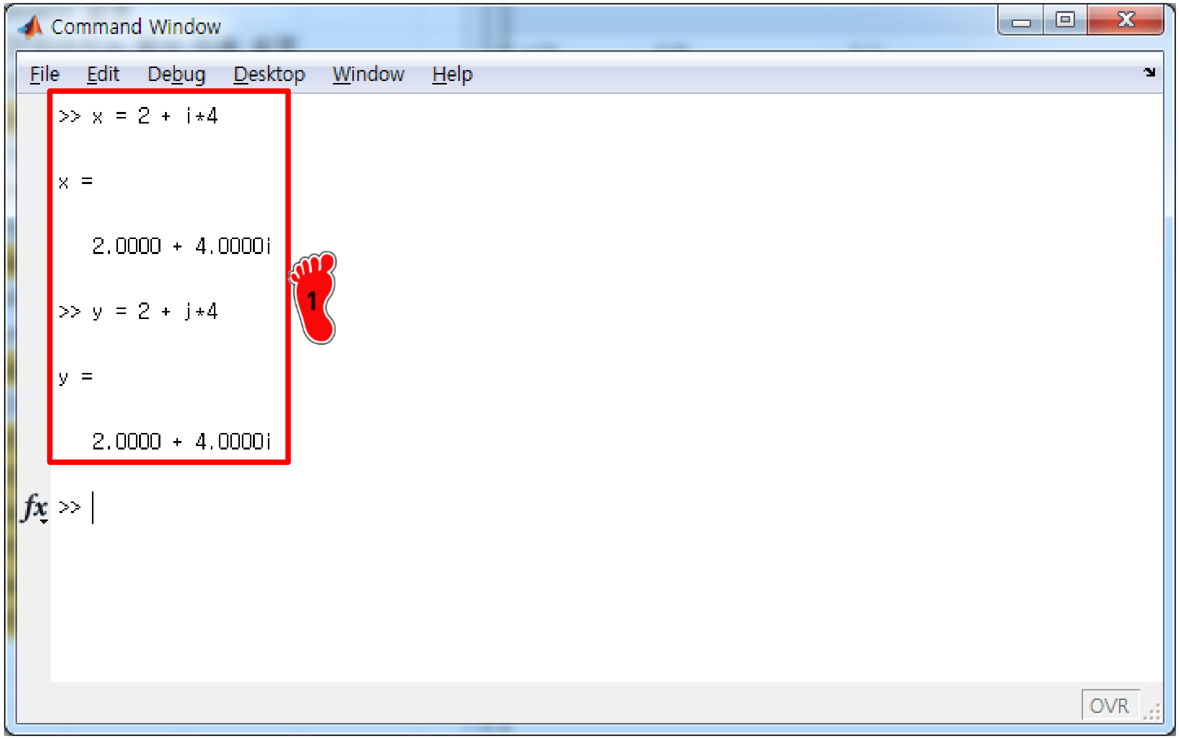
SCALARS

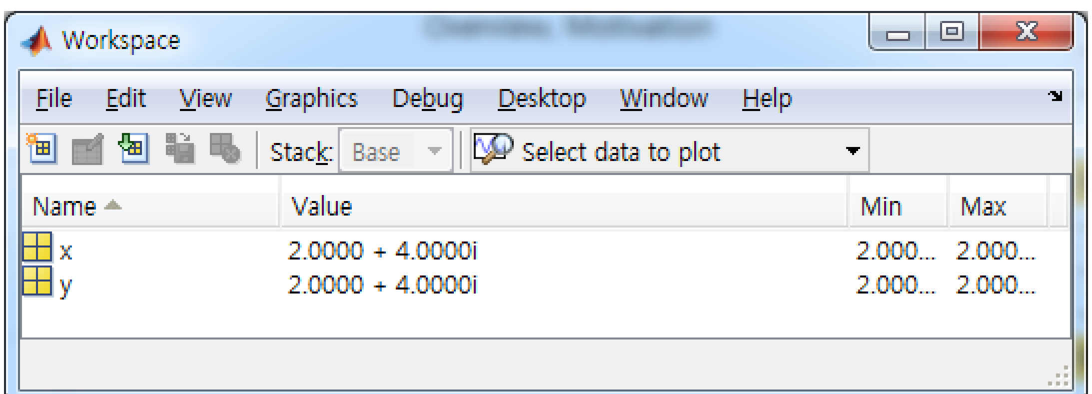




1 저장되어있는 상수의 값을 확인하고 싶을 경우 command line 에 상수 이름을 치면 확인 가능

COMPLEX VALUE

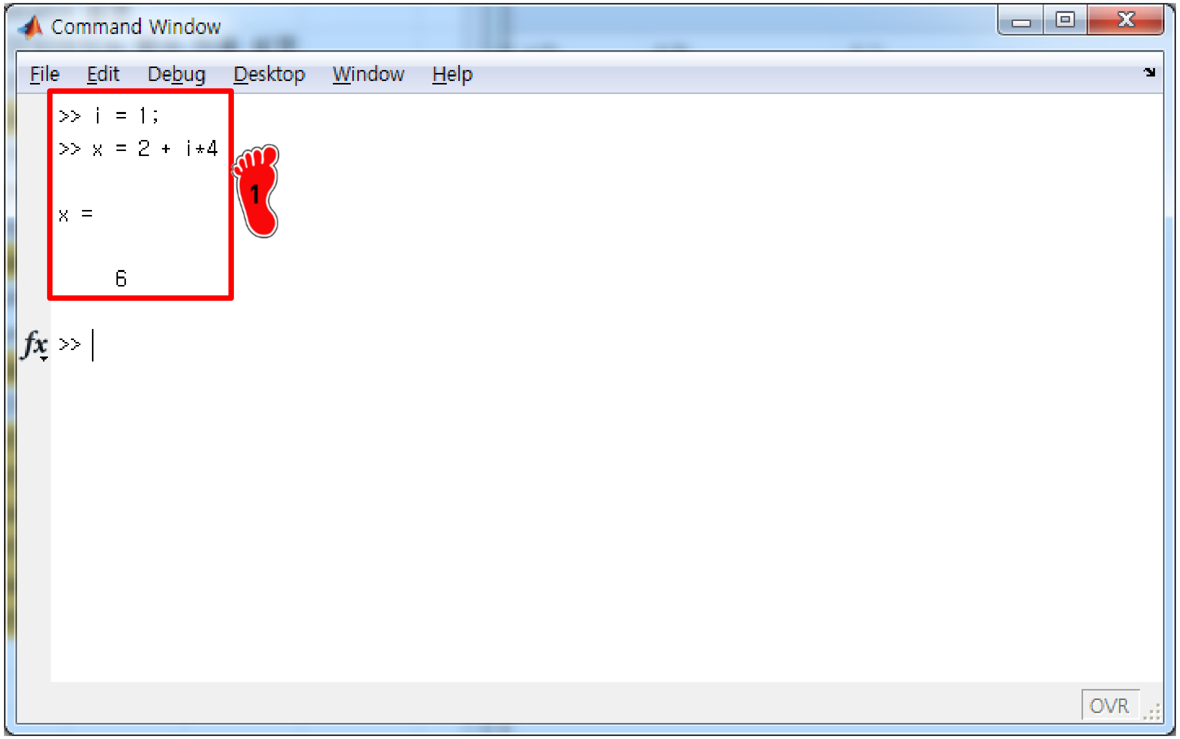


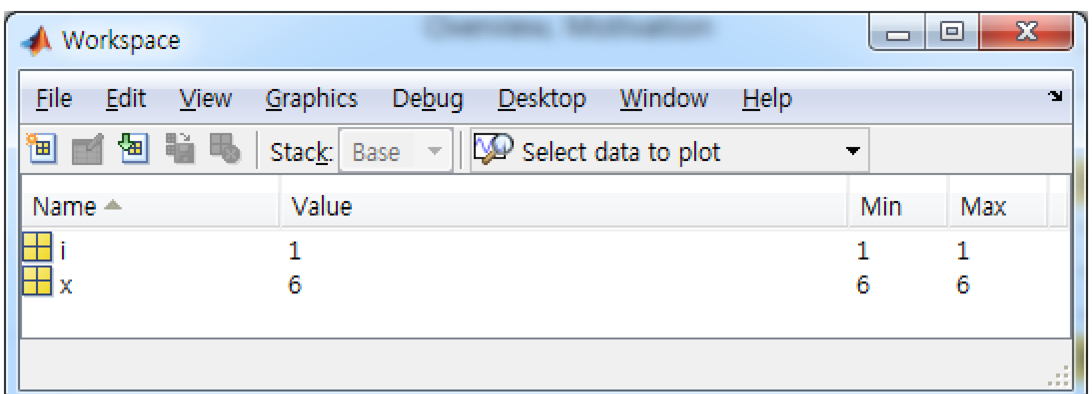


1 i 와 j 는 매틀랩에서 기본적으로 제공하는 허수를 의미함 ($\sqrt{-1}$)

workspace 에 i 와 j 가 상수로 선언이 되지 않을 경우 허수로 인식

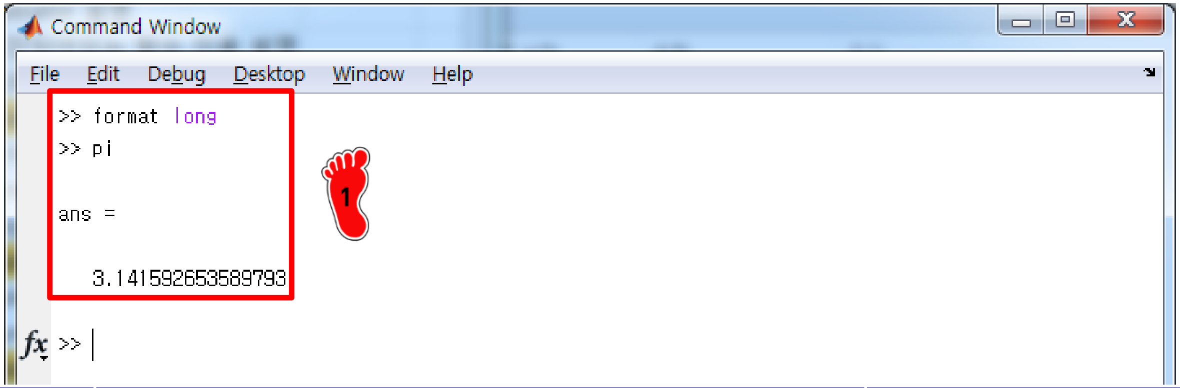
COMPLEX VALUE





1 i 가 선언이 될 경우 허수로 인식하지 않음

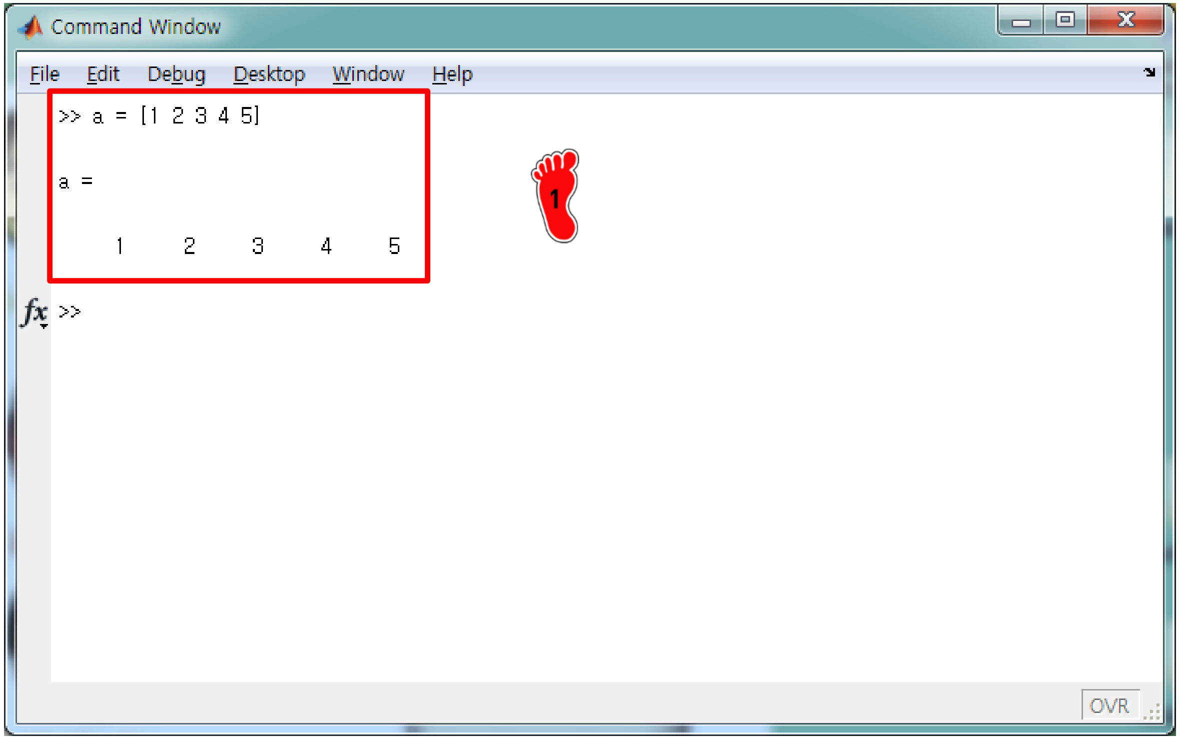
FORMAT

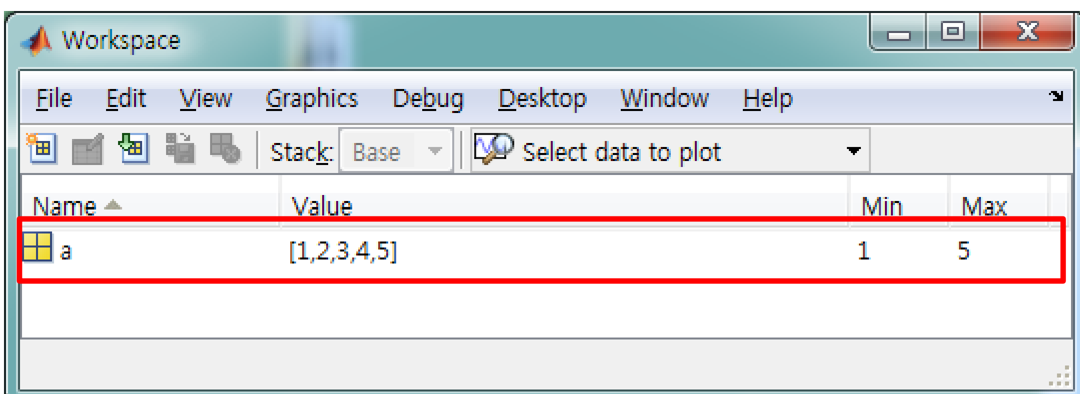


format 명령어를 이용하여 출력되는 자릿수와 형식을 결정할 수 있음

type	Result	Example
short	Scaled fixed-point format with 5 digit	3.1416
long	Scaled fixed-point with 15 digits for double and 7 digits for single	3.14159265358979
short e	Floating-point format with 5 digits	3.1416e+000
long e	Floating-point format with 15 digits for double and 7 digits for single	3.141592653589793e+000
short g	Best of fixed- or floating-point format with 5 digits	3.1416
long g	Best of fixed- or floating-point format with 15 digits for double and 7 digits for single	3.14159265358979
short eng	Engineering format with at least 5 digits and a power that is a multiple of 3	3.1416e+000
long eng	Enginerring format with exactly 16 significant digits and a power that is a multiple of 3	3.14159265358979e+000
bank	Fixed dollars and cents	3.14

ROW VECTOR

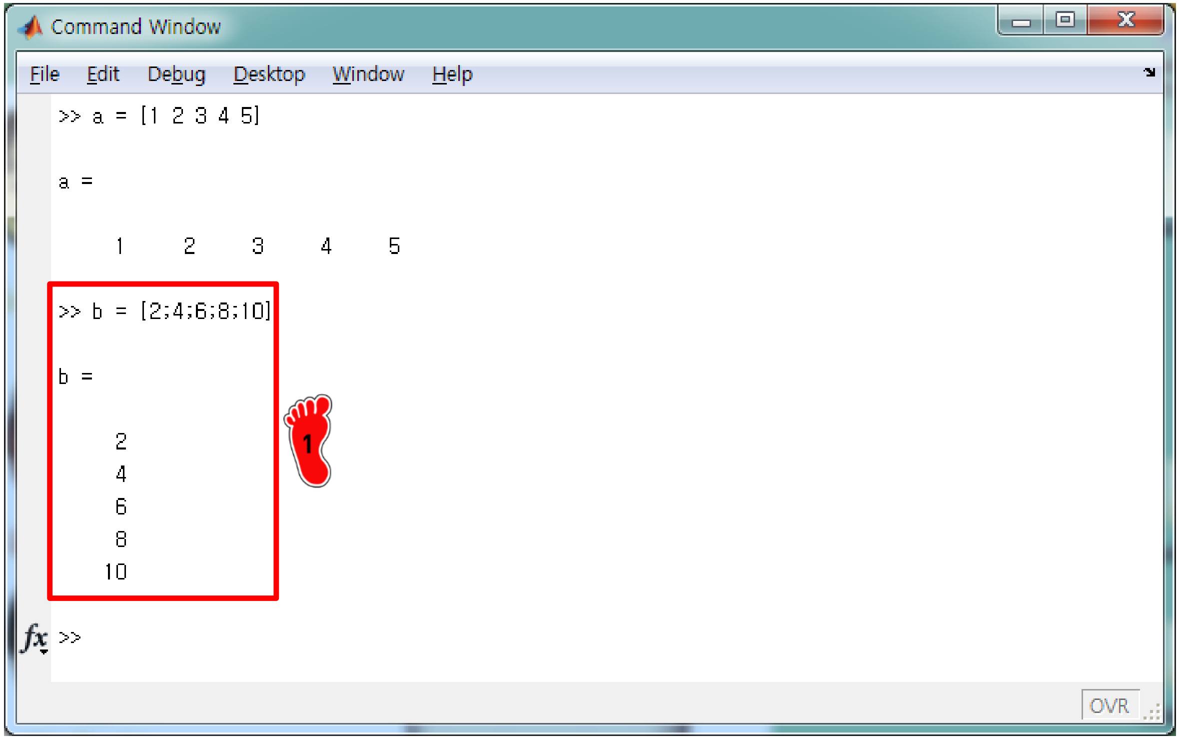


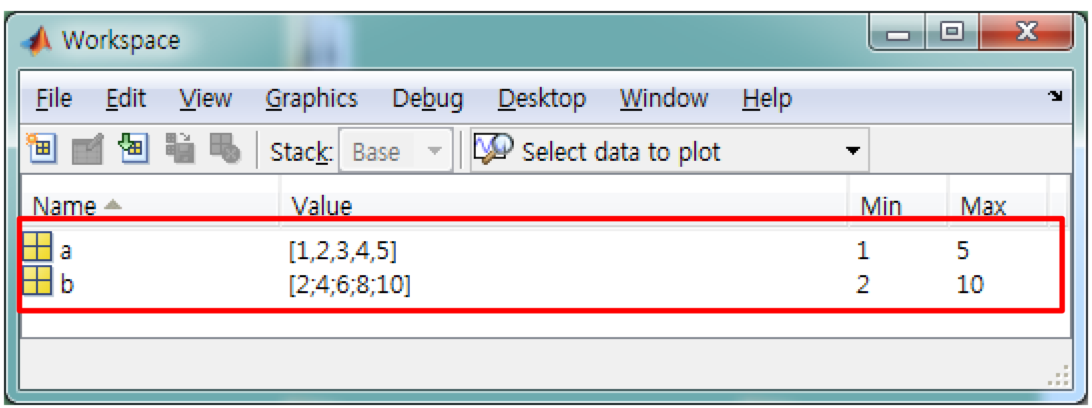


1. 괄호 기호 [ ] 를 이용하여 행 벡터를 저장할 수 있음

2. 저장된 데이터를 확인하면 상수와는 다르게 여러개의 데이터를 저장하고 있기 때문에 mim, max 값을 표기

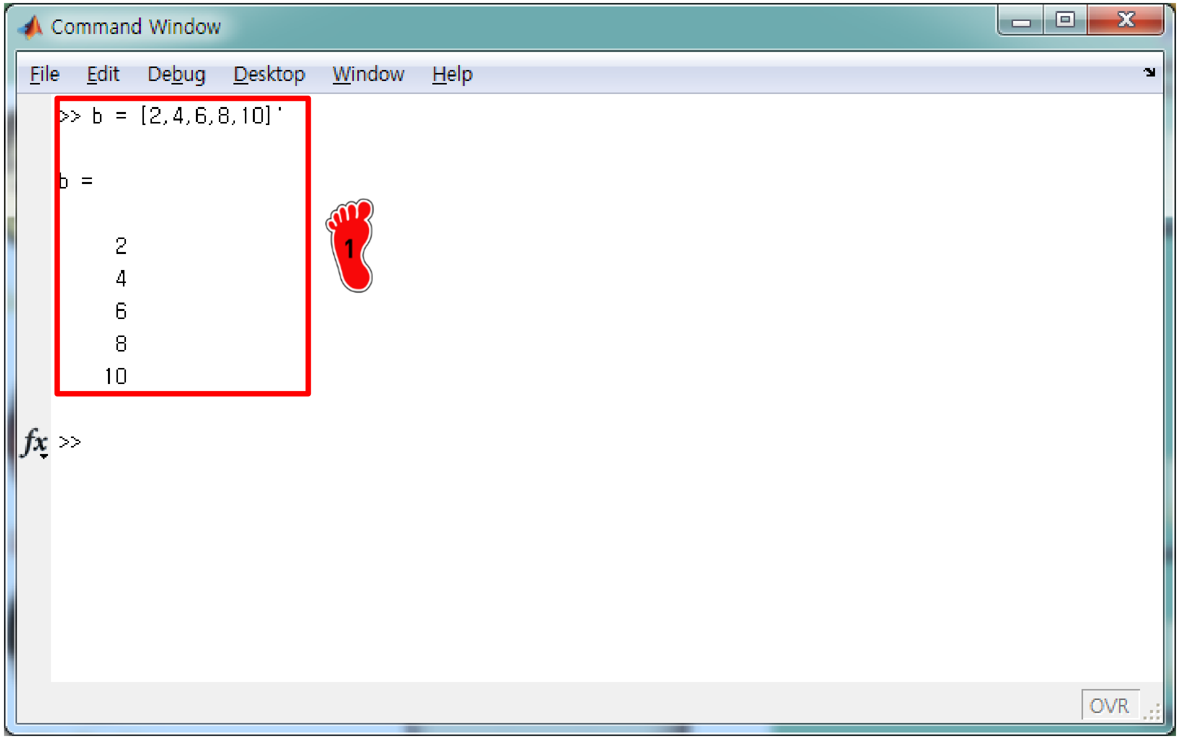
COLUMN VECTOR

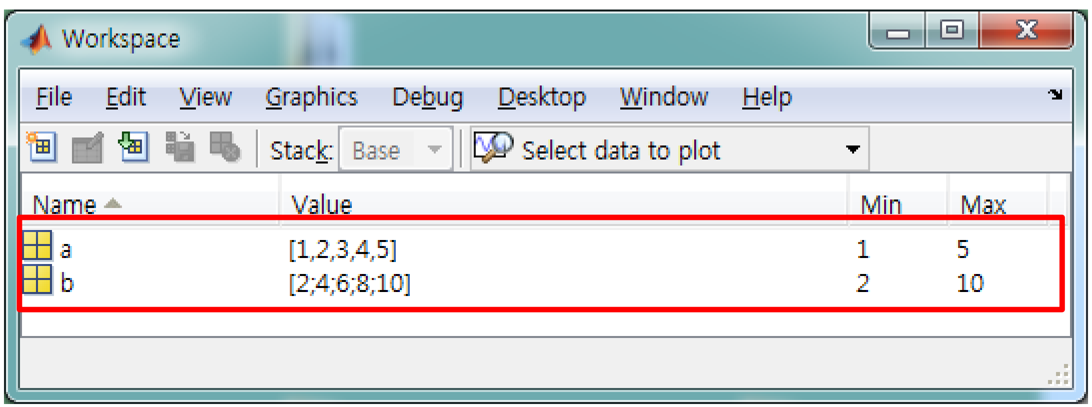




1 괄호 기호 [ ] 와 세미 콜론(;) 을 이용하여 열 벡터를 저장할 수 있음

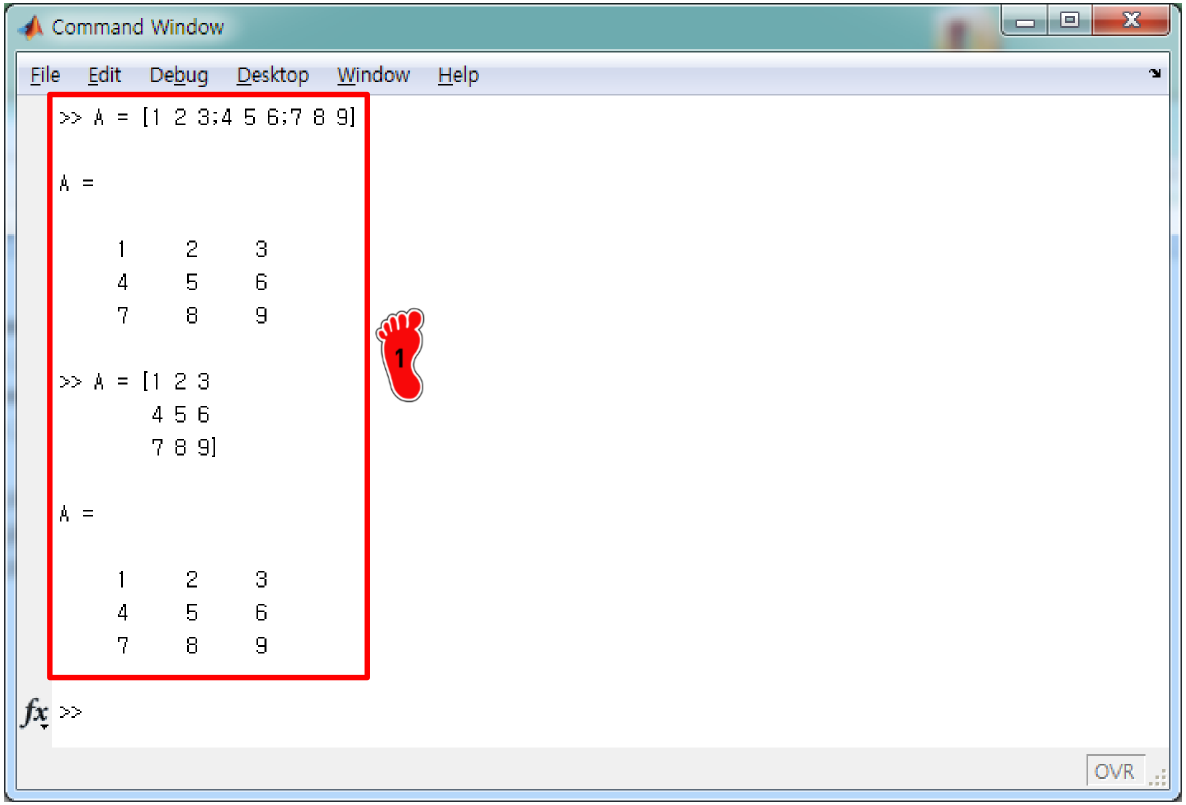
COLUMN VECTOR

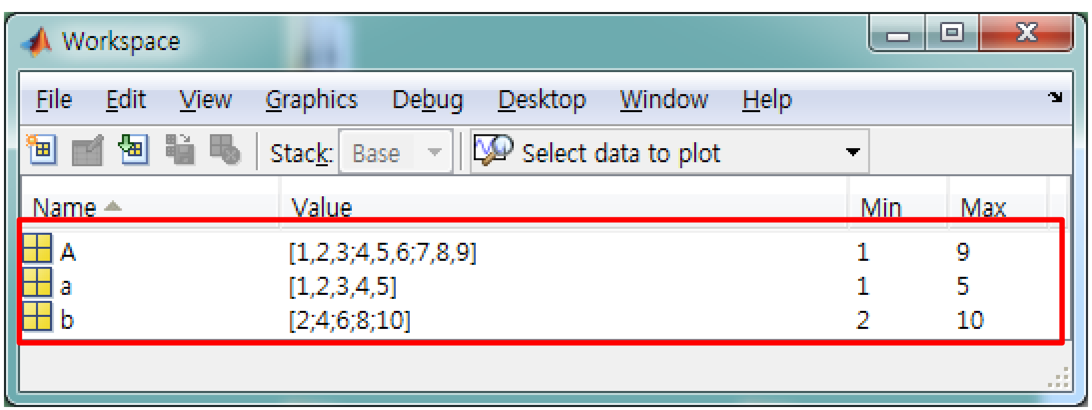




혹은 행 벡터로 입력 한 뒤 작은 따옴표 ' (transpose)를 이용하여 열 벡터로 입력 가능

MATRIX



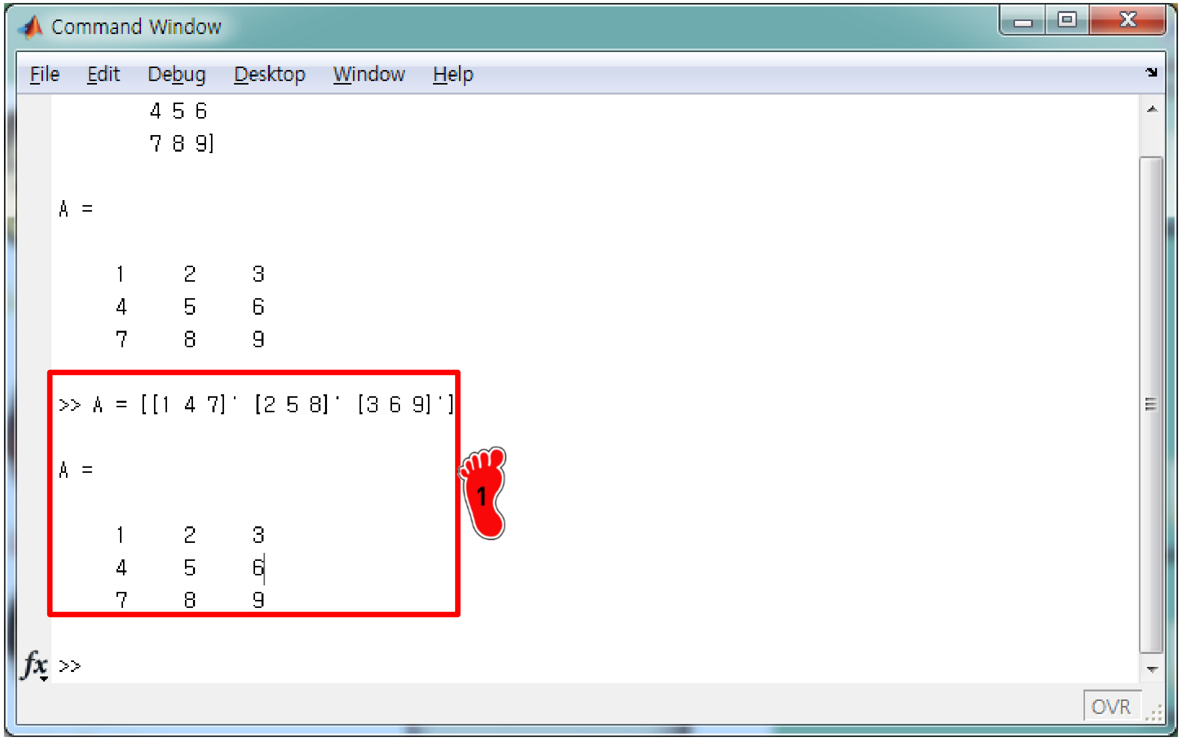


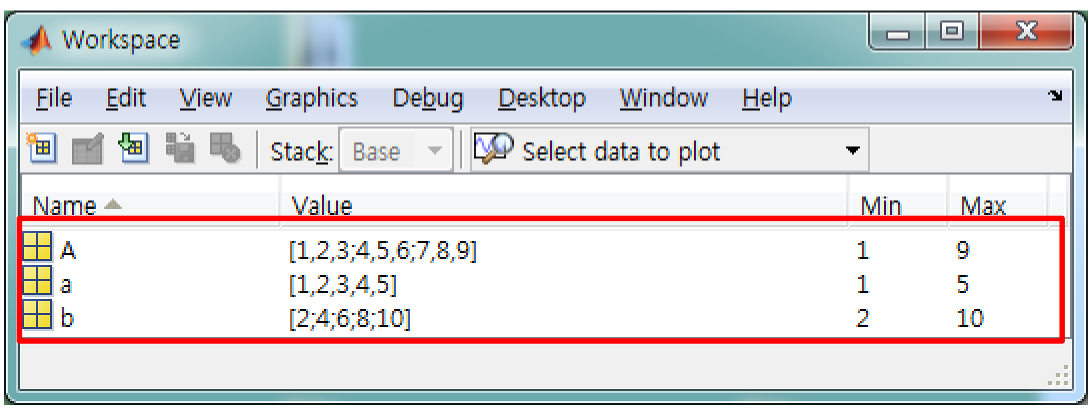
행렬을 입력하는 방법은 벡터를 입력하는 방식과 동일

행 벡터를 입력 한 후, 세미콜론 (;)으로 열을 구분하여 입력 가능

혹은 괄호 기호 [를 시작하면 엔터를 입력 할 경우 다음 줄로 넘어가기 때문에 두 번째 방식으로 입력할 수 있음

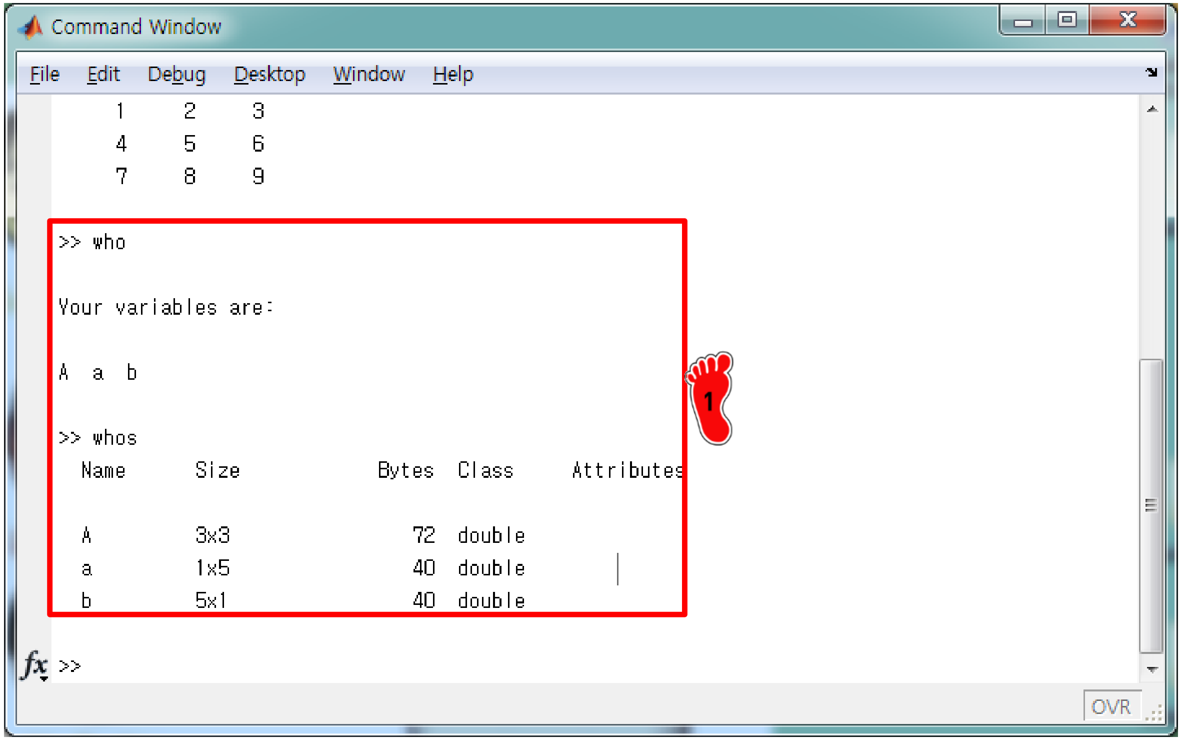
MATRIX

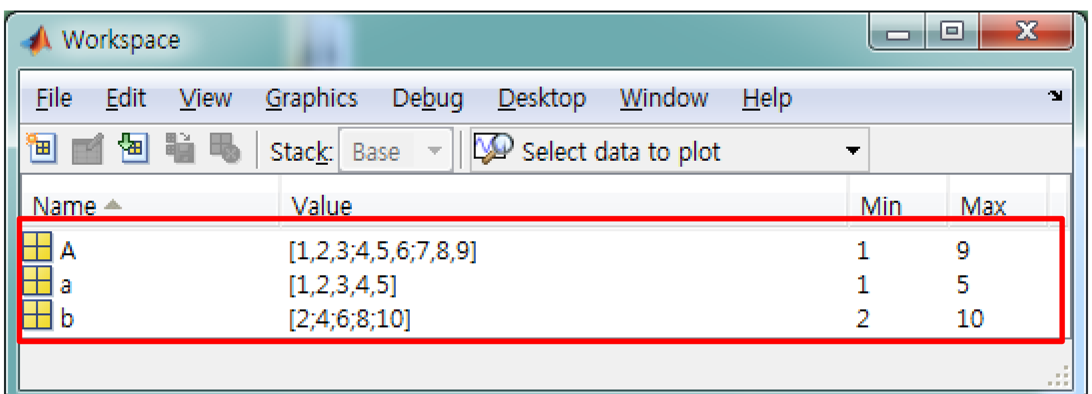




1 Transpose 기호(')를 사용, 다음과 같이 각각의 열 벡터를 이용하여 행렬을 구성할 수 있음

MATRIX: WHO(S)

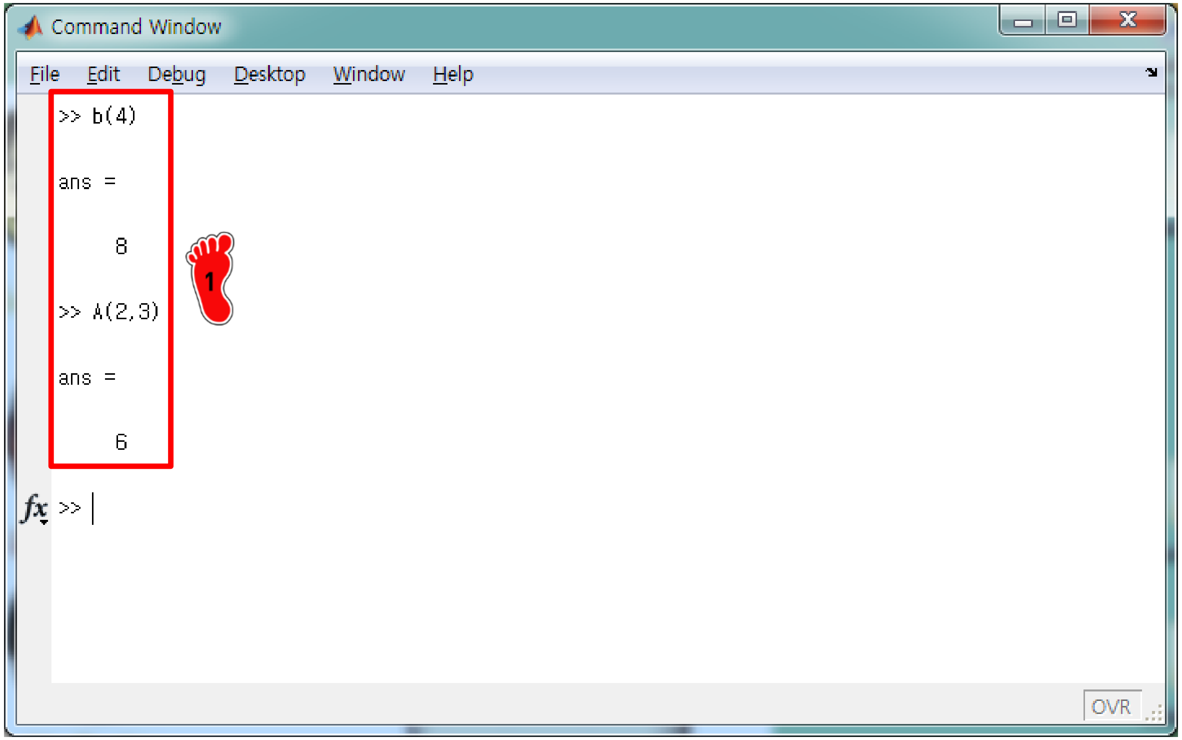


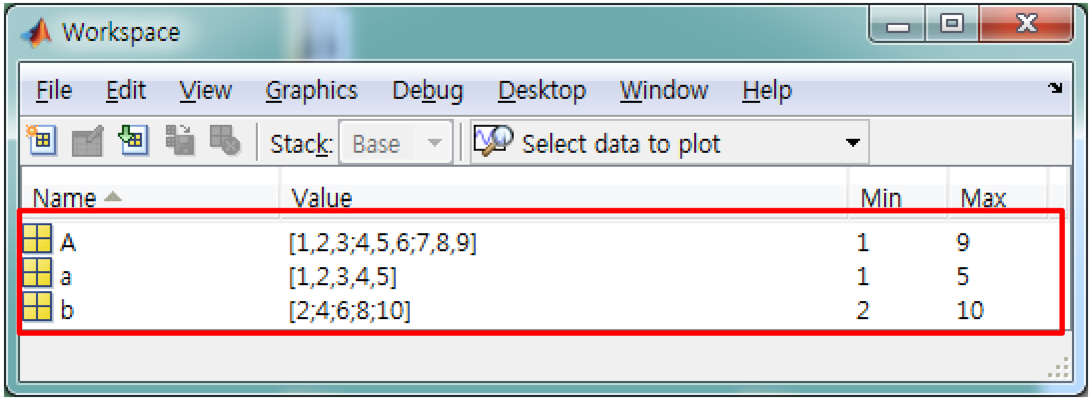


who
현재 저장 되어있는 변수들의 간략한 이름만 표시

whos
현재 저장 되어있는 변수들의 이름과 자세한 정보를 나타냄

MATRIX: ELEMENT

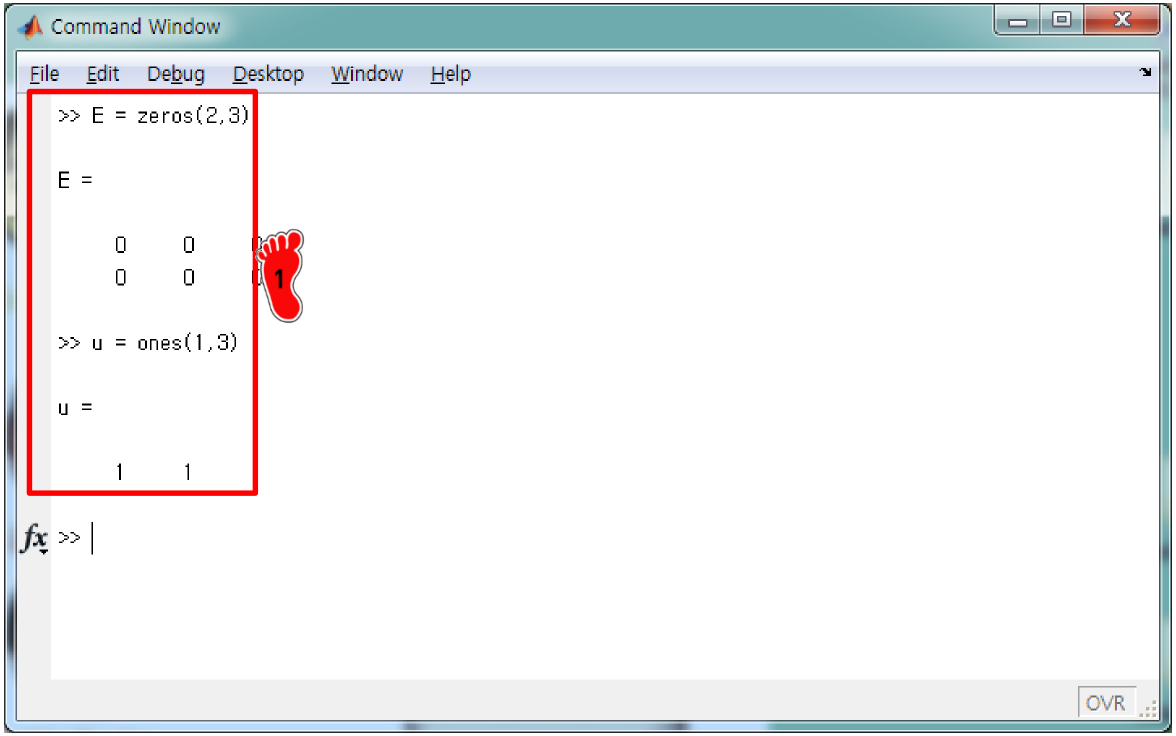


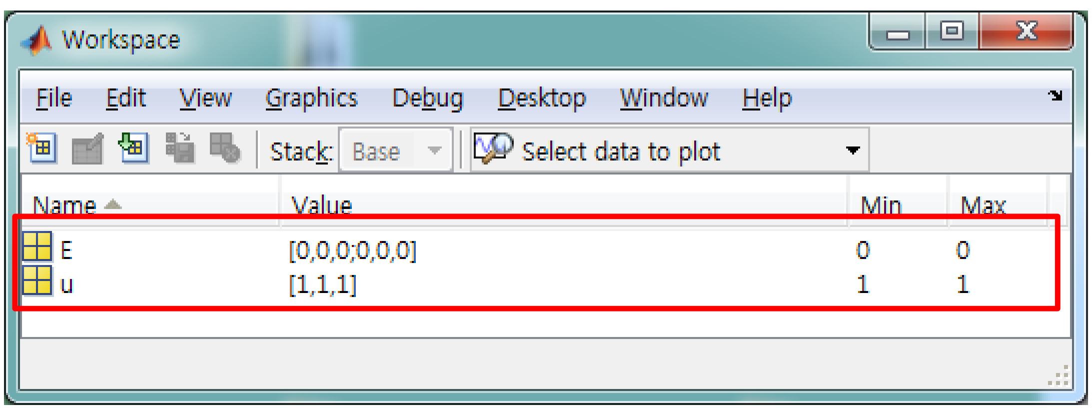


벡터 b 의 4행 1열에 저장되어있는 값을 확인할 경우 괄호 () 를 이용

행렬 A 도 같은 방식으로 저장되어있는 값 확인

MATRIX: BUILT IN FUNCTION



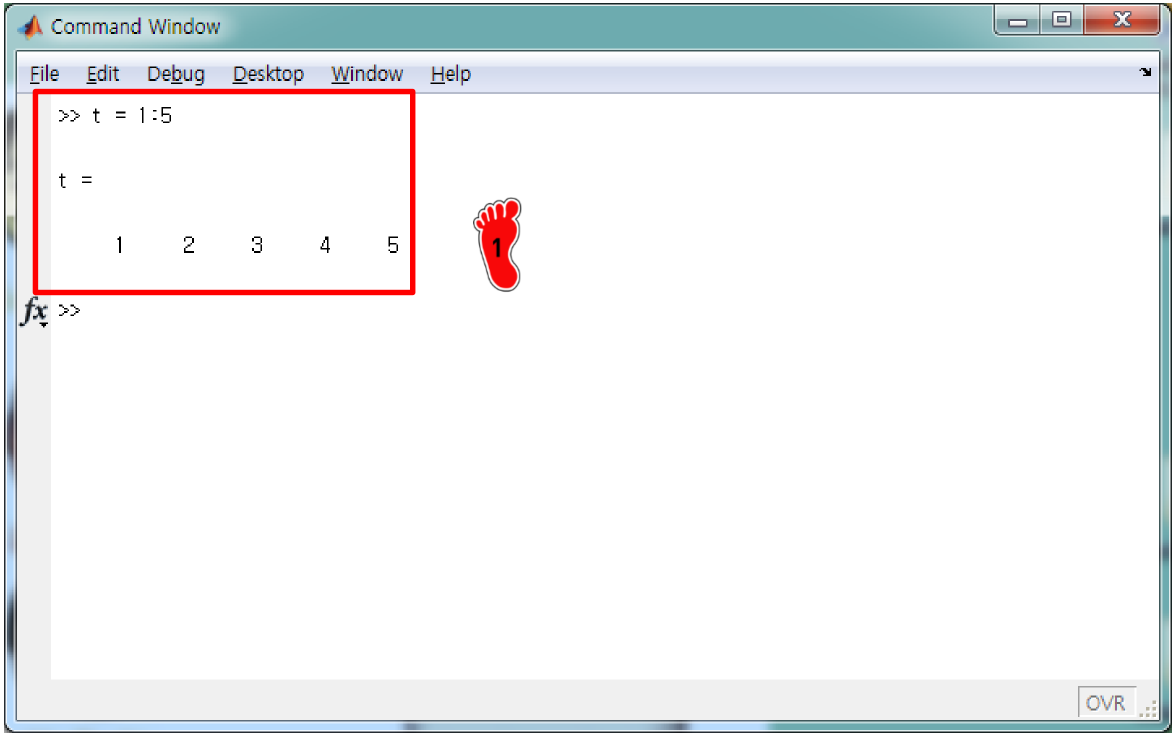


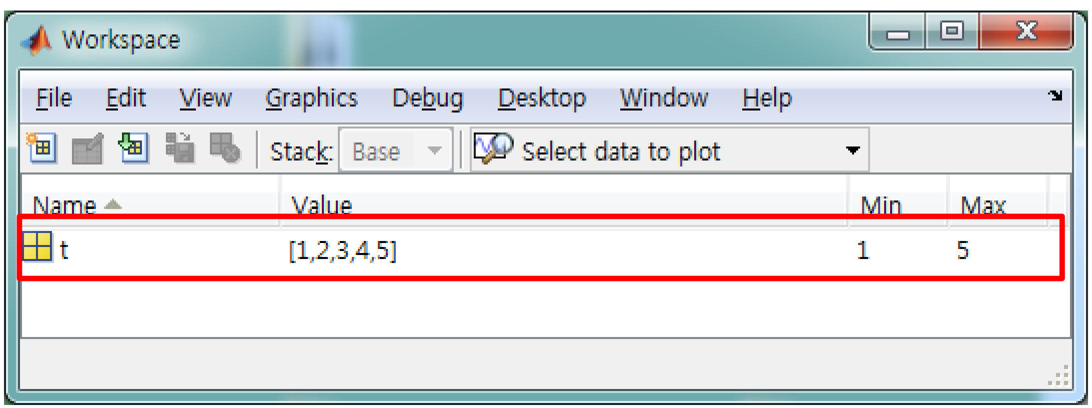
1

zeros(m,n)
m by n 의 0으로 채워진 매트릭스를 저장

ones(m,n)
m by n 의 1로 채워진 매트릭스를 저장

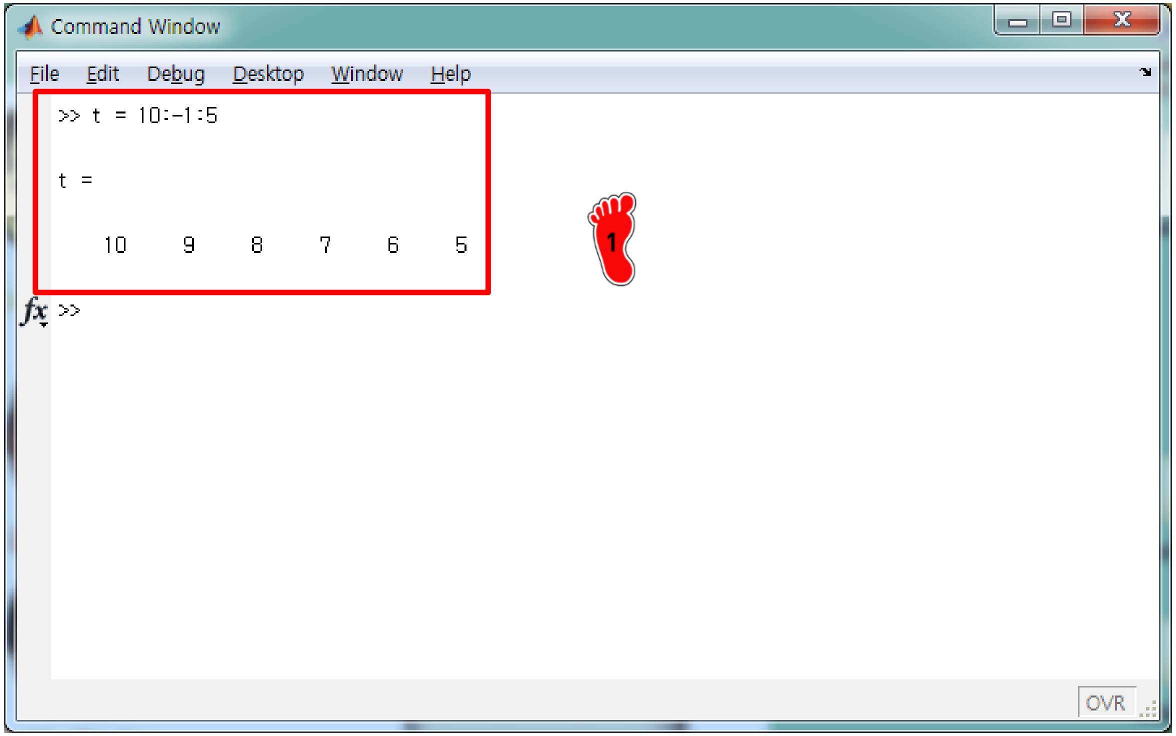
COLON OPERATOR

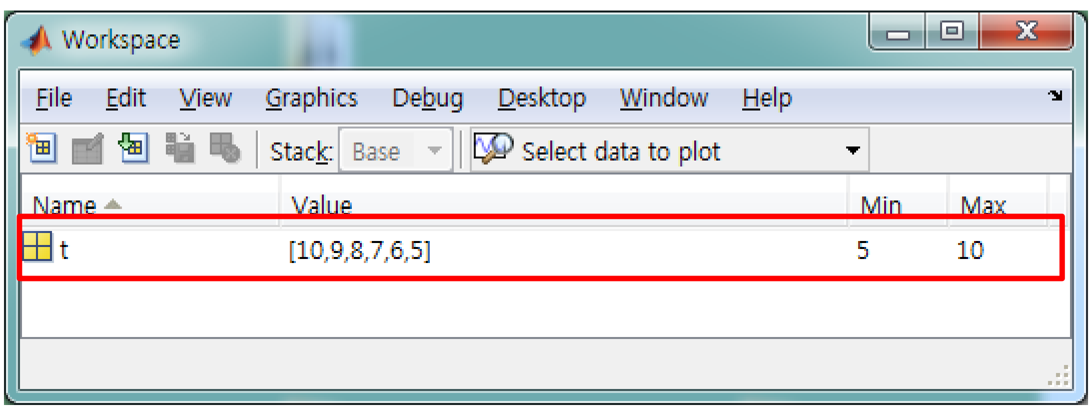




1 콜론 (:) 을 이용하면 1단위로 증가하는 배열을 저장할 수 있음

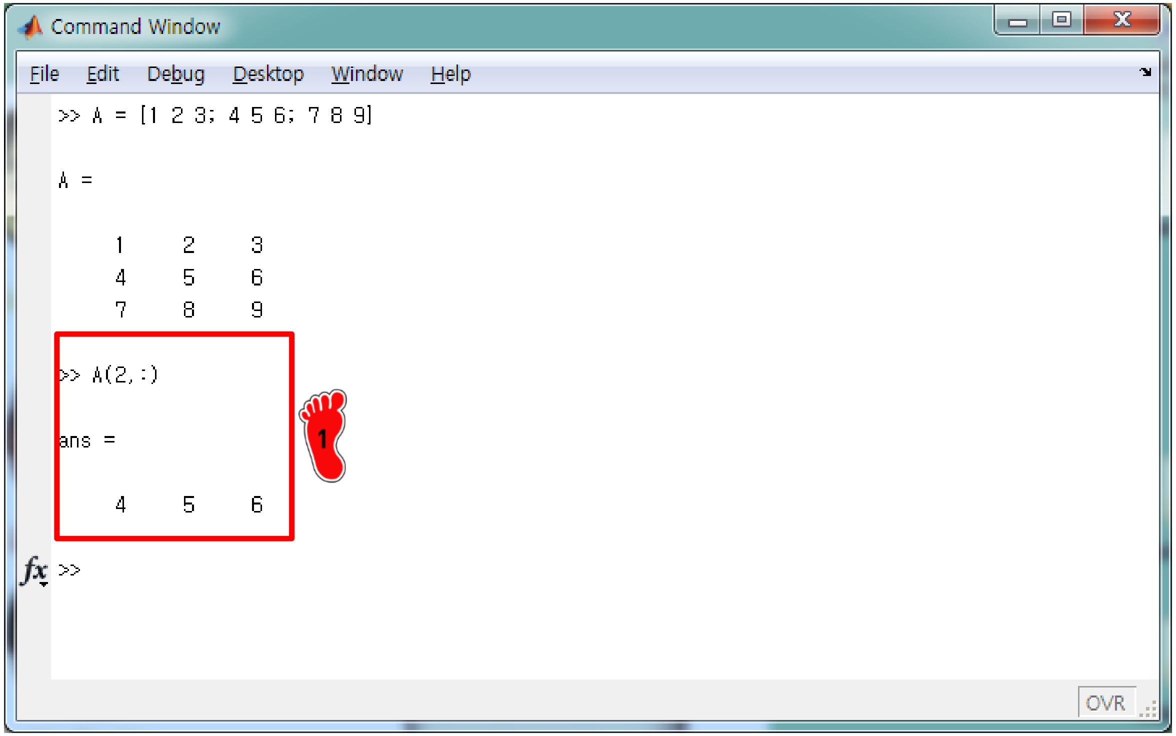
COLON OPERATOR

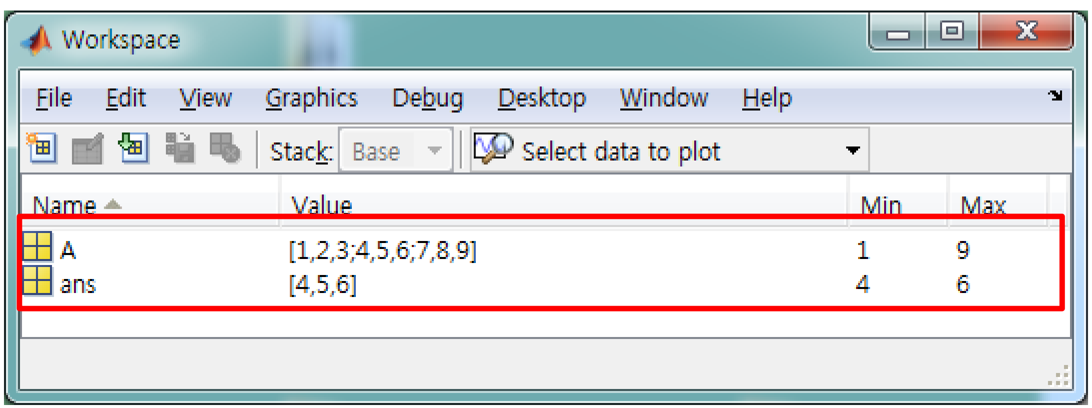




- 값을 이용하여 줄어드는 배열을 저장할 수 있음

COLON OPERATOR

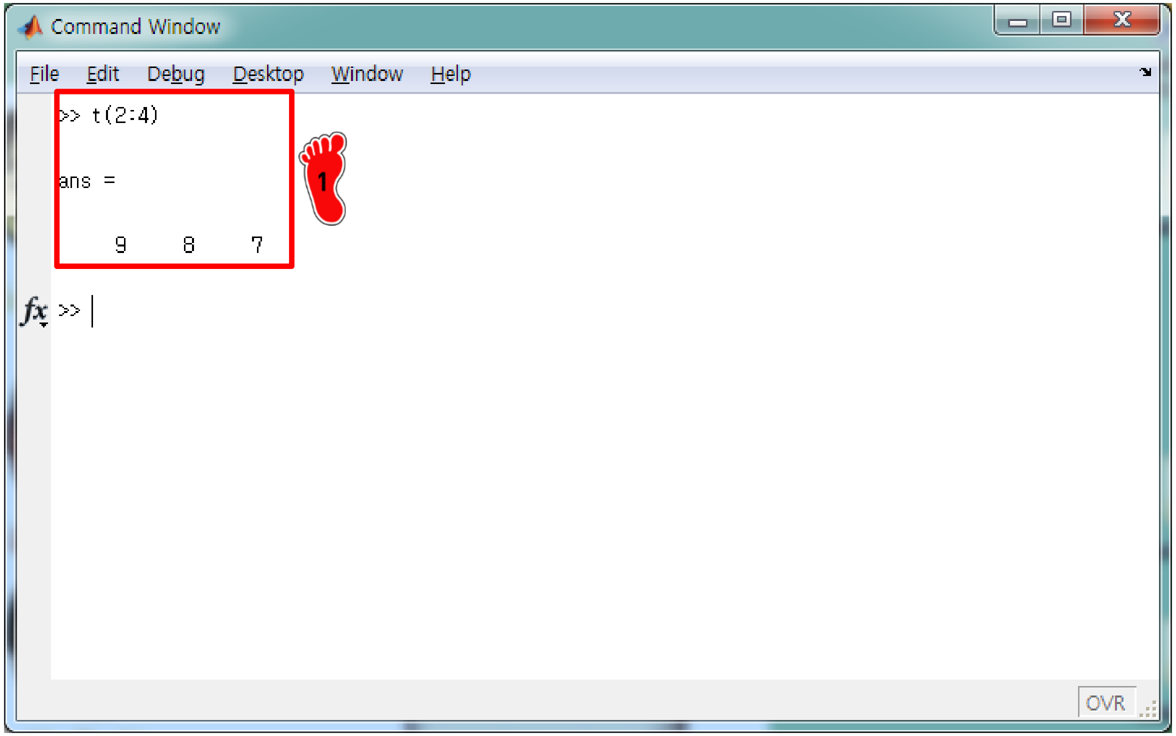


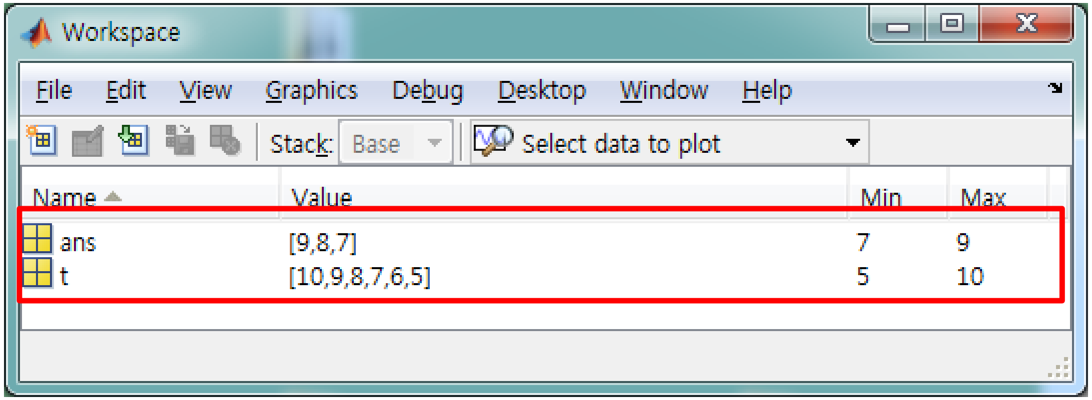


행렬의 값을 확인하는 괄호 () 를 이용할 때 콜론을 이용하면 전체 값을 확인 가능

예시에는 2행 전체 값을 확인

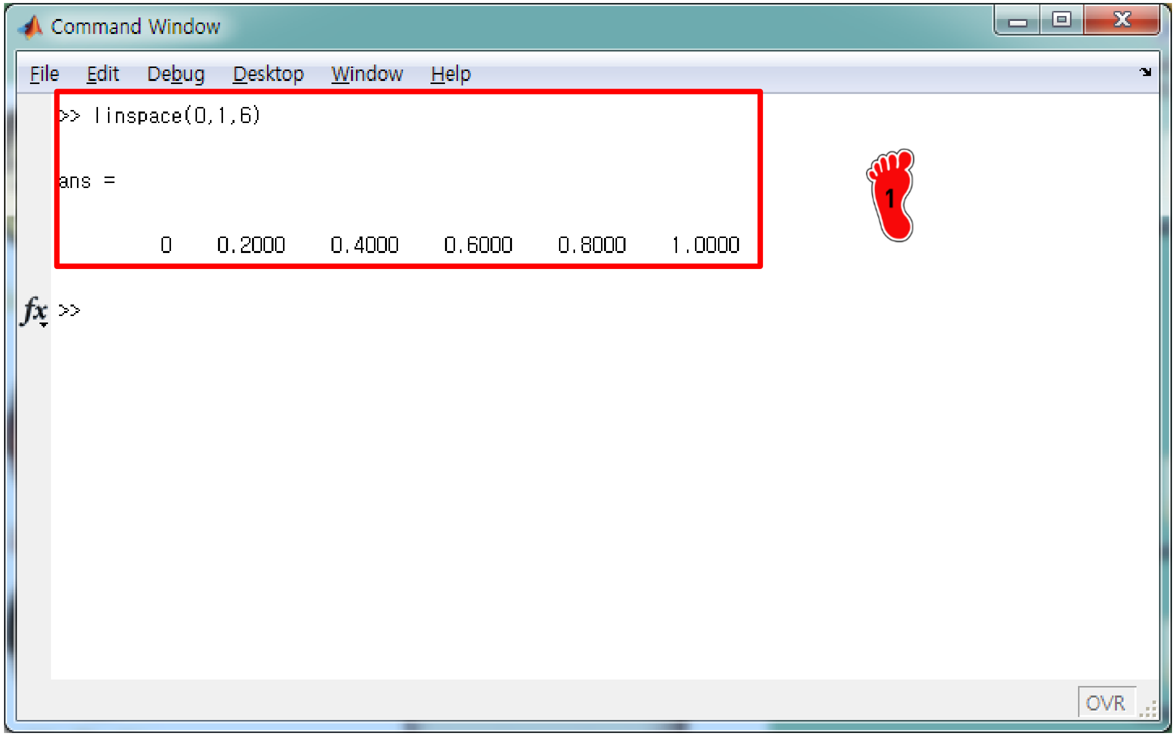
COLON OPERATOR

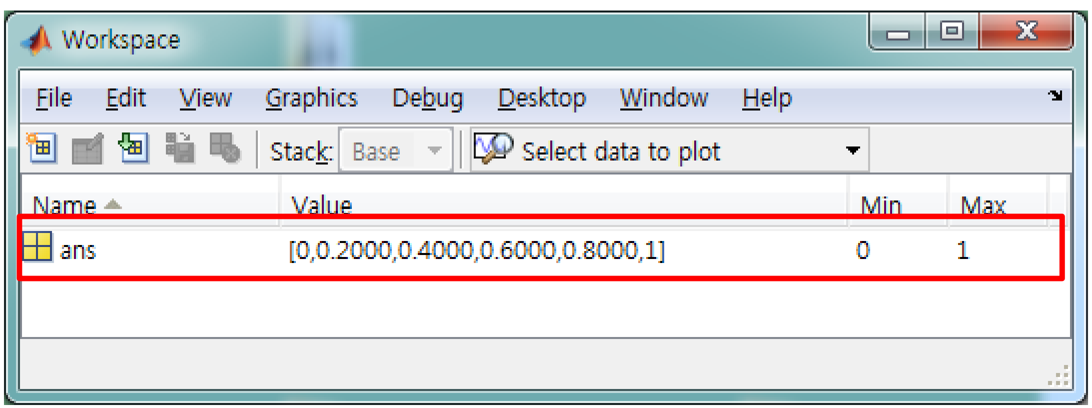




1 배열값 역시 콜론을 이용하여 원하는 위치의 값을 확인할 수 있음

LINSPACE FUNCTION

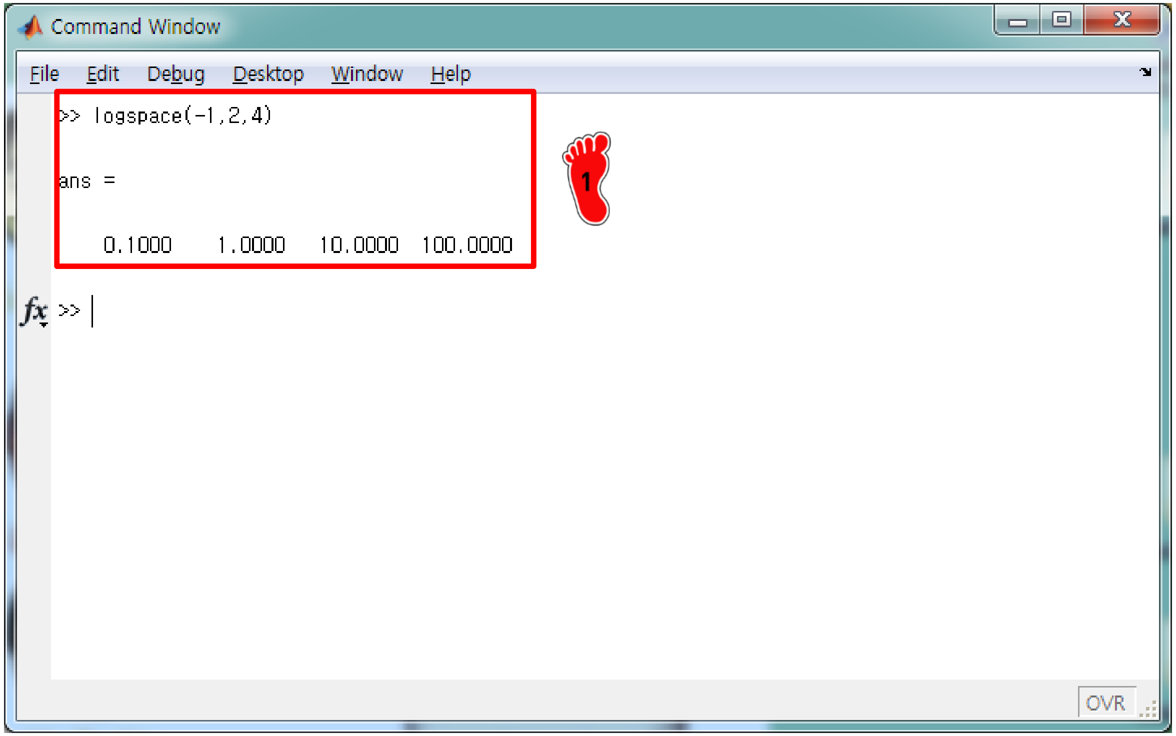


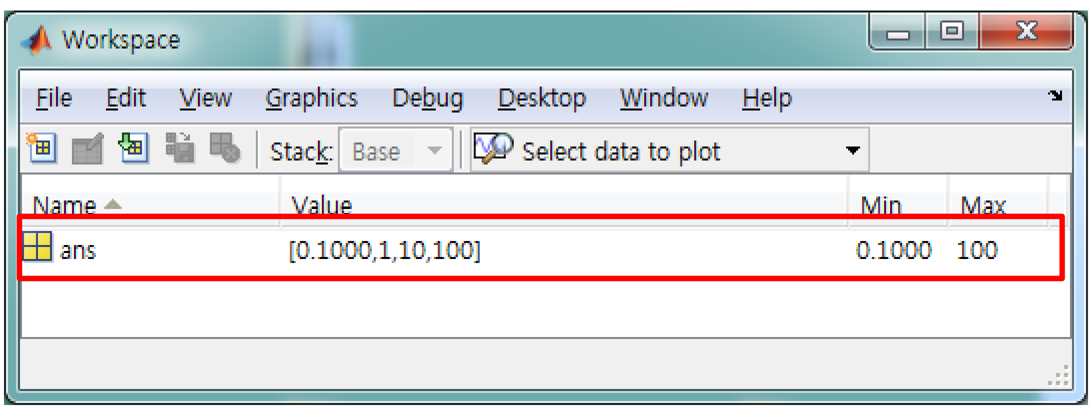


linspace(x1,x2,n)

값 x1 부터 x2 까지 선형적으로 n 등분한 결과값을 저장

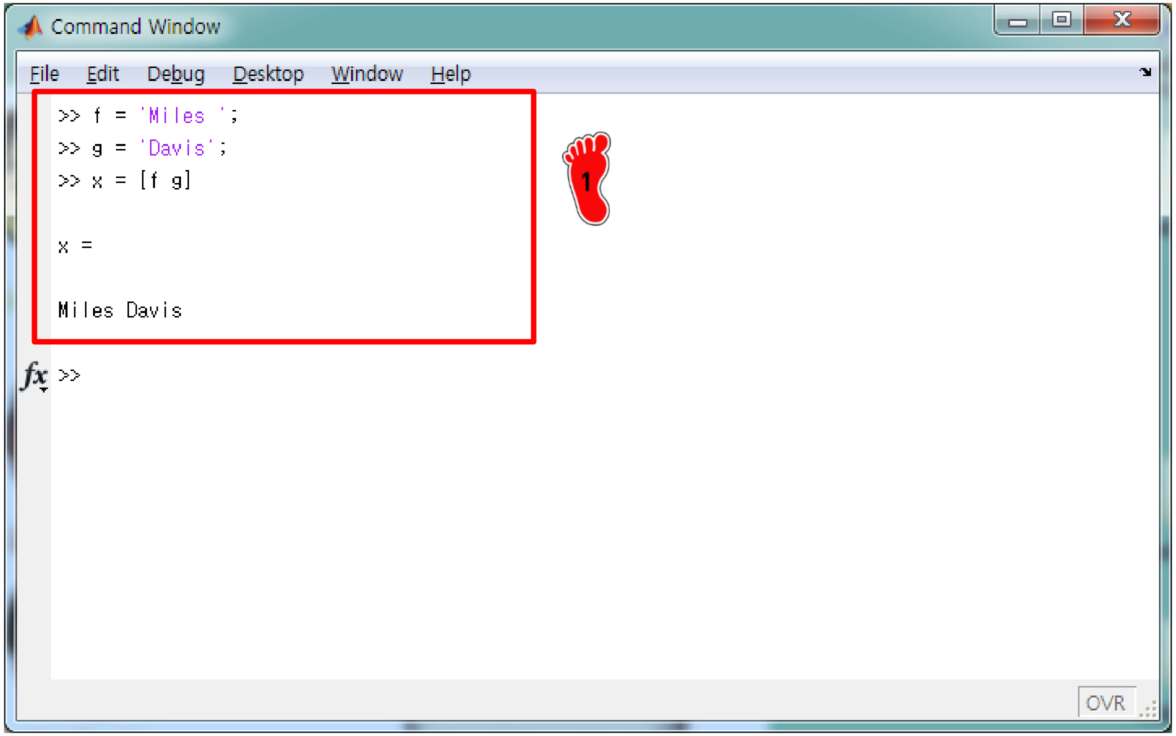
LOGSPACE FUNCTION

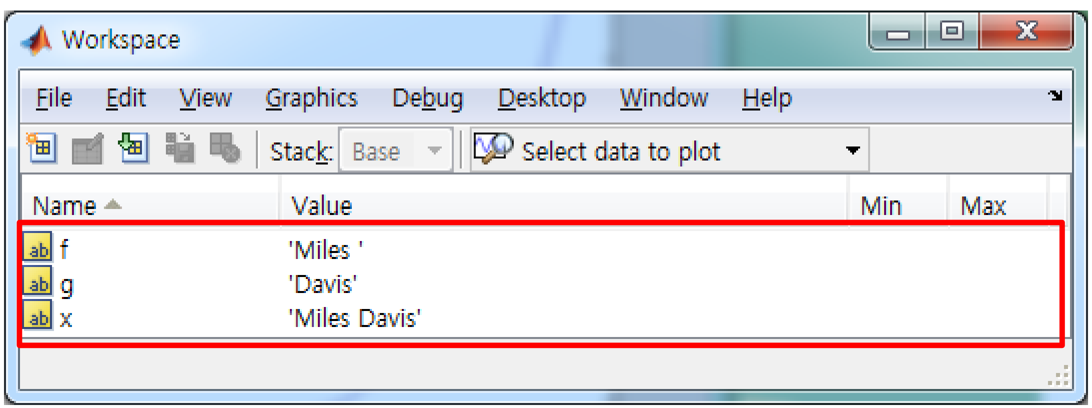




1 **logspace(x1,x2,n)**
값 10^{-1} 부터 10^{2} 까지 로그 스케일로 n 등분한 결과값을 저장

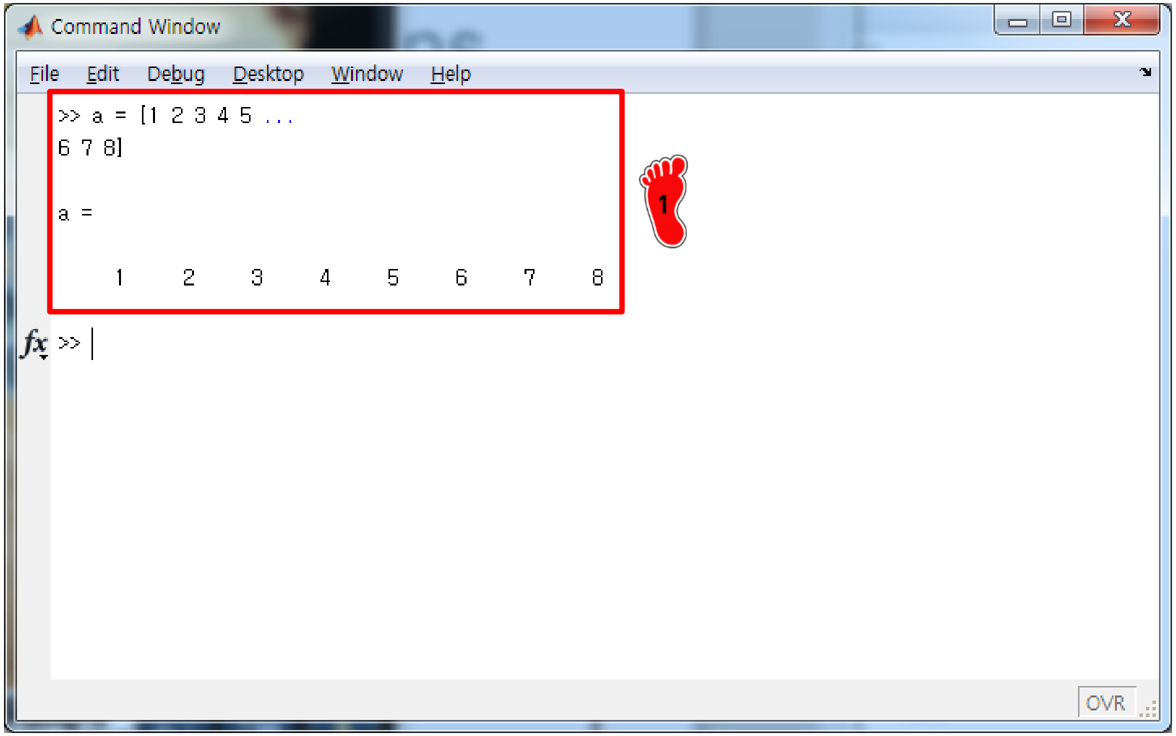
CHARACTER STRING

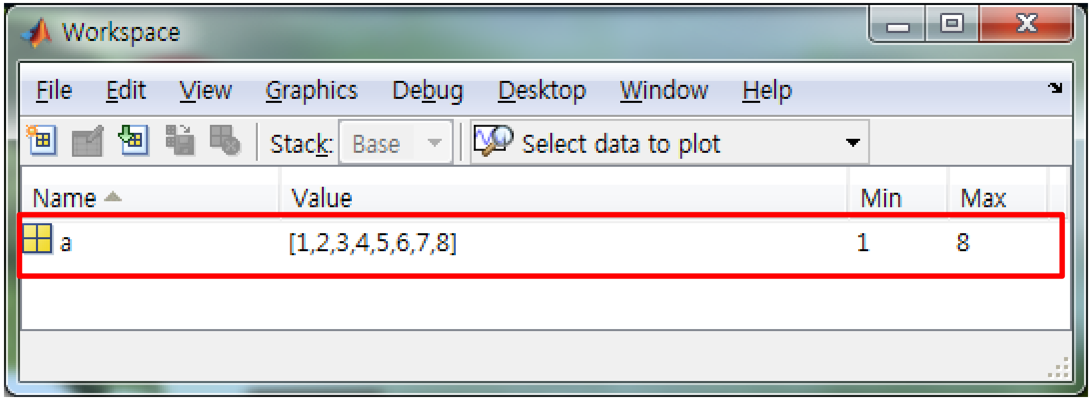




1 작은 따옴표 두 개를 이용하여 글자열을 저장할 수 있음

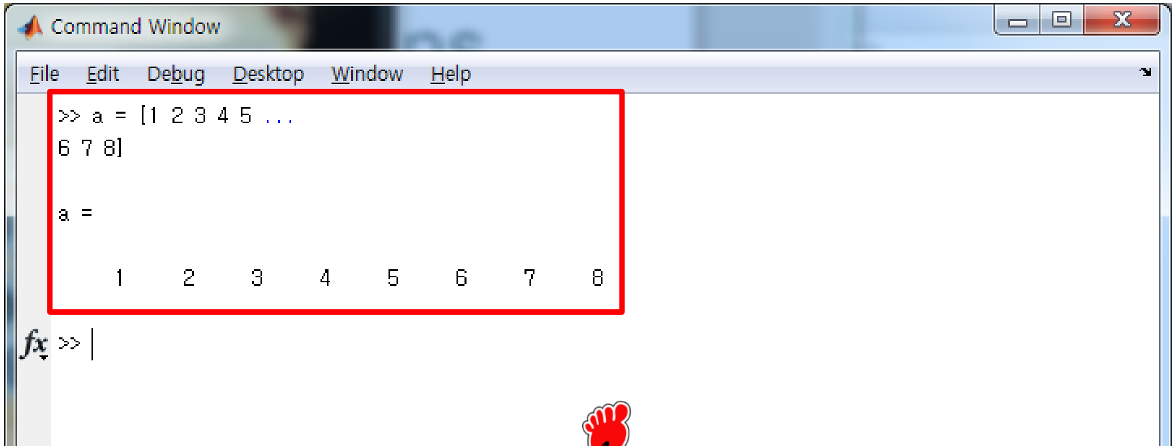
CHARACTER STRING

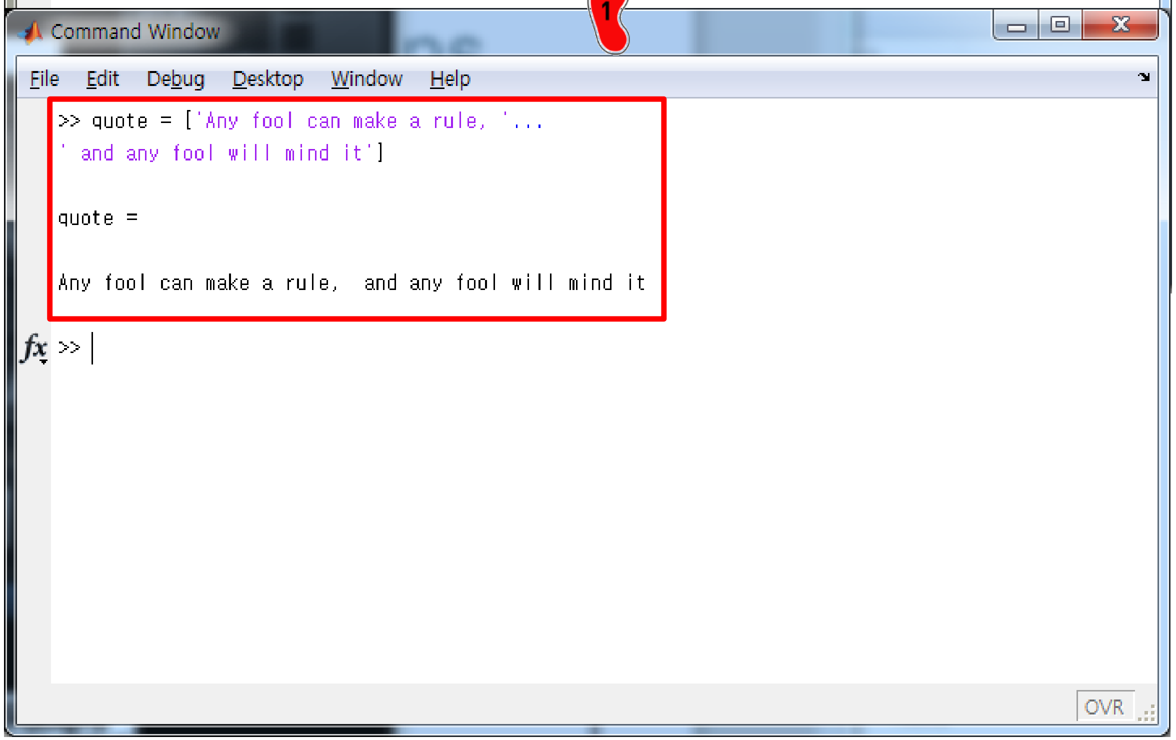




1 마침표를 이용하여 세 개를 입력하면 다음줄로 넘어 갈 수 있음

CHARACTER STRING





1 마침표를 이용하여 세 개를 입력하면 다음줄로 넘어 갈 수 있음

- **Mathematical operations**

  ✓ **Operators**

  ✓ **Mathematical operation**

  ✓ **Vector product**

  ✓ **Vector-matrix multiplication**

  ✓ **Matrix-matrix multiplication**

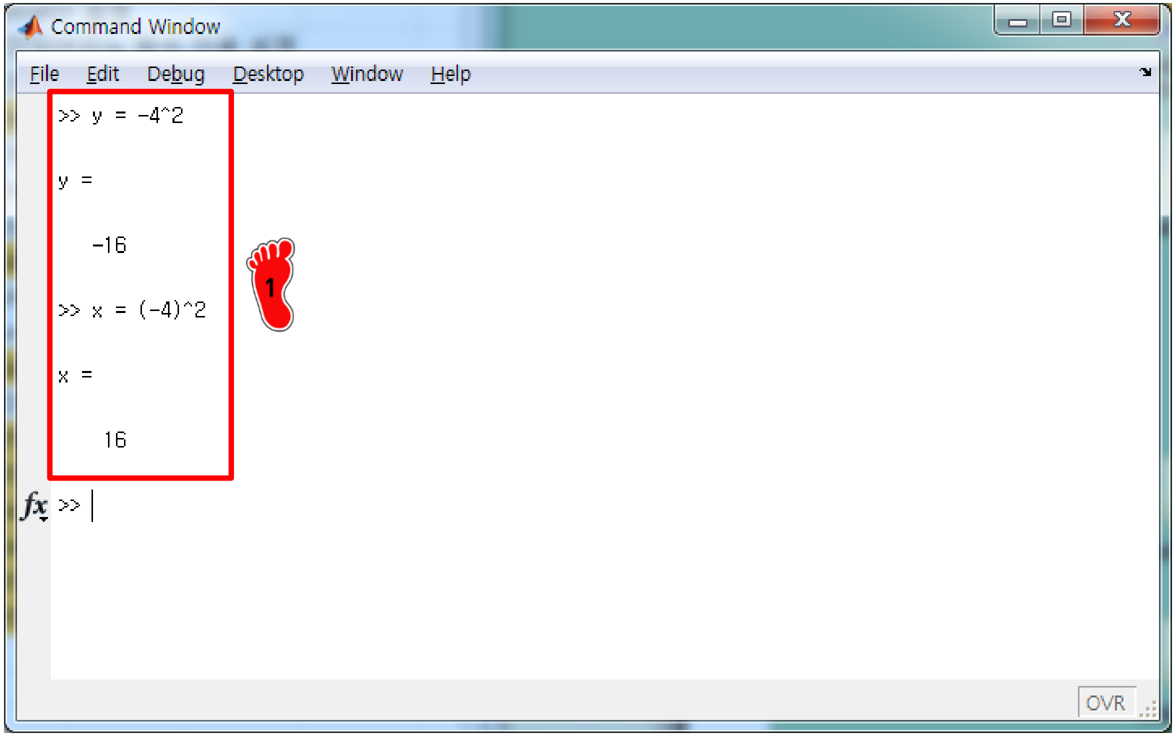
  ✓ **Mixed operation**

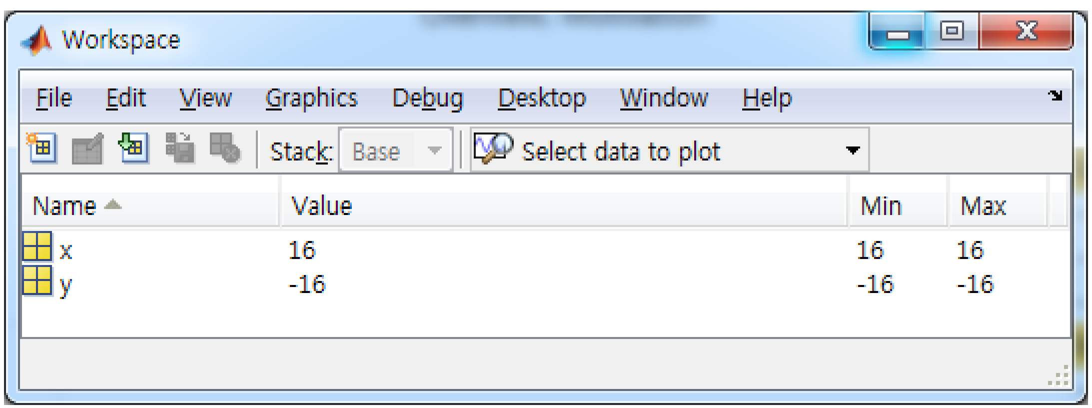
OPERATORS

매틀랩 연산 기호

^	Exponentiation	4^2 = 8
-	Negation (unary operation)	-8 = -8
* /	Multiplication and Division	2*pi = 6.2832 pi/4 = 0.7854
\	Left Division	6\2 = 0.3333
+ -	Addition and Subtraction	3+5 = 8 3-5 = -2

MATHEMATICAL OPERATION



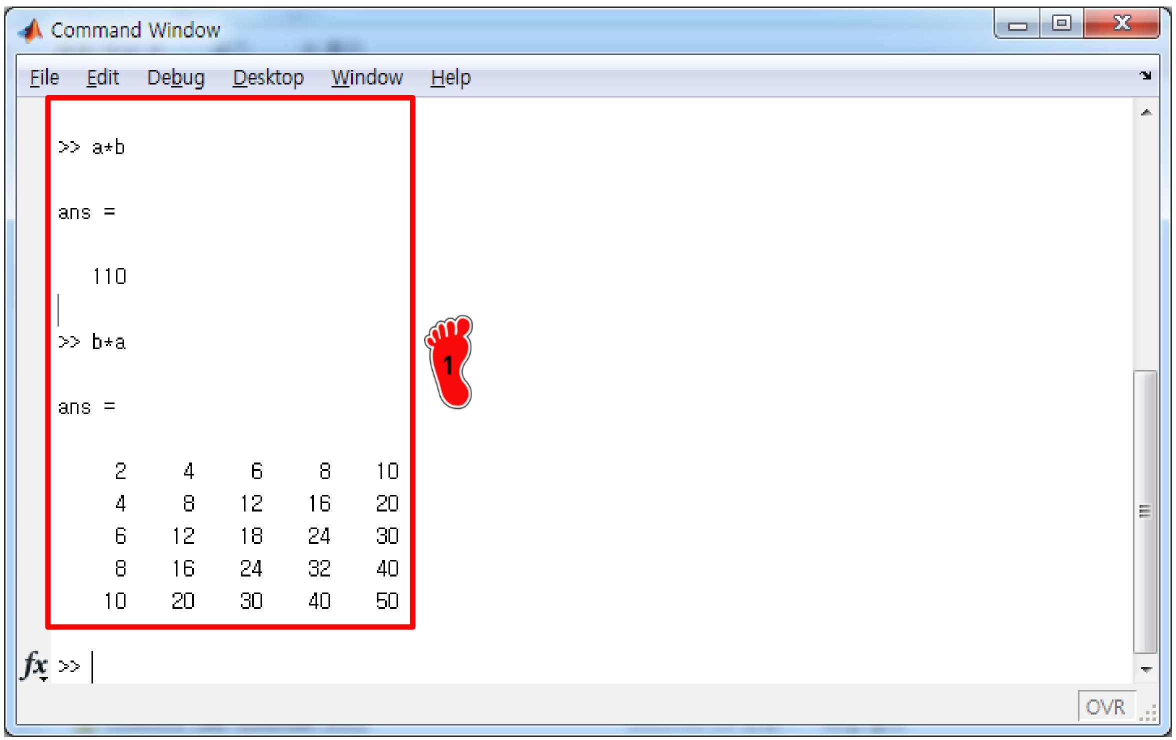


1 연산에 우선순위가 있는것을 확인

VECTOR PRODUCT

$$a = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}, b = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \\ 10 \end{bmatrix}$$

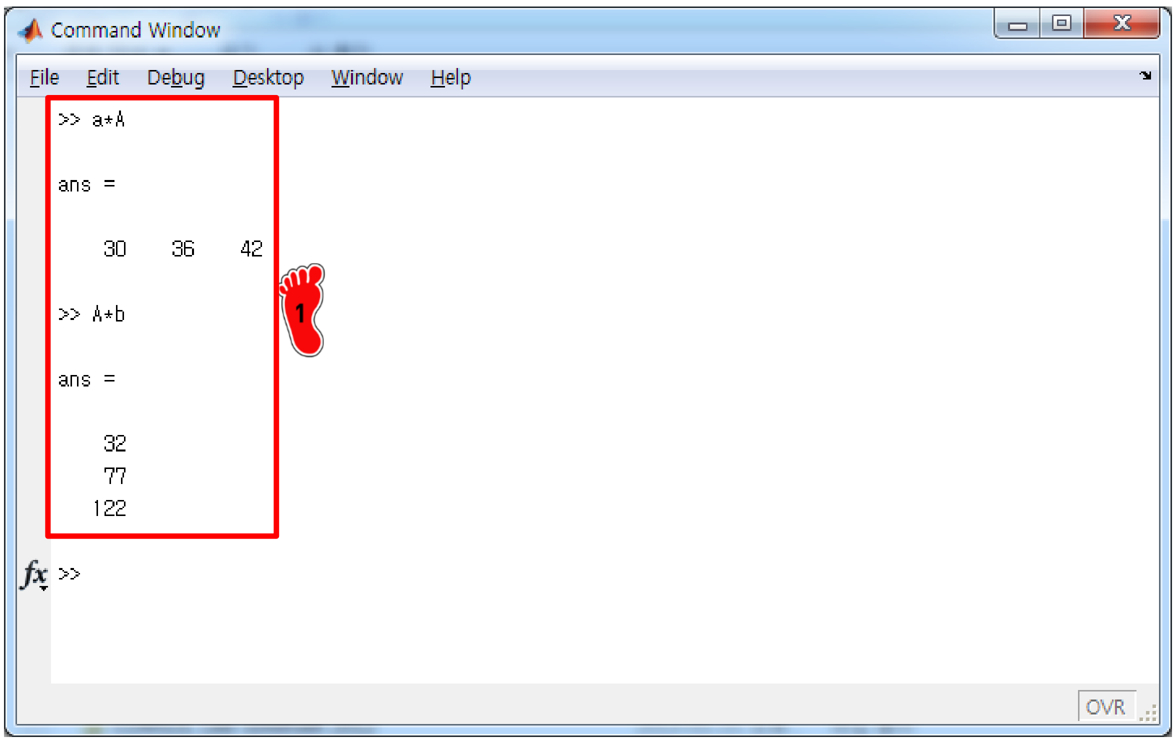
1 저장되어있는 변수가 벡터 혹은 행렬일 경우 벡터 연산으로 연산 작용이 됨



VECTOR-MATRIX MULTIPLICATION

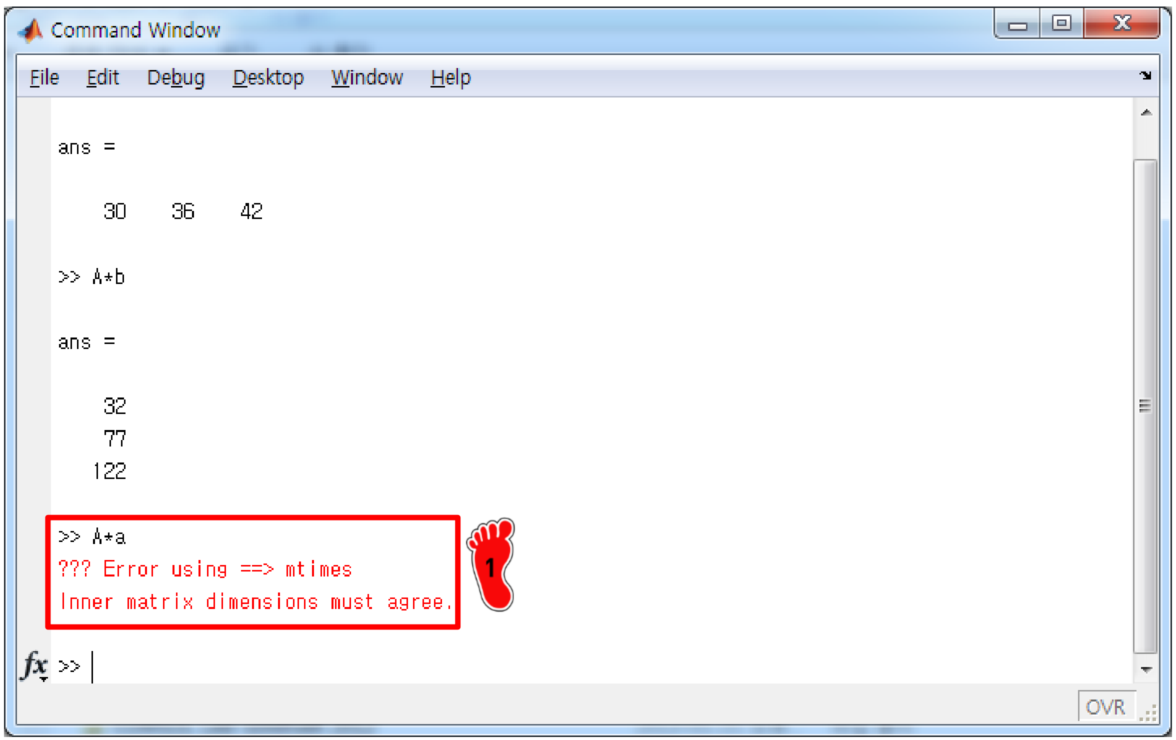
1 벡터와 행렬 간 연산

$$a=\begin{bmatrix}1 & 2 & 3\end{bmatrix}, b=\begin{bmatrix}4\\5\\6\end{bmatrix}, A=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$



VECTOR-MATRIX MULTIPLICATION

$$a = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}, b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



1 차수가 맞지 않을 경우 오류 메세지를 출력

MATRIX-MATRIX MULTIPLICATION

연산기호 ^ 을 행렬 연산에 적용할 경우 제곱을 의미함

$$a=\begin{bmatrix}1 & 2 & 3\end{bmatrix}, b=\begin{bmatrix}4\\5\\6\end{bmatrix}, A=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}$$

Command Window

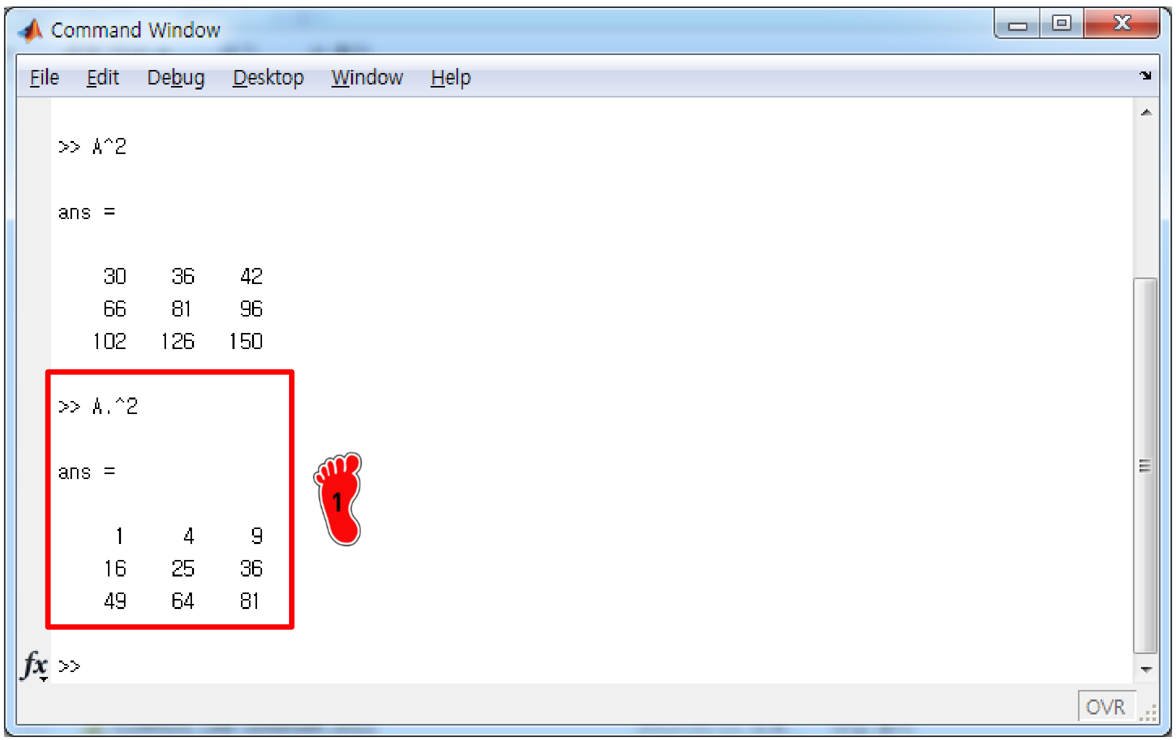
File Edit Debug Desktop Window Help

```
>> A*A

ans =

    30    36    42
    66    81    96
   102   126   150

>> A^2

ans =

    30    36    42
    66    81    96
   102   126   150

fx >>
```

OVR

MIXED OPERATION

$$a = [1 \quad 2 \quad 3], b = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$



1 만약 행렬의 각 항을 제곱한 결과를 보고 싶을때는 마침표를 연산기호 ^ 앞에 붙여서 .^ 연산을 해주게 되면 각 항 별로 제곱한 결과를 확인할 수 있음

- **Built-in functions**
  - ✓ Log
  - ✓ Elfun
  - ✓ Round
  - ✓ Ceil
  - ✓ Floor
  - ✓ Others

LOG

Command Window

File Edit Debug Desktop Window Help

```
>> help log
 LOG    Natural logarithm.
    LOG(X) is the natural logarithm of the elements of X.
    Complex results are produced if X is not positive.

    See also log1p, log2, log10, exp, logm, reallog.

    Overloaded methods:
       gf/log
       codistributed/log
       fints/log

    Reference page in Help browser
       doc log

fx >>
```

log

Natural logarithm

Syntax

```
Y = log(X)
```

Description

The log function operates element-wise on arrays. Its domain includes complex and negative numbers, which may lead to unexpected results if used unintentionally.

Y = log(X) returns the natural logarithm of the elements of X. For complex or negative z, where $z = x + y*i$, the complex logarithm is returned.

```
log(z) = log(abs(z)) + i*atan2(y,x)
```

Examples

The statement abs(log(-1)) is a clever way to generate π.

```
ans =

    3.1416
```

See Also

exp, log10, log2, logm, reallog

1 매트랩에 기본적으로 저장되어있는 함수들이 있음

예를들어 로그함수를 이용하고 싶을 경우, help 명령어를 이용하여 doc log 를 클릭하면 log 함수에 관한 문서가 팝업됨

팝업된 창에는 로그 함수의 설명과 이용방법에 대해 나와있음

ELFUN

Command Window

File Edit Debug Desktop Window Help

```
>> help elfun
  Elementary math functions.

  Trigonometric.
    sin         - Sine.
    sind        - Sine of argument in degrees.
    sinh        - Hyperbolic sine.
    asin        - Inverse sine.
    asind       - Inverse sine, result in degrees.
    asinh       - Inverse hyperbolic sine.
    cos         - Cosine.
    cosd        - Cosine of argument in degrees.
    cosh        - Hyperbolic cosine.
    acos        - Inverse cosine.
    acosd       - Inverse cosine, result in degrees.
    acosh       - Inverse hyperbolic cosine.
    tan         - Tangent.
    tand        - Tangent of argument in degrees.
    tanh        - Hyperbolic tangent.
    atan        - Inverse tangent.
    atand       - Inverse tangent, result in degrees.
    atan2       - Four quadrant inverse tangent.
    atanh       - Inverse hyperbolic tangent.
    sec         - Secant.
    secd        - Secant of argument in degrees.
    sech        - Hyperbolic secant.
    asec        - Inverse secant.
    asecd       - Inverse secant, result in degrees.
    asech       - Inverse hyperbolic secant.
    csc         - Cosecant.
    cscd        - Cosecant of argument in degrees.
    csch        - Hyperbolic cosecant.
    acsc        - Inverse cosecant.
    acscd       - Inverse cosecant, result in degrees.
    acsch       - Inverse hyperbolic cosecant.
    cot         - Cotangent.
    cotd        - Cotangent of argument in degrees.
    coth        - Hyperbolic cotangent.
    acot        - Inverse cotangent.
```

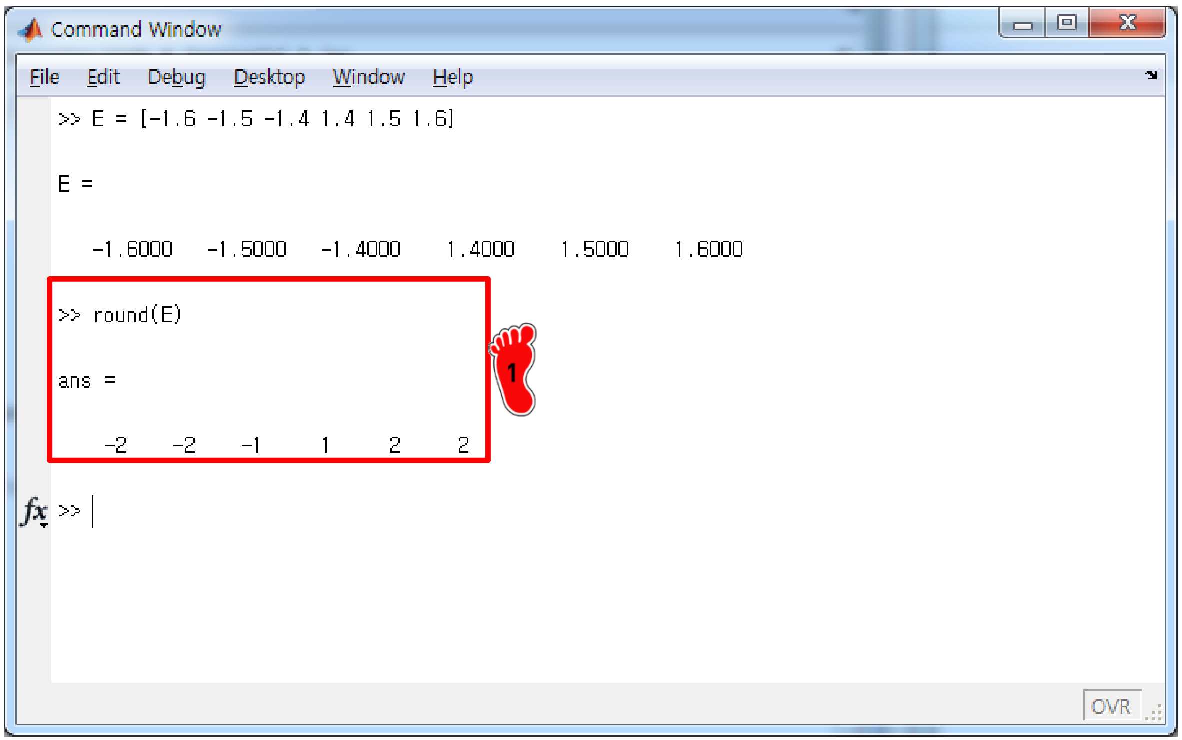
OVR

1 그 외 기본적인 수학 관련 함수는 help elfun 으로 확인 할 수 있음

ROUND

round 함수는 각 원소별로 반올림하여 정수로 만들어 주는 함수

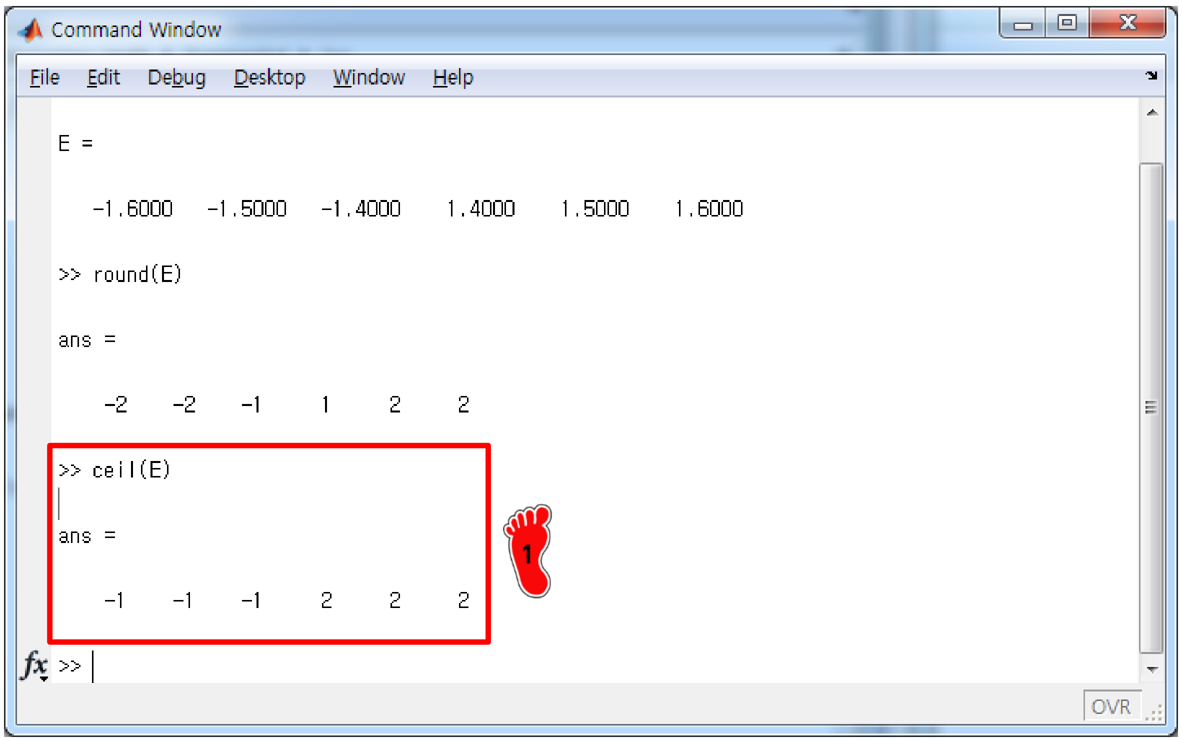
$$E = \begin{bmatrix} -1.6 & -1.5 & -1.4 & 1.4 & 1.5 & 1.6 \end{bmatrix}$$



CEIL

ceil 함수는 올림하여 정수로 만들어 주는 함수

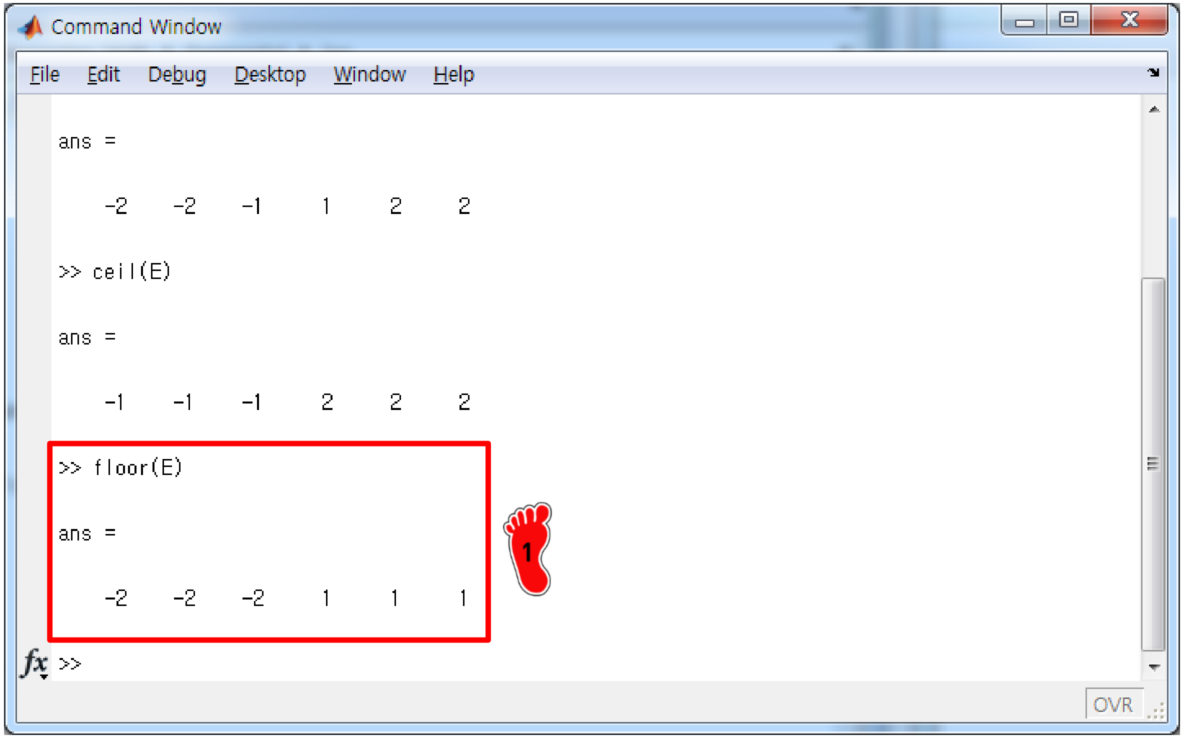
$$E = \begin{bmatrix} -1.6 & -1.5 & -1.4 & 1.4 & 1.5 & 1.6 \end{bmatrix}$$



FLOOR

floor 함수는 내림하여 정수로 만들어 주는 함수

$$E = \begin{bmatrix} -1.6 & -1.5 & -1.4 & 1.4 & 1.5 & 1.6 \end{bmatrix}$$



OTHERS

$$F = [3 \quad 5 \quad 4 \quad 6 \quad 1]$$

Command Window

File Edit Debug Desktop Window Help

```
>> sum(F), min(F), max(F), mean(F), prod(F), sort(F)

ans =

    19


ans =

     1


ans =

     6


ans =

    3.8000


ans =

   360


ans =

     1     3     4     5     6
```

OVR

1

sum: 각 원소를 전부 더한 결과를 출력

min: 행 또는 열 중에서 제일 작은값을 출력

max: 행 또는 열 중에서 제일 큰 값을 출력

mean: 행 또는 열 중에서 평균값을 계산

prod: 각 원소를 전부 곱한 결과를 출력

sort: 각 원소를 낮은 순서대로 배치

- **Graphics**
  - ✓ **Example**
  - ✓ **Plot**
  - ✓ **Specifiers**
  - ✓ **Subplot & 3D plot**

EXAMPLE

- Falling parachutist

$$F = ma = F_D - F_U = \begin{cases} mg - cv \\ mg - cv^2 \end{cases}$$

Case I: $\frac{dv}{dt} = g - \frac{c}{m}v \rightarrow$ nonhomogeneous linear differential equation

Case II: $\frac{dv}{dt} = g - \frac{c}{m}v^2 \rightarrow$ nonhomogeneous nonlinear differential equation

$<$ *Solution* $>$

Case I: $v_h = c_1 e^{\lambda t} \rightarrow \lambda = -\frac{c}{m}$

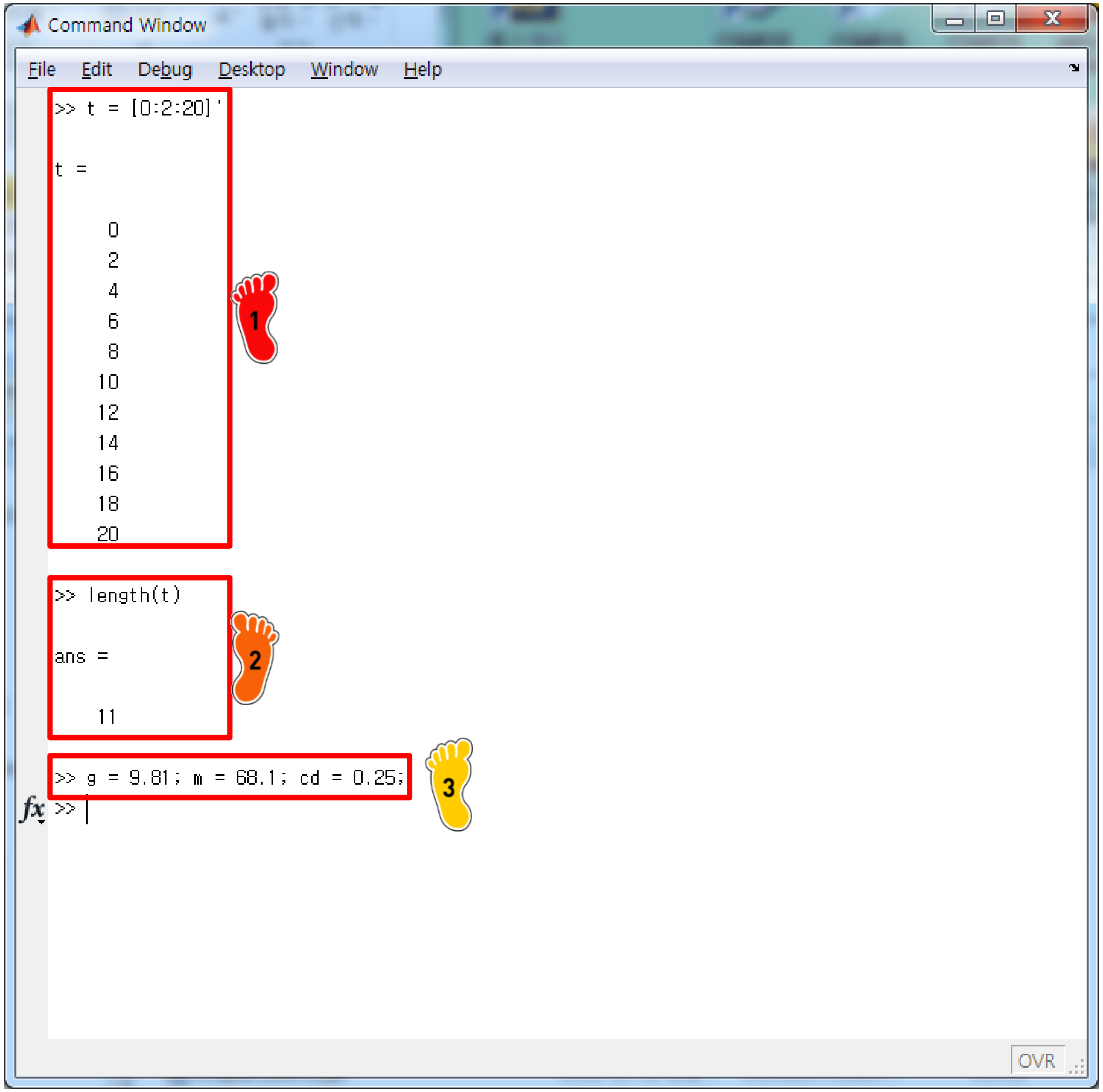
$$v = c_1 e^{\left(-\frac{c}{m}\right)t} + c_2 \rightarrow \frac{c}{m}c_2 = g \rightarrow c_2 = \frac{gm}{c}$$

$$v = c_1 e^{\left(-\frac{c}{m}\right)t} + \frac{gm}{c} \leftarrow (t = 0, v = 0) \Rightarrow c_1 = -\frac{gm}{c}$$

$$v = \frac{gm}{c}\left[1 - e^{\left(-\frac{c}{m}\right)t}\right]$$: analytical (or exact) solution

Case II: $v = \sqrt{\frac{gm}{c}} \tanh\left(\sqrt{\frac{gc}{m}}t\right)$

VARIABLES & CONSTANTS

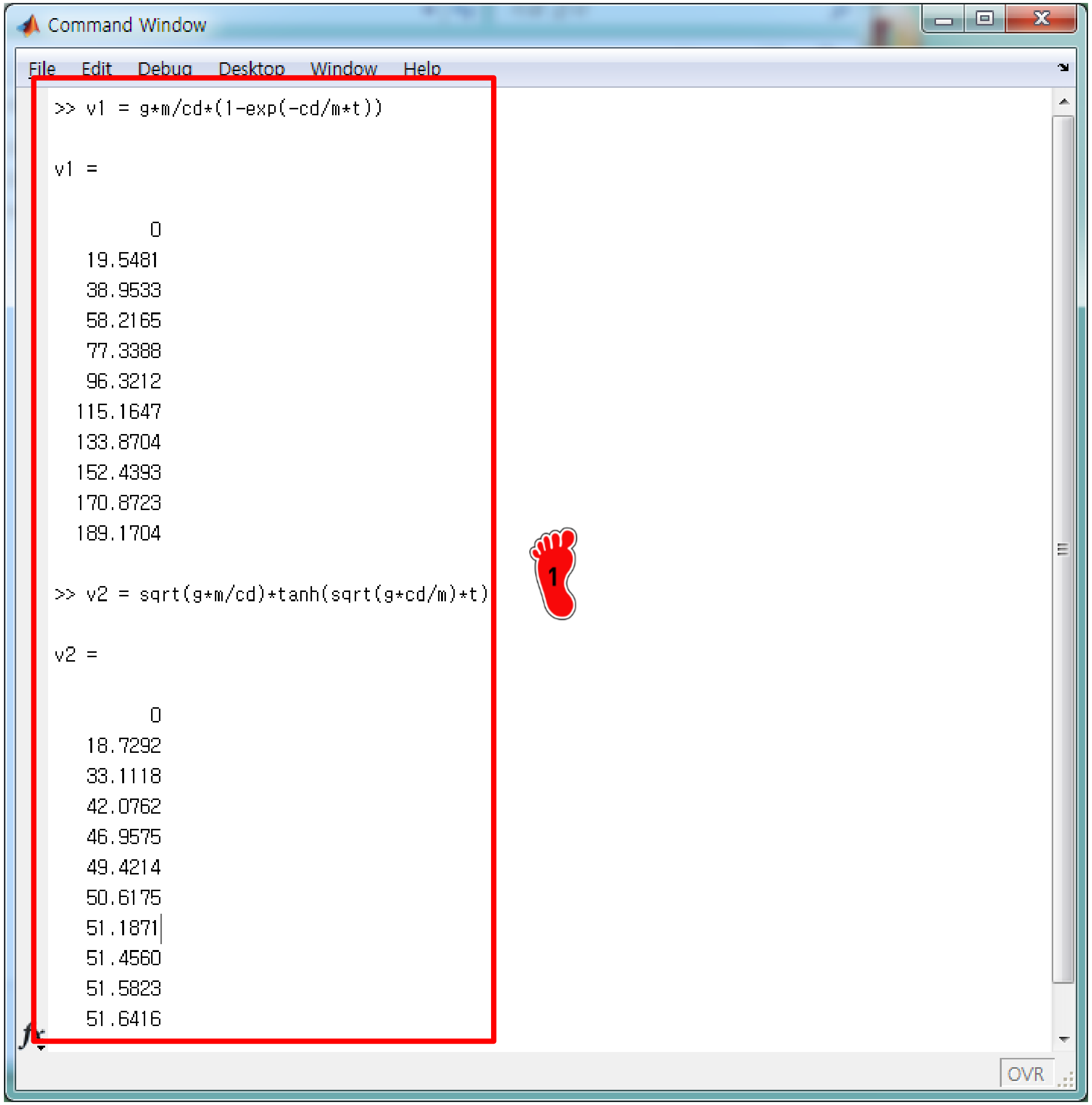


1 독립변수 t 를 0부터 20까지 2초 단위인 벡터로 저장

2 **length** 함수는 벡터 혹은 배열의 최대 크기를 출력 현재 11개의 원소로 벡터가 저장됨

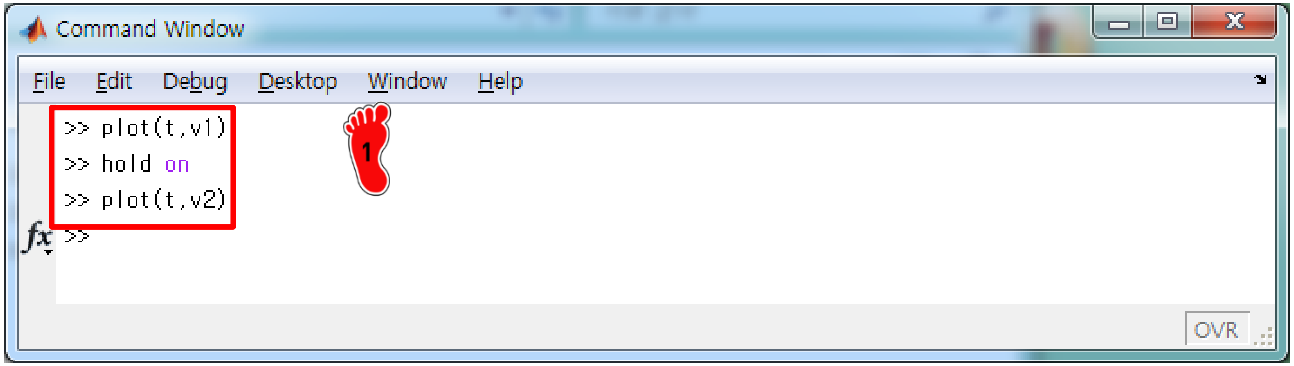
3 속도를 구하기 위한 각 상수를 저장

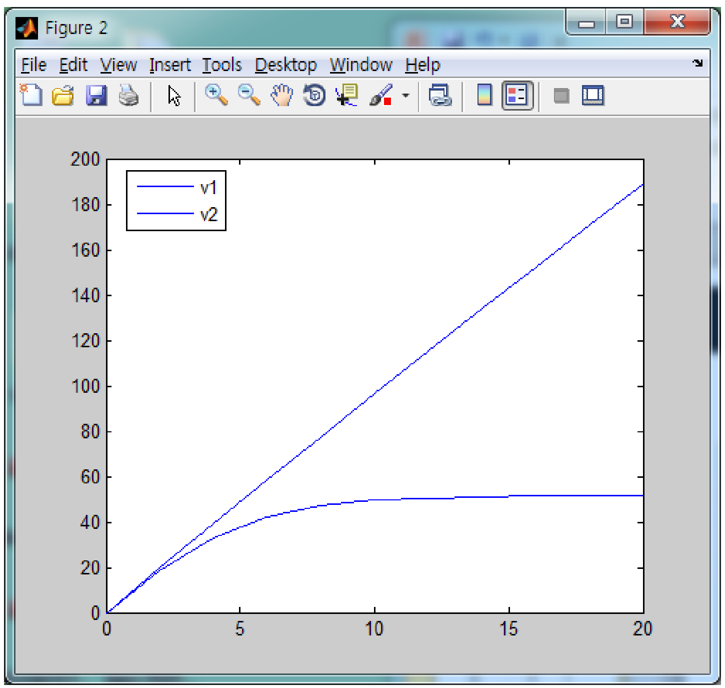
DEPENDENT VARIABLE



1 예제에서 구한 exact solution 에 맞게 case 1 은 v1 으로 case 2 는 v2 로 속도를 계산

PLOT



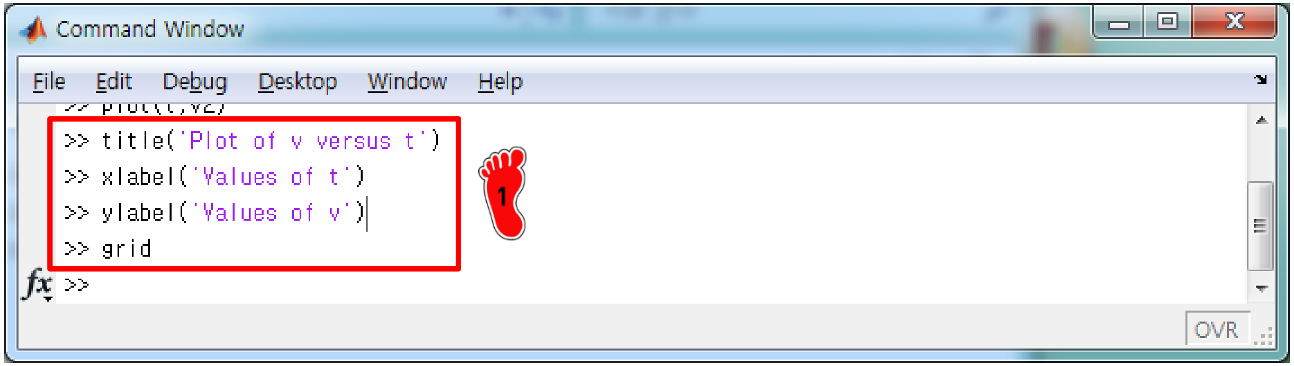


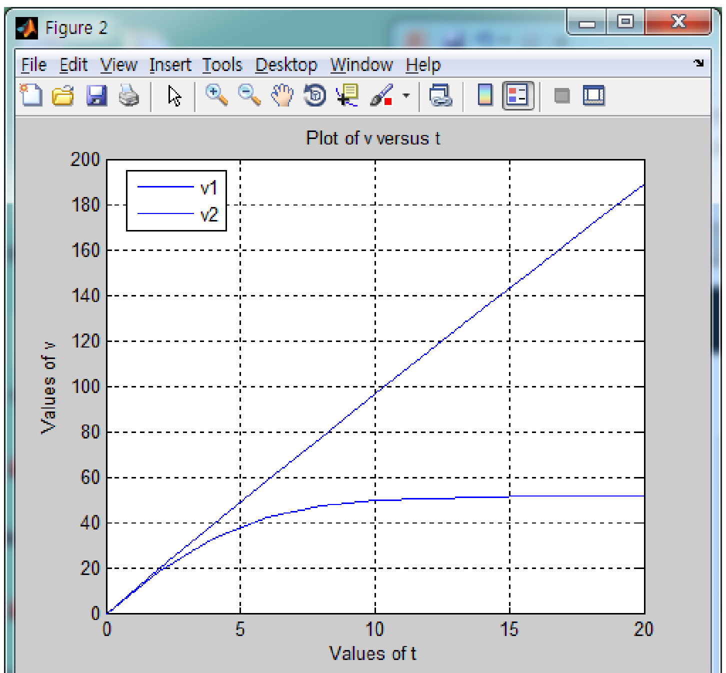
1 **plot(a,b)** 함수는 독립변수 a와 함수값 b 를 이용하여 그래프를 출력하는 함수

두 벡터 혹은 배열의 크기가 맞아야 그래프가 출력이 됨

hold on 명령어는 현재 plot 이 되어있는 figure 를 hold 시킴으로써 새로운 plot 을 그릴때 그 위에 덧대서 그림을 그릴 수 있게하는 명령어

NAMING





title: 그래프의 제목을 변경

xlabel: 그래프 가로축의 이름을 변경

ylabel: 그래프 세로축의 이름을 변경

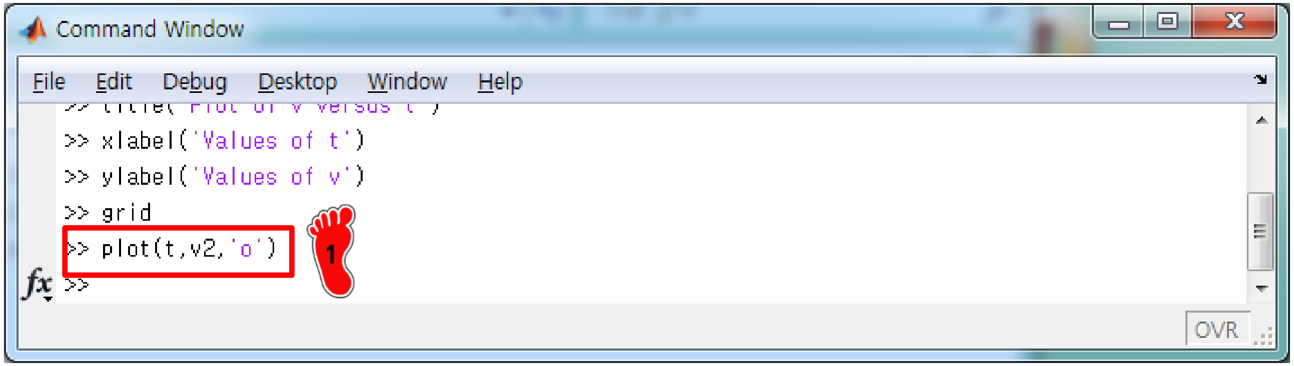
grid: 격자를 생성

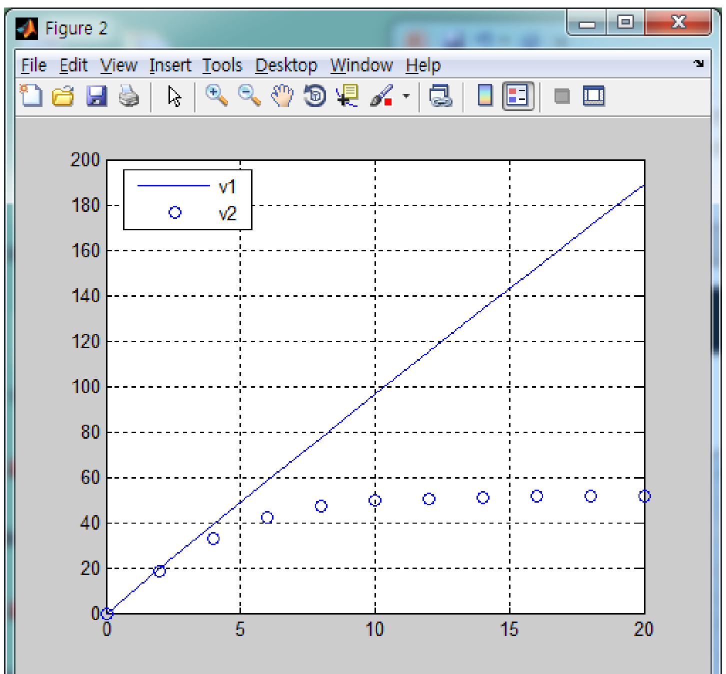
SPECIFIERS

Colors		Symbols		Line Types	
Blue	b	Point	.	Solid	-
Green	g	Cicle	o	Dotted	:
Red	r	X-mark	x	Dashdot	-.
Cyan	c	Plus	+	Dashed	--
Magenta	m	Star	*		
Yellow	y	Square	s		
Black	k	Diamond	d		
White	w	Triangle(down)	v		
		Triangle(up)	^		
		Triangle(left)	<		
		Triangle(right)	>		
		Pentagram	p		
		Hexagram	h		

그래프에 다양한 데이터가 존재하여 여러가지 형식으로 구분해야할 경우 색깔, 기호 및 선 형식에 따라 구분할 수 있음(help plot에서 plot 도움말 문서 참조)

SYMBOL





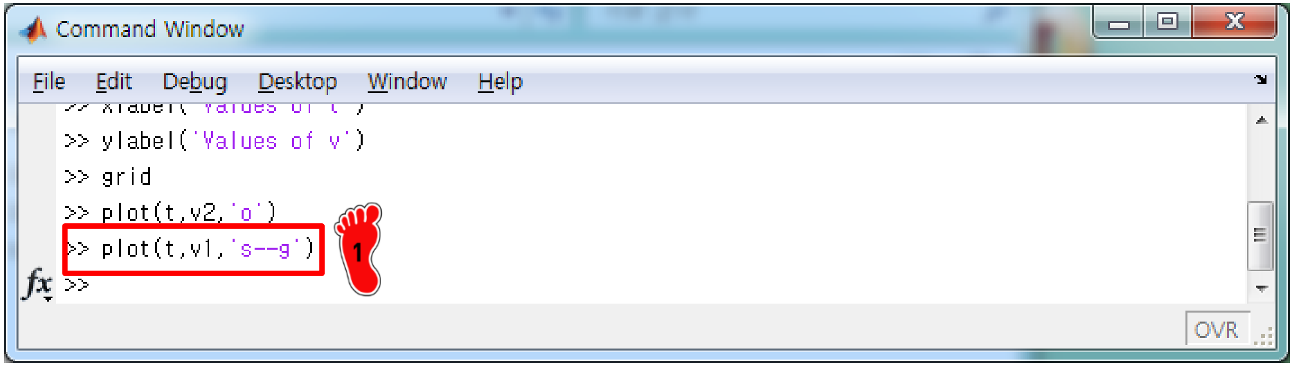
전 페이지의 명령어를 입력하는 방법은

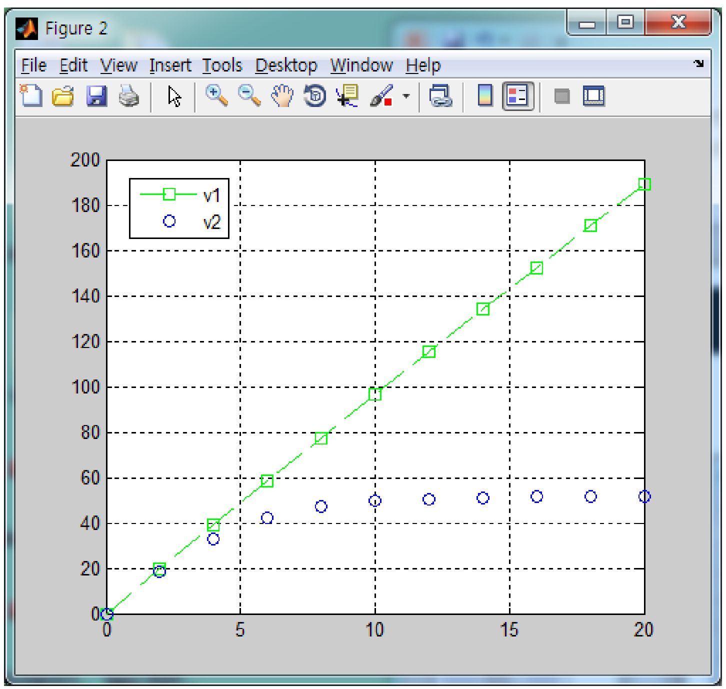
plot(x,y,'명령어')

방식으로 입력 가능

예시는 데이터를 o 표시로 그림을 출력하는 방법

COLLOR & DASHED LINE



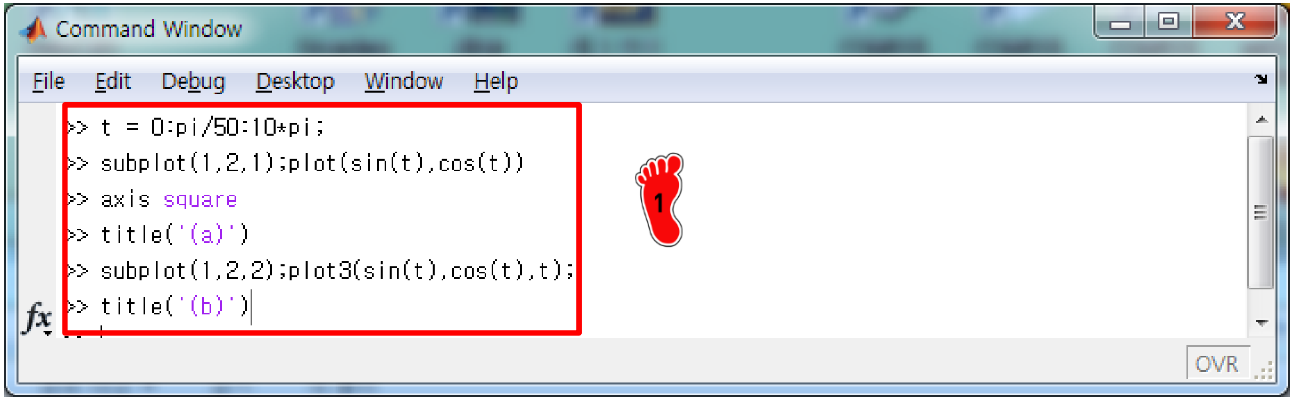


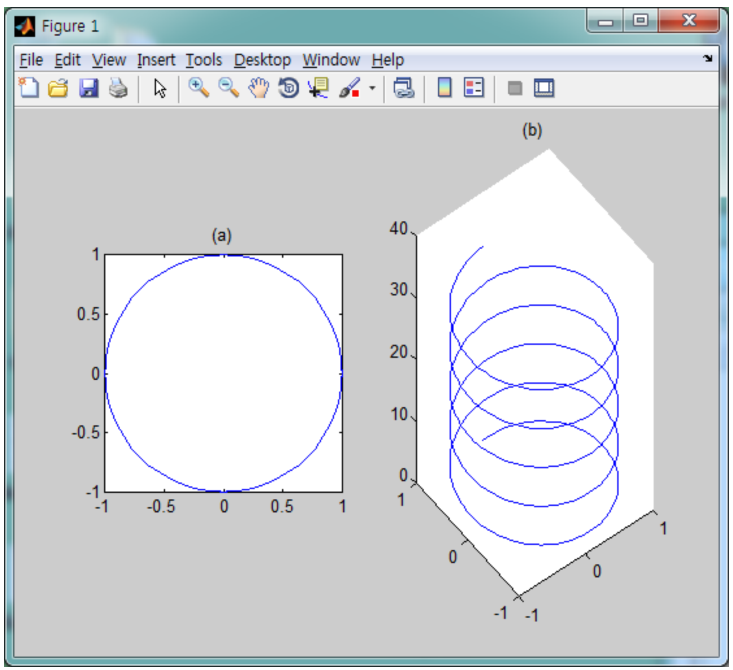
1 본 예시는

사각형 s
대쉬 –
초록색 g

명령어를 한 번에 적용하여
그래프를 출력한 경우

SUBPLOT & 3D PLOT





subplot(m,n,p): m by n 개의 그림칸을 생성, p 는 p 번째의 그림을 의미

plot3(x,y,z): 3d plot 을 의미하며 z 출은 세 번째 데이터를 의미

- **Case study**
- **Assignment**

CASE STUDY

Background. Your textbooks are filled with formulas developed in the past by renowned scientists and engineers. Although these are of great utility, engineers and scientists often must supplement these relationships by collecting and analyzing their own data. Sometimes this leads to a new formula. However, prior to arriving at a final predictive equation, we usually "play" with the data by performing calculations and developing plots. In most cases, our intent is to gain insight into the patterns and mechanisms hidden in the data.

In this case study, we will illustrate how MATLAB facilitates such exploratory data analysis. We will do this by estimating the drag coefficient of a free-falling human based on Eq. (2.1) and the data from Table 2.1. However, beyond merely computing the drag coefficient, we will use MATLAB's graphical capabilities to discern patterns in the data.

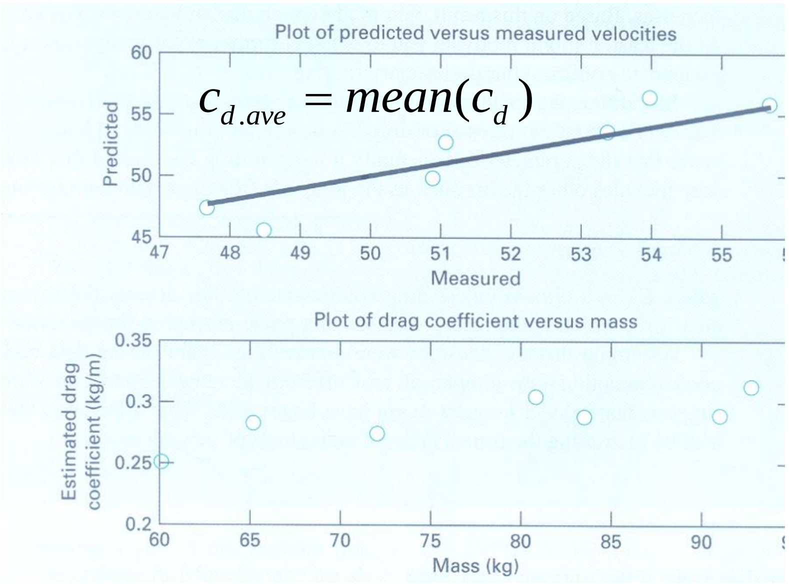
Data

TABLE 2.1 Data for the mass and associated terminal velocities of a number of jumpers.

m, kg	83.6	60.2	72.1	91.1	92.9	65.3	80.9
v_t, m/s	53.4	48.5	50.9	55.7	54	47.7	51.1

$$v_t = \sqrt{\frac{gm}{c_d}} \quad (2.1)$$

Results figures



$$c_{d.ave} = mean(c_d)$$

ASSIGNMENT

2.21 Figure P2.21*a* shows a uniform beam subject to a linearly increasing distributed load. As depicted in Fig. P2.21*b*, deflection y (m) can be computed with

$$y = \frac{w_0}{120EIL}(-x^5 + 2L^2x^3 - L^4x)$$

where E = the modulus of elasticity and I = the moment of inertia (m^4). Employ this equation and calculus to generate MATLAB plots of the following quantities versus distance along the beam:

(a) displacement (y),
(b) slope [$\theta(x) = dy/dx$],
(c) moment [$M(x) = EId^2y/dx^2$],
(d) shear [$V(x) = EId^3y/dx^3$], and
(e) loading [$w(x) = -EId^4y/dx^4$].

Use the following parameters for your computation: $L = 600$ cm, $E = 50{,}000$ kN/cm^2, $I = 30{,}000$ cm^4, $w_0 = 2.5$ kN/cm, and $\Delta x = 10$ cm. Employ the `subplot` function to display all the plots vertically on the same page in the order **(a)** to **(e)**. Include labels and use consistent MKS units when developing the plots.

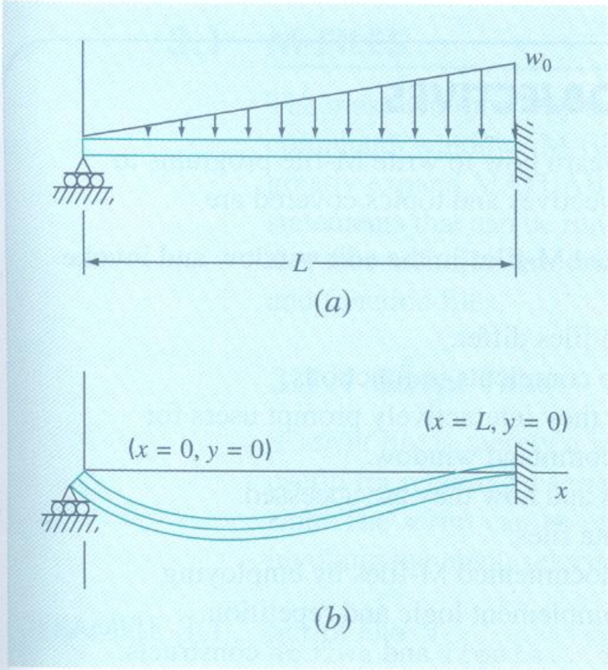


FIGURE P2.21

2.22 The *butterfly curve* is given by the following parametric equations:

$$x = \sin(t)\left(e^{\cos t} - 2\cos 4t - \sin^5\frac{t}{12}\right)$$

$$y = \cos(t)\left(e^{\cos t} - 2\cos 4t - \sin^5\frac{t}{12}\right)$$

Generate values of x and y for values of t from 0 to 100 with $\Delta t = 1/16$. Construct plots of **(a)** x and y versus t and **(b)** y versus x. Use `subplot` to stack these plots vertically and make the plot in **(b)** square. Include titles and axis labels on both plots and a legend for **(a)**. For **(a)**, employ a dotted line for y in order to distinguish it from x.

2.23 The butterfly curve from Prob. 2.22 can also be represented in polar coordinates as

$$r = e^{\sin\theta} - 2\cos(4\theta) - \sin^5\left(\frac{2\theta - \pi}{24}\right)$$

Generate values of r for values of θ from 0 to 8π with $\Delta\theta = \pi/32$. Use the MATLAB function `polar` to generate the polar plot of the butterfly curve with a dashed red line. Employ the MATLAB Help to understand how to generate the plot.

ORIGINS OF MATLAB

Origins of MATLAB, by Cleve Moler

MATLAB is now a full-featured technical computing environment, but it started as a simple "Matrix Laboratory." Three men, J.H. Wilkinson, George Forsythe, and John Todd, played important roles in the origins of MATLAB.

ORIGINS OF MATLAB

Wilkinson was a British mathematician who spent his entire carrer at the National Physical Laboratory (NPL) in Teddington, outside London. Working on a simplified version of a sophisticated design by Alan Turning, Wilkinson and colleagues at NPL bulit the Pilot Automatic Computing Engine (ACE), one of Britain`s first sotred program digital computers. THe Pilot ACE ran its first program in May 1950. Wikinson did matrix computations on the machine and went on to become the world`s leading authority on numerical linear algebra.

J. H. Wilkinson and the Pilot ACE, 1951, National Physical Laboratory, Teddington, England.

ORIGINS OF MATLAB

At about the same time, mathematicians at the Institue for Numerical Analysis (INA), a branch of the National Bureau of Standards, located at UCLA, were working with the Standards Western Automatic Computer (SWAC), one of the USA's first computers. Researchers at INA included George Forsythe, John Todd, and Olga Taussky-Todd. When the INA dissolved in 1957, Forsythe joined the faculty at Stanford and the Todds joined the faculty at Caltech

Institute for Numerical Analysis, early 1950s, UCLA. George Forsythe is in the center, and John Todd is looking over Forsythe's shoulder.

ORIGINS OF MATLAB

I went to Caltech as a freshman in 1957 and two years later took John Todd's Math 105, Numerical Analysis. We did some of our homework with mechanical calculators, but we also used the Burroughs 205 Datatron, one of only a few dozen computers in southern California at the time.

The Burroughs 205 Datatron, 1959, a vacuum tube computer with 4,000 words of magnetic drum memory. Programs for the Datatron were written in absolute numeric machine language and punched on paper tape.

ORIGINS OF MATLAB

One of the projects that I did under Todd's direction in 1960 involved Hilbert matrices. These are famous, ill-conditioned test matrices with elements

$h_{i,j} = 1/(i+j-1),\ i,j = 1,\ \ldots,\ n$

I wrote my programs in absolute numeric machine language and punched them on paper tape.

If I'd had MATLAB at the time, my project would have involved computing

```
H = single(hilb(6))
```

```
H =
        1.0000000 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667
        0.5000000 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571
        0.3333333 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000
        0.2500000 0.2000000 0.1666667 0.1428571 0.1250000 0.1111111
        0.2000000 0.1666667 0.1428571 0.1250000 0.1111111 0.1000000
        0.1666667 0.1428571 0.1250000 0.1111111 0.1000000 0.0909091
```

ORIGINS OF MATLAB

I would then compute the inverse of H.

```
inv(H)
```

```
ans =
    35.80     -624.43      3322.96     -7464.70      7455.60     -2731.09
  -624.41    14546.09    -87179.65    209060.31   -217634.53     82038.44
  3322.81   -87180.05    557732.38  -1393901.88   1493100.13   -574728.88
 -7464.46   209065.14  -1393927.13   3584568.00  -3920712.00   1533448.13
  7455.48  -217644.66   1493161.50  -3920799.75   4357413.00  -1725818.00
 -2731.10    82044.30   -574766.00   1533516.50  -1725855.38    690537.44
```

The exact inverse of the Hilbert matrix has integer elements. A function in MATLAB computes it with a recursive algorithm.

```
invhilb(6)
```

```
ans =
    36.00     -630.00      3360.00     -7560.00      7560.00     -2772.00
  -630.00    14700.00    -88200.00    211680.00   -220500.00     83160.00
  3360.00   -88200.00    564480.00  -1411200.00   1512000.00   -582120.00
 -7560.00   211680.00  -1411200.00   3628800.00  -3969000.00   1552320.00
  7560.00  -220500.00   1512000.00  -3969000.00   4410000.00  -1746360.00
 -2772.00    83160.00   -582120.00   1552320.00  -1746360.00    698544.00
```

ORIGINS OF MATLAB

I would compare these last two matrices and say the difference was the result of roundoff error introduced by the inversion process. I was wrong. I would learn a few years later from Wilkinson that the roundoff error introduced in computing H in the first place has more effect on the final answer than the error introduced by the inversion process.

In 1961, it was time for graduate school. Todd recommended that I go to Stanford and work with his friend George Forsythe. At the time, Forsythe was a professor in the math department, but he was starting the process that would create Stanford's computer science department, one of the world's first, in 1965.

ORIGINS OF MATLAB

In 1962, after Forsythe's numerical analysis course and a visit to Stanford by Wilkinson, I wrote a Fortran program to solve systems of simultaneous linear equations. Decks of punched cards for the program were distributed fairly widely at the time, including via SHARE, the IBM User's Group.

A few punched cards from a Fortran program for solving simultaneous linear equations, 15 years before the introduction of the MATLAB backslash operator.

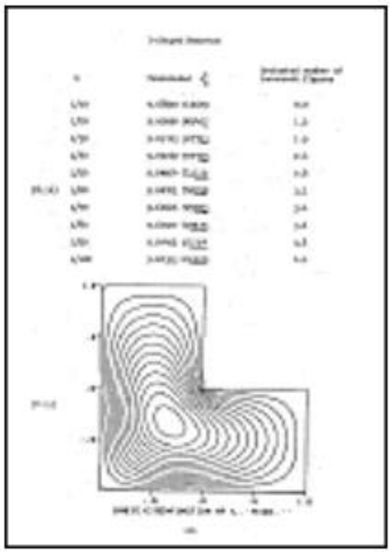
ORIGINS OF MATLAB

Alston Householder from Oak Ridge National Laboratory and the University of Tennessee began a series of research conferences on numerical algebra in the late 1950s. These are now held every three or four years and are called the Householder Conferences. As a graduate student, I went to the third conference in the series in 1964 and obtained a photo of the organizing committee. Much later, that photo was used in the first documentation of the image function in MATLAB.

The organizing committee for the 1964 Gatlinburg/Householder meeting on Numerical Algebra. All six members of the committee – J. H. Wilkinson, Wallace Givens, George Forsythe, Alston Householder, Peter Henrici, and F. L. Bauer – have influenced MATLAB.

ORIGINS OF MATLAB

My 1965 Ph.D. thesis under Forsythe's direction was entitled "Finite Difference Methods for the Eigenvalues of Laplace's Operator." The primary example, on which both Forsythe and Wilkinson had worked earlier, was the L-shaped membrane, now the MathWorks logo.



The first eigenvalue and eigenfunction of the L-shaped membrane. Click on image to see enlarged view.

This 1967 textbook contained working code in Algol, Fortran, and PL/I.

ORIGINS OF MATLAB

Forsythe and I published a textbook about matrix computation in 1967 that was later listed by the Association for Computing Machinery as an important early text in computer science because it contained working software: programs in Algol, Fortran, and PL/I for solving systems of simultaneous linear equations.

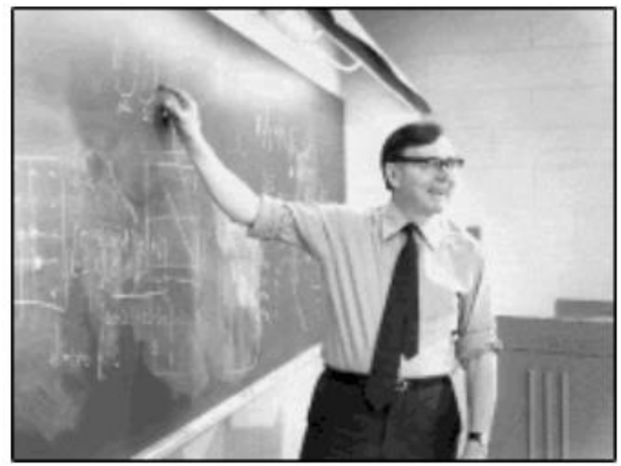
Over several years in the late 1960s, Wilkinson and a number of colleagues published papers in *Numerische Mathematik* that included algorithms in Algol for various aspects of matrix computation. These algorithms were eventually collected in a 1971 book edited by Wilkinson and Reinsch.

Handbook for
Automatic Computation

J. H. Wilkinson · C. Reinsch

Linear Algebra

Even today, more than 30 years after its publication, this collection of algorithms for matrix computation is an important reference. Click on image to see enlarged view.

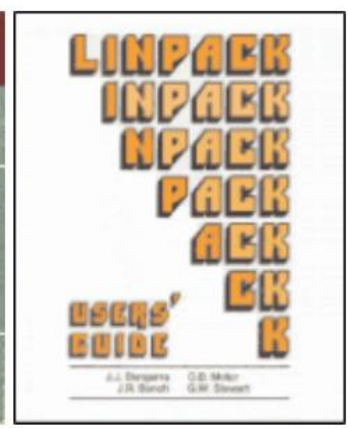


Wilkinson describing a matrix algorithm to an audience at Argonne in the early 1970s.

ORIGINS OF MATLAB

Every summer for 15 years, Wilkinson lectured in a short course at the University of Michigan and then visited Argonne National Laboratory for a week or two. Researchers at Argonne translated the Algol code for matrix eigenvalue computation from the Wilkinson and Reinsch handbook into Fortran to produce EISPACK. This was followed by LINPACK, a package of Fortran programs for solving linear equations.

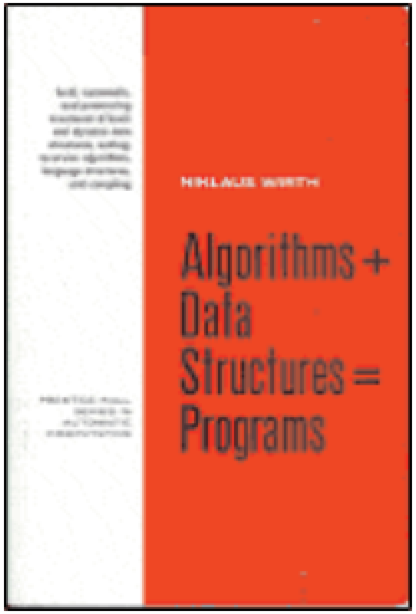
The authors of LINPACK: Jack Dongarra, Cleve Moler, Pete Stewart, and Jim Bunch in 1978.



The EISPACK manual was published in 1976 and the LINPACK manual in 1979.

ORIGINS OF MATLAB

When we were developing EISPACK and LINPACK, I was a math professor at the University of New Mexico, teaching numerical analysis and matrix theory. I wanted my students to be able to use our new packages without writing Fortran programs, so I studied a book by Niklaus Wirth to learn about parsing computer languages.



A 1975 textbook by Niklaus Wirth, who later developed PASCAL.

ORIGINS OF MATLAB

In the late 1970s, following Wirth's methodology, I used Fortran and portions of LINPACK and EISPACK to develop the first version of MATLAB. The only data type was "matrix." The HELP command listed all of the available functions, with their names abbreviated.

```
ABS   ANS   ATAN  BASE  CHAR  CHOL  CHOP  CLEA  COND  CONJ  COS
DET   DIAG  DIAR  DISP  EDIT  EIG   ELSE  END   EPS   EXEC  EXIT
EXP   EYE   FILE  FLOP  FLPS  FOR   FUN   HESS  HILB  IF    IMAG
INV   KRON  LINE  LOAD  LOG   LONG  LU    MACR  MAGI  NORM  ONES
ORTH  PINV  PLOT  POLY  PRIN  PROD  QR    RAND  RANK  RCON  RAT
REAL  RETU  RREF  ROOT  ROUN  SAVE  SCHU  SHOR  SEMI  SIN   SIZE
SQRT  STOP  SUM   SVD   TRIL  TRIU  USER  WHAT  WHIL  WHO   WHY
```

There were only 80 functions. There were no M-files or toolboxes. If you wanted to add a function, you had to modify the Fortran source code and recompile the entire program. Here is a sample program. If you change long to format long, it works with today's MATLAB.

ORIGINS OF MATLAB

The first graphics were very primitive.

```
pi = 4*atan(1);
x = 0:pi/40:2*pi;
y = x.*sin(3*x);
plot(x,y)
```

Many of the first plots were made by printing asterisks on the teletypes and typewriters that served as terminals.

This first Fortran MATLAB was portable and could be compiled to run on many of the computers that were available in the late 1970s and early 1980s. We installed it on the interactive time-sharing systems that were hosted by mainframe computers at universities and national laboratories.

ORIGINS OF MATLAB

The first “personal computer” that I used was the Tektronix 4081, which Argonne acquired in 1978. The machine was the size of a desk and consisted of a Tektronix graphics display attached to an Interdata 7/32, the first 32-bit minicomputer. There was only 64K, that’s 64 *kilobytes* of memory. But there was a Fortran compiler, and so, by using memory overlay, we were able to run MATLAB.

The Tektronix 4081, 1978. A step on the way from time-sharing to workstations and PCs.

ORIGINS OF MATLAB

I visited Stanford in 1979 and taught CS237, the graduate numerical analysis course. I had the students use MATLAB for some of the homework. Half of the students in the class were from math and computer science, and they were not impressed by my new program. It was based on Fortran, it was not a particularly powerful programming language, and it did not represent current research work in numerical analysis. The other half of the students were from engineering, and they liked MATLAB. They were studying subjects that I didn’t know anything about, such as control analysis and signal processing , and the emphasis on matrices in MATLAB proved to be very useful to them.

A few of the Stanford engineering students from my class joined two consulting companies in Palo Alto. These companies extended MATLAB to have more capability in control analysis and signal processing and, in the early 1980s, offered the resulting software as commercial products.

ORIGINS OF MATLAB

Jack Little, a Stanford- and MIT-trained control engineer, was the principal developer of one of the first commercial products based on Fortran MATLAB. When IBM announced their first PC in August, 1981, Jack quickly anticipated the possibility of using MATLAB and the PC for technical computing. He and colleague Steve Bangert reprogrammed MATLAB in C and added M-files, toolboxes, and more powerful graphics.

Jack Little, founder and CEO of The MathWorks.