

Introudction of CAE Laboratory

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



CONTENTS

- **Lecturer**
- **CAE Overview**
- **System Simulation**
- **Finite Element Analysis**

LECTURER

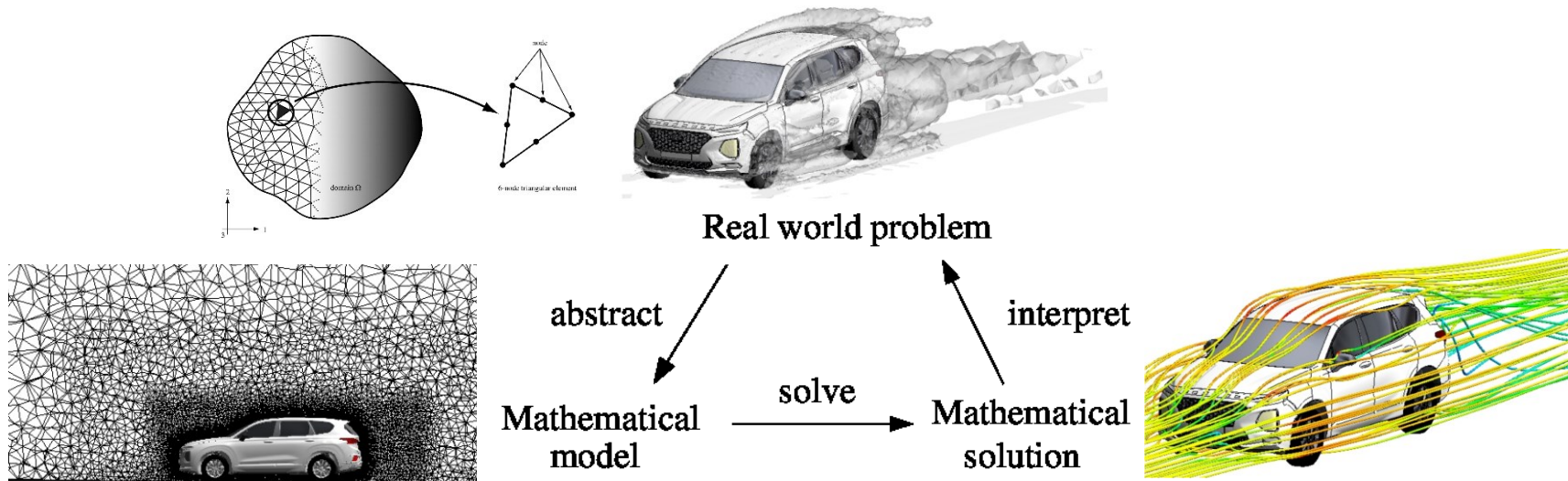
- 송원석 Teaching Fellow
- Contact

E-mail : cdlcaesws@gmail.com

Phone : 010-9052-1988

CAE OVERVIEW

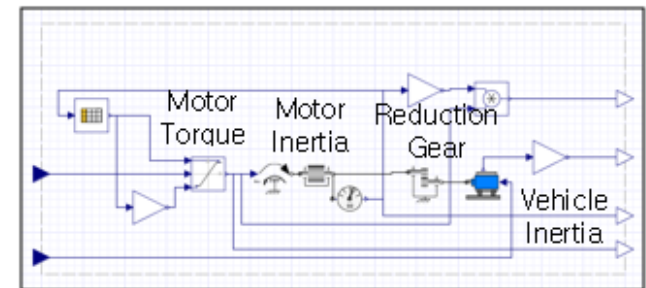
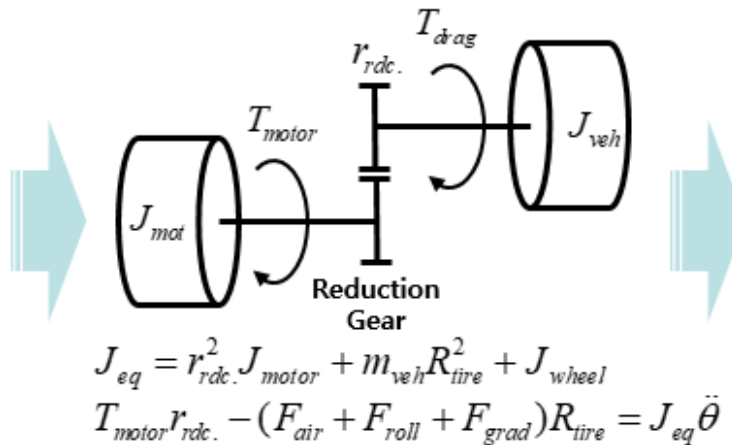
모든 물리현상은 편미분방정식(Partial Differential Equation)로 표현 가능
 CAE : 컴퓨터를 활용, 편미분방정식을 수치적인 방법으로 풀어 물리현상을 해석
 대표적으로 System Simulation과 FEA의 두 가지 방법을 통해 원하는 현상 분석



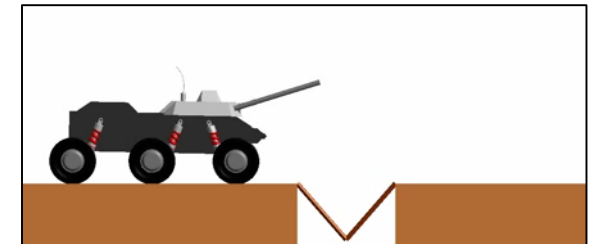
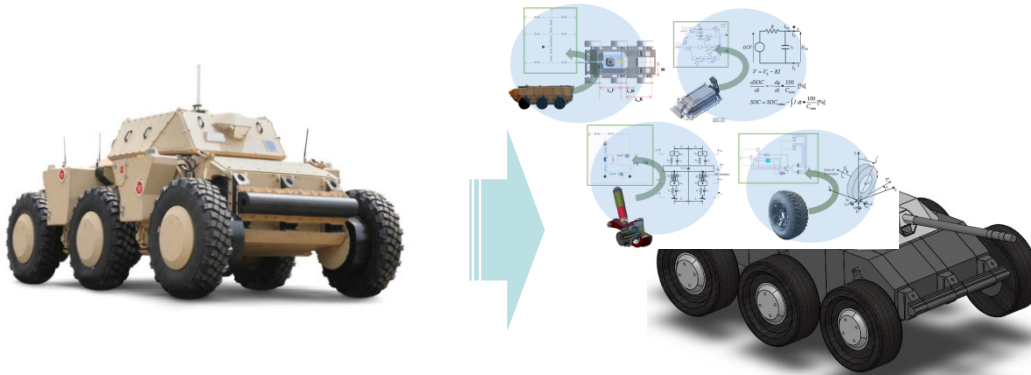
	System Simulation	Finite Element Analysis
분석 대상	System	Component
모델 종류	Function	Shape
해석 요소	System Component	Finite Element
Domain	Time	Space(+Time)
주요 분석 현상	Integrated Performance	Deformation, Endurance

SYSTEM SIMULATION

각 구성품 간의 상관관계에 의한 복잡한 시스템의 통합적인 성능 분석
 Model-Based Design 기법을 활용하여 복잡한 시스템의 직관적 모델링
 주로 차량 시스템 관점에서의 동력성능, 연비분석 등 FEA로 구현하기 힘든 부분에 활용

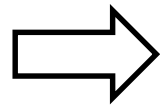


[MathWorks : MBD overview](#)

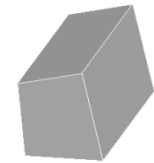
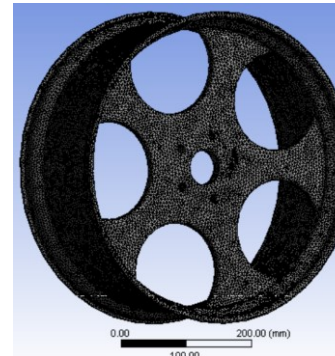


FINITE ELEMENT ANALYSIS

유한요소법(Finite Element Method)을 통해 특정 컴포넌트의 국부적 거동 예측
부품 내구 강도, 충돌 변형, 유체 흐름, 열전도 등의 세부적인 거동을 모사하기 위해 사용



Discretization

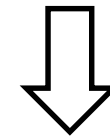


$$\{\sigma\} = [E]\{\epsilon\}$$

Continuum Governing Equation

$$\{\sigma\} = [E]\{\epsilon\} \rightarrow [K^e]\{\Delta^e\} = \{F^e\}$$

Element Equation



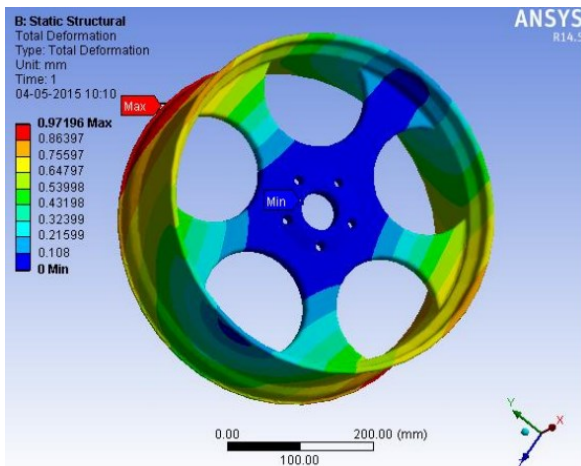
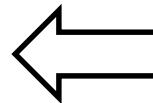
Assembly



$$[K]\{\Delta\} = \{F\}$$

System Equation

Solving



CAE PLAN

Numerical Method
for ODE/PDE

System Simulation

Finite Element Analysis

Week 주	Lecture Topics 강의주제	Lab Topics 실습주제
1	Overview	MATLAB introduction & fundamentals
2	Approximations and Errors	MATLAB programming
3	Chusuk	MATLAB: ODE
4	ODE: Runge-Kutta Methods, Boundary-Value and Eigenvalue Problems	MATLAB: PDE
5	PDE/FDM: Elliptic/Parabolic Equations	MATLAB/Simulink
6	Model Based Design: Eco-Vehicle	AMESim
7	Model Based Design: Powertrain Component	MATLAB/Simulink, AMESim
8	Model Based Design: Full Vehicle	Midterm Exam
9	Midterm Exam	COMSOL: Direct Stiffness Method, beam
10	FEM: Overview, Direct Stiffness Method	COMSOL: plane stress/strain
11	FEM Modeling	COMSOL: plate, shell, 3D
12	Plane Stress Problem	COMSOL: modal analysis, buckling
13	Multifreedom Constraints	COMSOL: heat transfer, fluid
14	Variational Formulation of Plane Bar, Plane Beam	COMSOL: electromagnetics
15	2D Isoparametric elements	Project Presentation
16	Convergence	Final Exam
17	Final Exam	

MATLAB Fundamental

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



OUTLINE


- **Lecture Goals**

- ✓ 공학분야에서 많이 사용하는 MATLAB에서 벡터/행렬 정의 및 연산, 제공 함수 활용, 가시화 방법(그래픽스) 등 기초적인 명령어를 실습한다.

- **Content**

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

OVERVIEW

- **What is MATLAB?**
 - ✓ **MATLAB = Matrix + Laboratory**
- **History**
 - ✓ **Cleve Moler, the chairman of the computer science department at the University of New Mexico, started developing MATLAB to provide easy access to matrix software in the late 1970s. Set up a Mathworks company in 1984**
- **Logo** 
 - ✓ **picture (generated by MATLAB) of a numerical approximation to the fundamental mode of a vibrating L-shaped membrane, a topic that Moler discussed in his Stanford Ph.D. thesis in 1965**

ENGINEERING MATH ON PC (1984)

Engineers—Mathematicians—Scientists

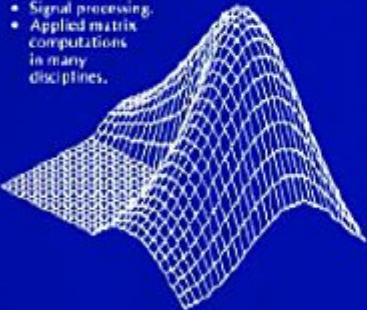
PC-MATLAB Features

PC-MATLAB

A Personal Computer becomes a powerful scientific and engineering workstation.

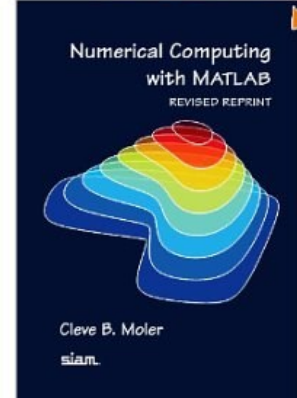
Applications:

- Numerical analysis.
- Matrix theory.
- Statistics.
- Data analysis.
- Control theory.
- Signal processing.
- Applied matrix computations in many disciplines.



PC-MATLAB turns your IBM PC or compatible into a scientific workstation. Offering performance in the range of machines costing hundreds of times more, PC-MATLAB gives you rapid, accurate numeric computation. With a widely acclaimed and unique user interface, PC-MATLAB lets you do scientific and engineering matrix calculations using natural commands. For displaying results, two and three dimensional graphics commands provide personal, automatic graph paper. PC-MATLAB is the system you have been waiting for: Quality analytical tools, precision graphics, and a command format that is absolutely the best anywhere for flexibility and power.

- **Easy to learn, fun to use.**
Since it follows standard mathematical notation for matrix operations, there is no arbitrary command syntax to be learned. An online help facility provides all the information necessary for many applications. A user's guide provides more detailed information and extensive examples. You'll feel at home in only 15 minutes.
- **A data manipulation environment.**
Basic commands provide data input, format control, real and complex arithmetic, trigonometric functions, statistical functions, sorting, manipulation of vectors and matrices, and file handling. You won't have to write Fortran code to manipulate data.
- **Powerful analytical tools.**
Analytical commands include eigenvalues, eigenvectors, matrix arithmetic, matrix inversion, linear-equation solution, least-squares, regression, determinants, singular-value decomposition, condition estimates, root-finding, fast-Fourier transforms, digital filtering, convolution, and statistics. You won't have to use those unfriendly scientific subroutine libraries.
- **Extensible and programmable.**
Most problems are solved with a quick built-in command. Other specialized problems can be solved by creating new commands in the MATLAB language itself, without resorting to time-consuming "low-level" Fortran programming. Programming commands allow FOR and WHILE loops, IF-THEN-ELSE constructs, echoing, screen control, user prompting, macros, text manipulation, and other common language structures. PC-MATLAB is a system you won't outgrow.
- **Complete engineering graphics.**
Graphics commands include linear, loglog, semilog, polar, and 3-d mesh surface plots. Grid lines, titles, and labels may be added. Manual or autoranging axes are available. Hardcopy of the graphics on the screen may be obtained on Epson compatible dot-matrix printers. Graph paper is eliminated forever.



■ Eigenvalues

$$Ax = \lambda x$$

■ Ordinary Differential Equations

$$\frac{dy(t)}{dt} = f(t, y(t))$$

■ State Space

$$\begin{aligned} dx/dt &= Ax + Bu \\ y &= Cx + Du \end{aligned}$$

■ Discrete Fourier Transform

$$X(k) = \sum_{j=1}^N x(j) \omega_N^{(j-1)(k-1)}$$

■ Digital Filter

$$Y(z) = \frac{b(1) + b(2)z^{-1} + \dots + b(nb+1)z^{-nb}}{1 + a(2)z^{-1} + \dots + a(na+1)z^{-na}} X(z)$$

RELEASE HISTORY

Version ^[22]	Release name	Number	Bundled JVM	Year	Release Date	Notes
MATLAB 1.0				1984		
MATLAB 2				1986		
MATLAB 3				1987		
MATLAB 3.5				1990		Ran on MS-DOS but required at least a 386 processor. Version 3.5m required math coprocessor
MATLAB 4				1992		
MATLAB 4.2c				1994		Ran on Windows 3.1 . Required a math coprocessor
MATLAB 5.0	Volume 8			1996	December, 1996	Unified releases across all platforms.
MATLAB 5.1	Volume 9			1997	May, 1997	
MATLAB 5.1.1	R9.1					
MATLAB 5.2	R10				March, 1998	
MATLAB 5.2.1	R10.1					
MATLAB 5.3	R11				January, 1999	
MATLAB 5.3.1	R11.1				November, 1999	
MATLAB 6.0	R12	12	1.1.8	2000	November, 2000	First release with bundled Java Virtual Machine (JVM).
MATLAB 6.1	R12.1		1.3.0	2001	June, 2001	
MATLAB 6.5	R13		1.3.1	2002	July, 2002	
MATLAB 6.5.1	R13SP1	13		2003		
MATLAB 6.5.2	R13SP2					
MATLAB 7	R14		1.4.2	2004	June, 2004	
MATLAB 7.0.1	R14SP1				October, 2004	
MATLAB 7.0.4	R14SP2	14	1.5.0	2005	March 7, 2005	
MATLAB 7.1	R14SP3		1.5.0		September 1, 2005	
MATLAB 7.2	R2006a	15	1.5.0	2006	March 1, 2006	
MATLAB 7.3	R2006b	16	1.5.0		September 1, 2006	HDF5 -based MAT-file support
MATLAB 7.4	R2007a	17	1.5.0_07	2007	March 1, 2007	
MATLAB 7.5	R2007b	18	1.6.0		September 1, 2007	Last release for Windows 2000 and PowerPC Mac . License Server support for Windows Vista ^[23]
MATLAB 7.6	R2008a	19	1.6.0	2008	March 1, 2008	
MATLAB 7.7	R2008b	20	1.6.0_04		October 9, 2008	
MATLAB 7.8	R2009a	21	1.6.0_04	2009	March 6, 2009	First release for 32-bit & 64-bit Microsoft Windows 7.
MATLAB 7.9	R2009b		1.6.0_12		September 4, 2009	First release for Intel 64-bit Mac , and last for Solaris SPARC .
MATLAB 7.9.1	R2009bSP1	22	1.6.0_12		April 1, 2010	
MATLAB 7.10	R2010a	23	1.6.0_12	2010	March 5, 2010	Last release for Intel 32-bit Mac .
MATLAB 7.11	R2010b		1.6.0_17		September 3, 2010	
MATLAB 7.11.1	R2010bSP1	24	1.6.0_17		March 17, 2011	
MATLAB 7.12	R2011a	25	1.6.0_17	2011	April 8, 2011	
MATLAB 7.13	R2011b	26	1.6.0_17		September 1, 2011	
MATLAB 7.14	R2012a	27	1.6.0_17	2012	March 1, 2012	
MATLAB 8	R2012b	28			September 11, 2012	First release with Ribbon interface.

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

<https://kr.mathworks.com/>

MathWorks

제품 솔루션 아카데미아 지원 커뮤니티 이벤트

MATLAB 받기

MathWorks.com내 검색

COVID-19 R&D

MATLAB을 활용한 AI

AI 모델 및 AI 시스템 설계

머신러닝 딥러닝 강화 학습

MATLAB EXPO 2021 KOREA

세션 다시보기

MATLAB® SIMULINK® R2021a

HOMEPAGE

활용 분야



자율주행 시스템

자율주행 시스템을 설계, 시뮬레이션 및 테스트할 수 있습니다



계산 생물학

생물학 데이터와 시스템의 분석, 시각화, 모델링



제어 시스템

제어 시스템의 설계, 테스트, 구현



데이터 과학

데이터 탐색, 머신러닝 모델 구축, 예측 분석 수행



데이터 마이닝

심층 신경망에 사용할 수 있는 데이터 준비, 설계, 시뮬레이션 및 배포



임베디드 시스템

임베디드 시스템 설계, 코딩 및 검증



엔터프라이즈 및 IT 시스템

IT 시스템과 함께 작동하는 MATLAB



FPGA, ASIC 및 SoC 개발

알고리즘 개발부터 하드웨어 설계 및 검증에 이르기까지의 워크플로 자동화



영상 처리 및 컴퓨터 비전

알고리즘 개발과 시스템 설계를 위한 영상 및 비디오의 수집, 처리 및 분석



사물 인터넷

임베디드 기기를 인터넷에 연결하여 데이터로부터 정보 확보



머신 러닝

모델을 학습시키고 파라미터를 조정하며 생산 시스템 또는 애저 기기에 배포



메카트로닉스

메카트로닉 시스템의 설계, 최적화 및 검증



혼합 신호 시스템

아날로그 및 혼합 신호 시스템의 분석, 설계 및 검증



전력 전자 제어 설계

모터 전력 변환기 및 배터리 시스템용 디지털 제어 설계 및 구현



전력 시스템 분석 및 설계

전력망 및 수송 시스템의 설계 및 시뮬레이션



예측 정비

상태 감시 및 예측 정비 소프트웨어 개발 및 배포



로봇공학

로봇공학 관련 아이디어 및 개념을 실제 환경에서 원활하게 작동하는 자율 시스템으로 변환



신호 처리

신호 및 시계열 데이터 분석, 신호 처리 시스템 모델링, 설계 및 시뮬레이션



테스트 및 측정

데이터 수집, 분석, 탐색 및 테스트 자동화



무선 통신

무선 통신 시스템 제작, 설계, 테스트 및 검증

HOMEPAGE

산업 분야



항공우주 및 국방

안전 필수 및 임무 필수 시스템의 설계, 시뮬레이션, 테스트 및 배포

- 우주 시스템



자동차

미래의 모빌리티 설계, 시뮬레이션, 배포



생명공학 및 제약

신약 개발 워크플로 전체에 대한 PK/PD 모델링 및 분석 수행



통신

통신 시스템 설계 및 시뮬레이션



전자

전자 시스템과 기기 개발, 시뮬레이션 및 테스트



에너지 생산

모델 개발 및 구현, 빅 데이터 분석 및 공정 자동화

- 유틸리티 및 에너지
- 화학 및 석유 화학
- 석유 및 가스



산업 자동화 및 기계

산업 및 에너지 관련 장비에 사용할 임베디드 제어 및 신호 처리 응용 프로그램 개발

- 발전 및 송전 장비
- 기계 제조
- 빌딩 자동화
- 전기 구동 및 자동화 구성요소



의료 기기

규제 준수 과정을 가속화함과 동시에 차세대 의료 기기 설계, 시뮬레이션 및 개발

- 의료 기기 개발
- FDA 소프트웨어 검증



금속, 자재 및 광업

센서 데이터 분석, 제어 전략 구현 및 예측 정비 시스템 생성



정량적 금융 및 위험 관리

데이터 가져오기, 알고리즘 개발, 코드 디버깅, 처리 성능 확장 등

- 금융 분야의 머신 러닝
- 모델 위험 관리
- 중앙 은행



철도 시스템

철도 응용 프로그램 모델링, 시뮬레이션 및 최적화



반도체

아날로그, 디지털 및 혼합 신호 장치 설계



소프트웨어 및 인터넷

소프트웨어 및 인터넷 시스템에 대한 데이터 탐색 및 분석, 알고리즘 개발 및 응용 프로그램 배포

학문 분야



생명 과학

생물학적 시스템 모델링, 시뮬레이션 및 분석



신경 과학

데이터 처리 및 분석, 실험 수행 및 뇌 회로 모델 시뮬레이션



물리학

실험 제어, 데이터 수집 및 분석, 시뮬레이션과 비교

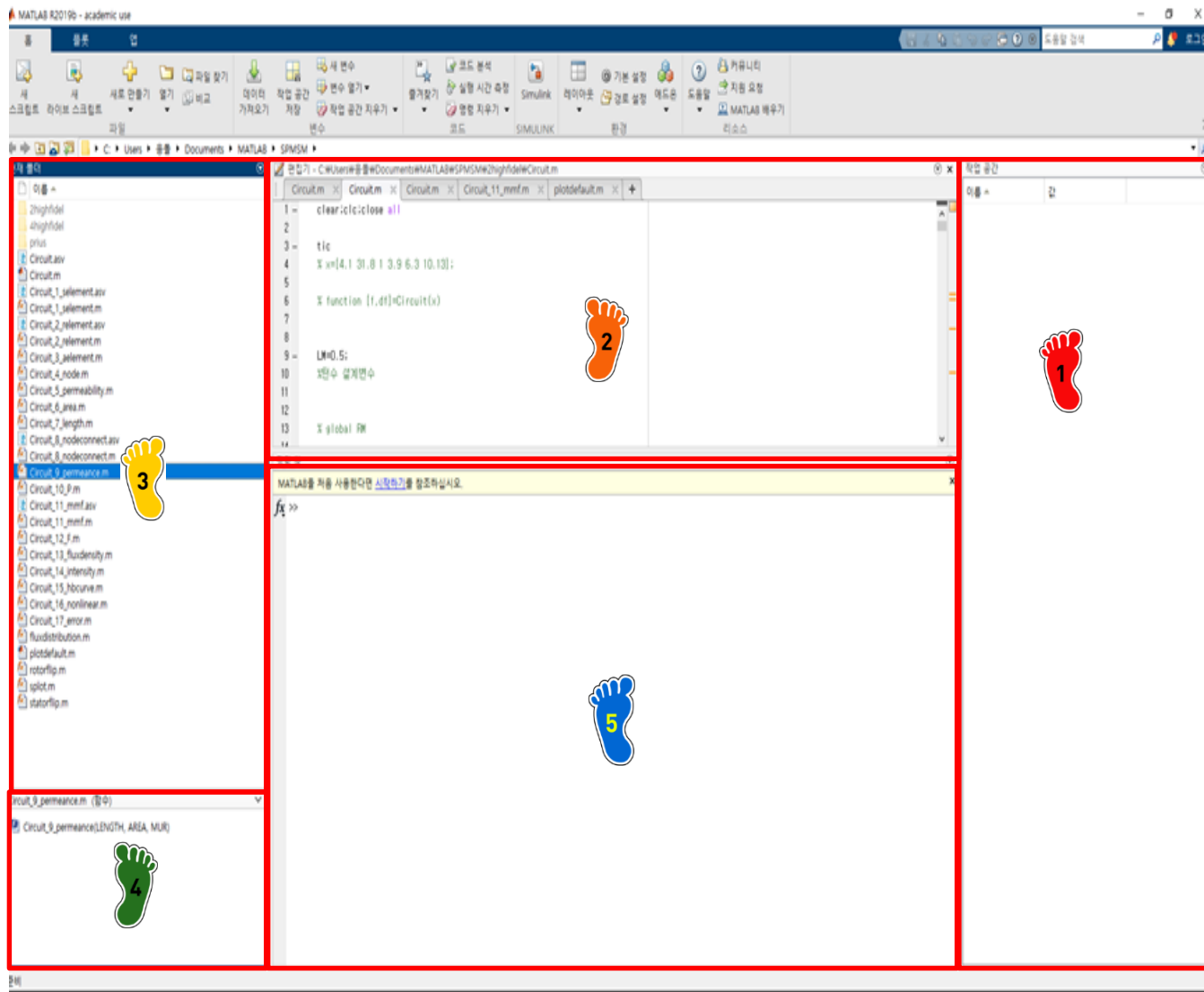


지구, 해양 및 대기 과학

복잡한 지질 동향 분석 및 이해

- **The MATLAB Environment**
 - ✓ **Interface**
 - ✓ **Command Window / Workspace**
 - ✓ **Scalars**
 - ✓ **Arrays, Vectors and Matrices**
 - ✓ **The Colon Operator**
 - ✓ **Character Strings**

INTERFACE



1 Workspace: 저장된 데이터를 보여주는 창

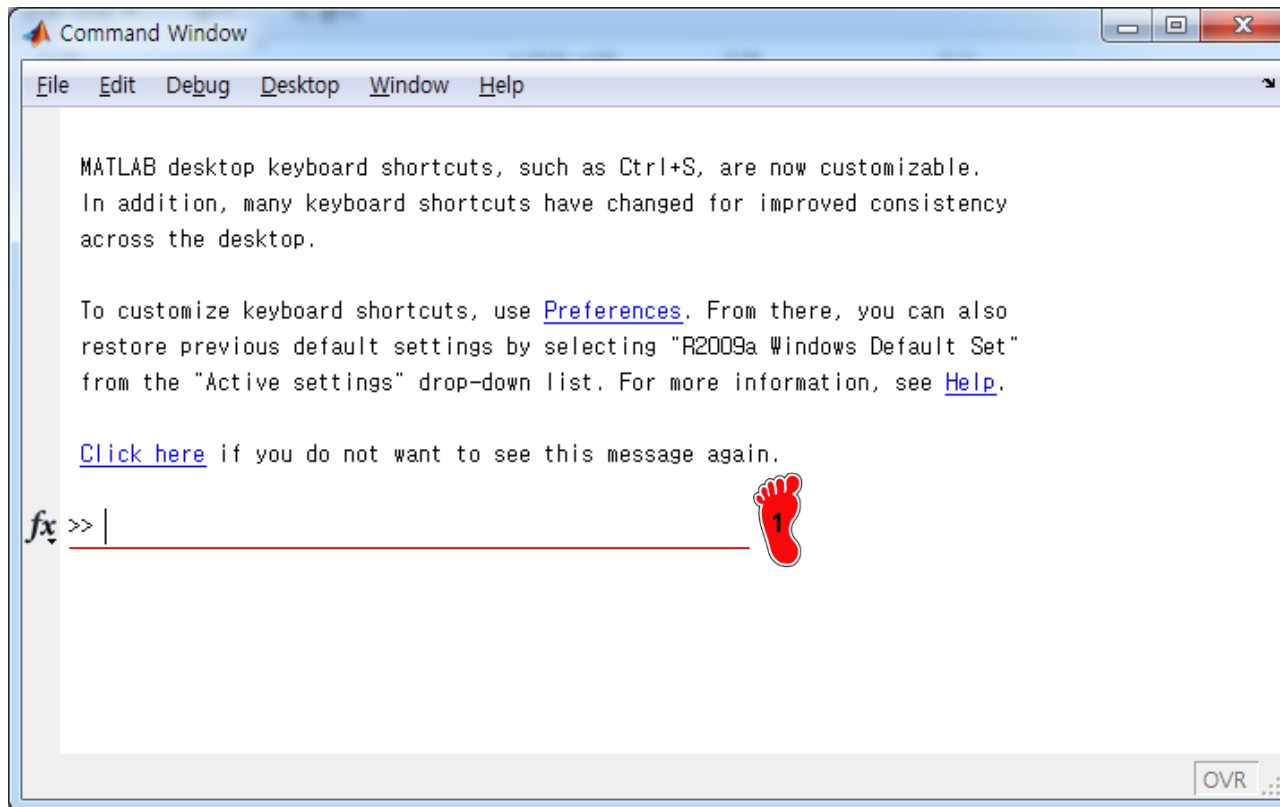
2 Editor: m-file 프로그래밍

3 Current folder: MATLAB 경로로 지정된 폴더

4 Details : 선택된 파일의 상세 정보

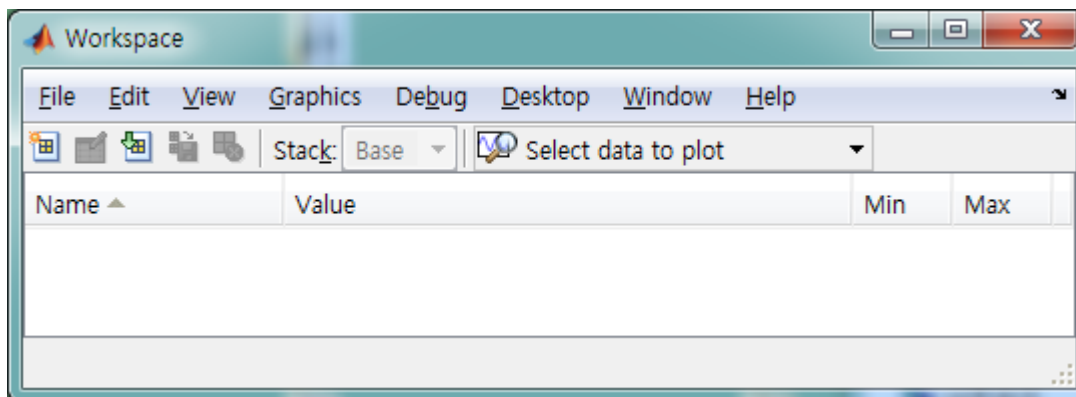
5 Command Window: 매틀랩 명령어를 입력하는 곳

COMMAND/WORKSPACE

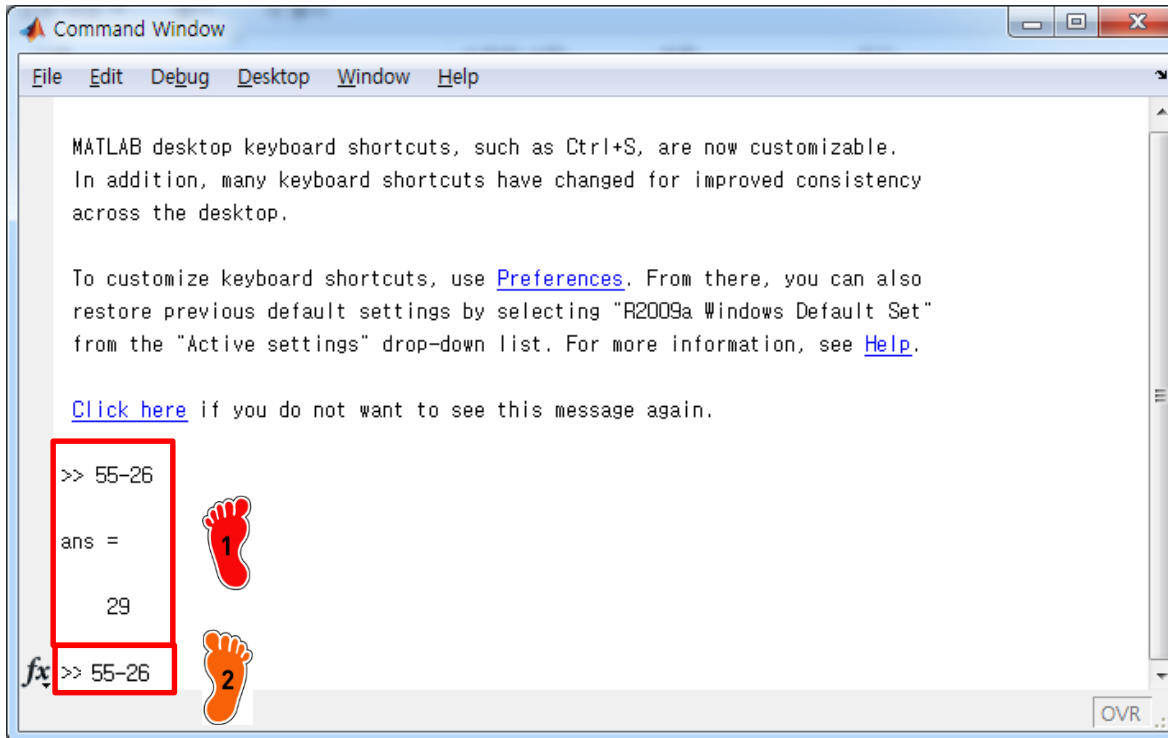


1 매틀랩 명령어를 입력하는 command line

>>



COMMAND/WORKSPACE

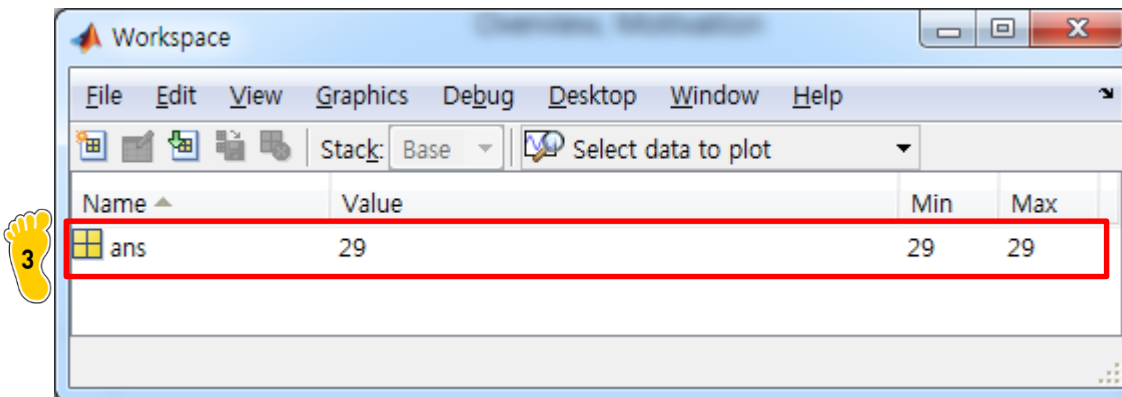


1 55 - 26 엔터를 입력하면
ans = 29 라는 결과를 확인

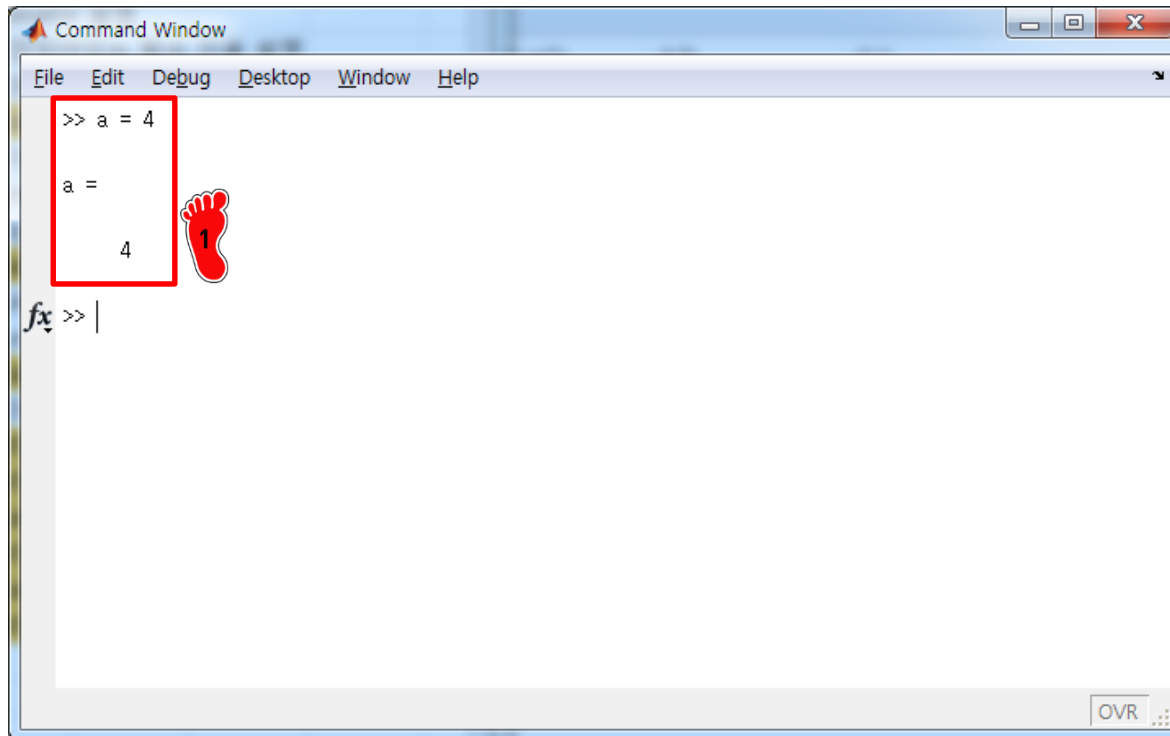
ans 는 answer 의 약자

2 방향키 ↑를 누르면 이전에
입력한 명령어를 불러오기
할 수 있음

3 workspace 에 ans 라는 변
수가 저장됨

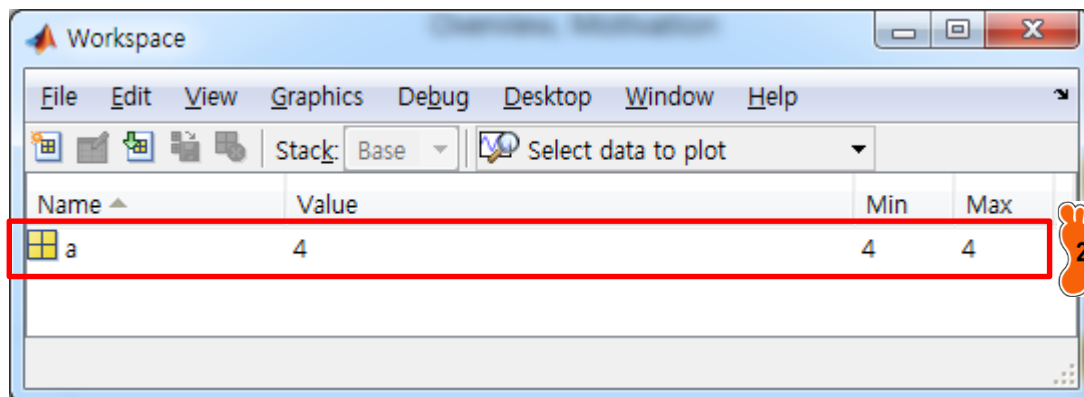


SCALARS

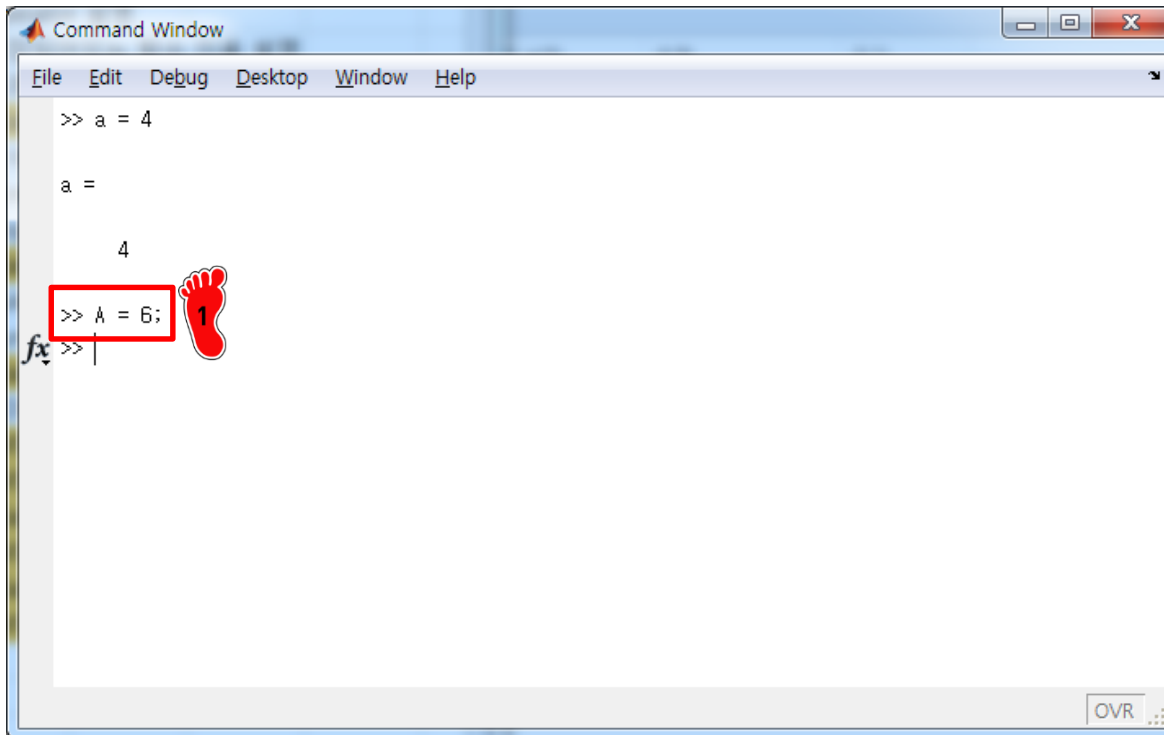


1 a = 4 Enter를 입력하면

2 workspace에 상수 a 가 저장됨

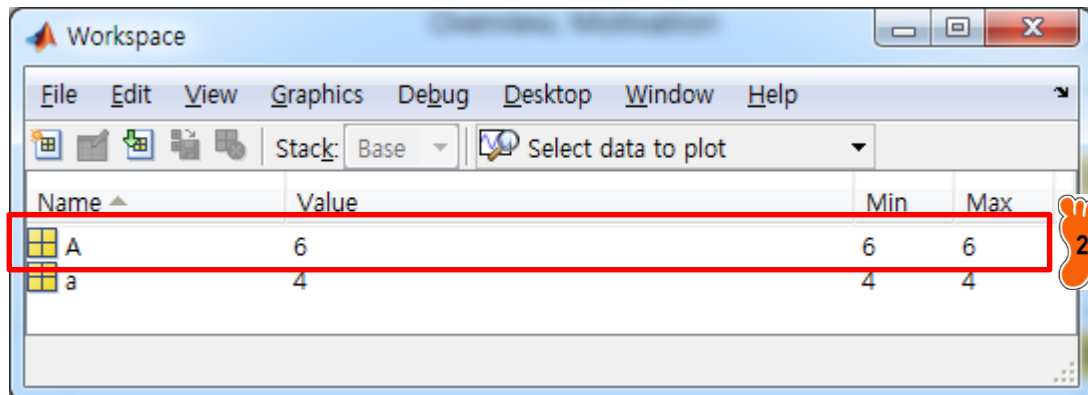


SCALARS

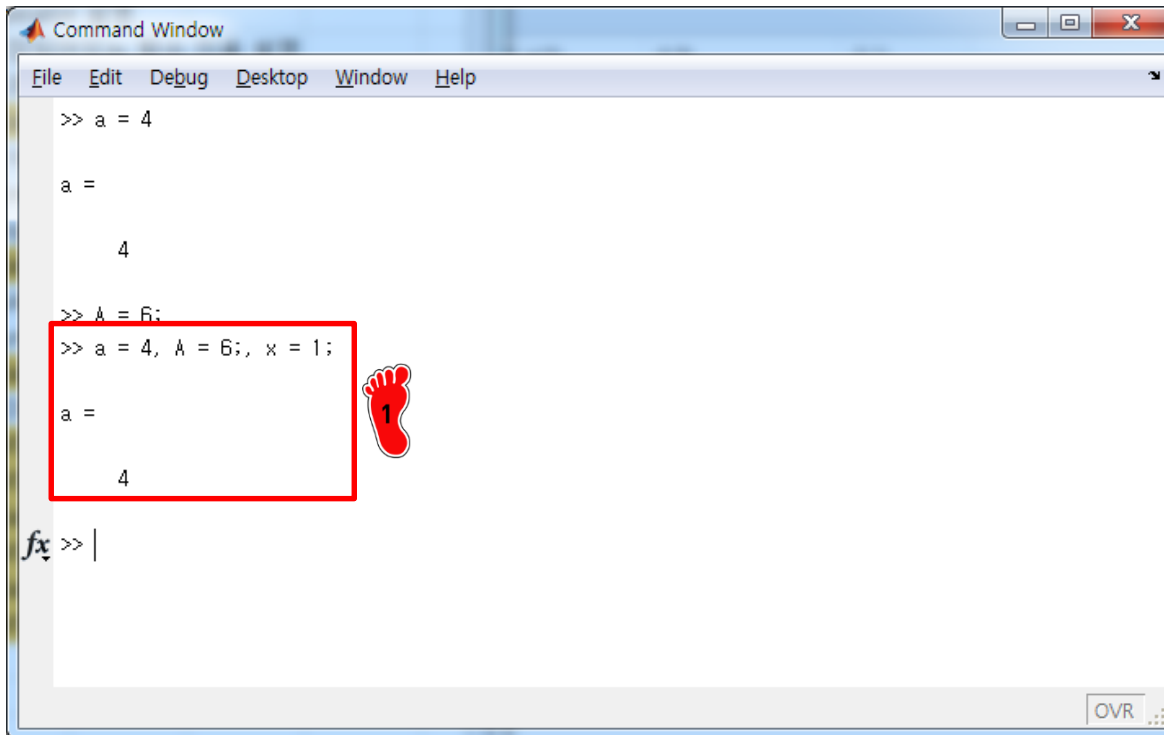


1 상수를 입력할 때 뒤에 세미콜론(;) 을 붙이면 결과값이 창에 뜨지 않음

2 하지만 workspace 에 상수 A 가 저장됨



SCALARS



```

>> a = 4

a =

    4

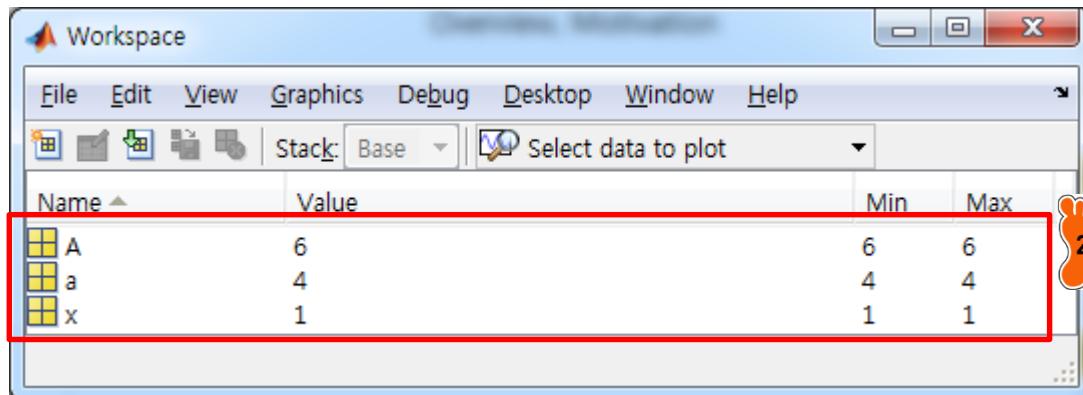
>> A = 6;
>> a = 4, A = 6; x = 1;

a =

    4
  
```

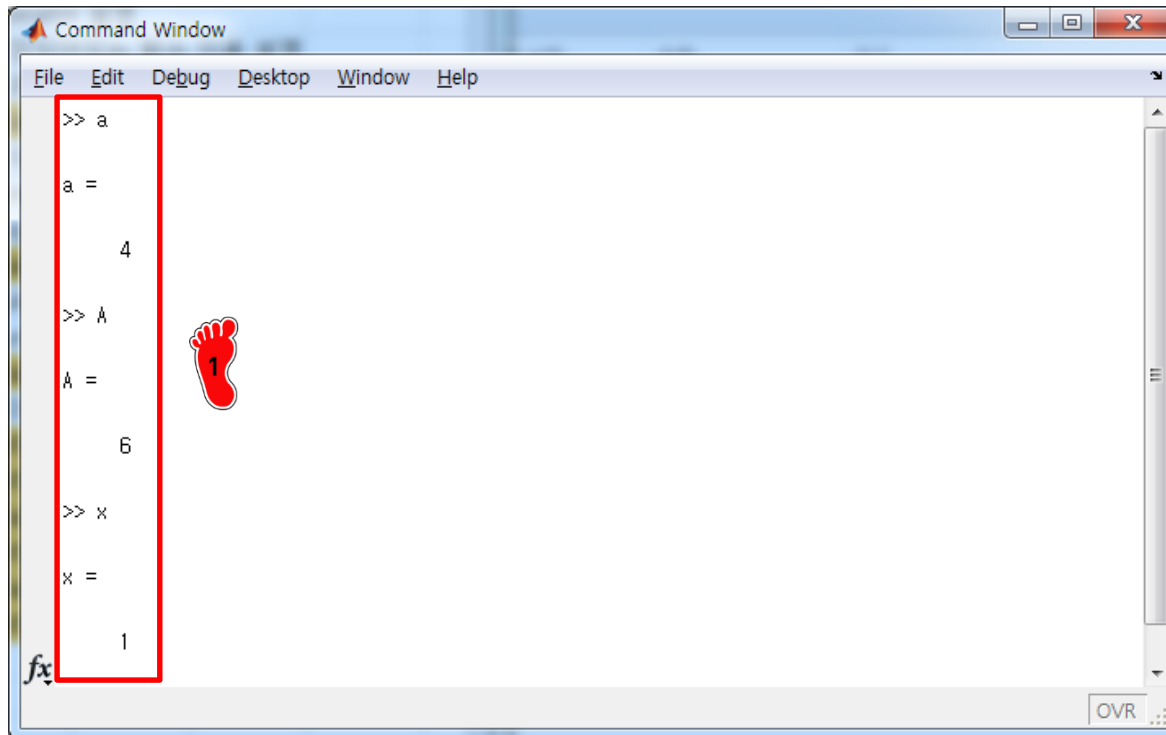
1 심표(,) 를 이용하여 여러가지 상수를 한꺼번에 입력할 수 있음

2 workspace에 저장된 값 확인

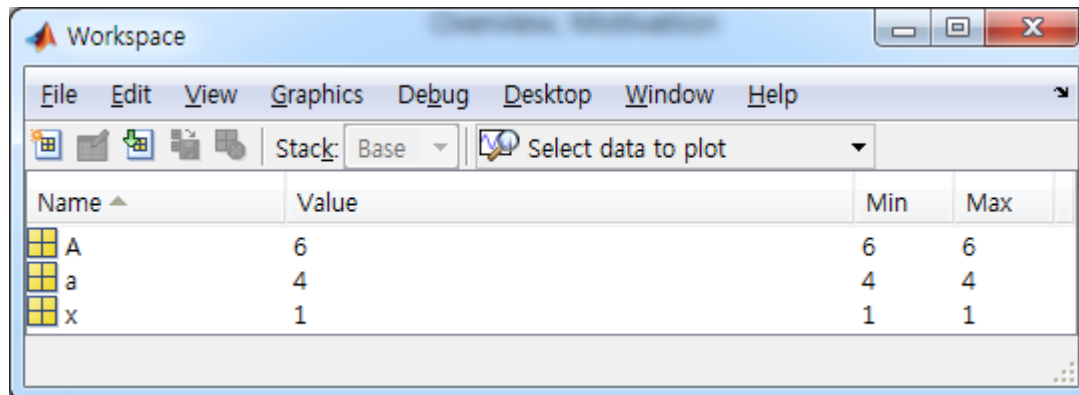


Name	Value	Min	Max
A	6	6	6
a	4	4	4
x	1	1	1

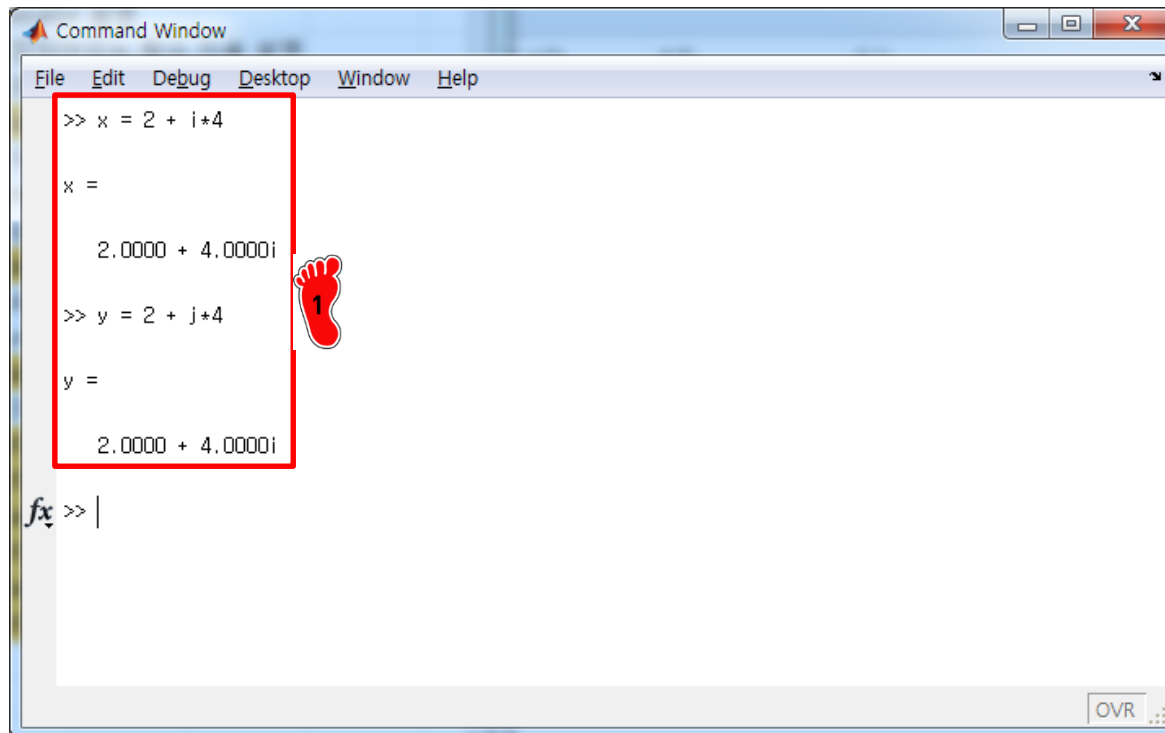
SCALARS



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



COMPLEX VALUE



Command Window

```
>> x = 2 + i*4
x =
    2.0000 + 4.0000i
>> y = 2 + j*4
y =
    2.0000 + 4.0000i
```

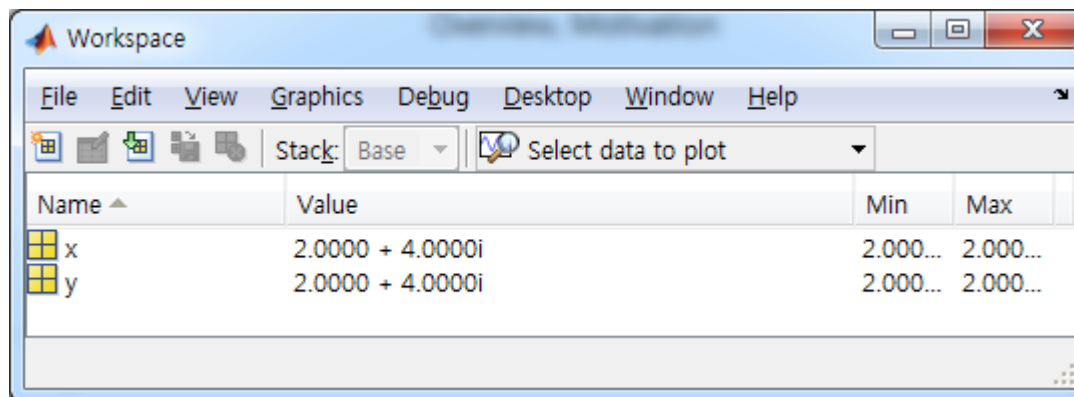
fx >> |

A red box highlights the input and output for both assignments. A red footprint icon with the number '1' is placed next to the output of the second assignment.



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

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



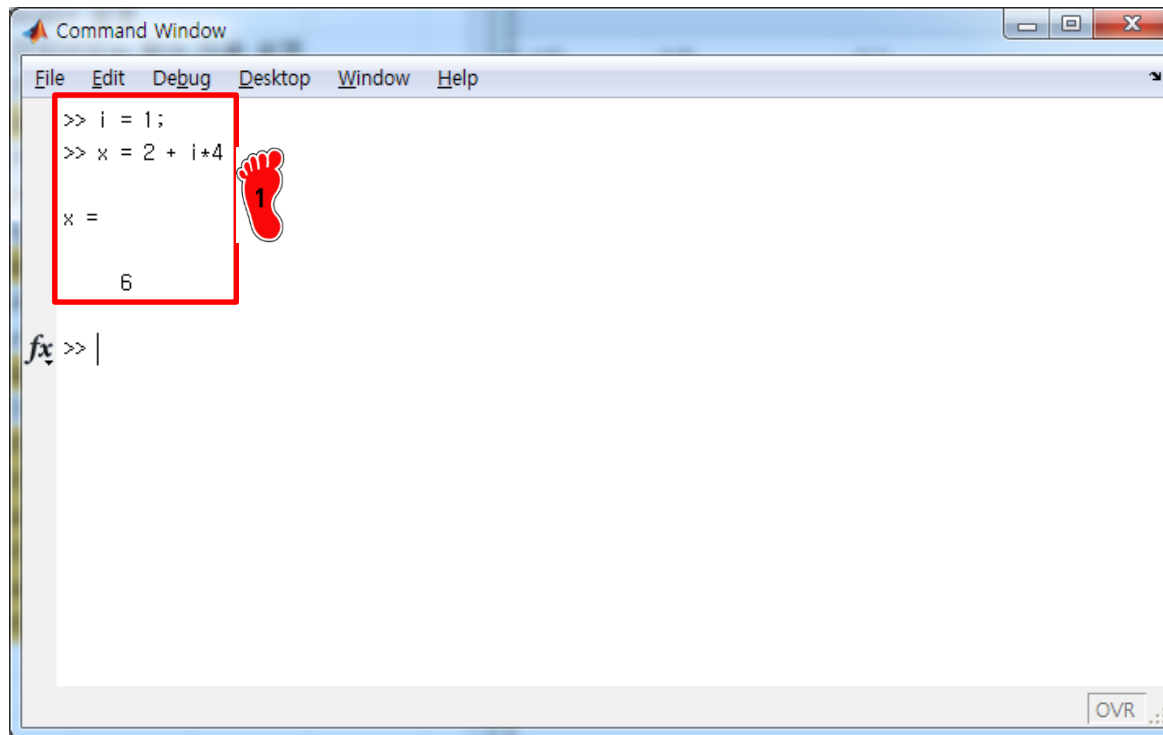
Workspace


File Edit View Graphics Debug Desktop Window Help

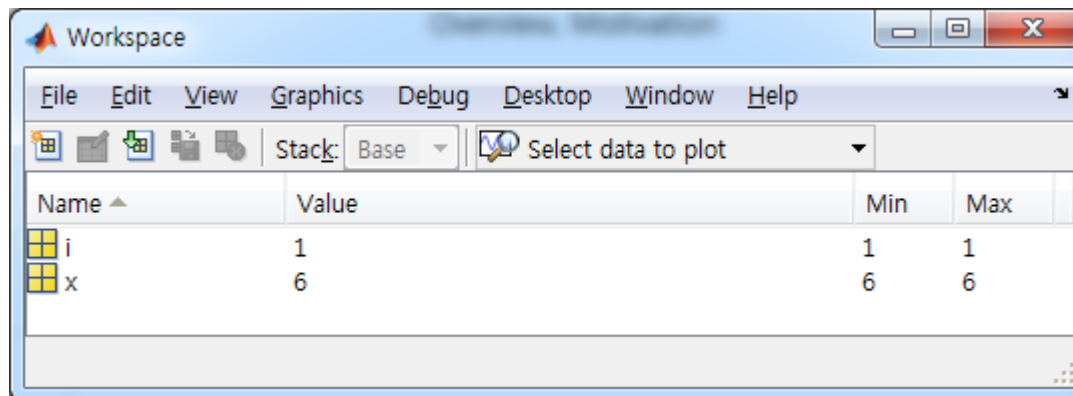
Stack: Base Select data to plot

Name	Value	Min	Max
x	2.0000 + 4.0000i	2.000...	2.000...
y	2.0000 + 4.0000i	2.000...	2.000...

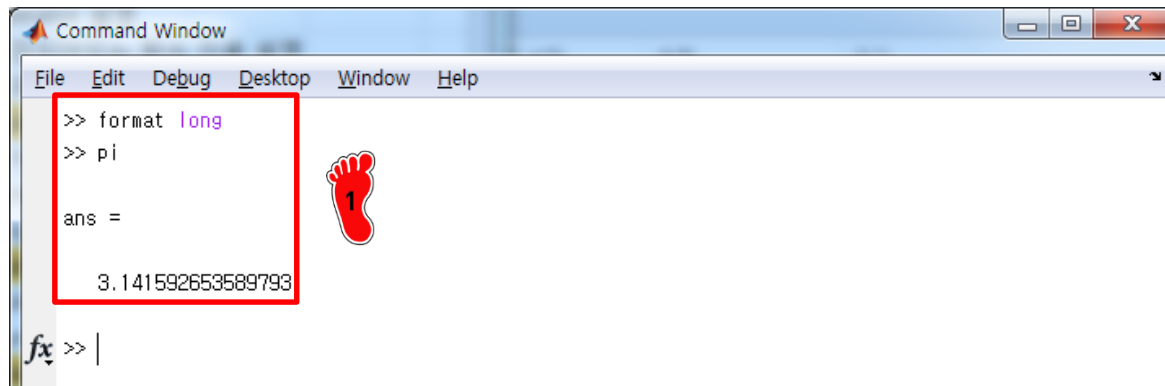
COMPLEX VALUE



 i 가 선언이 될 경우 허수로 인식하지 않음



FORMAT



```

Command Window
File Edit Debug Desktop Window Help
>> format long
>> pi

ans =

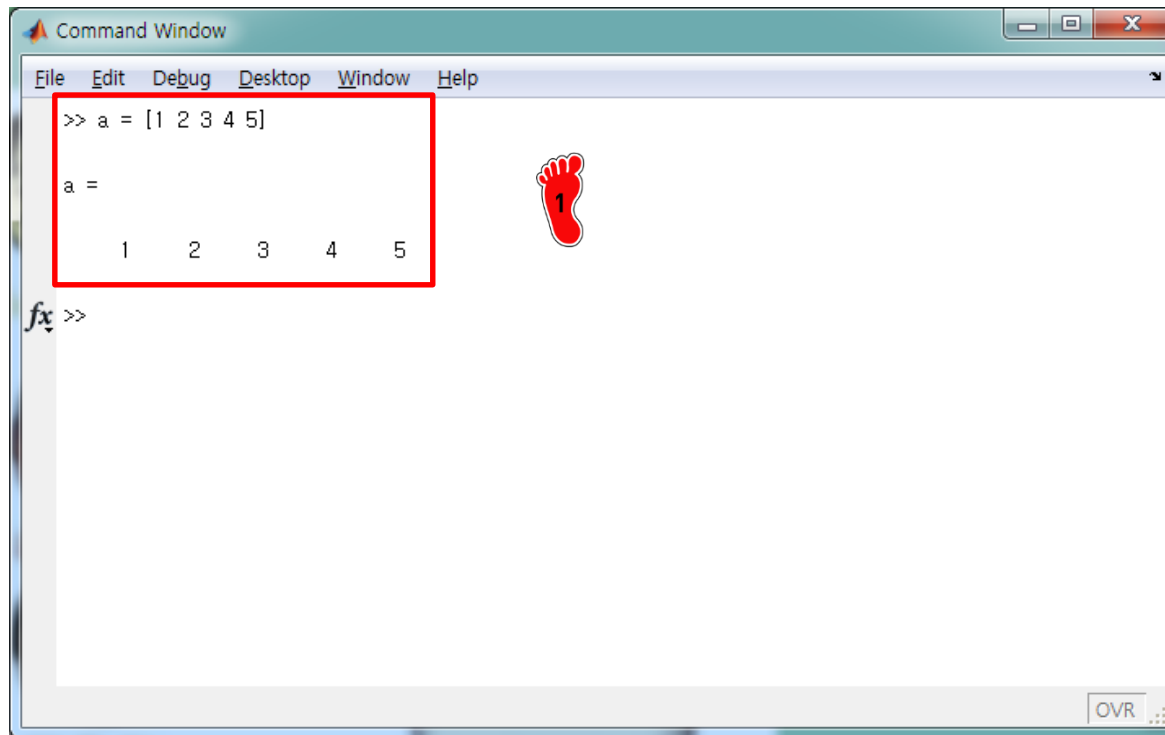
    3.141592653589793
  
```



format 명령어를 이용하여 출력되는 자릿수와 형식을 결정할 수 있음

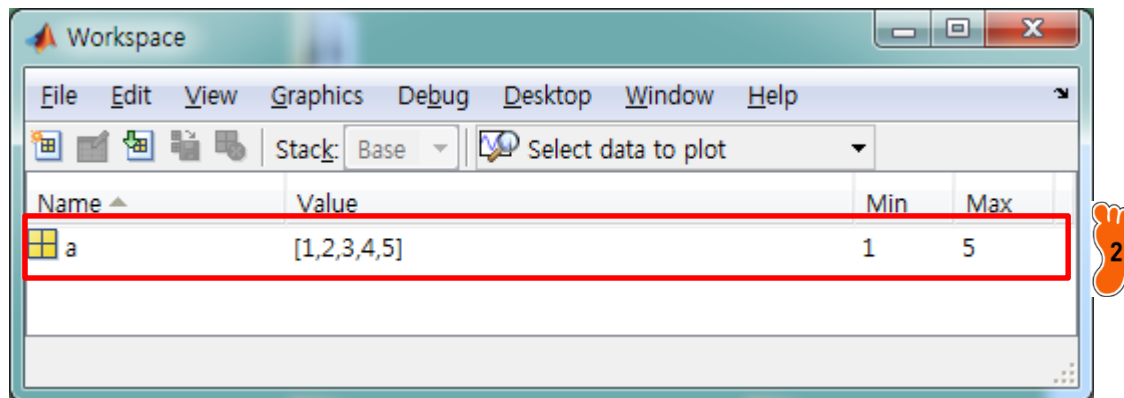
Type	Result	Example
short	Scaled fixed-point format with 5 digit	3.1416
long	Scaled fixed-point with 15 digits for double and 7 digits for single	3.14159265358979
short e	Floating-point format with 5 digits	3.1416e+000
long e	Floating-point format with 15 digits for double and 7 digits for single	3.141592653589793e+000
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	Engineering 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

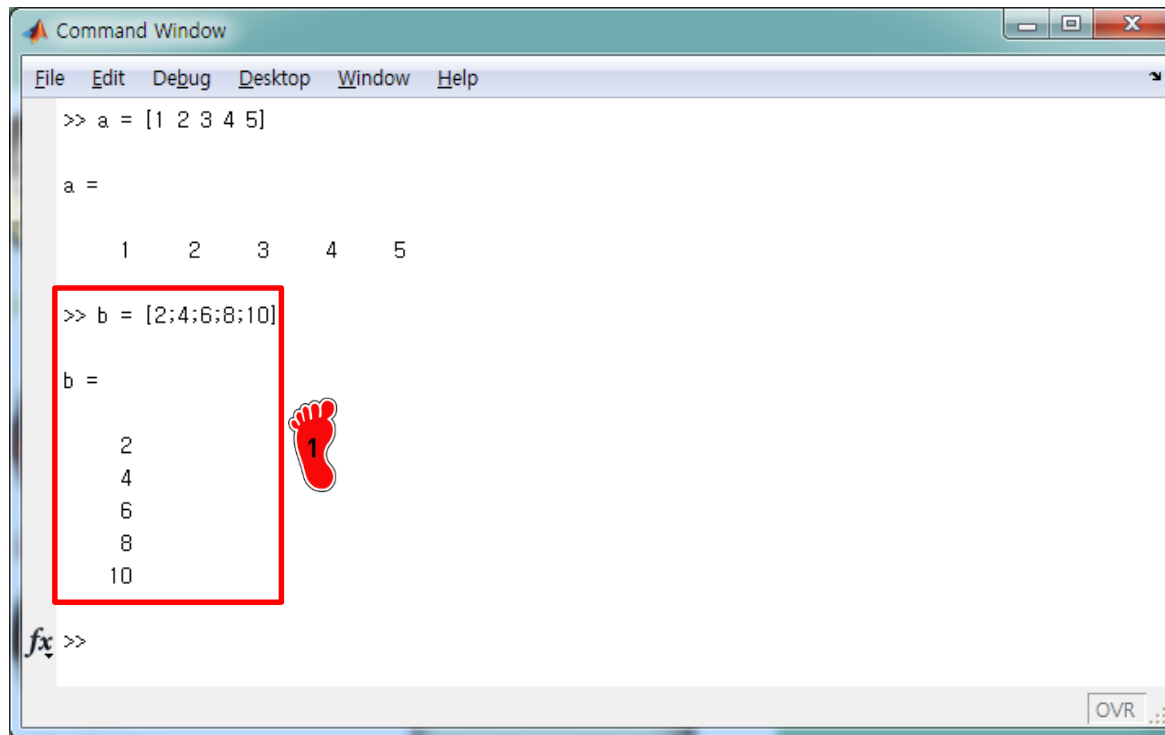


1 괄호 기호 [] 를 이용하여
행 벡터를 저장할 수 있음

2 저장된 데이터를 확인하면
상수와는 다르게 여러개의
데이터를 저장하고 있기 때
문에 min, max 값을 표기



COLUMN VECTOR



```

>> a = [1 2 3 4 5]

a =

     1     2     3     4     5

>> b = [2;4;6;8;10]

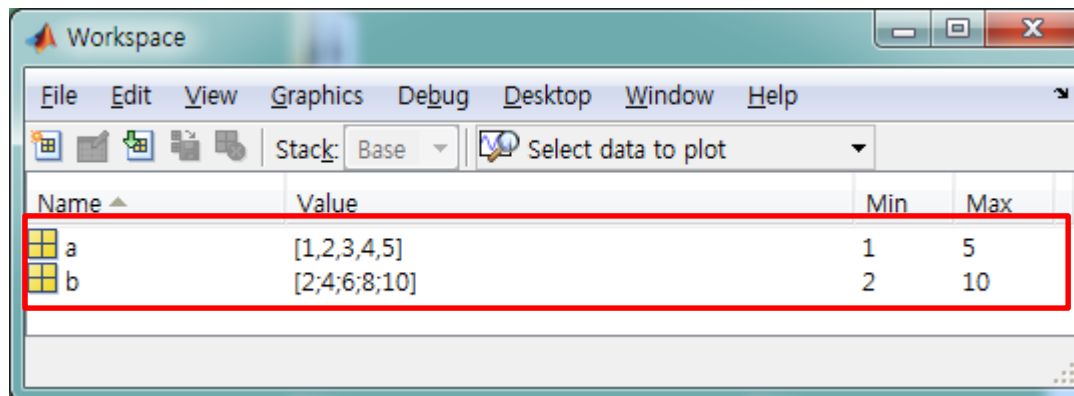
b =

     2
     4
     6
     8
    10
  
```

The Command Window shows the creation of a column vector `b` using semicolons. A red box highlights the command `b = [2;4;6;8;10]` and its output, with a red footprint icon pointing to the semicolons.



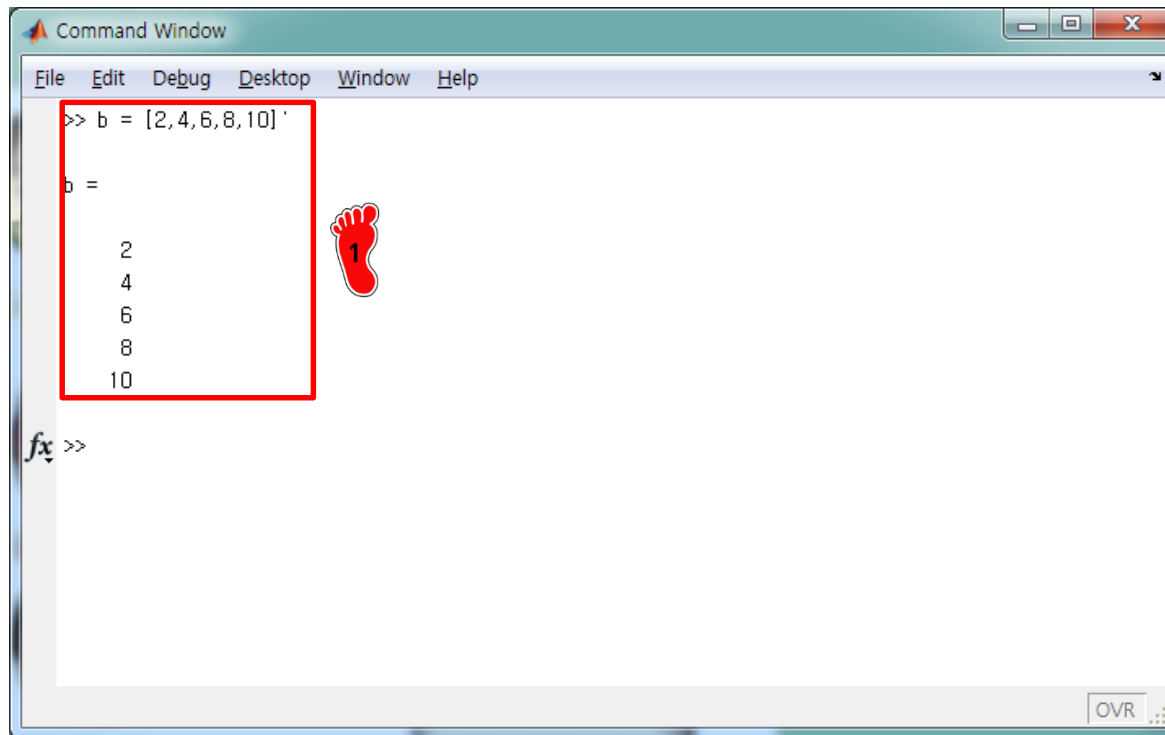
괄호 기호 [] 와 세미 콜론(;) 을 이용하여 열 벡터를 저장할 수 있음



The Workspace window displays the current variables in the MATLAB environment. A red box highlights the variables `a` and `b`.

Name	Value	Min	Max
a	[1,2,3,4,5]	1	5
b	[2;4;6;8;10]	2	10

COLUMN VECTOR



Command Window

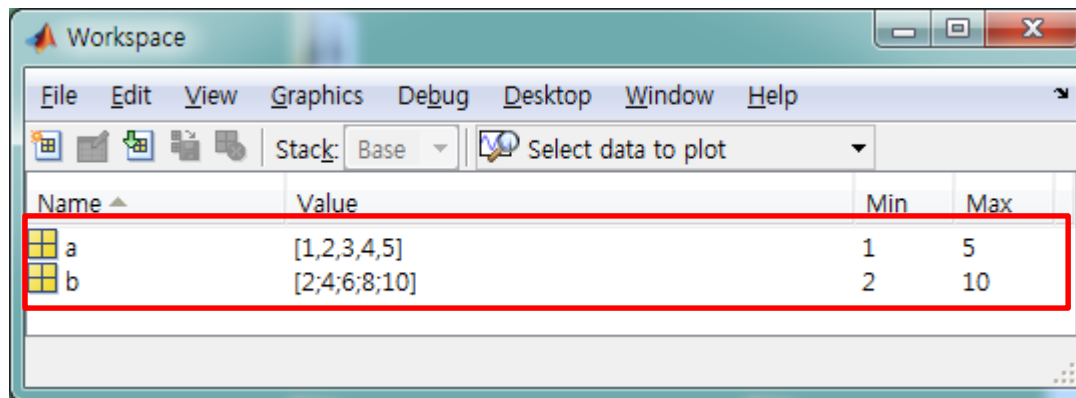
```
>> b = [2,4,6,8,10]'
```

b =

```
2
4
6
8
10
```

A red box highlights the output of the command, and a red footprint icon with the number 1 is placed next to it.

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



Workspace

Name	Value	Min	Max
a	[1,2,3,4,5]	1	5
b	[2;4;6;8;10]	2	10

A red box highlights the row for variable 'b' in the workspace table.

MATRIX

```

>> A = [1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> A = [1 2 3
        4 5 6
        7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9
  
```



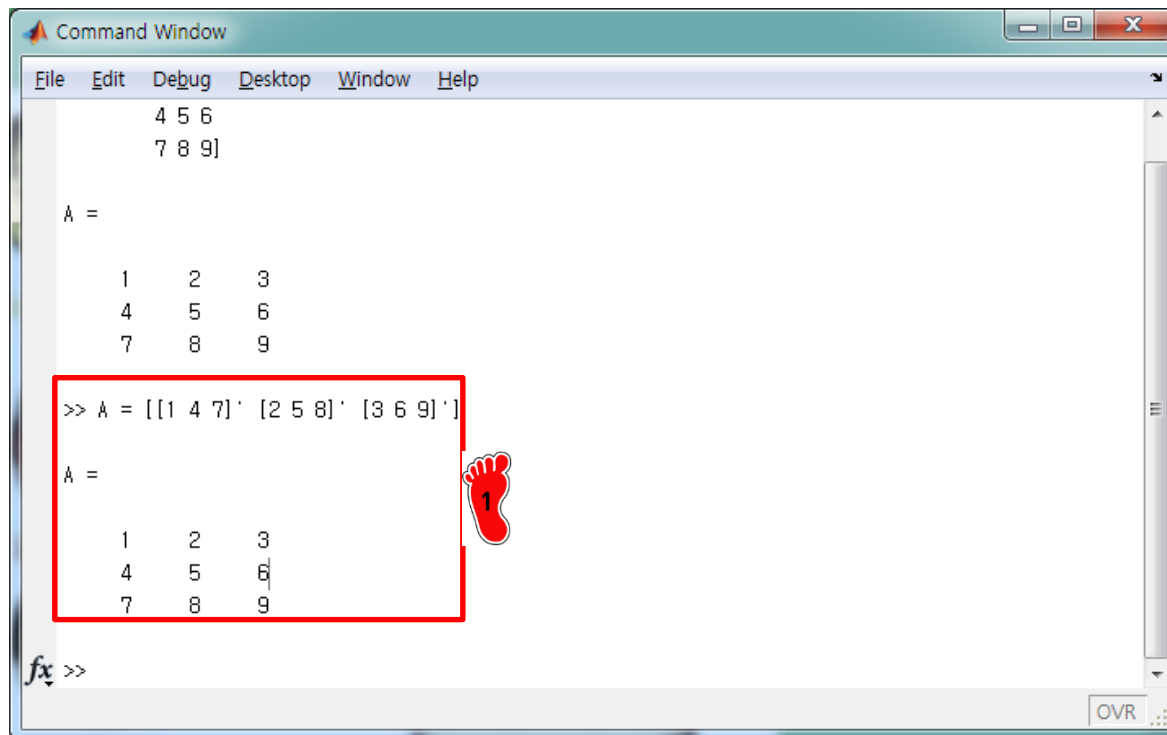
행렬을 입력하는 방법은
벡터를 입력하는 방식과 동일

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

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

Name	Value	Min	Max
A	[1,2,3;4,5,6;7,8,9]	1	9
a	[1,2,3,4,5]	1	5
b	[2;4;6;8;10]	2	10

MATRIX



Command Window

```

File Edit Debug Desktop Window Help
4 5 6
7 8 9]

A =

1 2 3
4 5 6
7 8 9

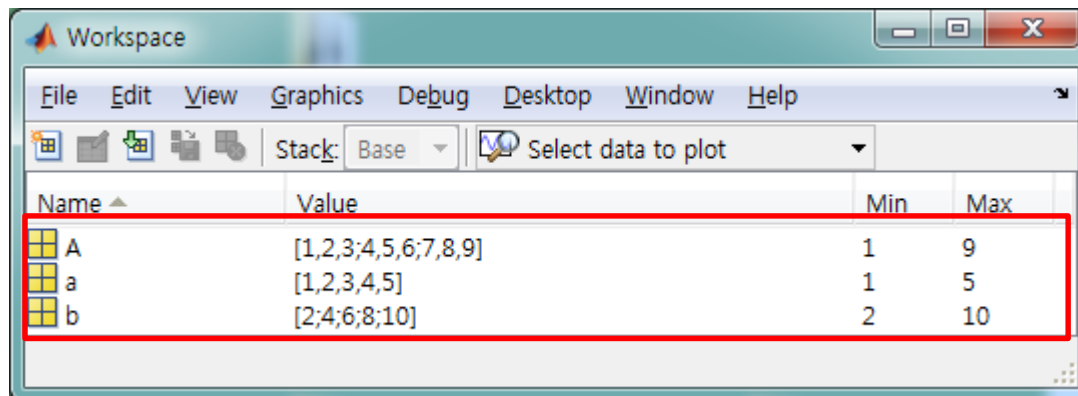
>> A = [[1 4 7]' [2 5 8]' [3 6 9]']
A =

1 2 3
4 5 6
7 8 9
fx >>
OVR

```



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



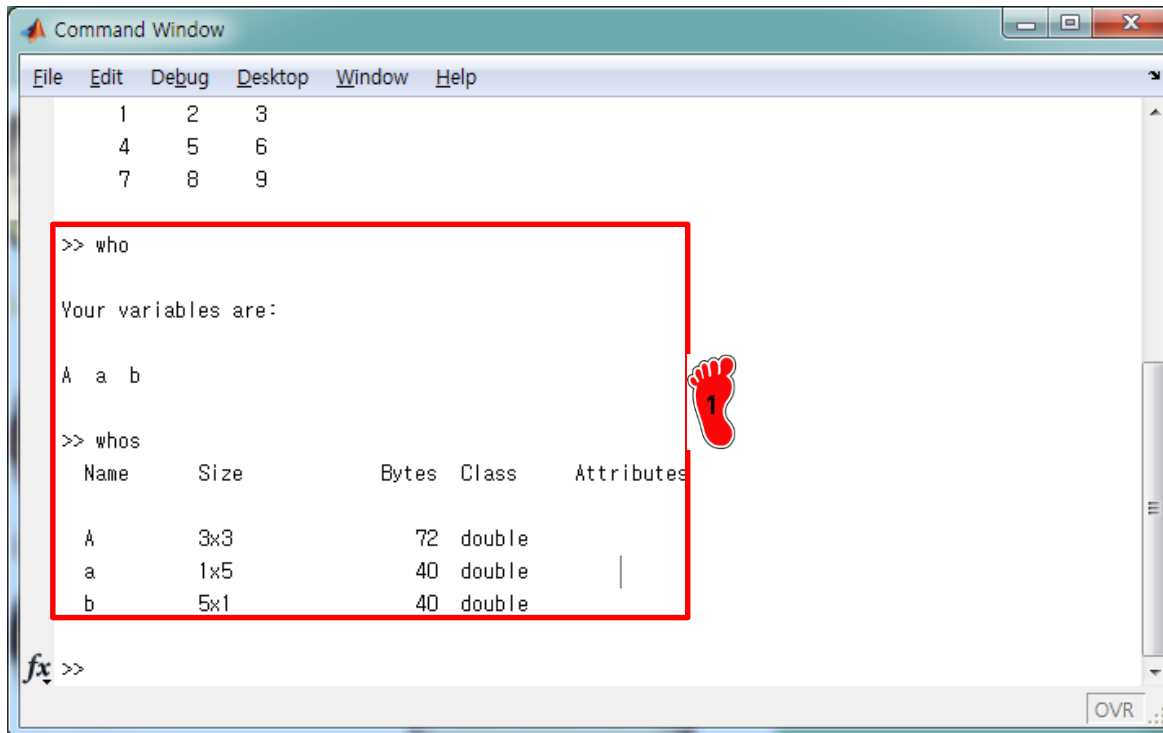
Workspace

File Edit View Graphics Debug Desktop Window Help

Stack: Base Select data to plot

Name	Value	Min	Max
A	[1,2,3;4,5,6;7,8,9]	1	9
a	[1,2,3,4,5]	1	5
b	[2;4;6;8;10]	2	10

MATRIX: WHO(S)



```

Command Window
File Edit Debug Desktop Window Help
1 2 3
4 5 6
7 8 9

>> who

Your variables are:

A a b

>> whos

Name      Size      Bytes  Class  Attributes
-----
A         3x3         72  double
a         1x5         40  double
b         5x1         40  double
  
```

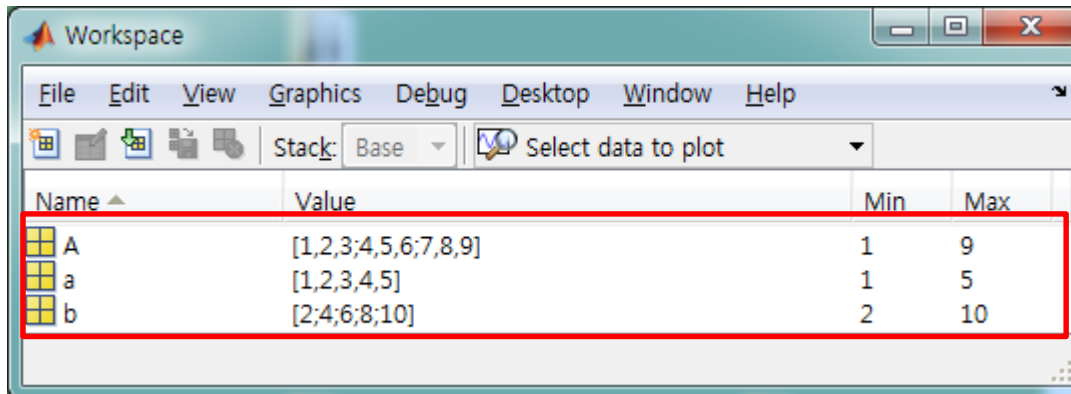


who

현재 저장 되어있는 변수들의 간략한 이름만 표시

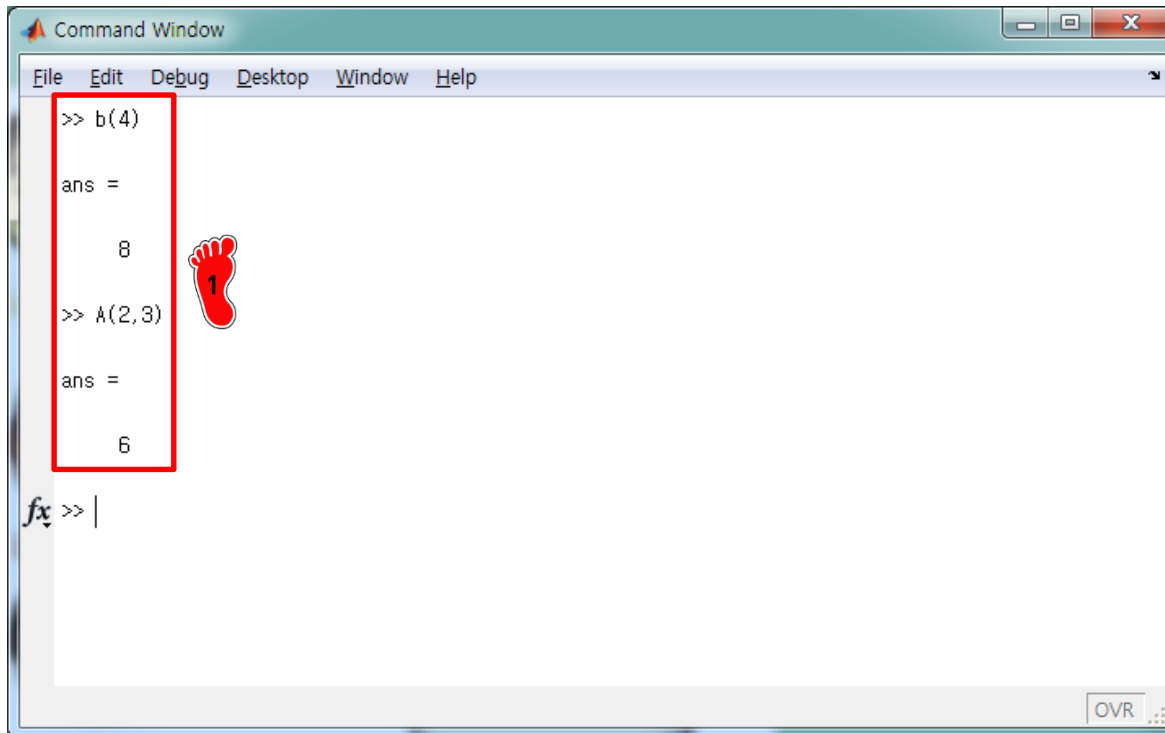
whos

현재 저장 되어있는 변수들의 이름과 자세한 정보를 나타냄



Name	Value	Min	Max
A	[1,2,3;4,5,6;7,8,9]	1	9
a	[1,2,3,4,5]	1	5
b	[2;4;6;8;10]	2	10

MATRIX: ELEMENT



```

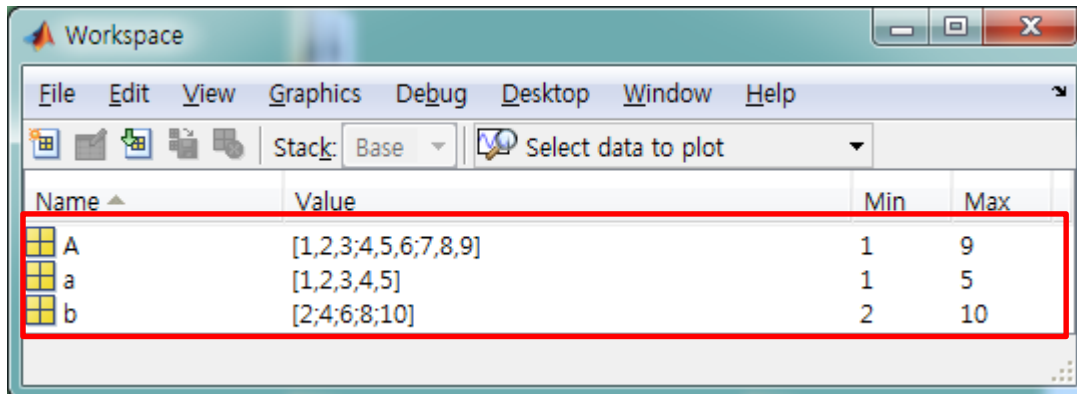
Command Window
File Edit Debug Desktop Window Help
>> b(4)
ans =
      8
>> A(2,3)
ans =
      6
fx >> |
OVR

```



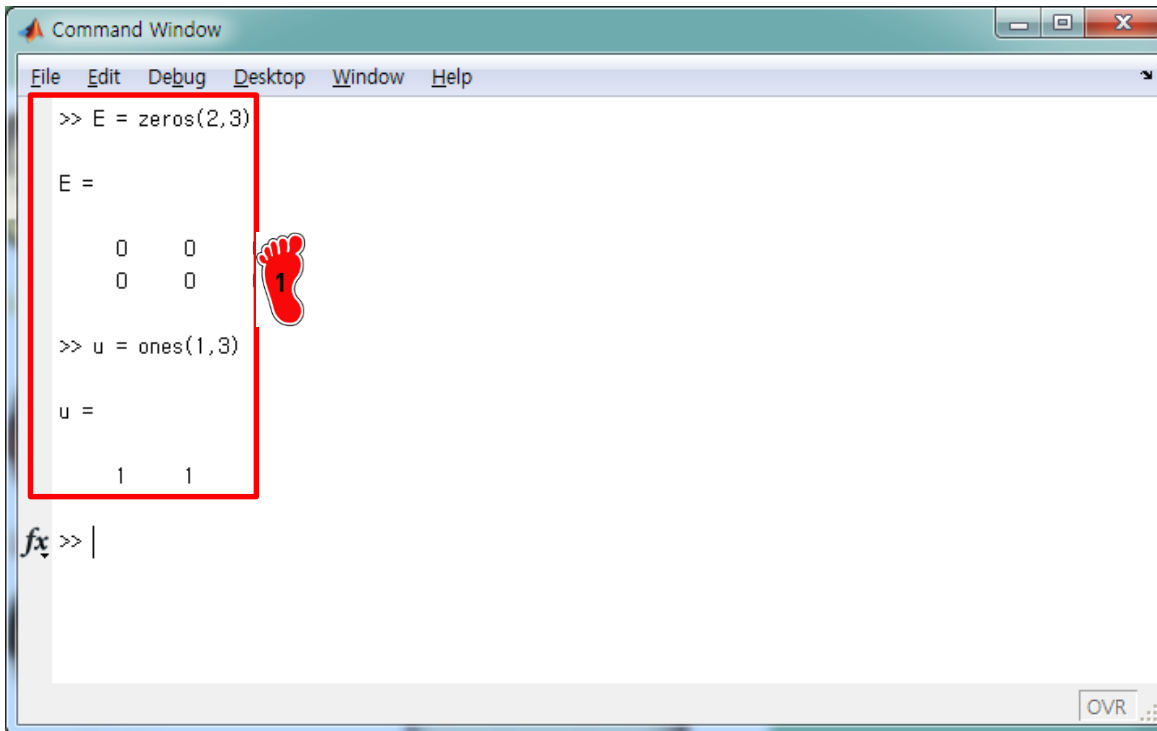
벡터 b 의 4행 1열에 저장
되어있는 값을 확인할 경우
괄호 () 를 이용

행렬 A 도 같은 방식으로 저
장되어있는 값 확인



Name	Value	Min	Max
A	[1,2,3;4,5,6;7,8,9]	1	9
a	[1,2,3,4,5]	1	5
b	[2;4;6;8;10]	2	10

MATRIX: BUILT IN FUNCTION



```

Command Window
File Edit Debug Desktop Window Help
>> E = zeros(2,3)

E =

     0     0
     0     0

>> u = ones(1,3)

u =

     1     1
  
```

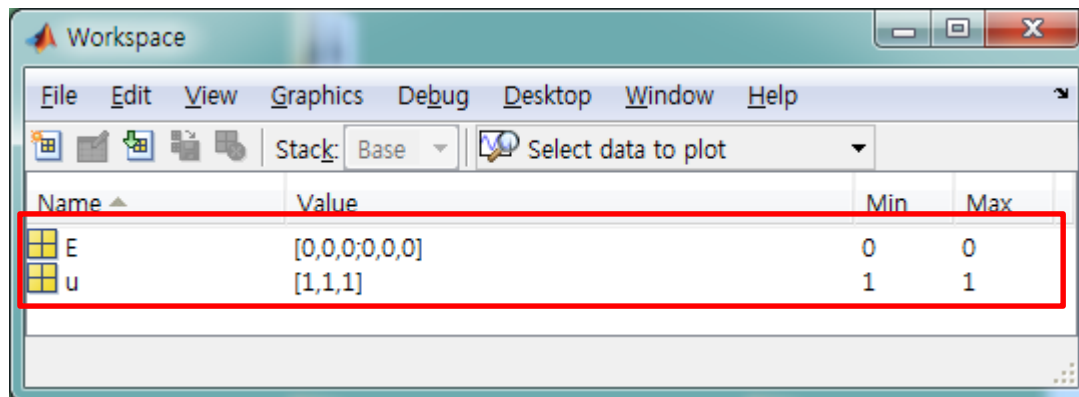


zeros(m,n)

m by n 의 0으로 채워진 매트릭스를 저장

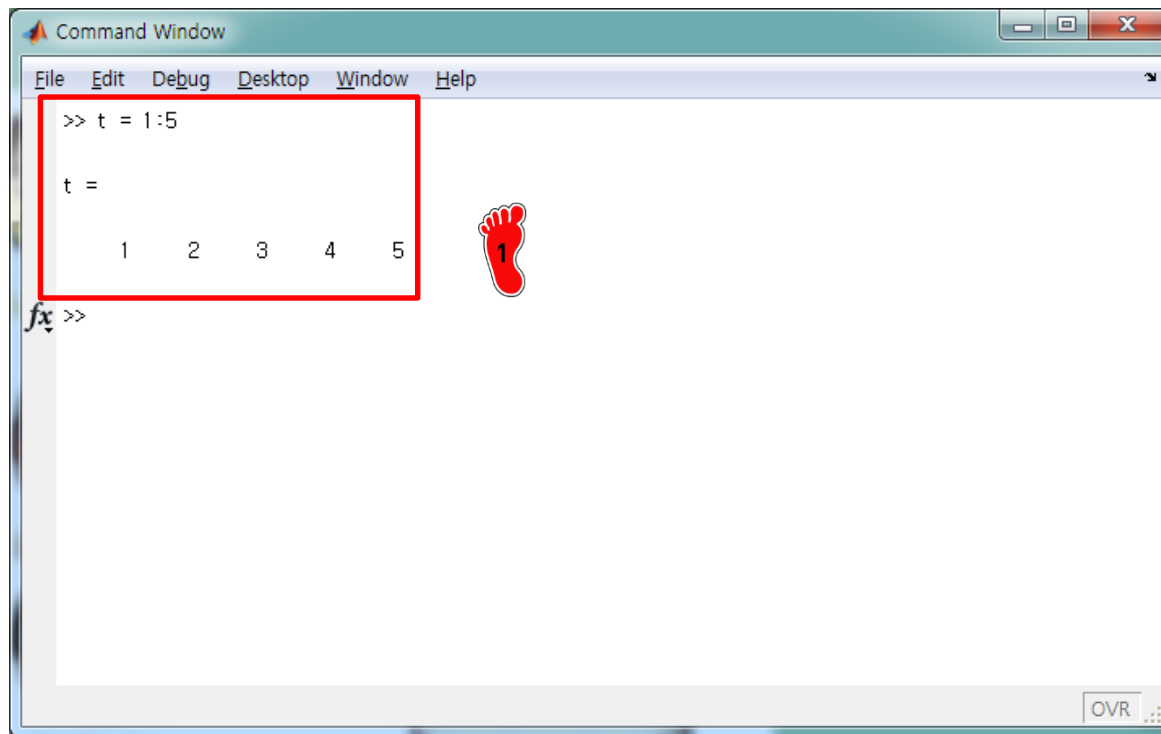
ones(m,n)

m by n 의 1로 채워진 매트릭스를 저장

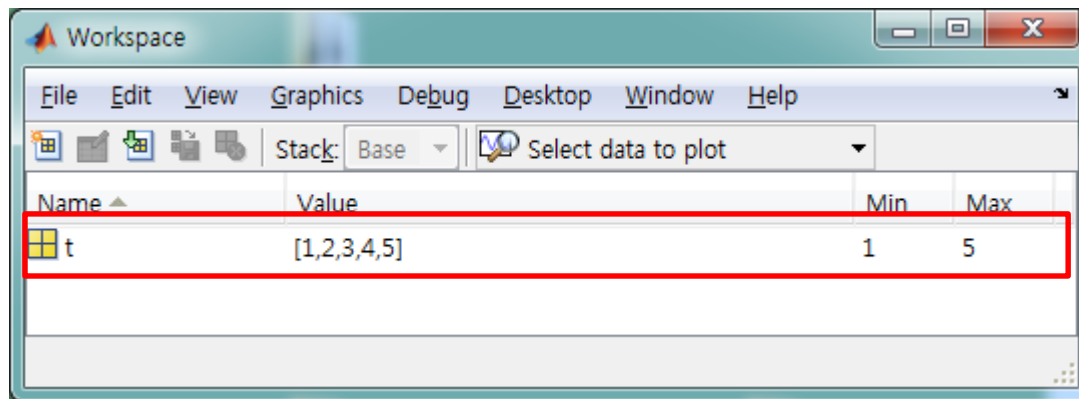


Name	Value	Min	Max
E	[0,0,0;0,0,0]	0	0
u	[1,1,1]	1	1

COLON OPERATOR



콜론 (:) 을 이용하면 1단위로 증가하는 배열을 저장할 수 있음



COLON OPERATOR



Command Window

```
>> t = 10:-1:5
```

t =

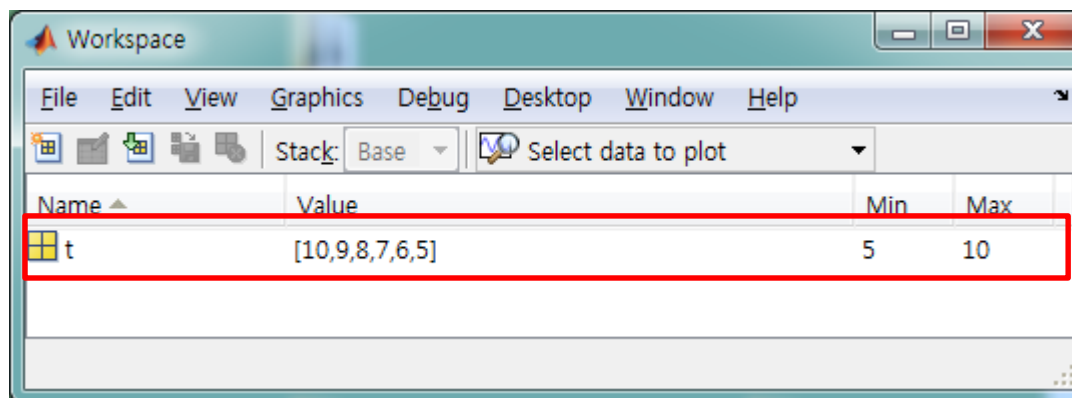
10	9	8	7	6	5
----	---	---	---	---	---

fx >>

OVR



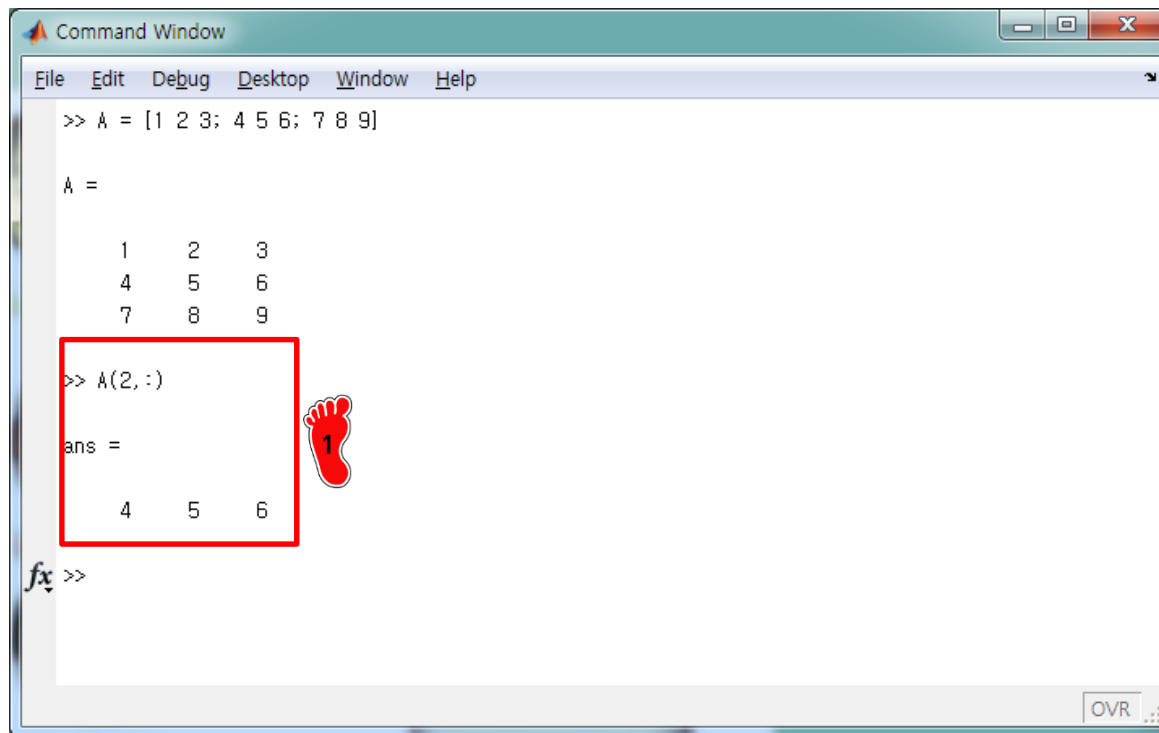
- 값을 이용하여 줄어드는 배열을 저장할 수 있음



Workspace

Name	Value	Min	Max
t	[10,9,8,7,6,5]	5	10

COLON OPERATOR



Command Window

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

A =

1	2	3
4	5	6
7	8	9

```
>> A(2,:)
ans =
```

4	5	6
---	---	---

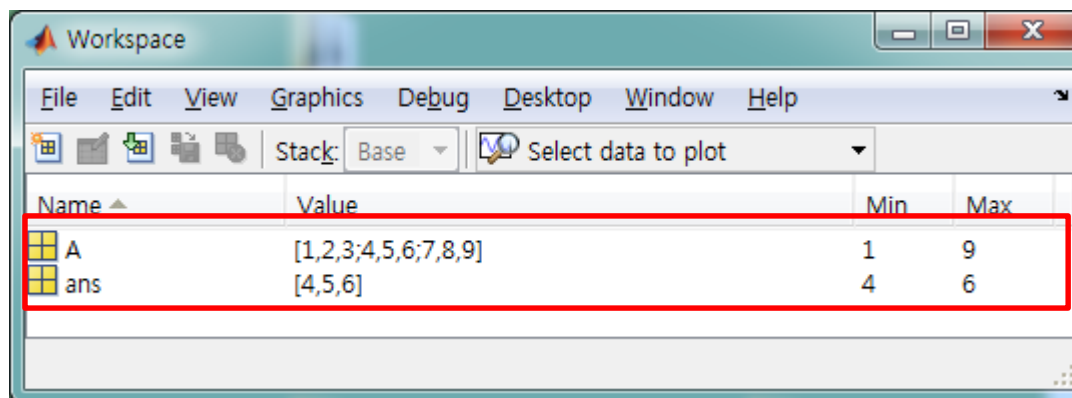
fx >>

OVR



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

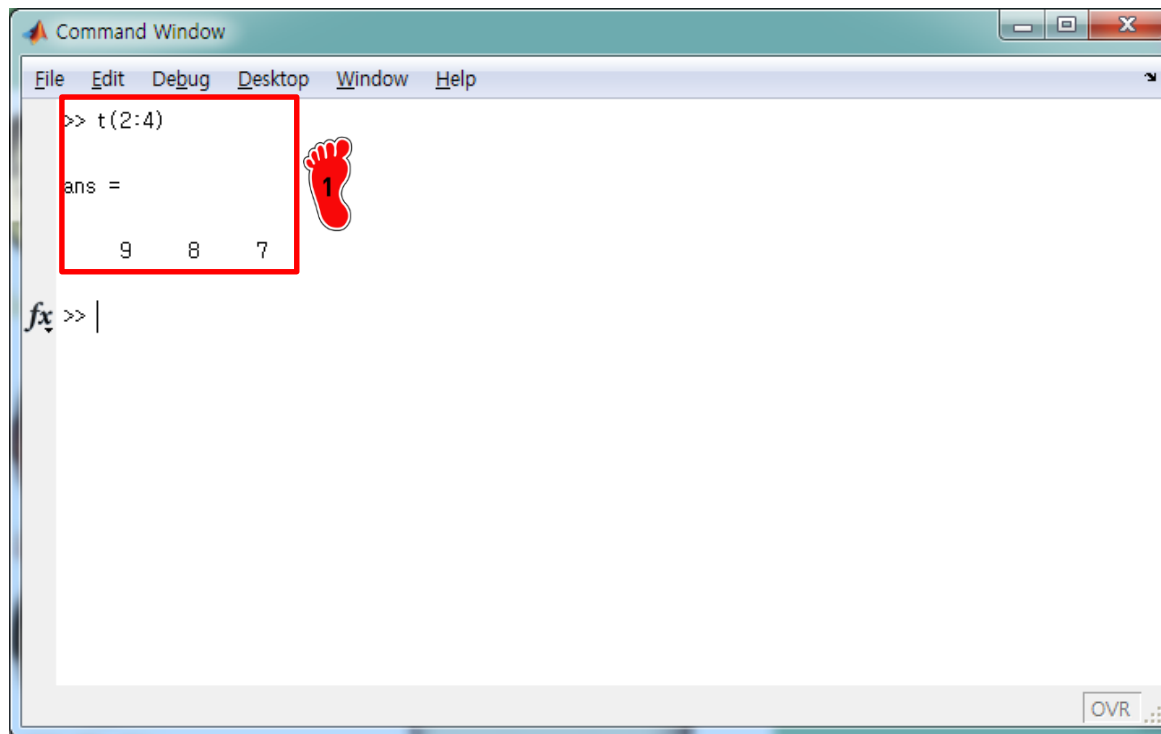
예시에는 2행 전체 값을 확인



Workspace

Name	Value	Min	Max
A	[1,2,3;4,5,6;7,8,9]	1	9
ans	[4,5,6]	4	6

COLON OPERATOR



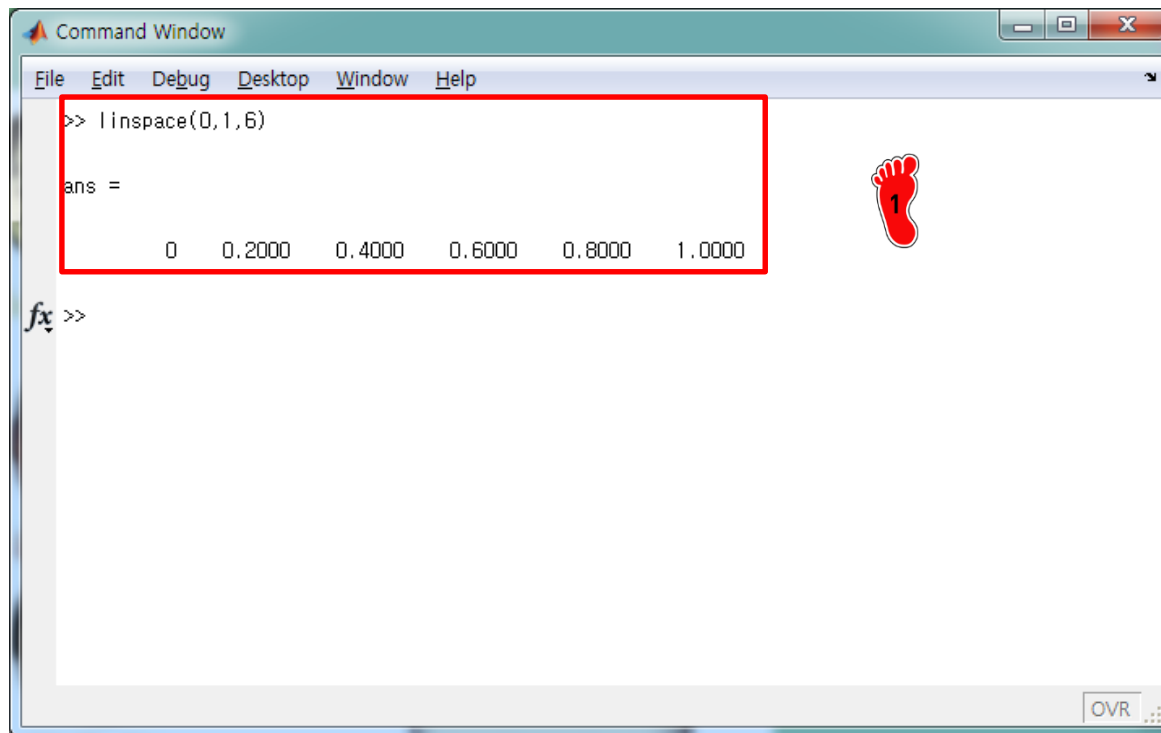
배열값 역시 콜론을 이용하여 원하는 위치의 값을 확인할 수 있음

Workspace

Name	Value	Min	Max
ans	[9,8,7]	7	9
t	[10,9,8,7,6,5]	5	10

A red box highlights the workspace table.

Linspace Function



Command Window

```
>> linspace(0,1,6)
```

ans =

0	0.2000	0.4000	0.6000	0.8000	1.0000
---	--------	--------	--------	--------	--------

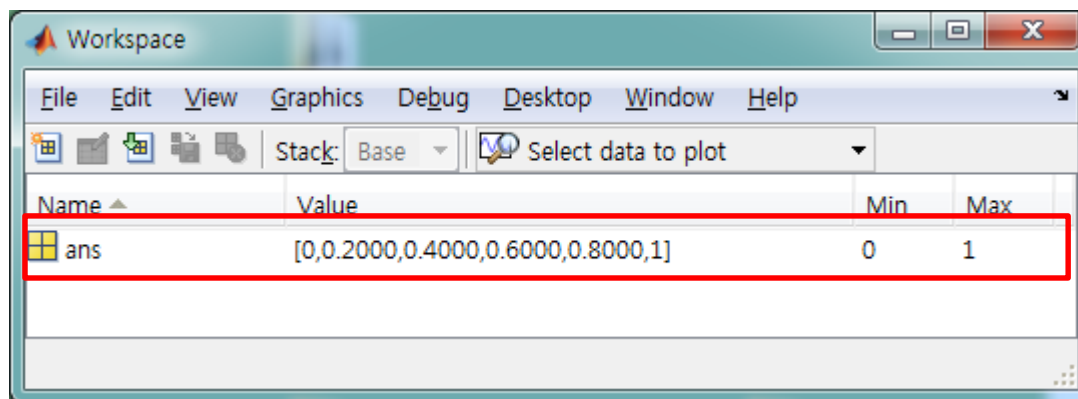
fx >>

OVR



linspace(x1,x2,n)

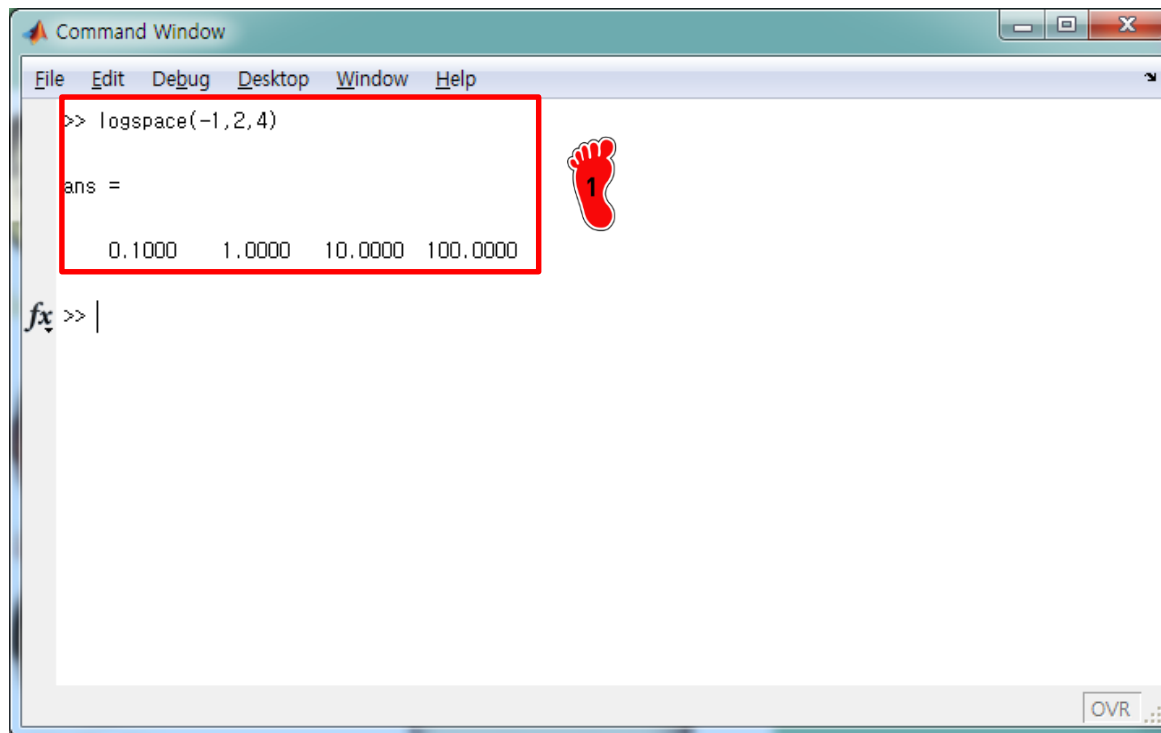
값 x1 부터 x2 까지 선형적으로 n 등분한 결과값을 저장



Workspace

Name	Value	Min	Max
ans	[0,0.2000,0.4000,0.6000,0.8000,1]	0	1

LOGSPACE FUNCTION



Command Window

```
>> logspace(-1,2,4)
```

ans =

0.1000	1.0000	10.0000	100.0000
--------	--------	---------	----------

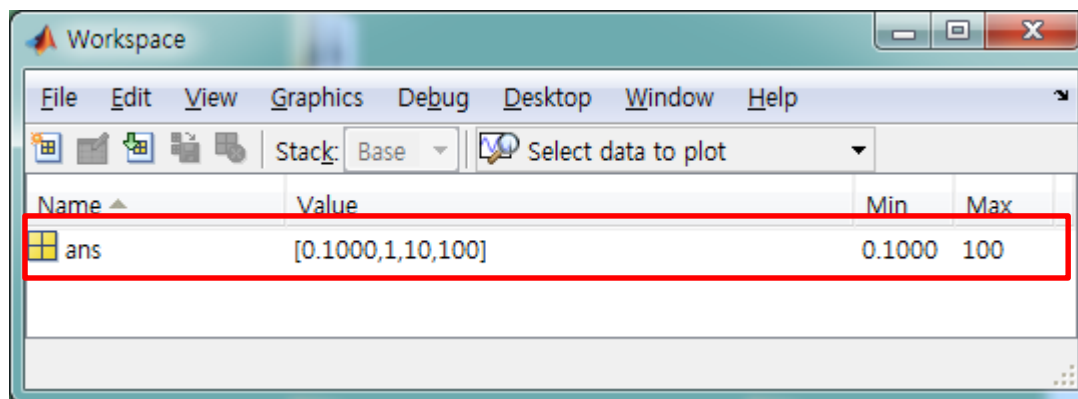
fx >> |

OVR



logspace(x1,x2,n)

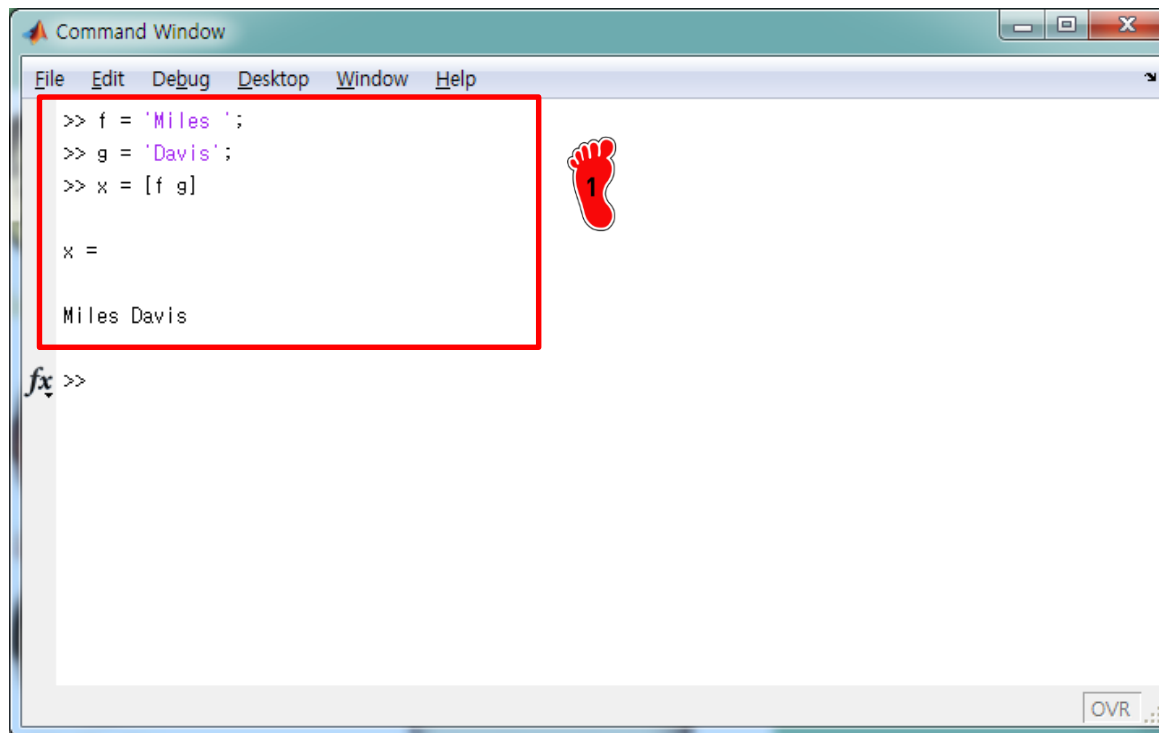
값 10^{-1} 부터 10^2 까지 로그 스케일로 n 등분한 결과값을 저장



Workspace

Name	Value	Min	Max
ans	[0.1000,1,10,100]	0.1000	100

CHARACTER STRING



Command Window

```
>> f = 'Miles ';
>> g = 'Davis';
>> x = [f g]

x =

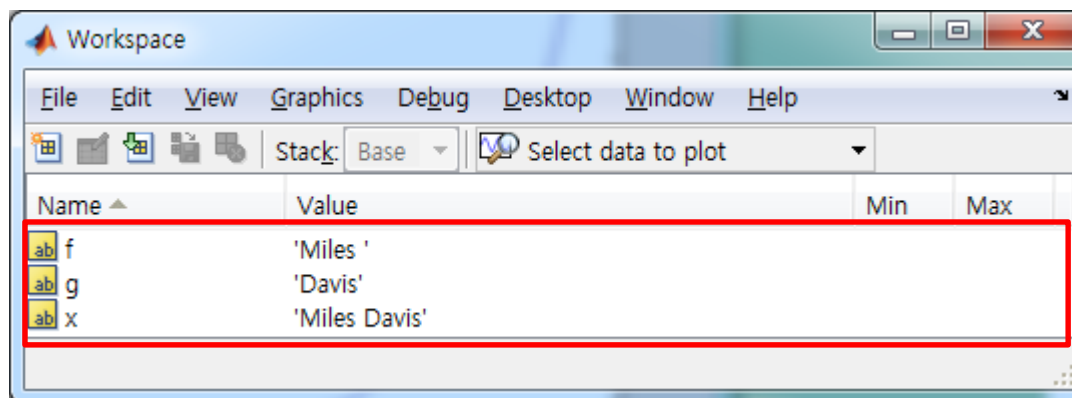
Miles Davis
```

fx >>

OVR



작은 따옴표 두 개를 이용하여 글자열을 저장할 수 있음



Workspace

Name	Value	Min	Max
f	'Miles '		
g	'Davis'		
x	'Miles Davis'		

CHARACTER STRING



Command Window

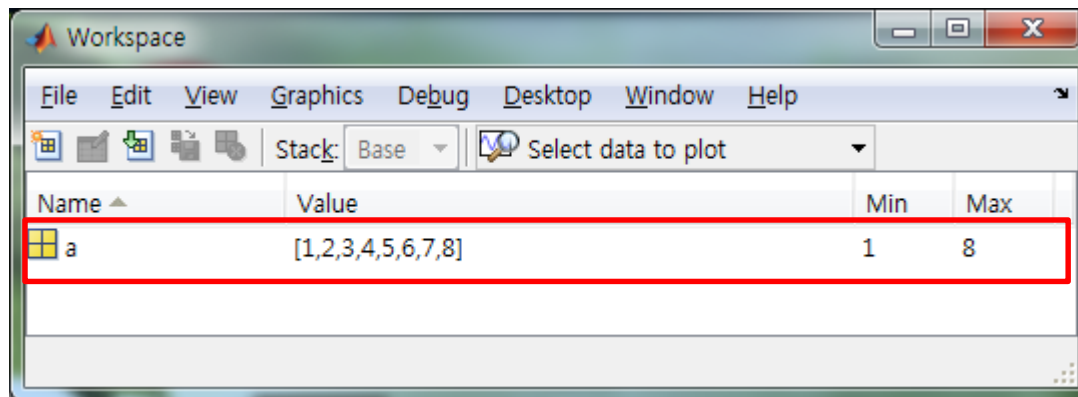
```
>> a = [1 2 3 4 5 ...  
6 7 8]  
  
a =  
  
    1    2    3    4    5    6    7    8
```

fx >> |

A red rectangle highlights the command and the resulting array. A red footprint icon with the number 1 is placed next to the array output.



마침표를 이용하여 세 개를
입력하면 다음줄로 넘어 갈
수 있음

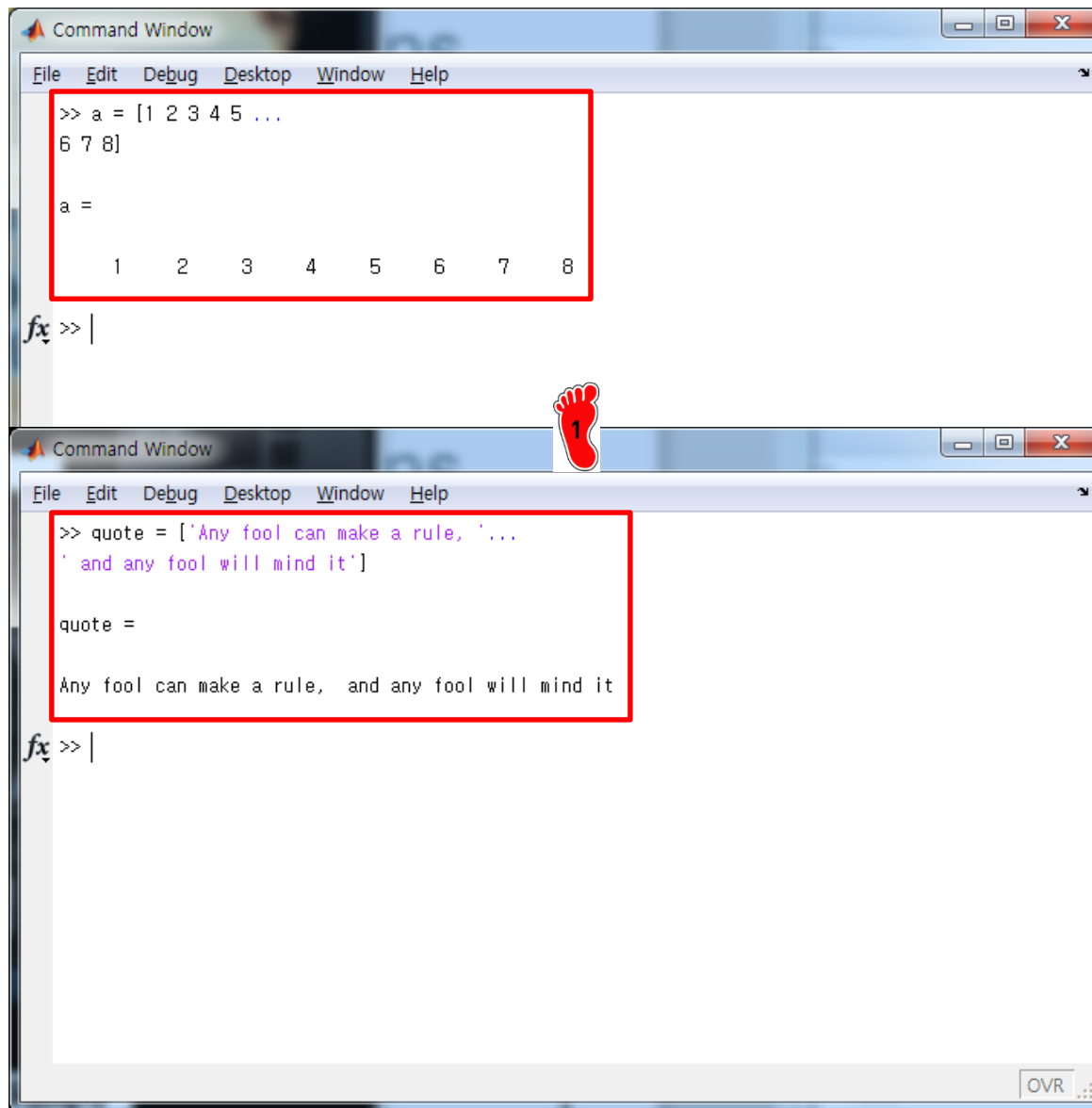


Workspace

Name	Value	Min	Max
a	[1,2,3,4,5,6,7,8]	1	8

A red rectangle highlights the row for variable 'a'.

CHARACTER STRING



```
Command Window
File Edit Debug Desktop Window Help
>> a = [1 2 3 4 5 ...
6 7 8]
a =
     1     2     3     4     5     6     7     8
fx >> |

Command Window
File Edit Debug Desktop Window Help
>> quote = ['Any fool can make a rule, '...
' and any fool will mind it']
quote =
Any fool can make a rule, and any fool will mind it
fx >> |
```



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

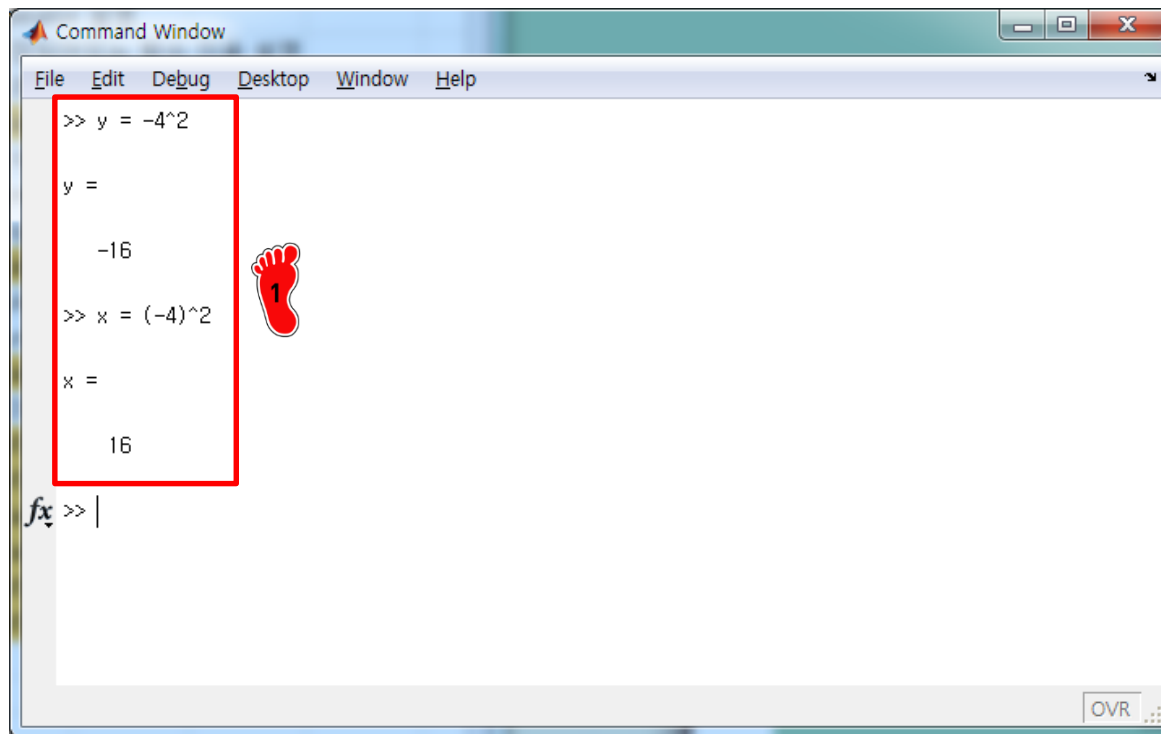


매틀랩 연산 기호



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

MATHEMATICAL OPERATION



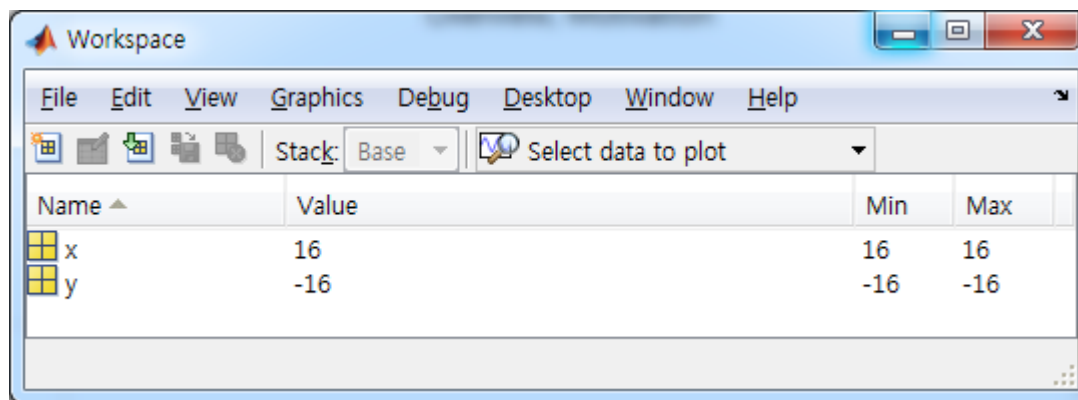
Command Window

```
>> y = -4^2
y =
    -16
>> x = (-4)^2
x =
    16
```

A red box highlights the first two lines of code: `>> y = -4^2` and `y = -16`. A red footprint icon with the number 1 is placed next to the box.



연산에 우선순위가 있는것
을 확인



Workspace

Name	Value	Min	Max
x	16	16	16
y	-16	-16	-16

VECTOR PRODUCT

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



저장되어있는 변수가 벡터
혹은 행렬일 경우 벡터 연산
으로 연산 작용이 됨

```

Command Window
File Edit Debug Desktop Window Help
>> a+b
ans =
    110
>> b*a
ans =
     2     4     6     8    10
     4     8    12    16    20
     6    12    18    24    30
     8    16    24    32    40
    10    20    30    40    50
fx >> |
OVR
  
```

VECTOR-MATRIX MULTIPLICATION

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



벡터와 행렬 간 연산

```

Command Window
File Edit Debug Desktop Window Help
>> a*A
ans =
    30    36    42
>> A*b
ans =
    32
    77
   122
fx >>
OVR
  
```

VECTOR-MATRIX MULTIPLICATION

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



차수가 맞지 않을 경우 오류
메세지를 출력

```

Command Window
File Edit Debug Desktop Window Help

ans =

    30    36    42

>> A*b

ans =

    32
    77
   122

>> A*a
??? Error using ==> mtimes
Inner matrix dimensions must agree.

fx >> |
OVR
  
```



MATRIX-MATRIX MULTIPLICATION

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



연산기호 ^ 을 행렬 연산에 적용할 경우 제곱을 의미함

```

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

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



만약 행렬의 각 항을 제공한 결과를 보고 싶을 때는 마침표를 연산기호 ^ 앞에 붙여서 .^ 연산을 해주게 되면 각 항 별로 제공한 결과를 확인할 수 있음

```

Command Window
File Edit Debug Desktop Window Help

>> A^2

ans =

    30    36    42
    66    81    96
   102   126   150

>> A.^2

ans =

     1     4     9
    16    25    36
    49    64    81
  
```



- **Built-in functions**

- ✓ **Log**
- ✓ **Elfun**
- ✓ **Round**
- ✓ **Ceil**
- ✓ **Floor**
- ✓ **Others**

LOG



```

>> 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
  
```

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

$$\log(z) = \log(\text{abs}(z)) + i*\text{atan2}(y,x)$$

Examples

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

```
ans =
    3.1416
```

See Also

[exp](#), [log10](#), [log2](#), [logm](#), [reallog](#)



매틀랩에 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.

```



그 외 기본적인 수학 관련 함수는 help elfun 으로 확인할 수 있음

ROUND

$$E = [-1.6 \quad -1.5 \quad -1.4 \quad 1.4 \quad 1.5 \quad 1.6]$$



round 함수는 각 원소별로 반올림하여 정수로 만들어 주는 함수

```

Command Window
File Edit Debug Desktop Window Help
>> E = [-1.6 -1.5 -1.4 1.4 1.5 1.6]

E =

    -1.6000    -1.5000    -1.4000     1.4000     1.5000     1.6000

>> round(E)
ans =

     -2     -2     -1      1      2      2
  
```



ceil 함수는 올림하여 정수로 만들어 주는 함수

$$E = [-1.6 \quad -1.5 \quad -1.4 \quad 1.4 \quad 1.5 \quad 1.6]$$

```
Command Window
File Edit Debug Desktop Window Help
E =
    -1.6000    -1.5000    -1.4000     1.4000     1.5000     1.6000

>> round(E)

ans =
     -2     -2     -1      1      2      2

>> ceil(E)
|
ans =
     -1     -1     -1      2      2      2
```

The screenshot shows the MATLAB Command Window. The matrix E is defined as [-1.6, -1.5, -1.4, 1.4, 1.5, 1.6]. The command round(E) is executed, resulting in ans = [-2, -2, -1, 1, 2, 2]. The command ceil(E) is then executed, and the result ans = [-1, -1, -1, 2, 2, 2] is displayed. A red box highlights the ceil(E) command and its result. A red footprint icon with the number 1 is placed next to the result of the ceil function.

FLOOR

$$E = [-1.6 \quad -1.5 \quad -1.4 \quad 1.4 \quad 1.5 \quad 1.6]$$



floor 함수는 내림하여 정수로 만들어 주는 함수

```

Command Window
File Edit Debug Desktop Window Help
ans =
    -2    -2    -1     1     2     2

>> ceil(E)

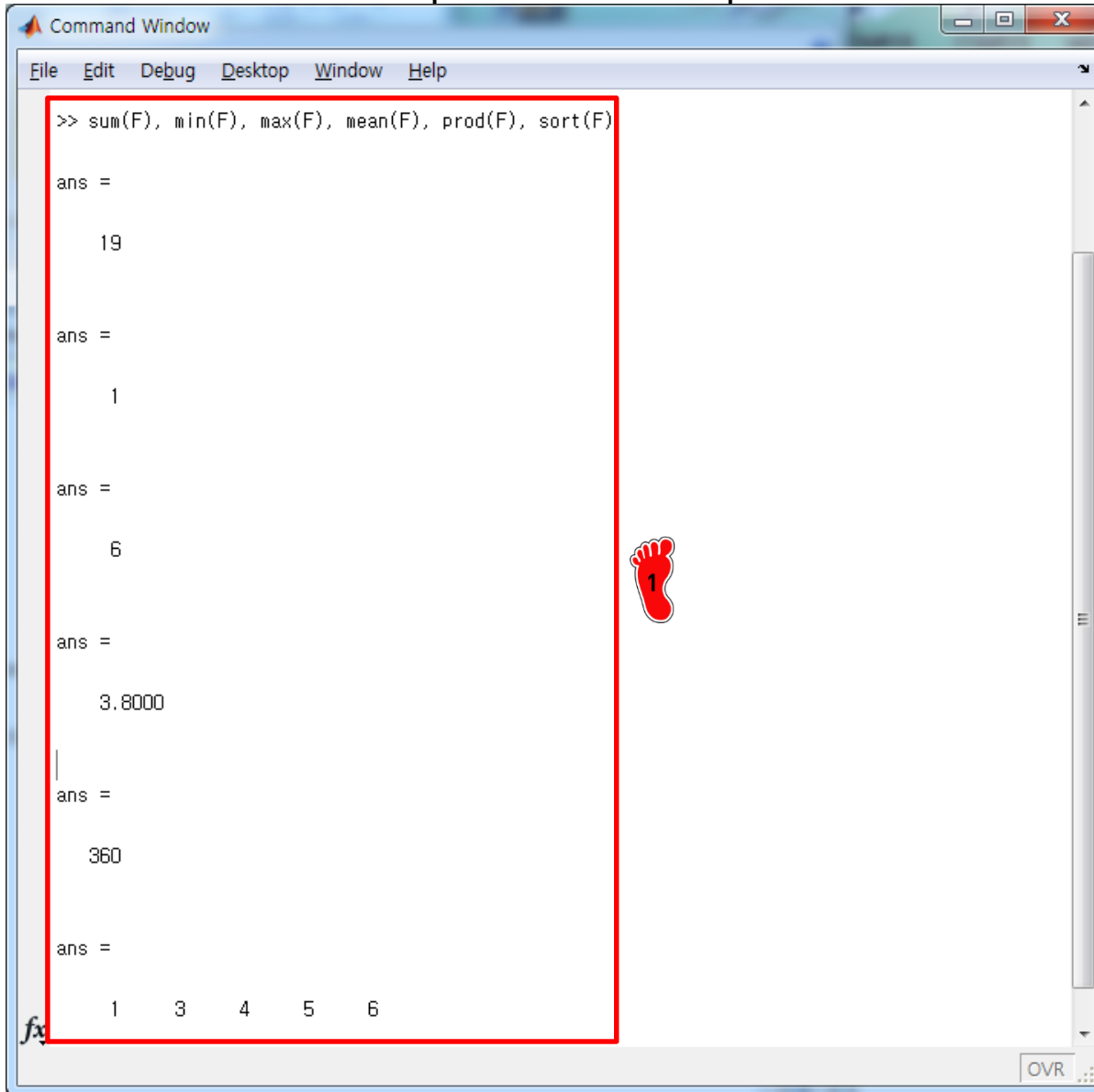
ans =
    -1    -1    -1     2     2     2

>> floor(E)
ans =
    -2    -2    -2     1     1     1
  
```

The screenshot shows the MATLAB Command Window with the following content: The first part shows the variable 'ans' with the values [-2, -2, -1, 1, 2, 2]. The second part shows the command '>> ceil(E)' followed by 'ans = [-1, -1, -1, 2, 2, 2]'. The third part, which is highlighted with a red rectangle, shows the command '>> floor(E)' followed by 'ans = [-2, -2, -2, 1, 1, 1]'. A red footprint icon with the number 1 is placed next to the 'floor(E)' command. The bottom of the window shows 'fx >>' and 'OVR ...'.

OTHERS

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



```

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
  
```



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 >

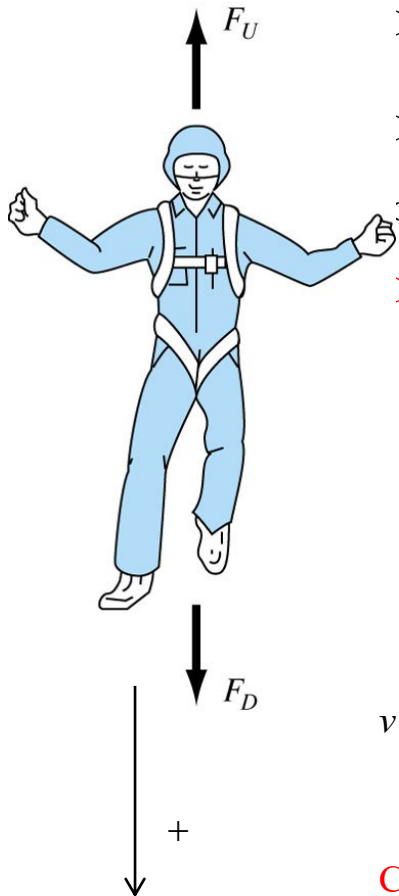
Case I: $v_h = c_1 e^{\lambda t} \rightarrow \lambda = -\frac{c}{m}$ (trivial solution $v = v_h + v_p$)

$$= c_1 e^{\left(-\frac{c}{m}\right)t} + c_2 \rightarrow \frac{c}{m}c_2 = g \rightarrow c_2 = \frac{gm}{c} \quad (\text{set } v_p = c_2, \text{ substitute } v \text{ into differential equation})$$

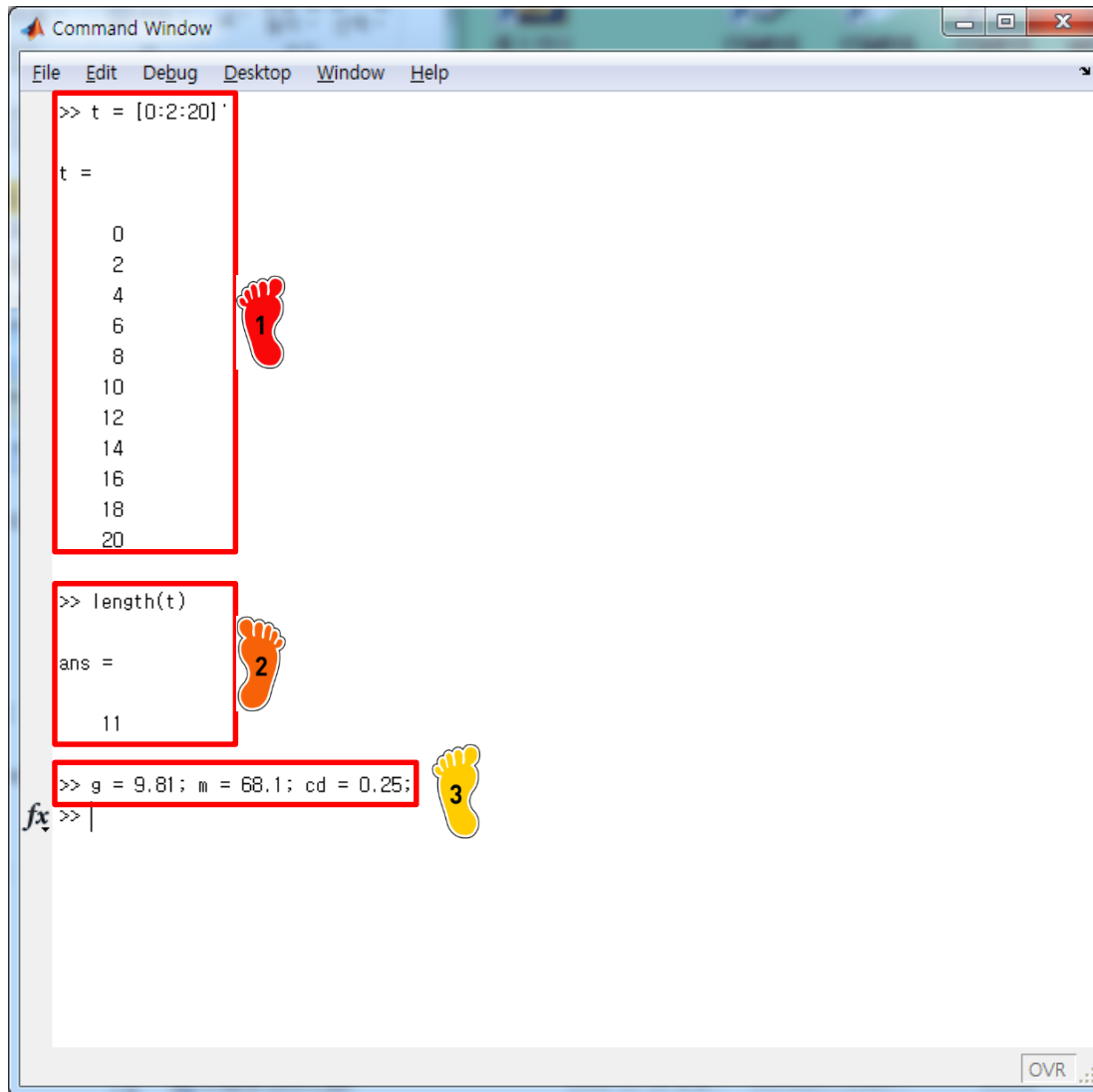
$$= c_1 e^{\left(-\frac{c}{m}\right)t} + \frac{gm}{c} \leftarrow (t=0, v=0) \Rightarrow c_1 = -\frac{gm}{c}$$

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

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



VARIABLES & CONSTANTS



```
Command Window
File Edit Debug Desktop Window Help

>> t = [0:2:20]
t =
    0
    2
    4
    6
    8
   10
   12
   14
   16
   18
   20

>> length(t)
ans =
    11

>> g = 9.81; m = 68.1; cd = 0.25;
fx >>
```

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 상수
를 저장

DEPENDENT VARIABLE



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

```

Command Window
File Edit Debug Desktop Window Help
>> v1 = g*m/cd*(1-exp(-cd/m*t))

v1 =

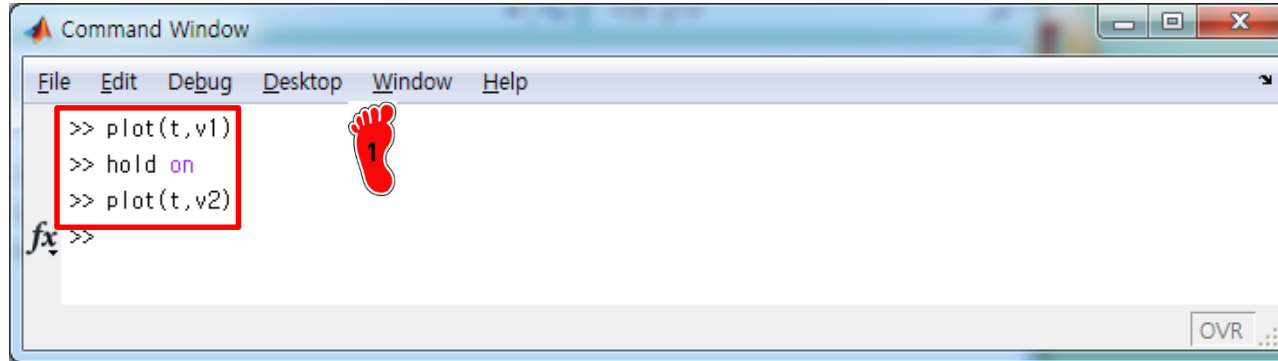
    0
 19.5481
 38.9533
 58.2165
 77.3388
 96.3212
115.1647
133.8704
152.4393
170.8723
189.1704

>> v2 = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t)

v2 =

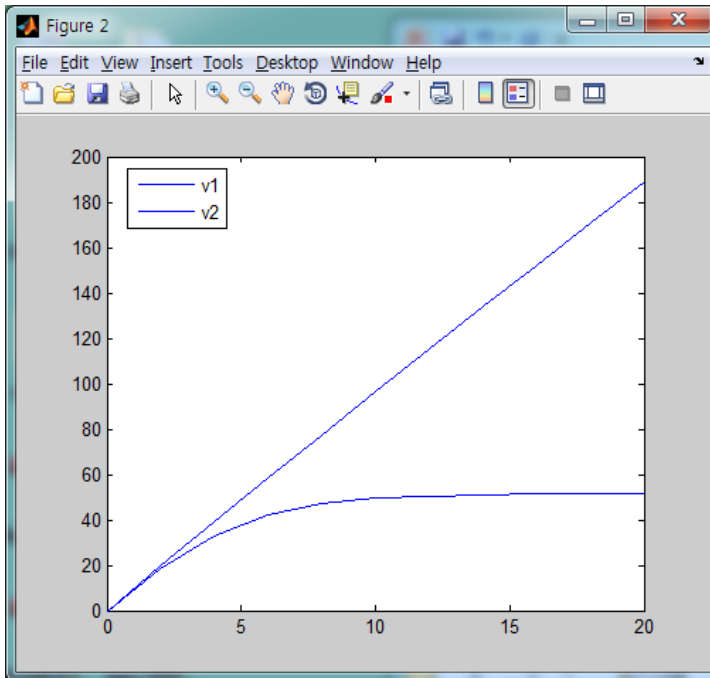
    0
 18.7292
 33.1118
 42.0762
 46.9575
 49.4214
 50.6175
 51.1871
 51.4560
 51.5823
 51.6416
  
```

PLOT



```

>> plot(t,v1)
>> hold on
>> plot(t,v2)
fx >>
  
```

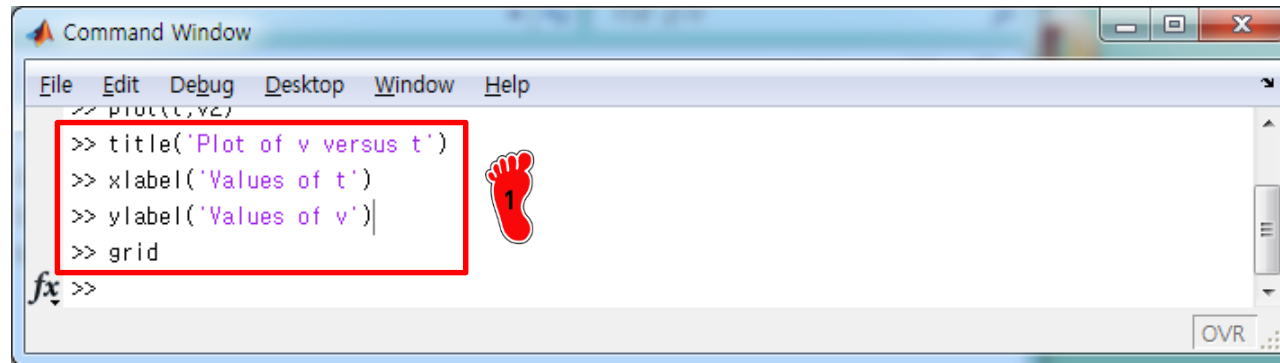


plot(a,b) 함수는 독립변수 a와 함수값 b를 이용하여 그래프를 출력하는 함수

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



```

>> plot(t,v2)
>> title('Plot of v versus t')
>> xlabel('Values of t')
>> ylabel('Values of v')
>> grid
fx >>
  
```

A red box highlights the four plotting commands: `title`, `xlabel`, `ylabel`, and `grid`. A red footprint icon with the number 1 is placed next to the `title` command.

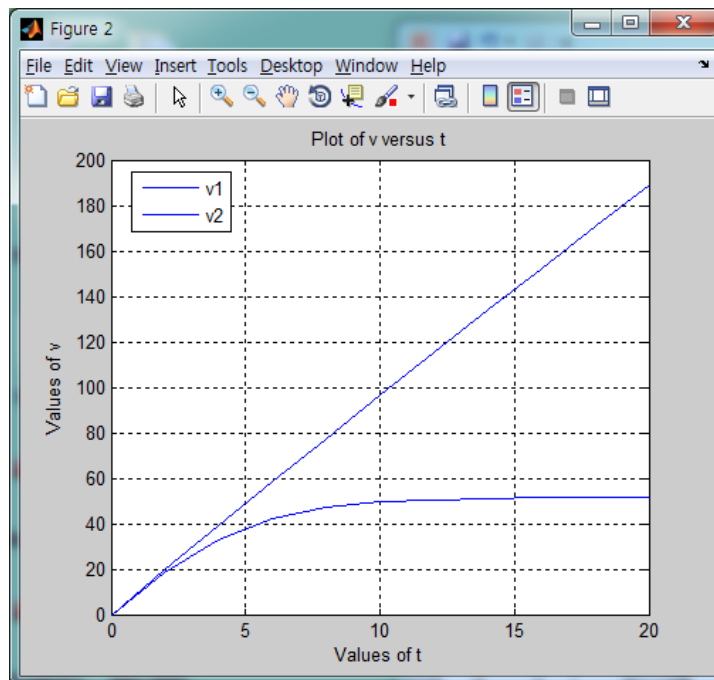


title: 그래프의 제목을 변경

xlabel: 그래프 가로축의 이름을 변경

ylabel: 그래프 세로축의 이름을 변경

grid: 격자를 생성

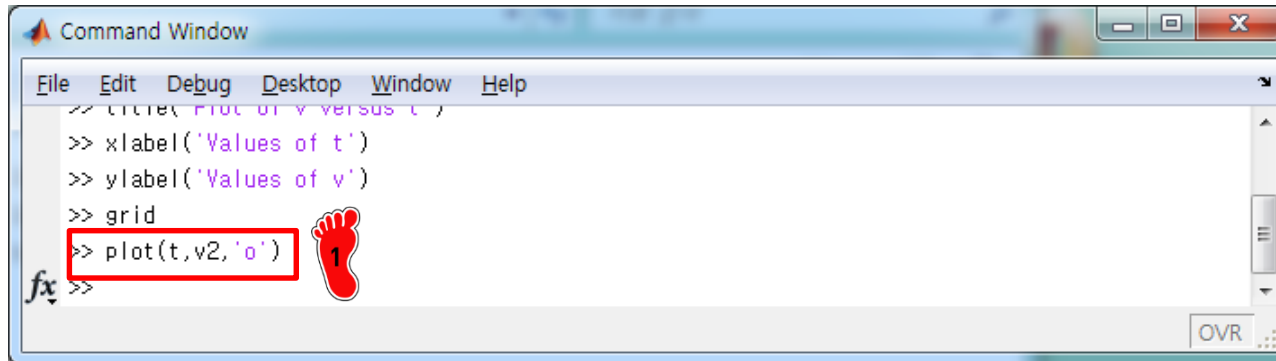


SPECIFIERS

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)

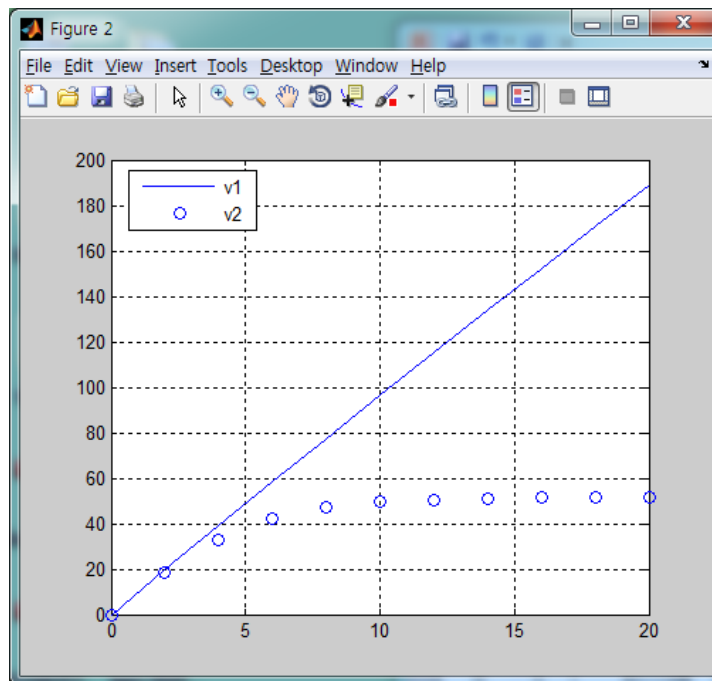
SYMBOL



```

Command Window
File Edit Debug Desktop Window Help
>> title('Plot of v versus t')
>> xlabel('Values of t')
>> ylabel('Values of v')
>> grid
>> plot(t,v2,'o')
fx >>
OVR

```



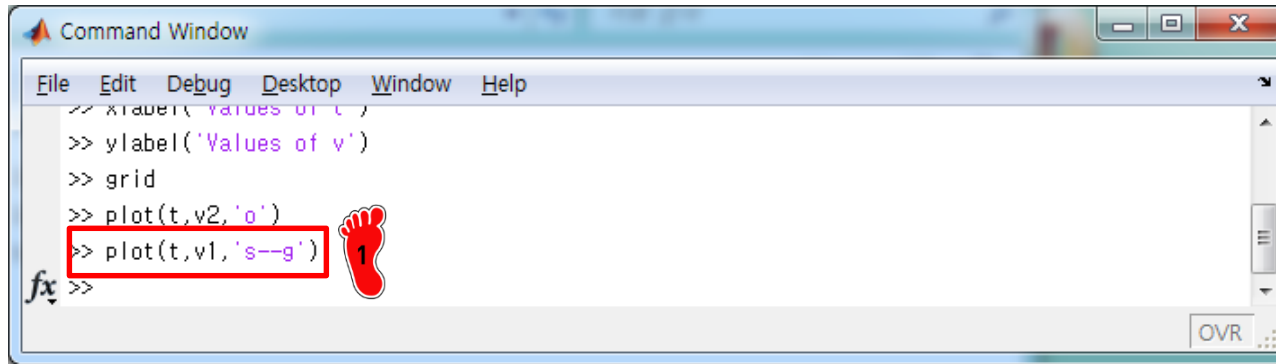
전 페이지의 명령어를 입력하는 방법은

plot(x,y,'명령어')

방식으로 입력 가능

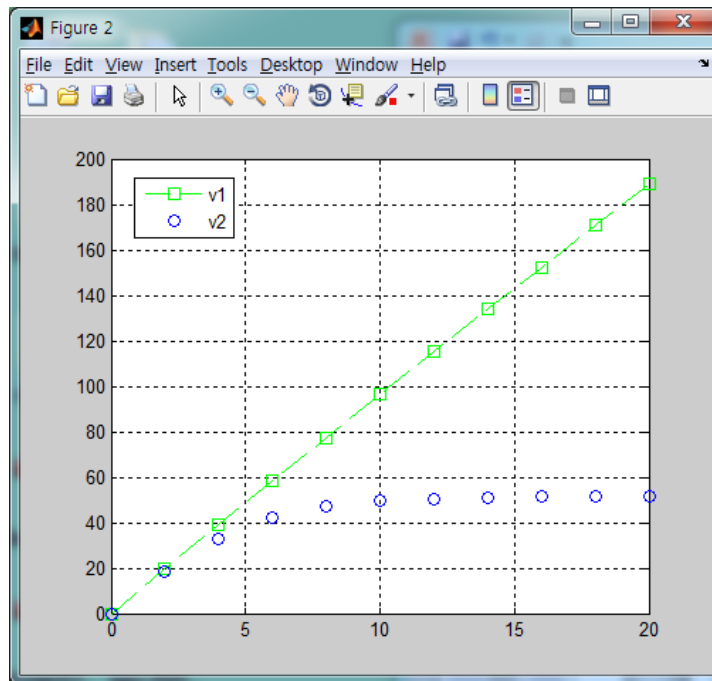
예시는 데이터를 o 표시로 그림을 출력하는 방법

COLLOR & DASHED LINE



```

Command Window
File Edit Debug Desktop Window Help
>> xlabel('Values of t')
>> ylabel('Values of v')
>> grid
>> plot(t, v2, 'o')
>> plot(t, v1, 's--g')
fx >>
OVR
  
```

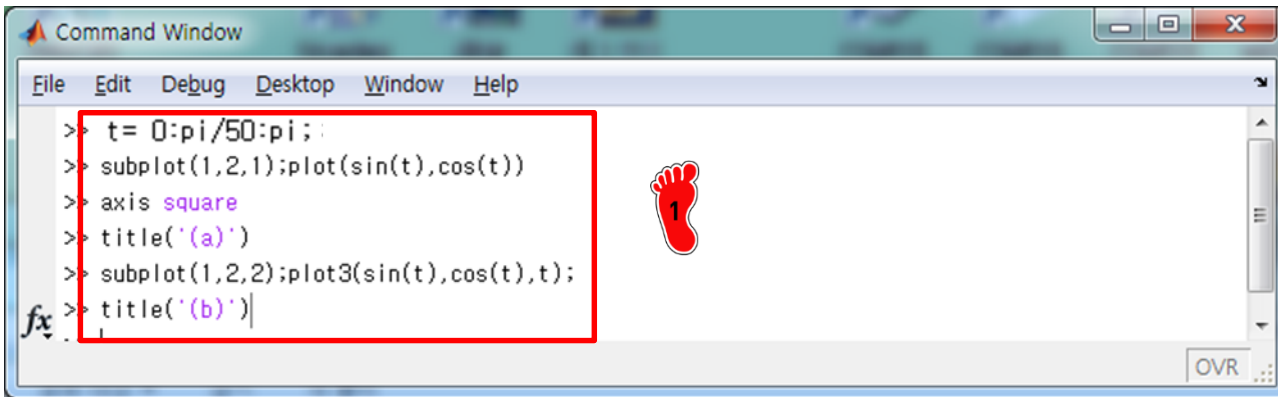


본 예시는

사각형 s
대쉬 -
초록색 g

명령어를 한 번에 적용하여
그래프를 출력한 경우

SUBPLOT & 3D PLOT



```

> t= 0:pi/50:pi;
> subplot(1,2,1);plot(sin(t),cos(t))
> axis square
> title('(a)')
> subplot(1,2,2);plot3(sin(t),cos(t),t);
> title('(b)')
  
```

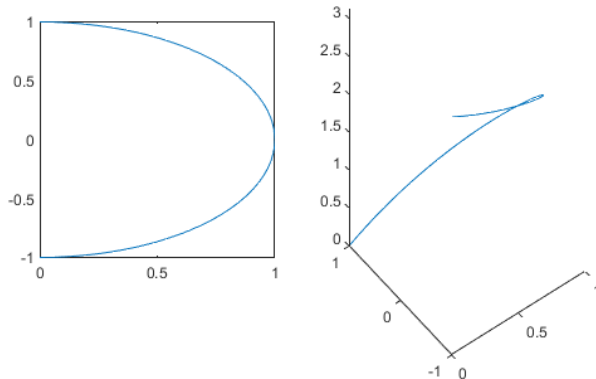


subplot(m,n,p): m by n 개의 그림칸을 생성, p 는 p 번째의 그림을 의미

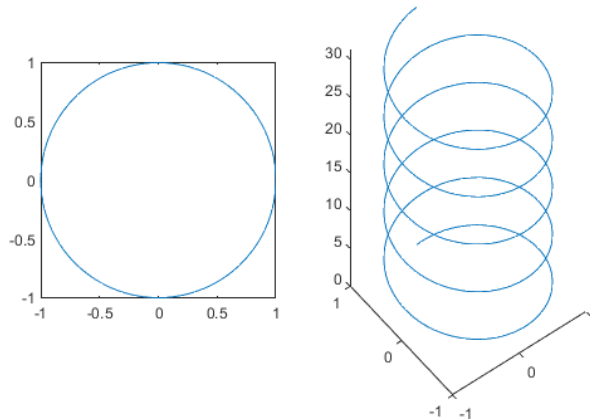
plot3(x,y,z): 3d plot 을 의미하며 z 축은 세 번째 데이터를 의미

T의 구간 설정에 따른 2d/3d plot 개형 변화 비교

$t = 0:\pi/50:\pi;$



$t = 0:\pi/50:10\pi;$



- **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.

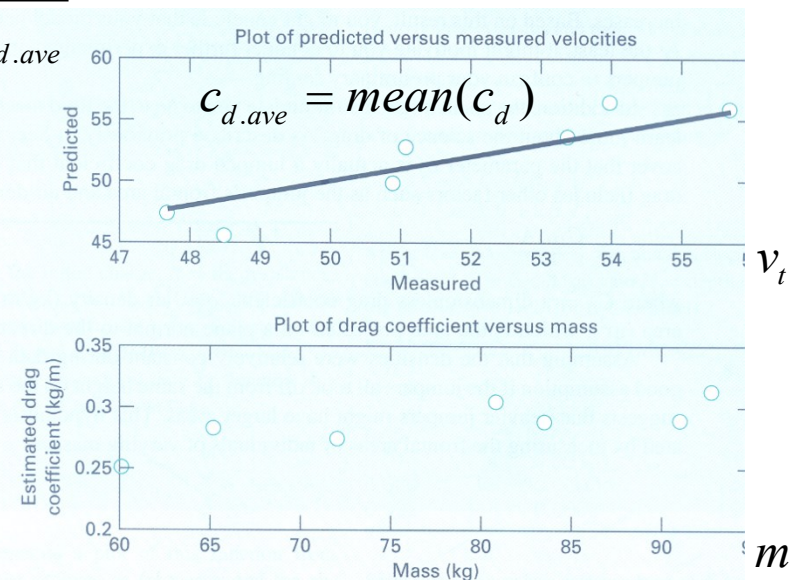
m , kg	83.6	60.2	72.1	91.1	92.9	65.3	80.9
v_t , m/s	53.4	48.5	50.9	55.7	54	47.7	51.1

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

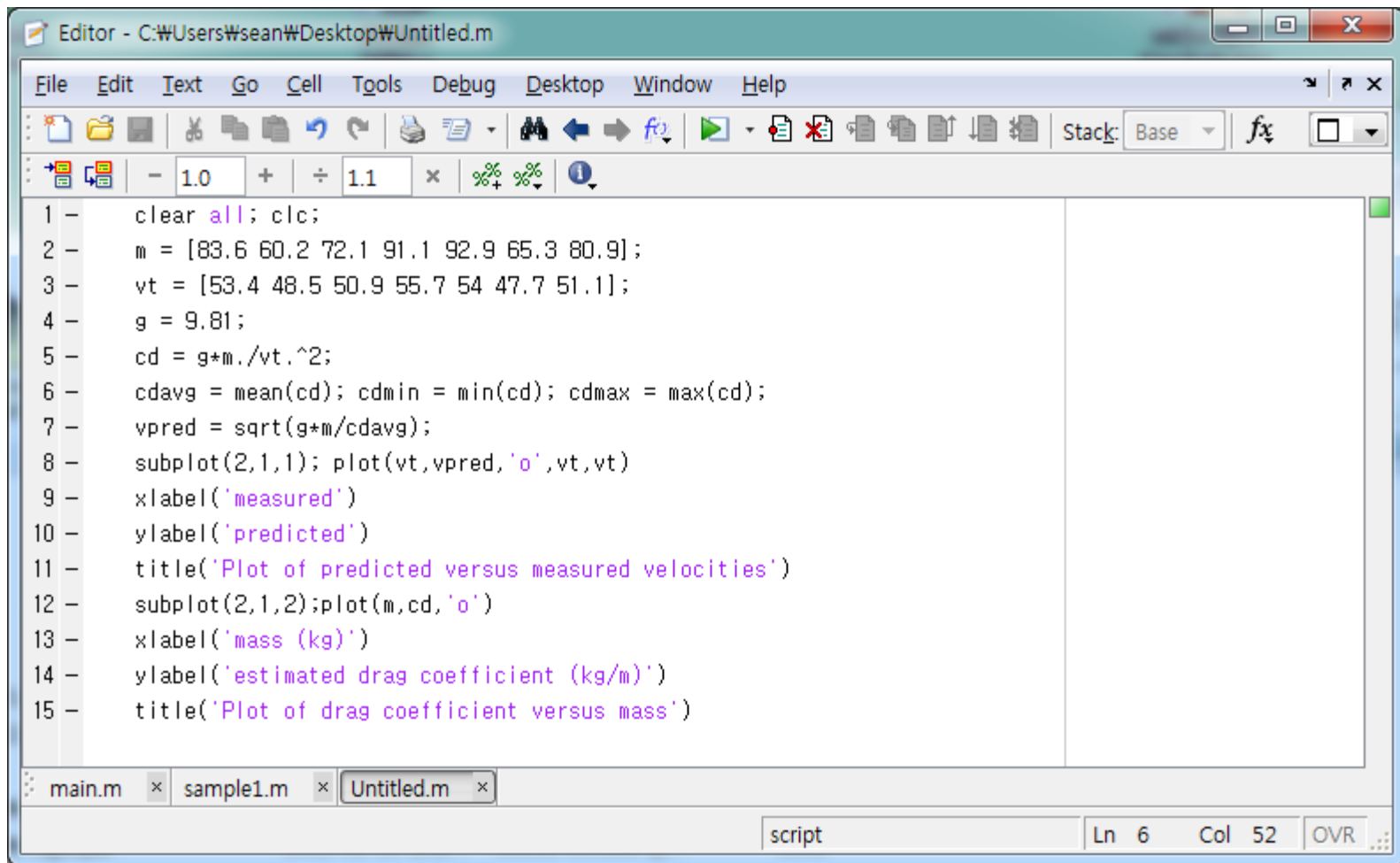
$$v_{pred} = \sqrt{\frac{gm}{c_{d,ave}}}$$

$$c_d = \frac{gm}{v_t^2}$$

Results figures



SOLUTION OF CASE STUDY

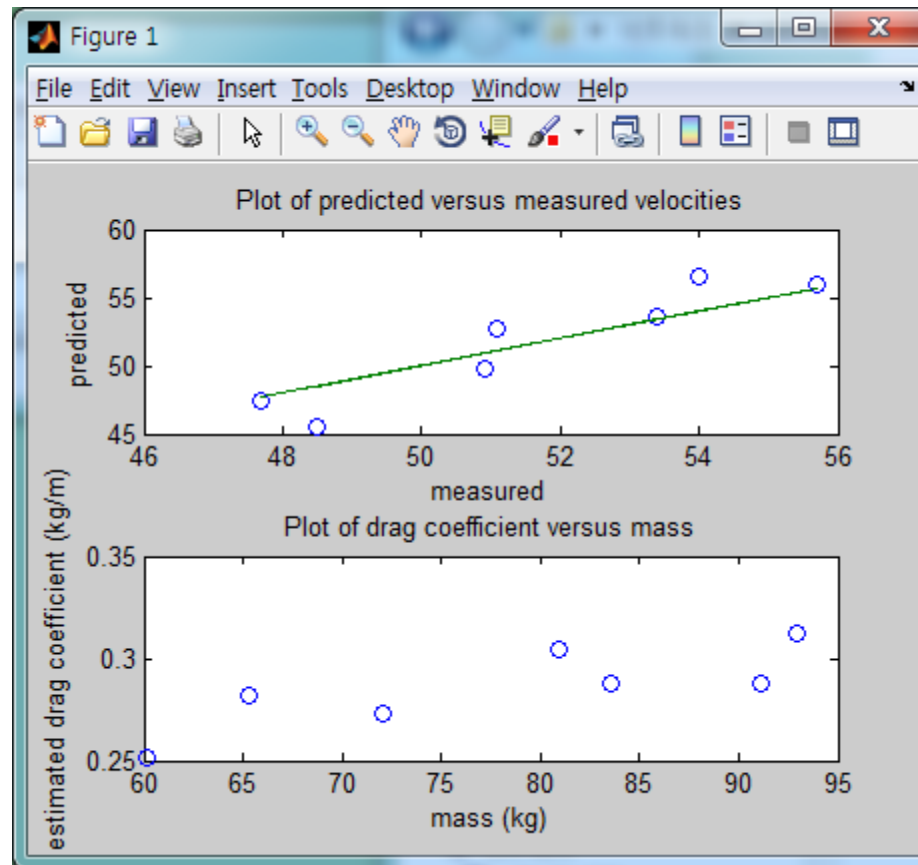


The image shows a MATLAB Editor window titled "Editor - C:\Users\sean\Desktop\Untitled.m". The window contains a script with 15 lines of MATLAB code. The code calculates the drag coefficient (cd) for a set of masses (m) and measured velocities (vt). It then plots the predicted velocity (vpred) against the measured velocity (vt) and the estimated drag coefficient (cd) against the mass (m). The script uses the following variables and operations:

```
1 - clear all; clc;
2 - m = [83.6 60.2 72.1 91.1 92.9 65.3 80.9];
3 - vt = [53.4 48.5 50.9 55.7 54 47.7 51.1];
4 - g = 9.81;
5 - cd = g*m./vt.^2;
6 - cdavg = mean(cd); cdmin = min(cd); cdmax = max(cd);
7 - vpred = sqrt(g*m/cdavg);
8 - subplot(2,1,1); plot(vt,vpred,'o',vt,vt)
9 - xlabel('measured')
10 - ylabel('predicted')
11 - title('Plot of predicted versus measured velocities')
12 - subplot(2,1,2); plot(m,cd,'o')
13 - xlabel('mass (kg)')
14 - ylabel('estimated drag coefficient (kg/m)')
15 - title('Plot of drag coefficient versus mass')
```

The window also shows a toolbar with various editing and debugging tools, a stack window, and a command window. The status bar at the bottom indicates the current line is 6, column is 52, and the overall status is "OVR".

SOLUTION OF CASE STUDY



ASSIGNMENT

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

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

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 [$\theta(x) = dy/dx$],

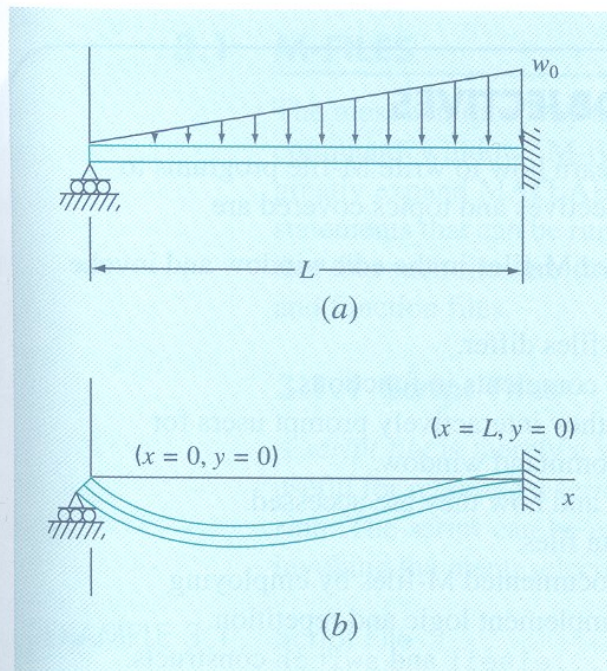


FIGURE P2.21

- (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/cm², $I = 30,000$ cm⁴, $w_0 = 2.5$ 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.

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.

<ftp://cdl.hanyang.ac.kr>

ID : cdl

PW : cdl

File name

학번_이름_week1

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 career at the National Physical Laboratory (NPL) in Teddington, outside London. Working on a simplified version of a sophisticated design by Alan Turing, Wilkinson and colleagues at NPL built the Pilot Automatic Computing Engine (ACE), one of Britain's first stored program digital computers. The Pilot ACE ran its first program in May 1950. Wilkinson 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

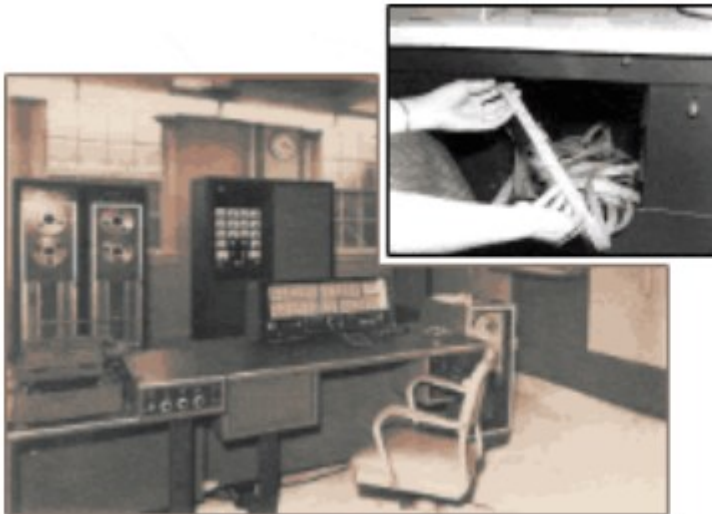
At about the same time, mathematicians at the Institute for Numerical Analysis (INA), a branch of the National Bureau of Standards, located at UCLA, were working with the Standards Western Automatic Computer (SWAC), one of the USA's first computers. Researchers at INA included George Forsythe, John Todd, 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

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

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.

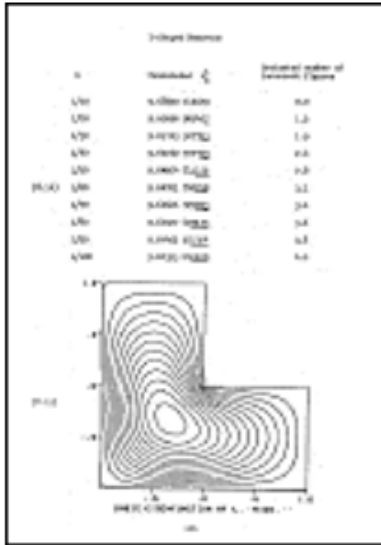
ORIGINS OF MATLAB

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.

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.

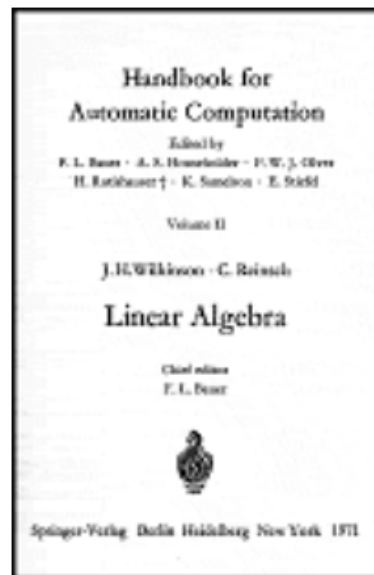


This 1967 textbook contained working code in Algol, Fortran, and PL/I.

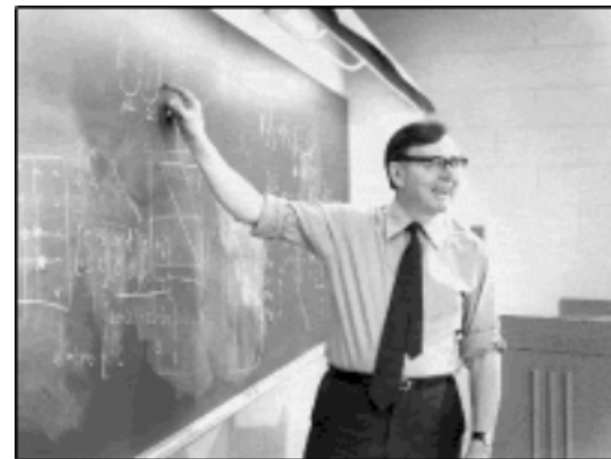
ORIGINS OF MATLAB

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.



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.](#)



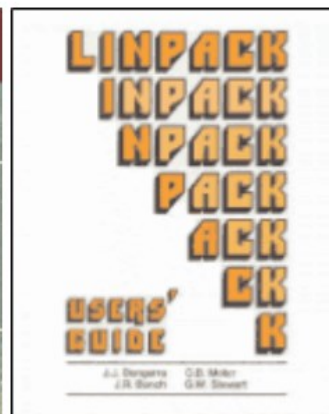
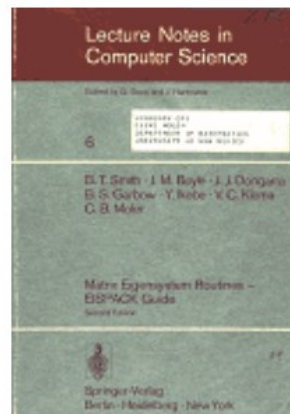
Wilkinson describing a matrix algorithm to an audience at Argonne in the early 1970s.

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.



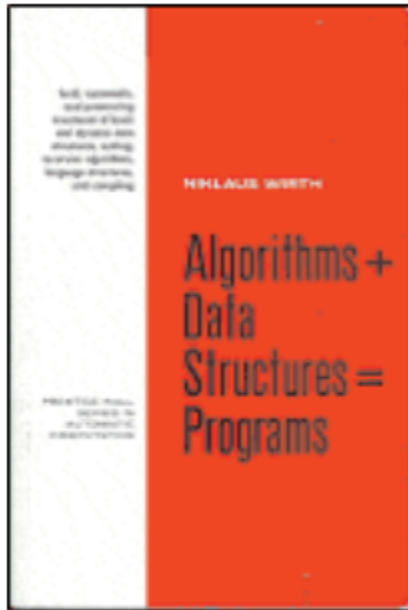
The authors of LINPACK: Jack Dongarra, Cleve Moler, Pete Stewart, and Jim Bunch in 1978.



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.