# Matlab Introduction

Dr. Antonio A. Trani
Professor
Dept. of Civil and Environmental Engineering

# Purpose of this Section

- To illustrate simple uses of the MATLAB™ Technical language
- To help you understand under what circumstances is MATLAB a better choice than spreadsheets and high-level languages
- To understand some of the MATLAB toolboxes used in specialized technical computation
- Just for the fun of learning something new (the most important reason)

™ trademark of the Mathworks (Natick, MA)

# What is MATLAB?

- A high-performance language for technical computing (Mathworks, 1998)
- Typical uses of MATLAB:
  - Mathematical computations
  - Algorithmic development
  - Model prototyping (prior to complex model development)
  - Data analysis and exploration of data (visualization)
  - Scientific and engineering graphics for presentation
  - Complex analysis using MATLAB toolboxes (i.e., statistics, neural networks, fuzzy logic, H-infinity control, economics, etc.)

# Why is MATLAB Good for Me?

- Because it simplifies the analysis of mathematical models
- It frees you from coding in high-level languages (saves a lot of time - with some computational speed penalties)
- Provides an extensible programming/visualization environment
- Provides professional looking graphs
- The learning curve of this language is moderate (my own bias)
- Our students learn the language in EF, Math and Physics. Perhaps we should exploit this fact in our junior and senior courses

# Where is MATLAB in the Scheme of Things?

Virginia Tech

Complimentary tool to spreadsheets and prog. languages

| Tool | My Remarks (subjective) |
|---|---|
| Spreadsheets (Excel) | • Easy to use<br>• Good for general purpose computation<br>• Nice standard graphics<br>• Good connectivity to other applications<br>• Platform independent |
| Numeric/Symbolic Tools (MATLAB, Mathematica/Mathcad) | • Moderate learning curve<br>• Good for general and scientific computations<br>• Excellent graphics<br>• Good connectivity to other applications<br>• Platform independent |
| Compiled Languages (C/C++) | • Require a fairly steep learning curve<br>• Best control over the development cycle<br>• Good graphics if a separate library is available<br>• Generally platform dependent |

# A Few More Facts About MATLAB

- MATLAB was created to be a numerical computation package (based on the LINPACK routines)
- MATLAB is usually faster than Mathematica and Maple in numeric intensive tasks
- MATLAB has more textbooks than other packages combined (850+ books). Perhaps this speaks on the acceptance by the user community
- Go to www.mathworks.com for a complete set of books on various subjects

# Tutorial Outline

- Basics of MATLAB (various modes of operation)
- Input-output commands
- Data analysis functions
- Matrices and vector operations
- Script files and programming issues
- Output graphics and plots (bar, 2D and 3D commands, interactive features)
- Numerical solutions to differential equations (queueing and dynamic system applications)
- Simulink and other MATLAB toolboxes (C compiler, Neural Networks, Statistics, etc.)

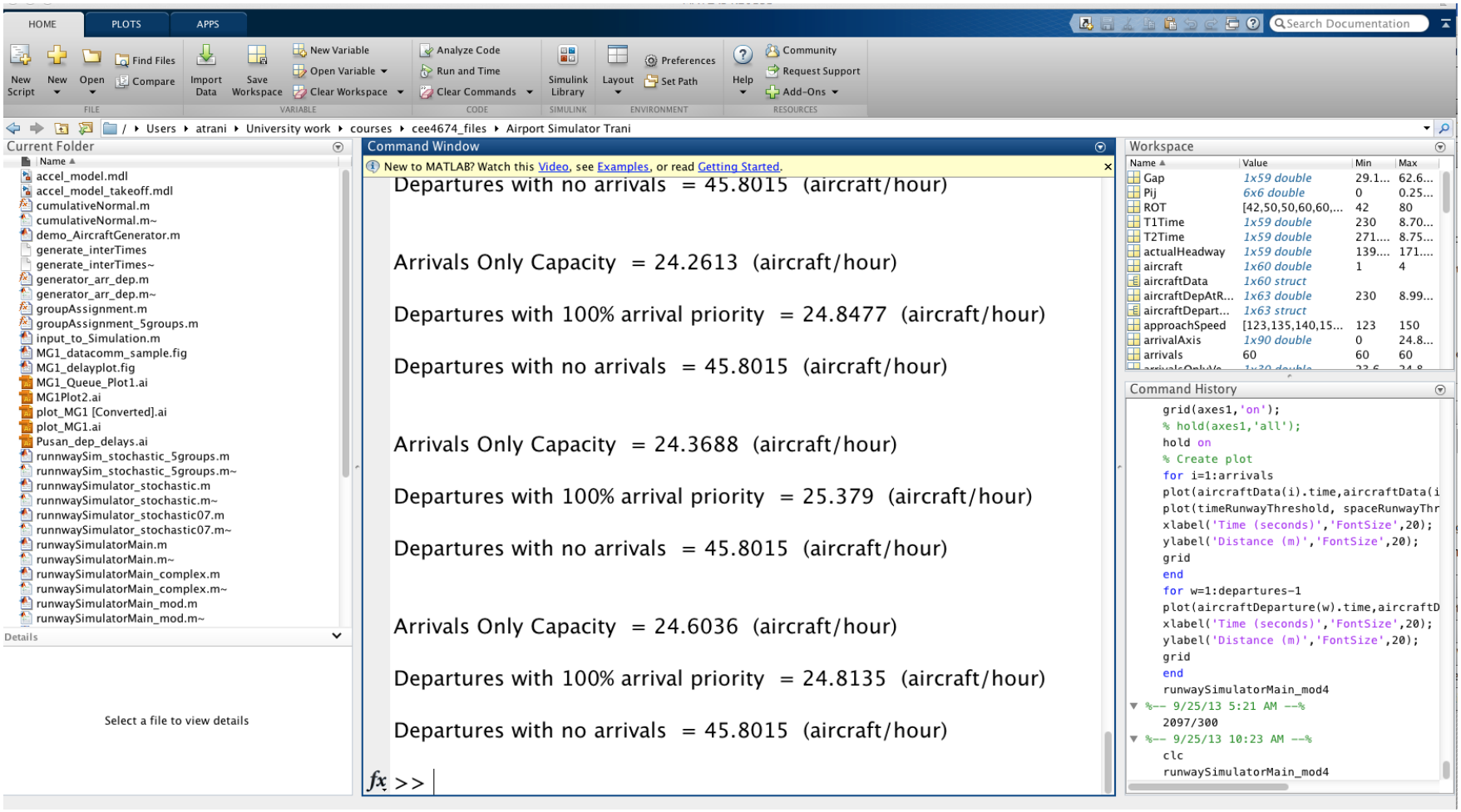
# Basics of the Technical Language

- MATLAB is a technical language to ease scientific computations
- The name is derived from MATrix LABoratory
- It provides many of the attributes of spreadsheets and programming languages
- MATLAB is a case sensitive language (a variable named “c” is different than another one called “C”)
- MATLAB can be used in interactive mode or in full compiled version (platform specific mode)
- In interactive mode MATLAB scripts are platform independent (good for cross platform portability)

# MATLAB Foundations

- MATLAB works with matrices
- Everything MATLAB understands is a matrix (from text to large cell arrays and structure arrays)
- Various data types exist within MATLAB
  - single precision
  - double precision
  - integer (8 bit)
- Performance of MATLAB scripts can be improved using vector operations (more on this later)
- MATLAB has advanced data structures including object-oriented programming functionality and overloadable operators

# The MATLAB Environment



# Basic Components of the MATLAB Environment

Virginia Tech

MATLAB has the following basic window components:

- Launch Pad Window
- to access all MATLAB services and toolboxes
- Command Window
  - to execute commands in the MATLAB environment
- Current Directory Window
  - to quickly access files on the MATLAB path
- Figure Window
  - to display graphical output from MATLAB code

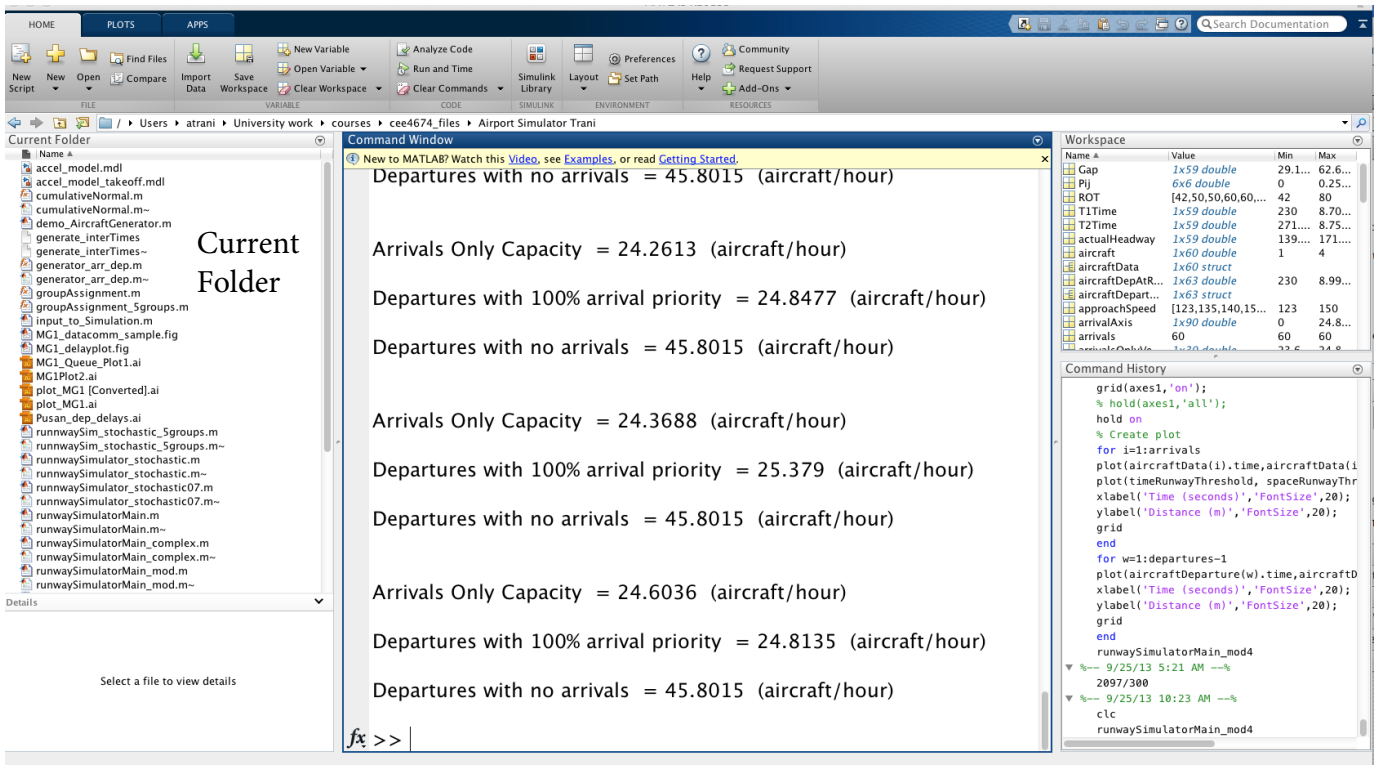
# Basic Components of the MATLAB Environment

Virginia Tech

- Workspace Window
  - to view variable definitions and variable memory allocations
- M-File Editor/Debugger Window
  - to write M-files (includes color-coded syntax features)
  - to debug M-files interactively (break points)
- MATLAB Path Window
  - to add and delete folders to the MATLAB path
- Command History Window
  - displays all commands issued in MATLAB since the last session (good for learning and verification)

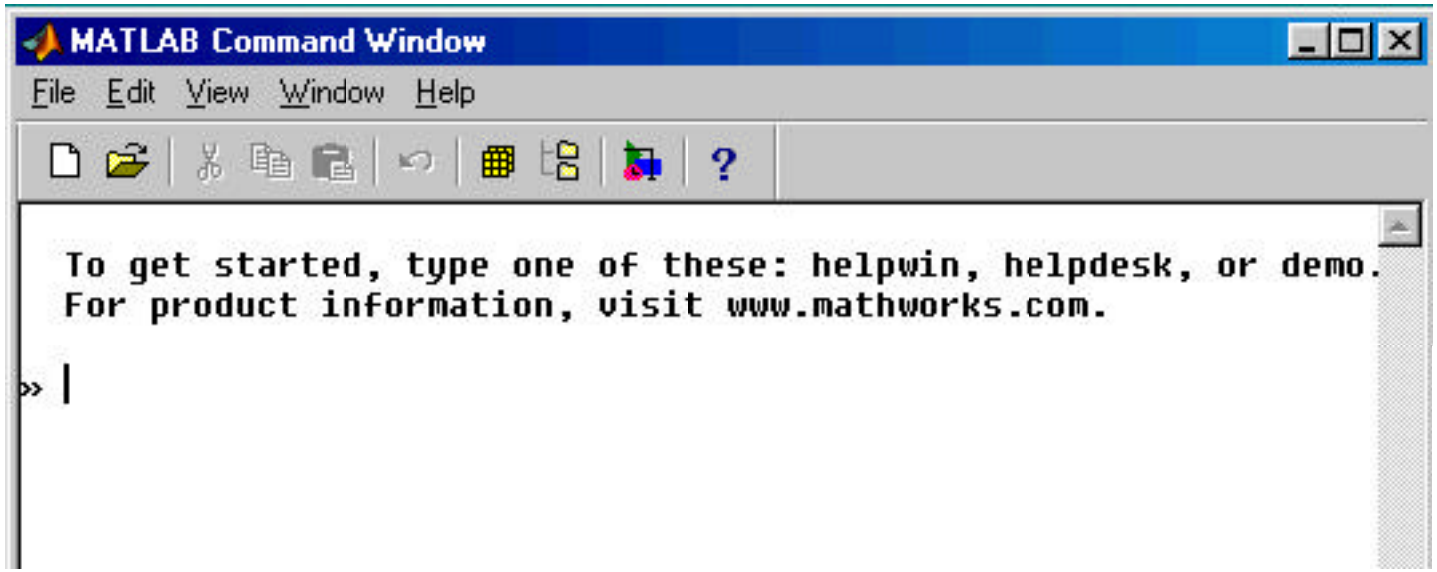
# Composite MATLAB Window Environment

- A Java-based GUI environment allows you to easily navigate between various windows



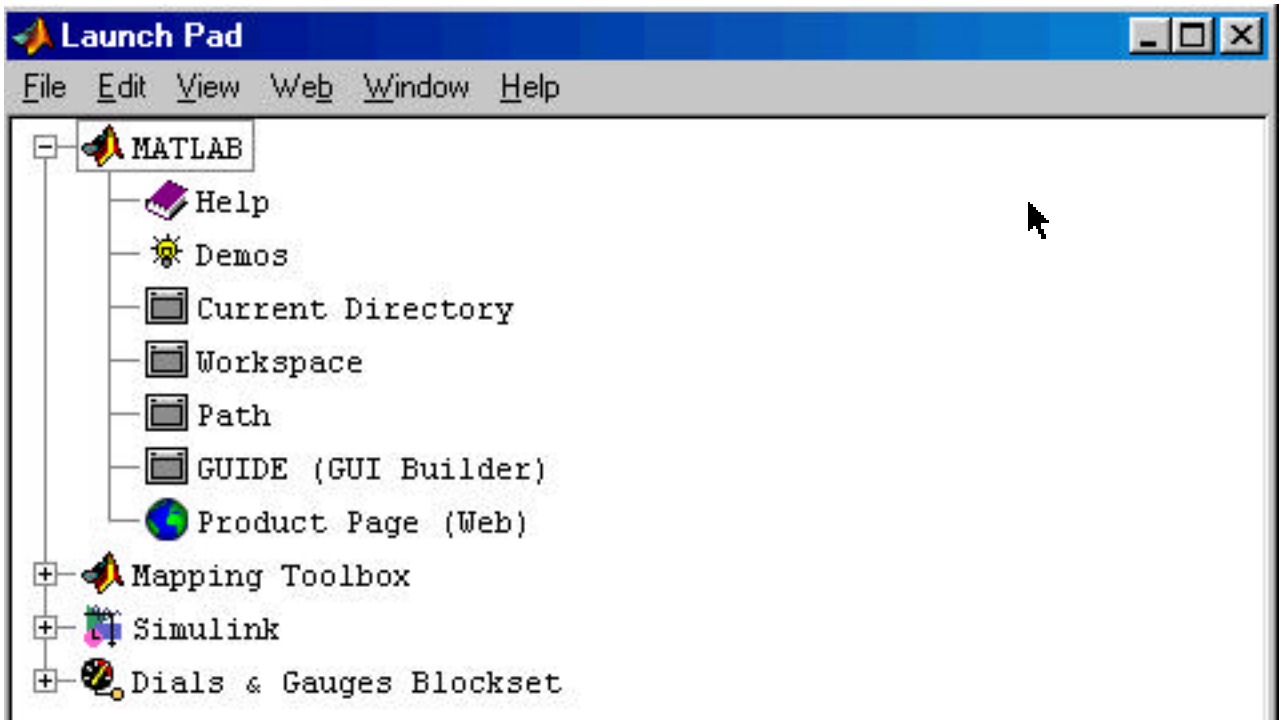
# MATLAB Command Window

- The command window allows you to interact with MATLAB just as if you type things in a calculator
- Cut and paste operations ease the repetition of tasks
- Use ‘up-arrow’ key to repeat commands (command history)



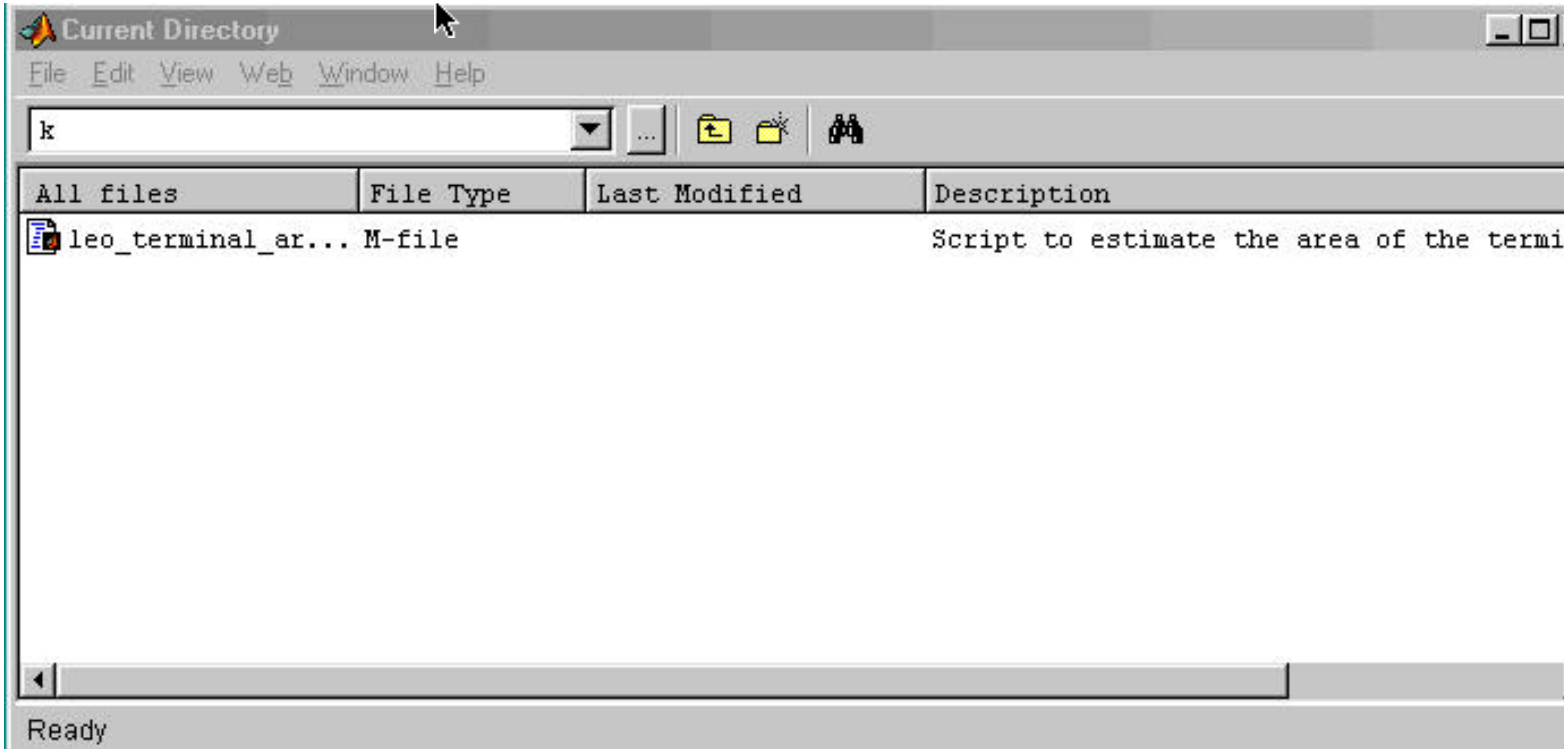
# MATLAB Launch Pad Window

- The launch window allows you to quickly select among various MATLAB components and toolboxes
- Shown below are MATLAB and three installed toolboxes in the launch window environment



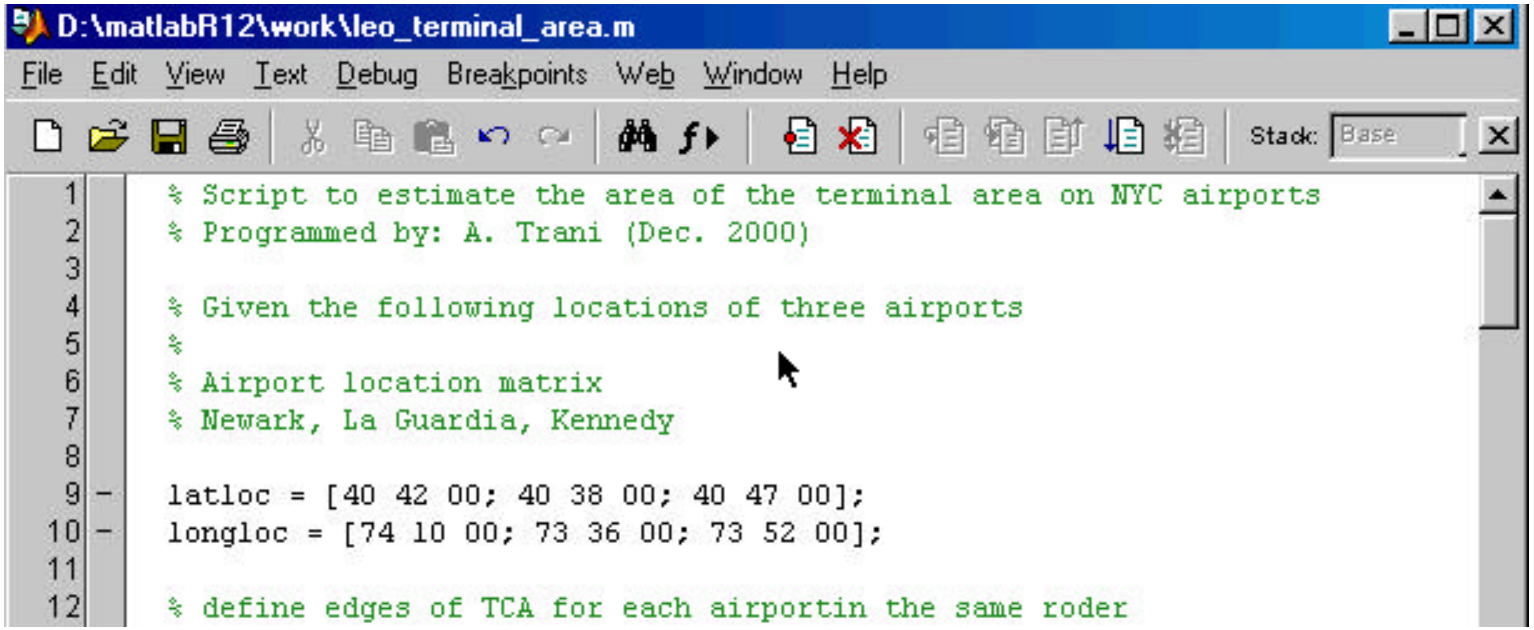
# MATLAB Current Directory Window

- Provides quick access to all files available in your Path
- Provides a brief description (when files are commented out) of each M-file



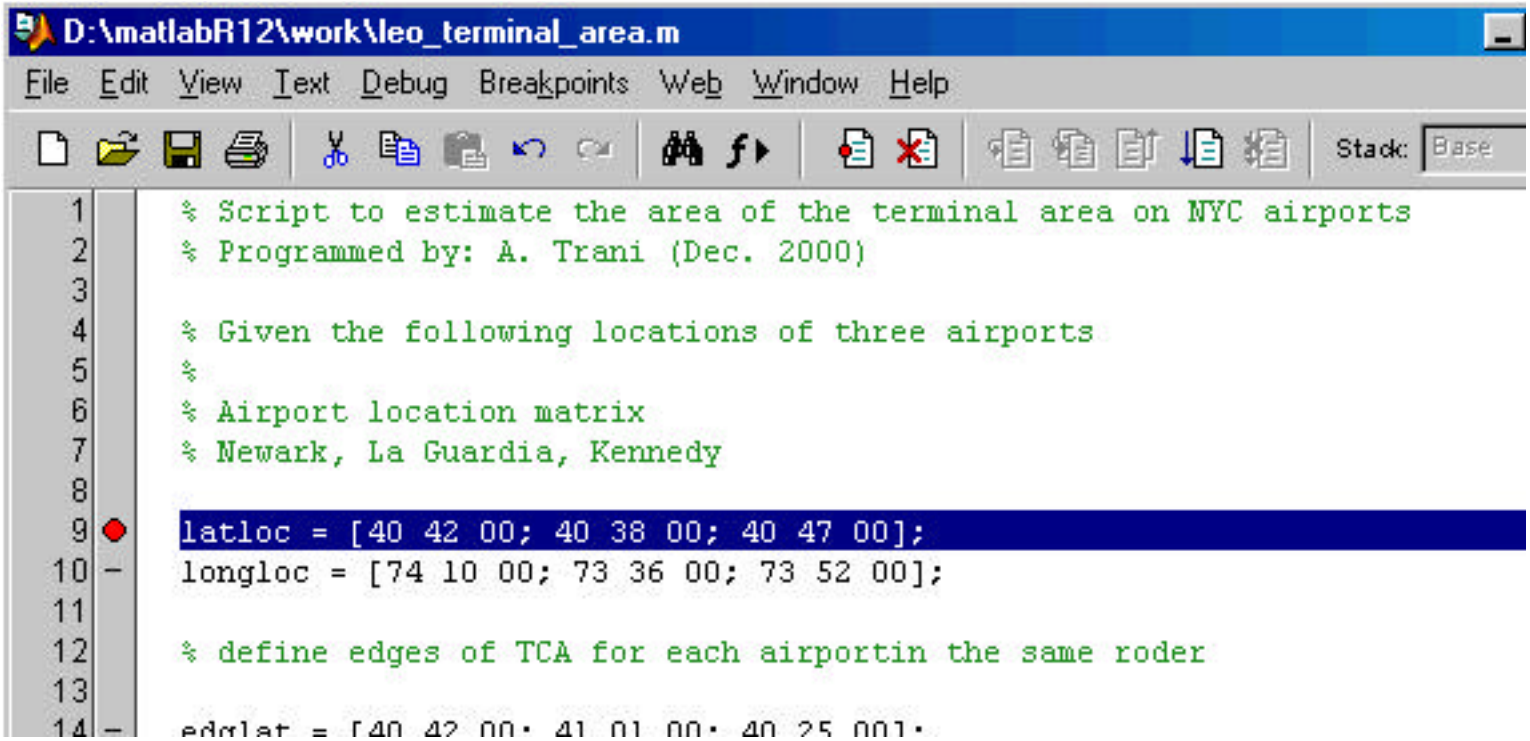
# MATLAB Editor/Debuger Window

- Provides the same functionality found in most programming language development environments
  - Color codes MATLAB built-in functions (blue color)
  - Easy access to cut, paste, print, and debug operations
  - Checks balance in MATLAB function syntax



# MATLAB Editor/Debugger

MATLAB has an interactive debugger to help you step through your source code. This debugger has many of the same functional features found in high-level programming languages (i.e., FORTRAN, C/C++, etc.).

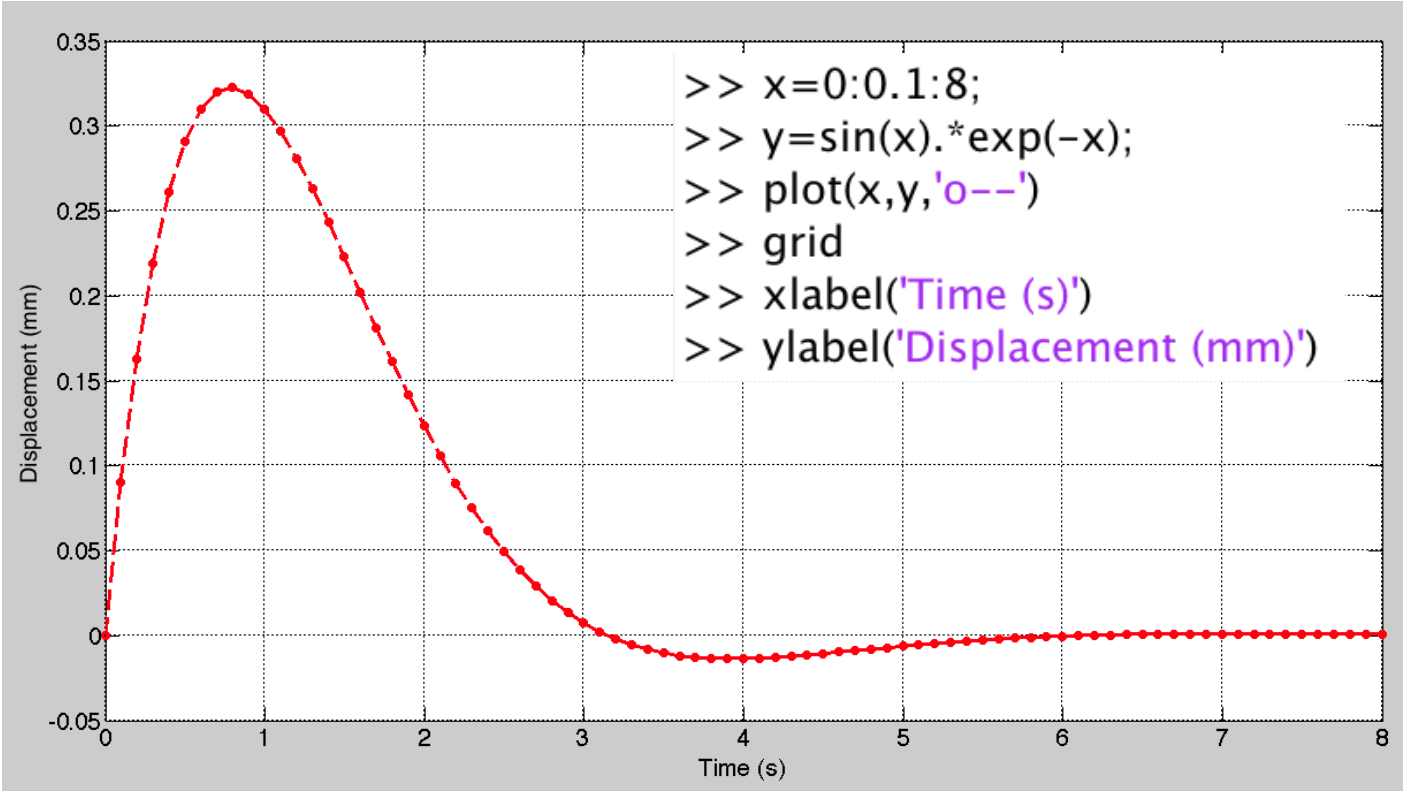


# MATLAB Debugger

- Allows standard programming techniques such:
  - Breakpoints
  - Break on error, warnings and overflows
  - Step in and out of script
  - Function dependencies

# MATLAB Figure Window

- Displays the graphic contents of MATLAB code (either from Command Window, an M-file, or output from MEX file)



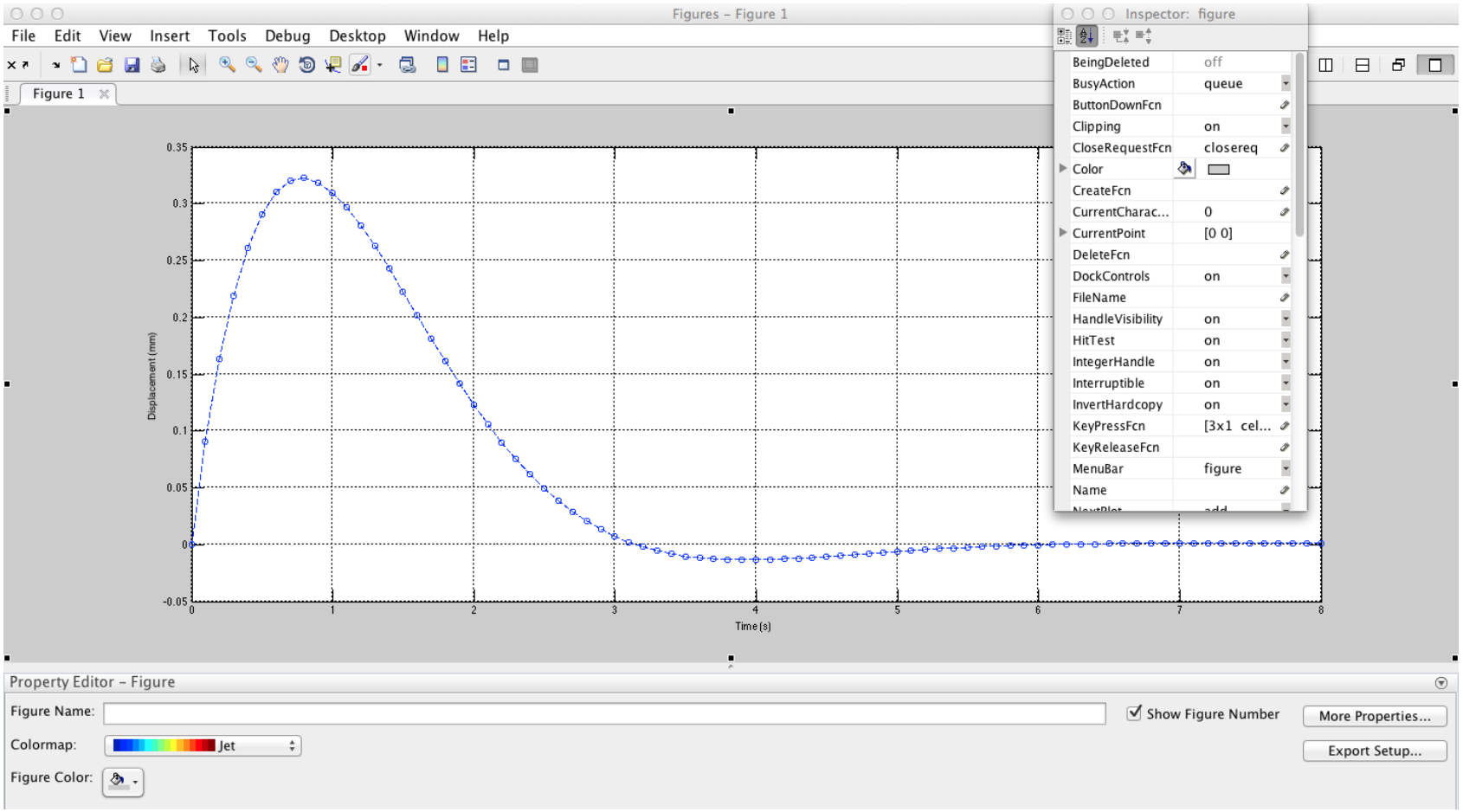
# MATLAB Figure Window (cont.)

Figure properties can be changed interactively using the following commands:

- PlotEdit
  - allows interactive changes to plots (add legend, lines, arrows, etc.)
  - This function is automatically invoked in MATLAB 5.3
- PropEdit
  - Allows changes to all Handle Graphic properties in a MATLAB plot
  - Requires knowledge of Handle Graphics (more on this later)

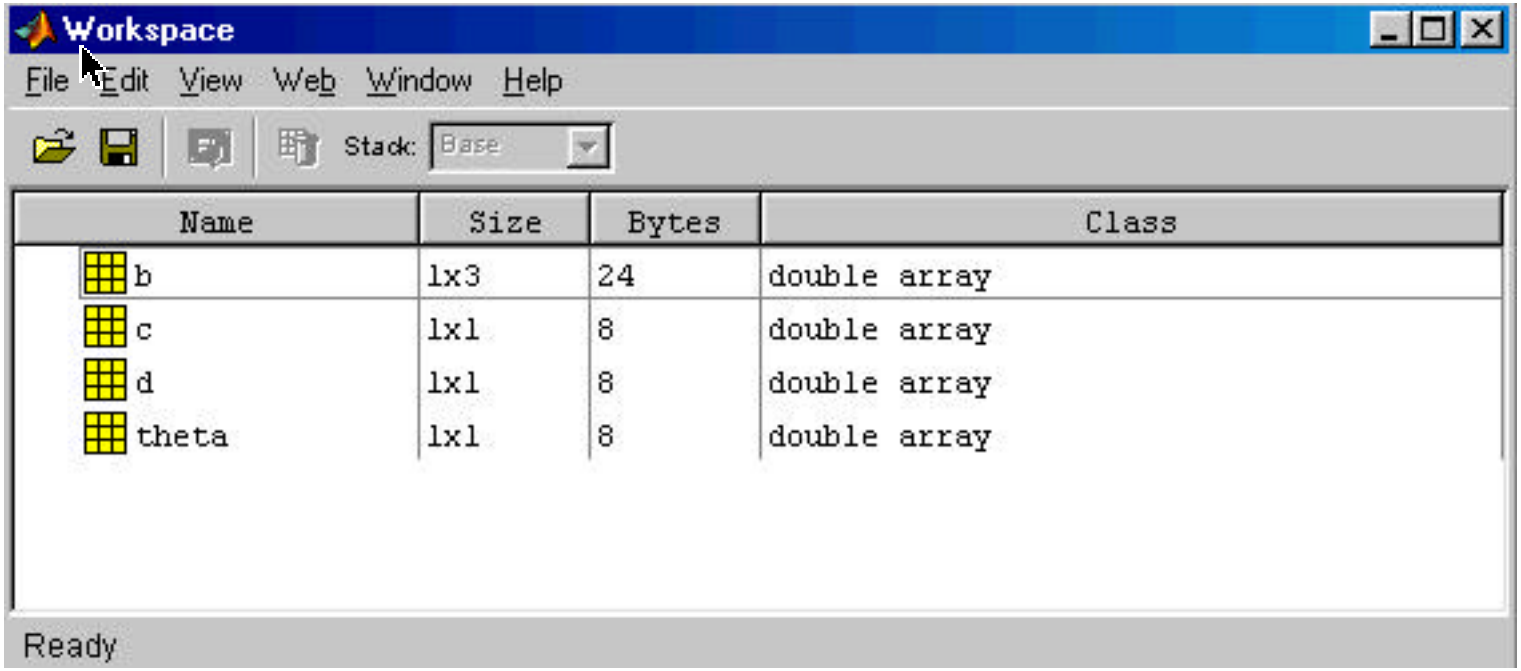
# MATLAB Figure Property Editor

- Propedit : Allows you to change properties of a plot



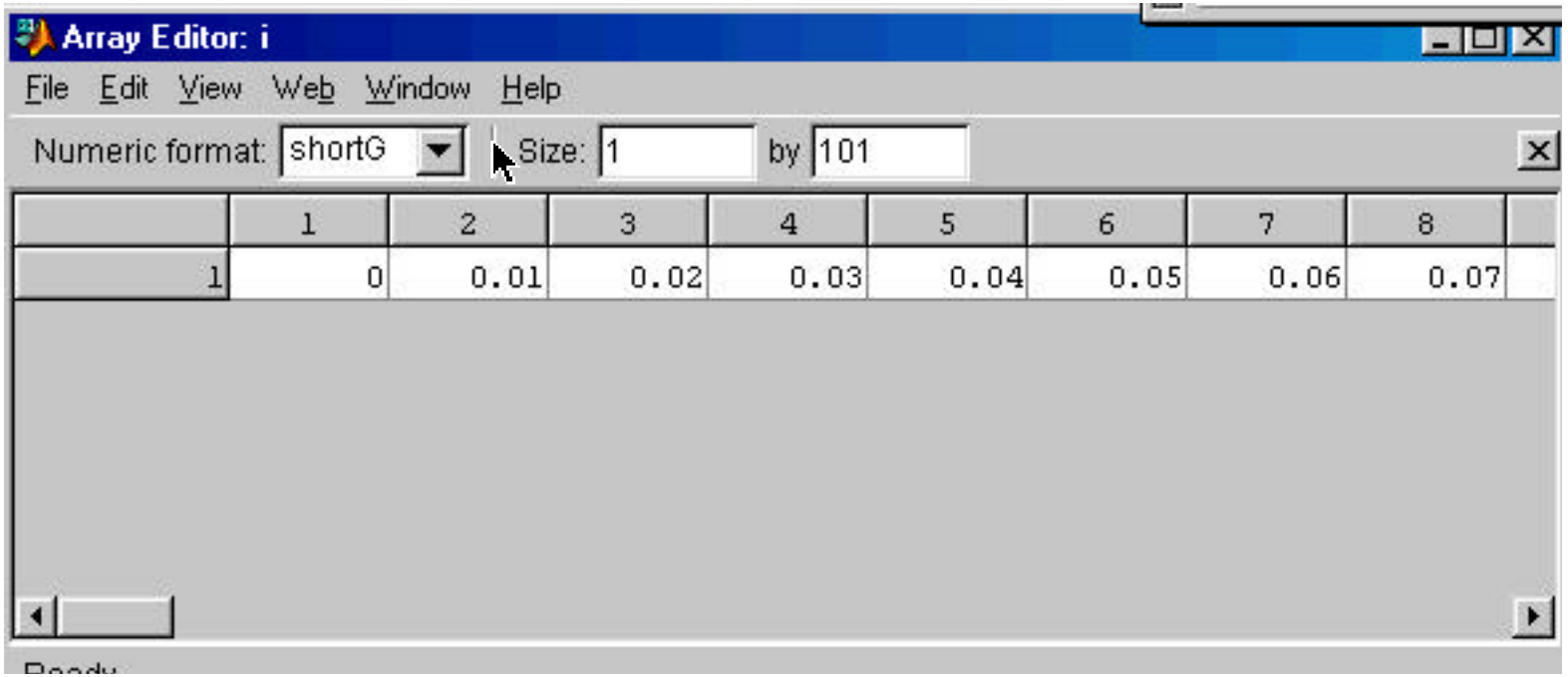
# MATLAB Workspace

As you develop and execute models in MATLAB the workspace stores all variables names and definitions for you. All variables are usually available to you unless the workspace is clear with the '>>clear' command.



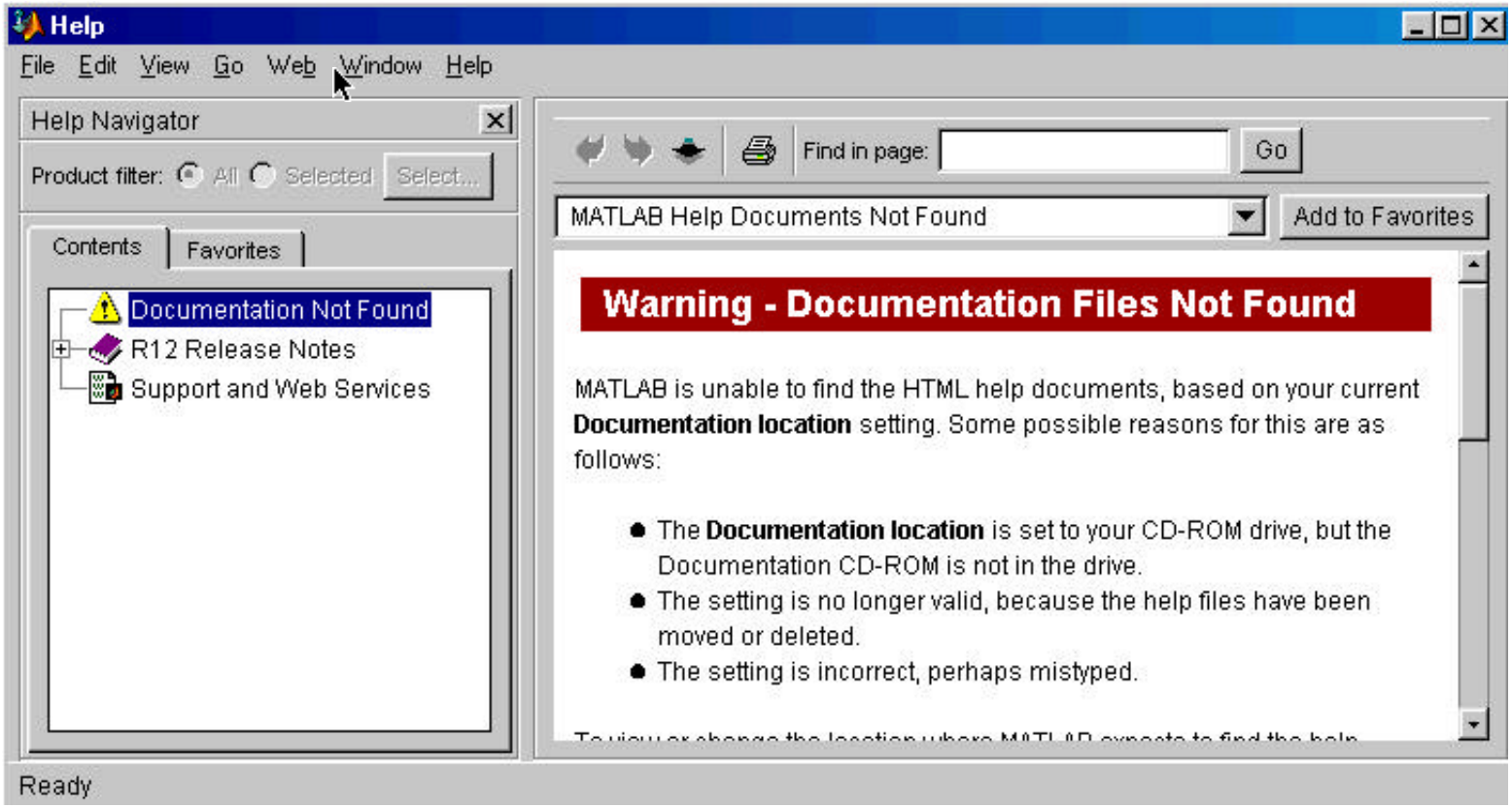
# Array Editor of Workspace Variables

- The workspace window allows you to inspect (and modify) variables in a spreadsheet-type window
- Cut and paste operations from the clipboard are also permitted from other applications

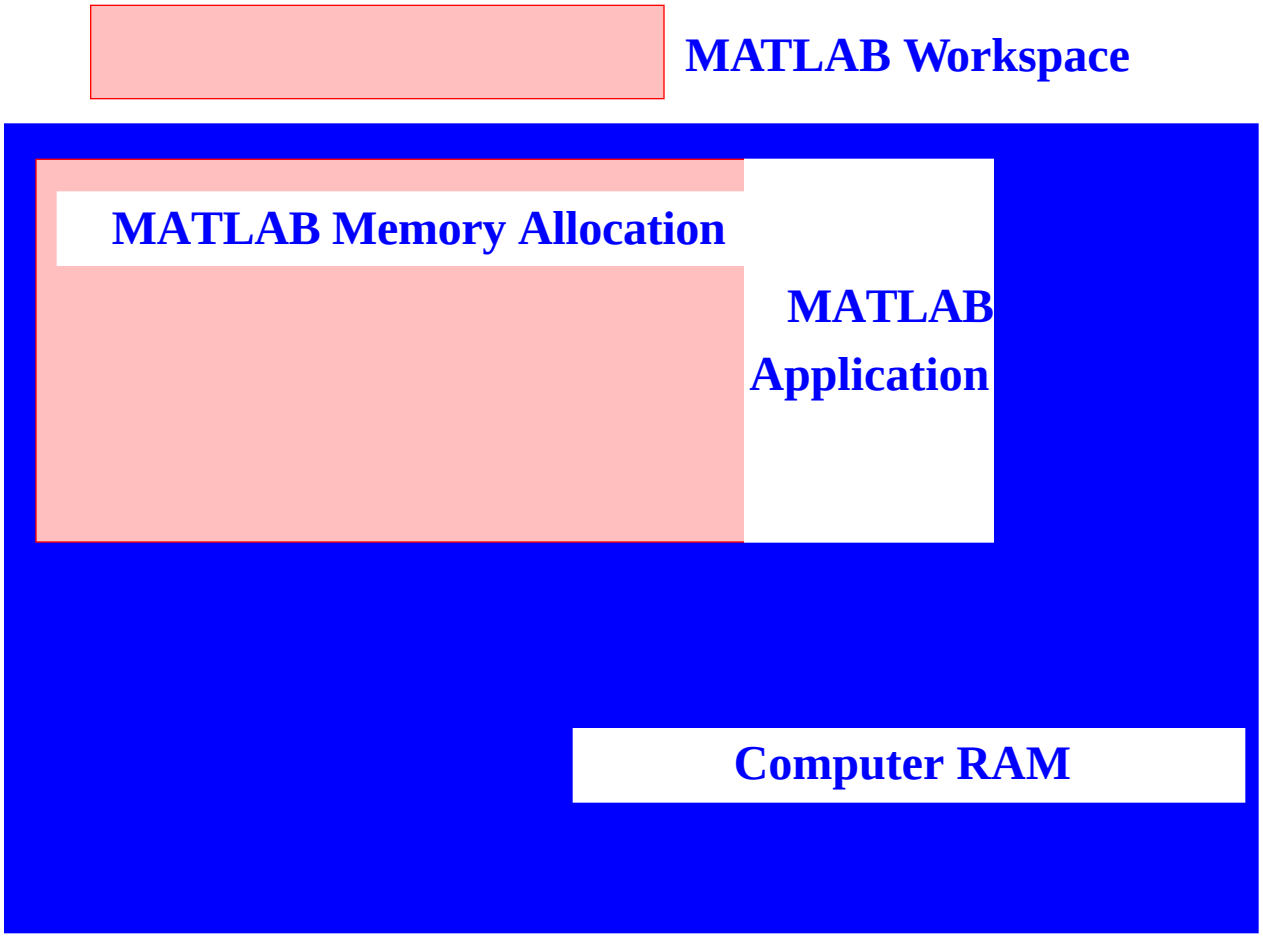


# Matlab Help Window

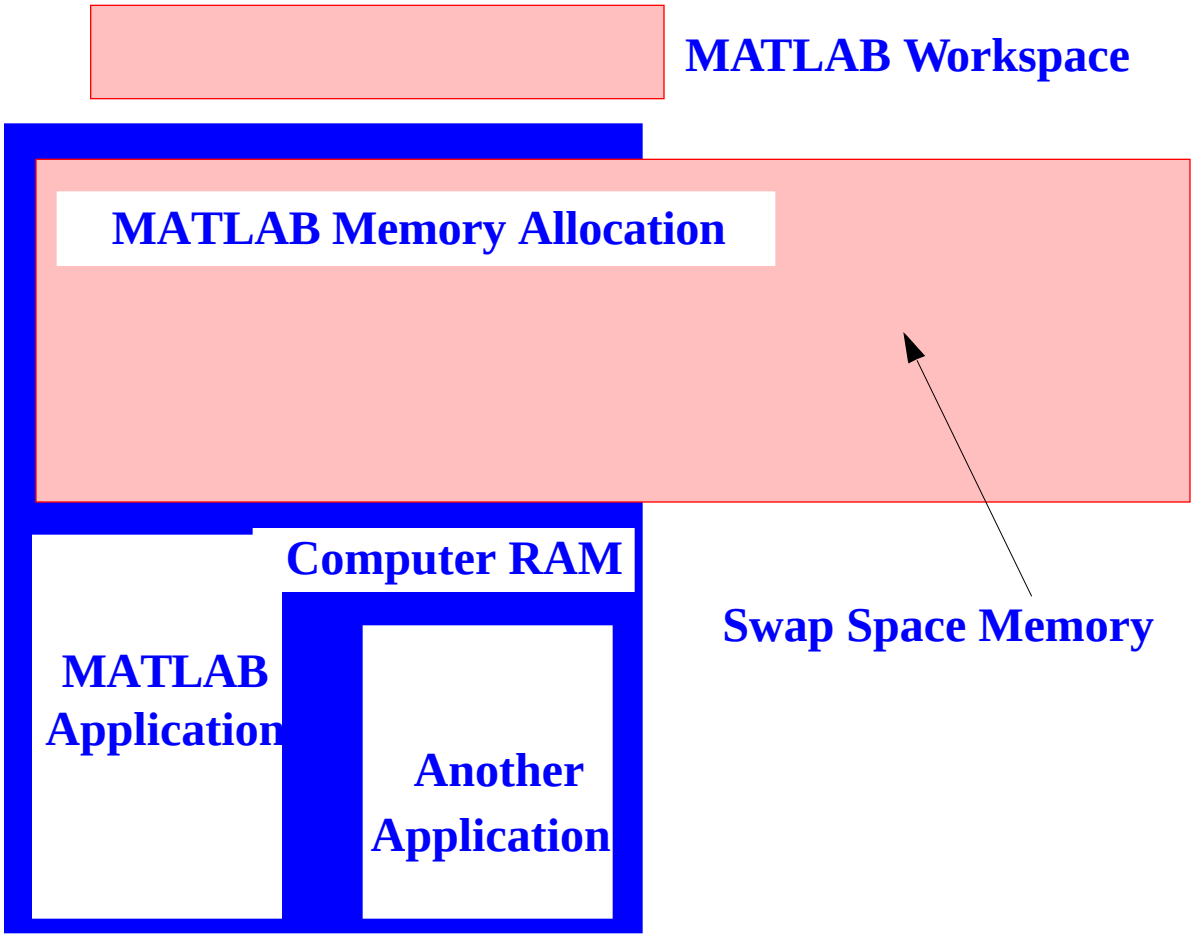
- Provides acces to various help files (both internal and on-line files available on the web)



# MATLAB Workspace (Macintosh Model)

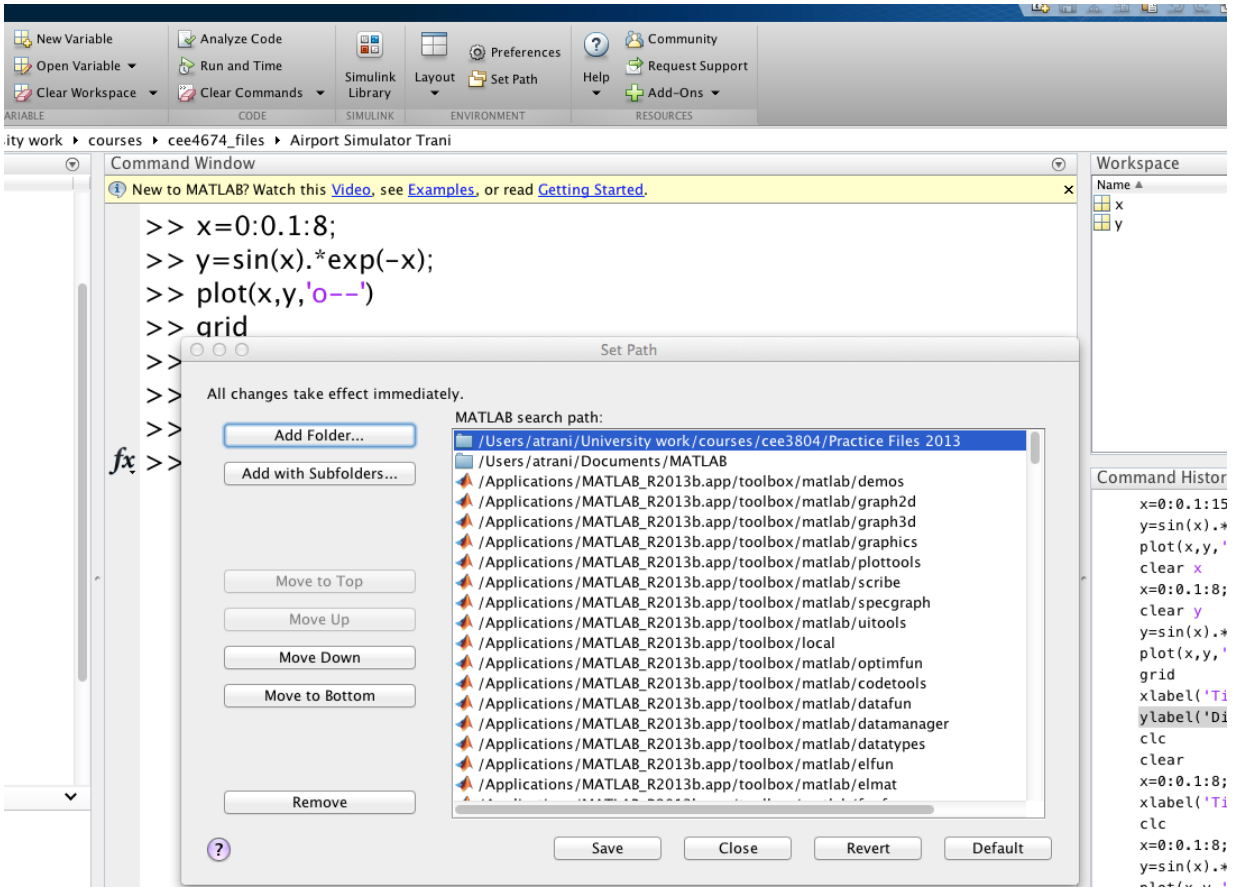


# MATLAB Workspace (Windows/UNIX Models)



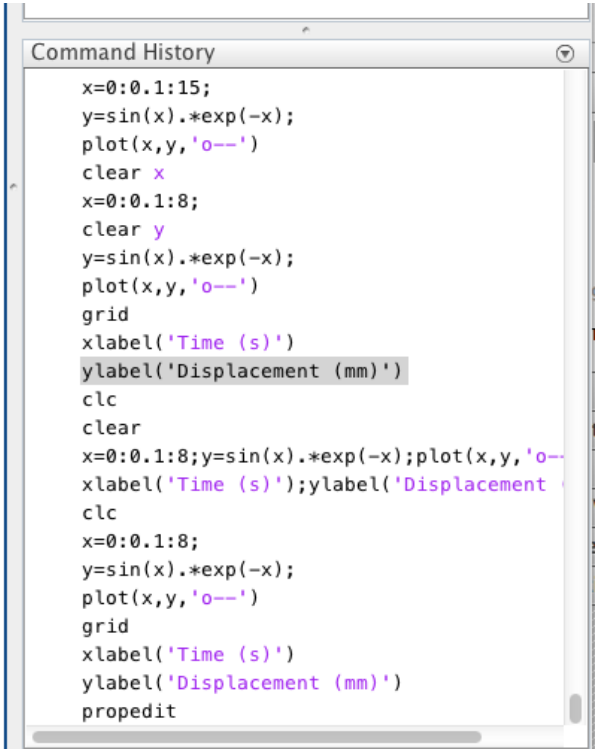
# MATLAB Path Window

- Shows all folders contained in the MATLAB path
- Allows you to include other folders from within MATLAB can be executed



# MATLAB Command History Window

- Displays all previous commands issued in a MATLAB session
- Good to verify computation sequences and for learning



# Interacting with MATLAB

There are several options to interact with MATLAB

| Mode | Remarks |
|---|---|
| Command line | • Interactive mode<br>• Good for quick computations or changes |
| M-files (script files) | • Semi-interactive mode<br>• Good to prototype small to complex models<br>• Used most of the time<br>• Platform independent |
| Executable MEX files | • Require a C/C++ compiler<br>• Fastest to execute<br>• Platform specific (target specific) |

# Interactive Mode (I)

- Use the MATLAB Command Window to interact with MATLAB in “calculator” mode

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

Try this out

- Multiple commands can be executed using the semi-colon “;” separator between commands

```
>> a=[3 2 4; 4 5 6; 1 2 3] ; b=[3 2 5]’ ; c=a*b
```

This single line defines two matrices (a and b) and computes their product (c)

# Interactive Mode (II)

- Use the semi-colon ";" separator to tell the MATLAB to inhibit output to the Command Window

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

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

Try this and see the difference

- Note that the semi-colon is also used to differentiate between rows in a matrix definition
- All commands that can be executed within the MATLAB Command Window

# General Purpose Commands

| | |
|---|---|
| helpwin | help window with hypertext navigation |
| demo | runs MATLAB demos from a MATLAB created Graphic User Interface (GUI) |
| helpdesk | troubleshooting with hypertext navigation |
| ver | tells you the version of MATLAB being used |
| who | lists all variables in the current workspace |
| whos | lists all variables in the workspace including array sizes |
| clear | clears all variables and functions from memory |

# General Purpose Commands (cont.)

| | |
|---|---|
| pack | consolidates workspace memory |
| load | load workspace variables from disk (from a previous session) |
| save | saves all variables and functions in the workspace to disk |
| quit | quits MATLAB session |
| what | lists MATLAB files in directory |
| edit | edits a MATLAB M-file |
| diary | save text of MATLAB session |

# Operating System Commands that Work in MATLAB

| | |
|---|---|
| cd | changes directory |
| copyfile | copy a file |
| dir | lists files in current directory |
| pwd | displays the working directory and its full path |
| delete | delete a file |
| mkdir | make a directory |
| dos | execute DOS command and return result |
| unix | execute UNIX command and return result |

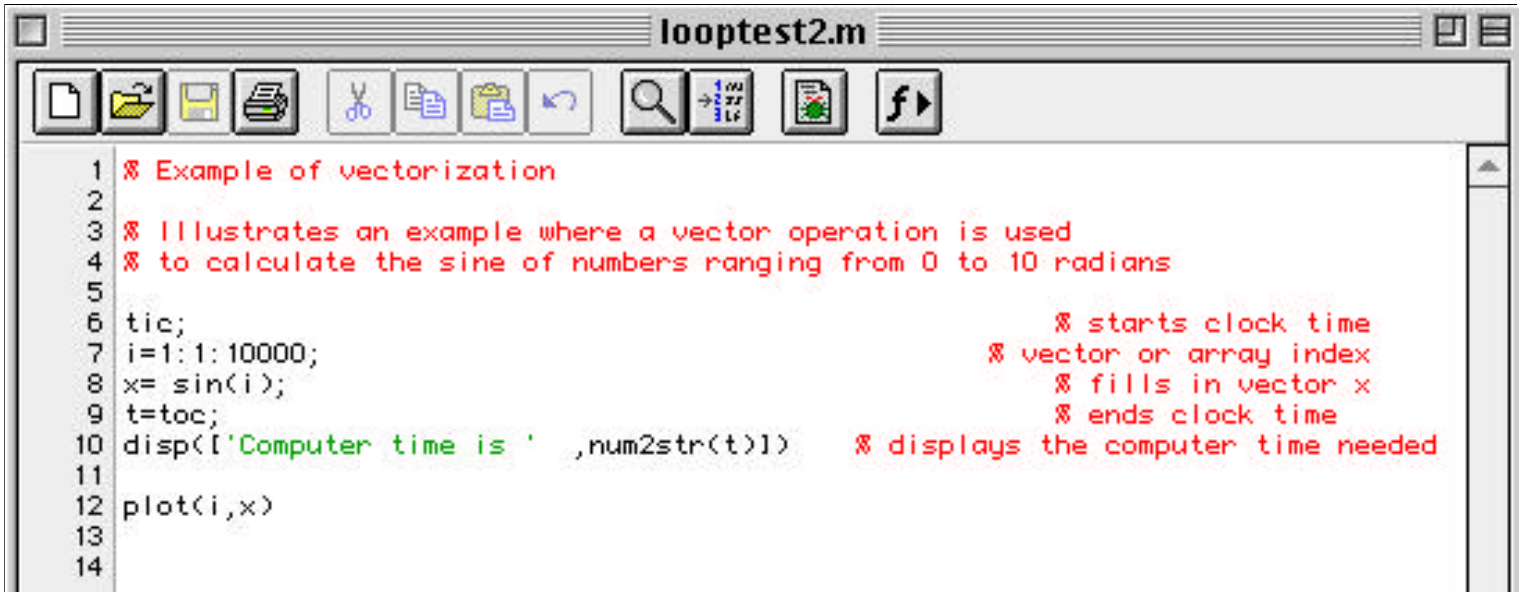
# Creating MATLAB Files

Two ways to interact with MATLAB:

- Interactive console mode - allows you to do computations and plots from the command line
- Through M-files - saves your “code” in a text file (with.m termination) allowing you to reuse any function or algorithm in it
- For this tutorial you will be working with M-files most of the time
- Other types of files in MATLAB are MAT (binary) and MEX (executable) files

# MATLAB M-Files

- They can be saved, refined and reused as needed
- These files end in ".m" in all platforms
- Use the MATLAB editor to accomplish this task
- Any wordprocessor can also be used (save files as text)



# Sample M-File

The following file generates random numbers

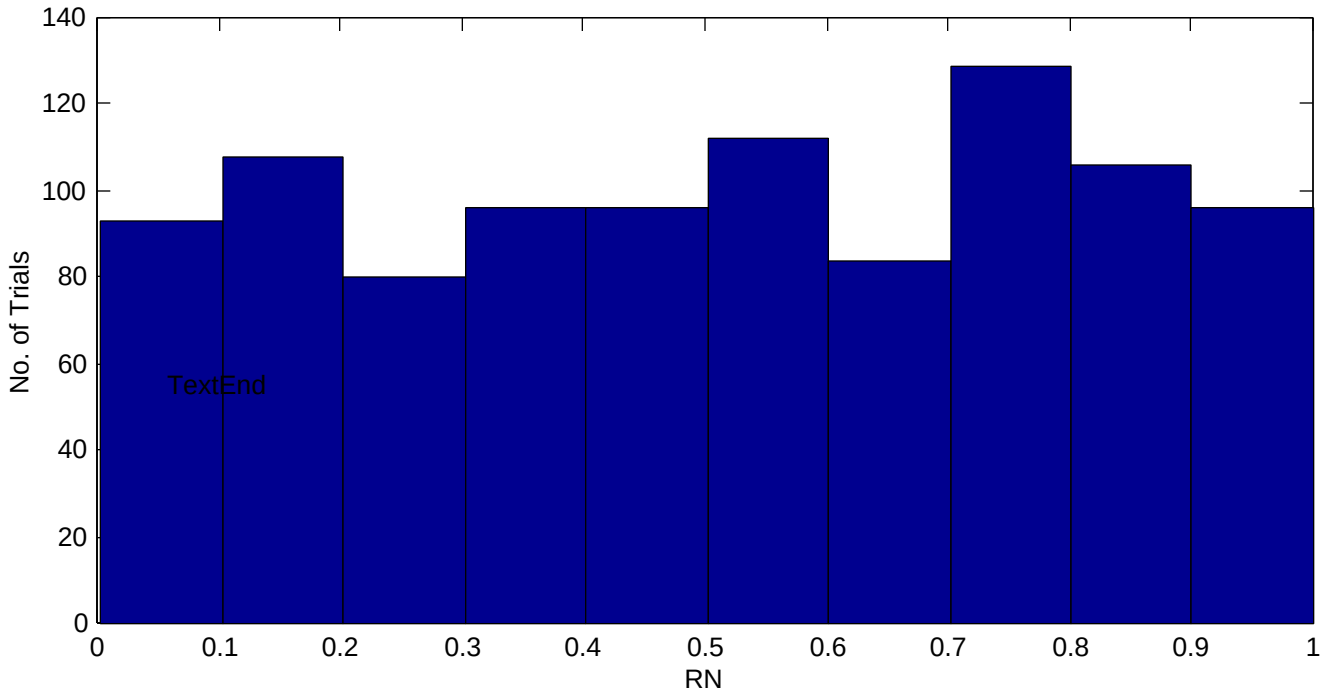
```
% Sample file to generate Random Numbers using
% MATLAB built-in functions

ntrials = 1000;                 % No. of trials to be simulated
i=1:1:ntrials;                  % defines a vector with 1k cells
RU(i) = rand(1,ntrials);        % uniform random number
                                % generator
RN(i) = randn (1,ntrials);      % normal random variate
                                % generator
hist(RU)                        % generates a histogram for
                                % variable RU
xlabel('RN')                    % adds the x-label to the plot
ylabel('No. of Trials')         % adds the y-label to the plot
```

# Executing the Sample M-File

- Type the previous file using the MATLAB Editor. Name and save the file as randem.m
- To execute the M-file type randem in the Command Window
- Or just go to **Run** from the **Debug** pull-down menu in the Editor/DebugWindow
- Alternatively (in the Mac OS) select the “Save and Execute” under the File menu
- Use the “up-arrow” key to go back to previous commands (cycle back through the MATLAB Command History)

# Output of randem.m

The following figure illustrates the output of randem.m



# Adding Comments to Your Code

It is a good practice to add comments to your source code.
Use the % operator to introduce comments in MATLAB

- Simplifies our task for code reviewing
- Easy to remember what you did in your code

# Few Tasks to Try on Your Own

1) Modify the randem.m M-file and plot a histogram of variable RN

2) Modify randem.m and plot the index variable i versus the values of RN and RU

- Use the plot command as follows:

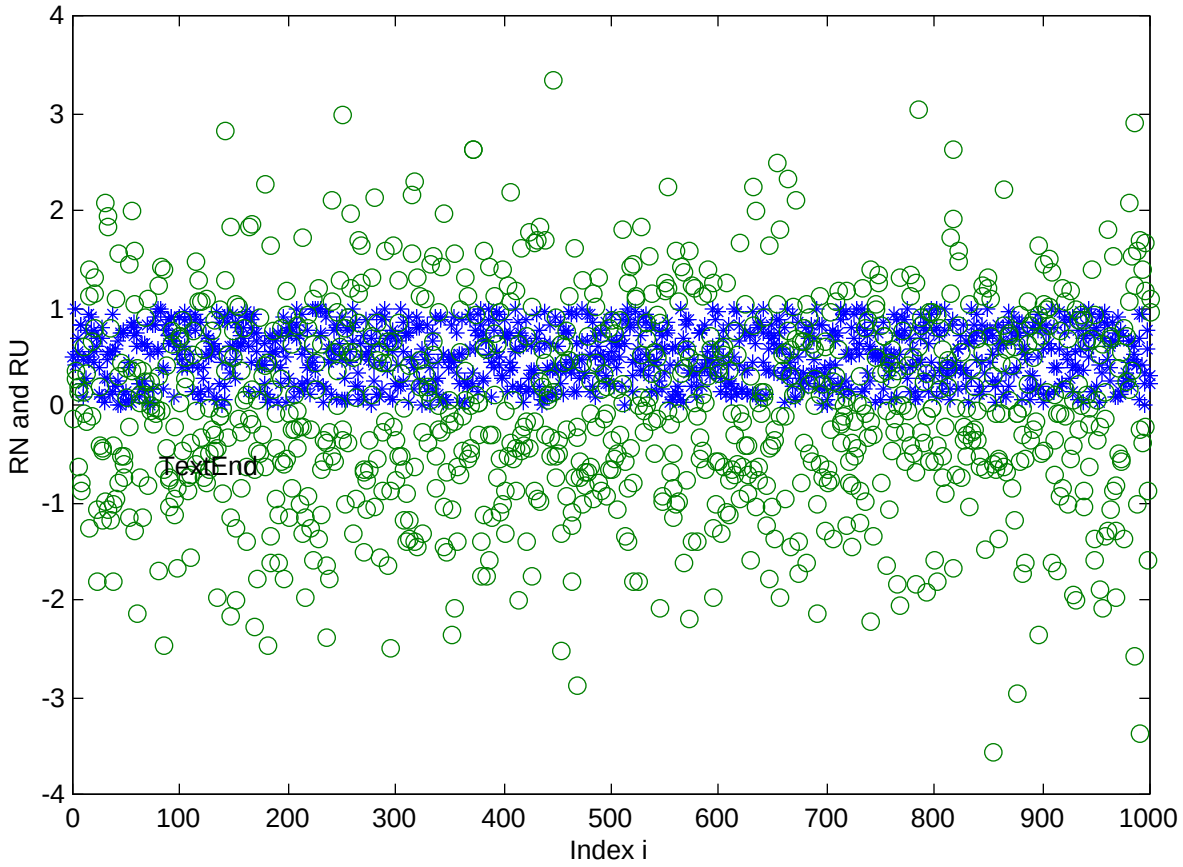
plot(x,y)

- where:
- x is the independent variable (index i in our case)
- y is the dependent variable (values of RU and RN)

3) From the Command Window execute the zoom command and select an area in the plot to view in more detail

# Plot of i vs. RN and RU

This plot shows index i versus the values of RU and RN



# MATLAB Binary Files

- These files are convenient to store information that needs to be reused
- MATLAB binary files end in .mat
- MATLAB mat files are platform independent
- Use the “save” command at the MATLAB command line.
  - save (saves all workspace variables to matlab.mat)
  - save fname (saves all workspace to fname.mat)
  - save fname x y (saves x and y to fname.mat)
  - save fname x y -ascii (saves x and y in 8-digit text format)
  - save fname x y -ascii -double -tabs (tab delimited format)

# Properties of Binary Files

Binary files are compact files only interpreted by MATLAB

- Good to store data to be reused later on
- Easy to transfer among PCs (compact size)
  - This works well across platforms
  - MATLAB 7/8 has good binary files backward compatibility
- Easy to retrieve and work with using the ‘load’command
- Fast retrieval

# Loading Binary Files

Binary files can be loaded simply issuing the 'load' MATLAB command.

Identified by .mat ending (e.g., traffic.mat)

For example if I want to load a file named traffic.mat (notice the termination) just invoke the load command and do not include the file type termination,

```
>>load traffic
>>who
>> observation density speed volume
>>
```

Note: that in this case the binary file has four variables

# Importing Data into MATLAB

There are several ways to enter data in MATLAB:

- Explicitly as elements of a matrix in MATLAB
- Creating data in an M-file
- Loading data from ASCII files
- Use the **Import Wizard** in MATLAB (7.0 version or later)
- Reading data using MATLAB's I/O functions (fopen, fread, etc.)
- Using specialized file reader functions (wk1read, imread, wavread, dlmread)
- Develop an MEX-file to read the data (if FORTRAN or C/C++ routines exist)

# Exporting Data from MATLAB

Virginia Tech

There are several ways to export data from MATLAB:

- Use the diary command (only for small arrays)
- ASCII (use the save command with '-ascii' option)
- Use the function dlmwrite to specify any delimiters needed
- Save data to a file in any specific format (use fopen, fwrite and other MATLAB I/O functions)
- Use specialized MATLAB write functions such as:
  - dlmwrite (user-defined delimeter ascii file)
  - wk1write (spreadsheet format)
  - imwrite and so on

# Importing Capabilities (I)

Suppose that we have a data file (called ohare_schedule) containing a typical schedule of daily aircraft operations at Chicago O'Hare Intl Airport. The information provided includes:

1) column 1 = local time (hours)

2) column 2 = number of  arrivals per hour

3) column 3 = number of departures per hour

4) column 4 = total operations

This file can be treated as a (24x4) matrix

# Sample Data File (ohare_schedule)

The following represents a subset of the ohare_schedule data file

0 4 7 11
1 3 2 5
2 2 2 4
3 4 2 6
4 2 8 10
.....

# Importing Data with Matlab

| Procedure | Remarks |
|---|---|
| **Load Comand** | Good to import Matlab binary files (.mat files) |
| **Import Built-in Wizard** | Good for well-defined formatted data (numbers and strings) |
| **Textscan Command** | Good for well-defined formatted data (numbers and strings) |
| **Xlsread** | Import Excel files directly |

# Importing Data with Matlab (2)

| Procedure | Remarks |
|---|---|
| **Xlsread** | Import Excel files directly |
| **fopen and fscanf Commands** | Low-level command to read more complex datasets |

# Reading the Sample Data File

Method 1 - Use the MATLAB load command

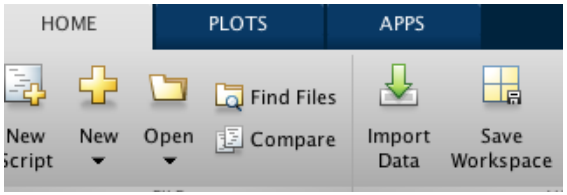
```
>> load ohare_schedule
```

- Loads the data file into the MATLAB Workspace and produces a new array variable called ohare_schedule
- This new array variable has dimensions 24 x 2
- All comment lines (if any) are neglected in the loading process. Only numerical data is read.

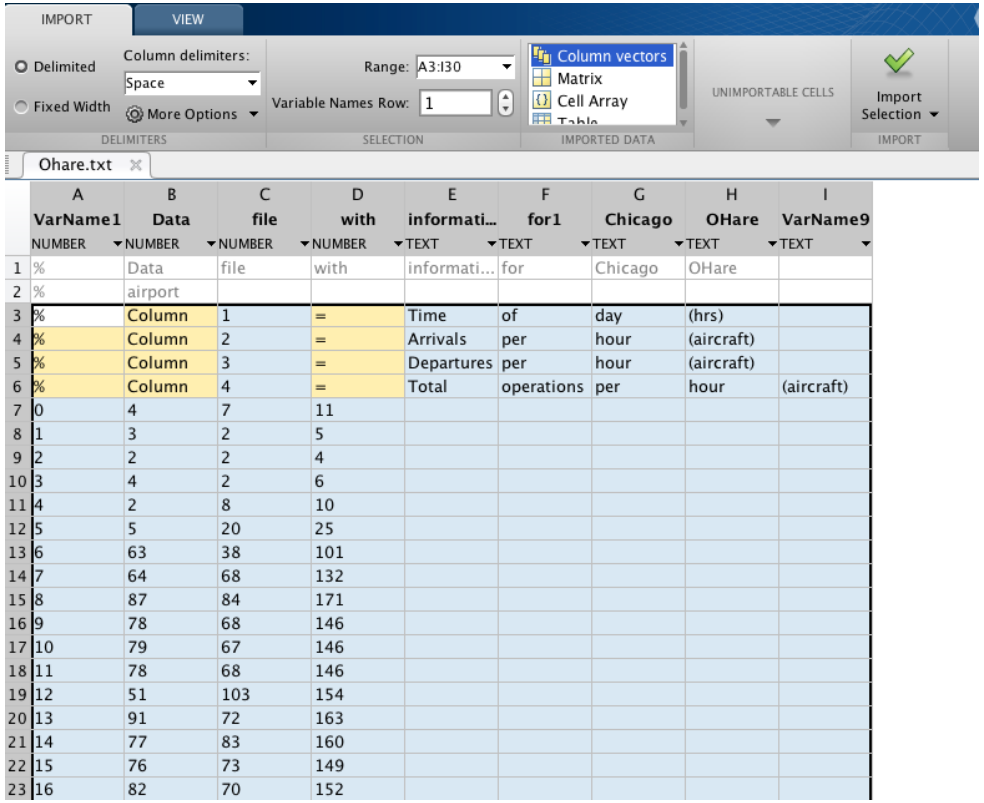
# MATLAB Import Screen (version 7.0)

Method 2 - To import data go to the Editor Window

- Select Import from the File pull-down menu

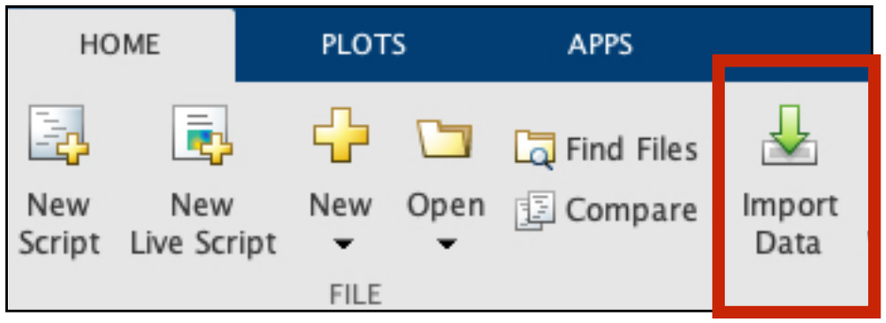



Import Command

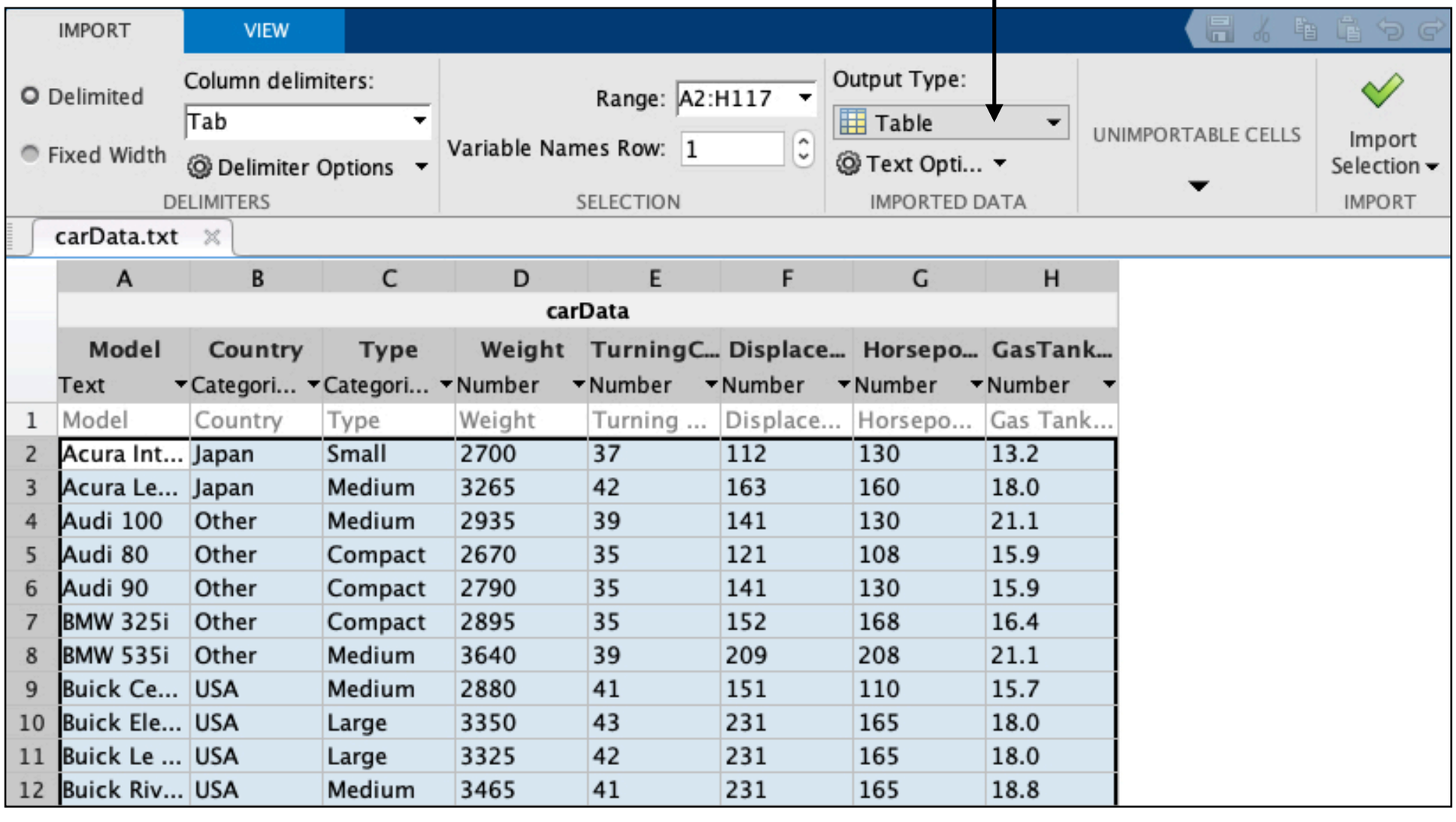



| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| | VarName1 | Data | file | with | informati... | for1 | Chicago | OHare | VarName9 |
| | NUMBER | NUMBER | NUMBER | NUMBER | TEXT | TEXT | TEXT | TEXT | TEXT |
| 1 | % | Data | file | with | informati... | for | Chicago | OHare | |
| 2 | % | airport | | | | | | | |
| 3 | % | Column | 1 | = | Time | of | day | (hrs) | |
| 4 | % | Column | 2 | = | Arrivals | per | hour | (aircraft) | |
| 5 | % | Column | 3 | = | Departures | per | hour | (aircraft) | |
| 6 | % | Column | 4 | = | Total | operations | per | hour | (aircraft) |
| 7 | 0 | 4 | 7 | 11 | | | | | |
| 8 | 1 | 3 | 2 | 5 | | | | | |
| 9 | 2 | 2 | 2 | 4 | | | | | |
| 10 | 3 | 4 | 2 | 6 | | | | | |
| 11 | 4 | 2 | 8 | 10 | | | | | |
| 12 | 5 | 5 | 20 | 25 | | | | | |
| 13 | 6 | 63 | 38 | 101 | | | | | |
| 14 | 7 | 64 | 68 | 132 | | | | | |
| 15 | 8 | 87 | 84 | 171 | | | | | |
| 16 | 9 | 78 | 68 | 146 | | | | | |
| 17 | 10 | 79 | 67 | 146 | | | | | |
| 18 | 11 | 78 | 68 | 146 | | | | | |
| 19 | 12 | 51 | 103 | 154 | | | | | |
| 20 | 13 | 91 | 72 | 163 | | | | | |
| 21 | 14 | 77 | 83 | 160 | | | | | |
| 22 | 15 | 76 | 73 | 149 | | | | | |
| 23 | 16 | 82 | 70 | 152 | | | | | |

# Matlab Import Wizard (Built-in Interactive Procedure)



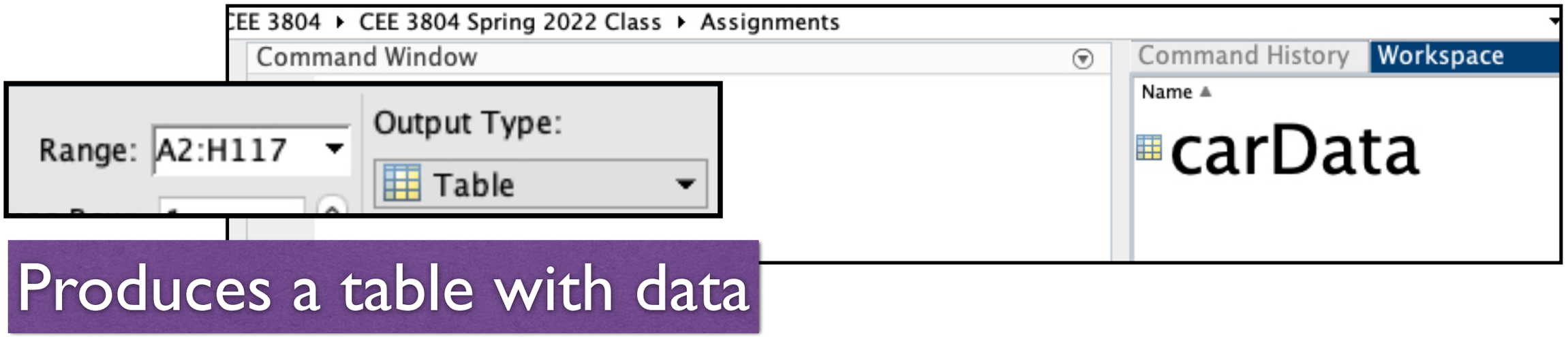

Select the output format



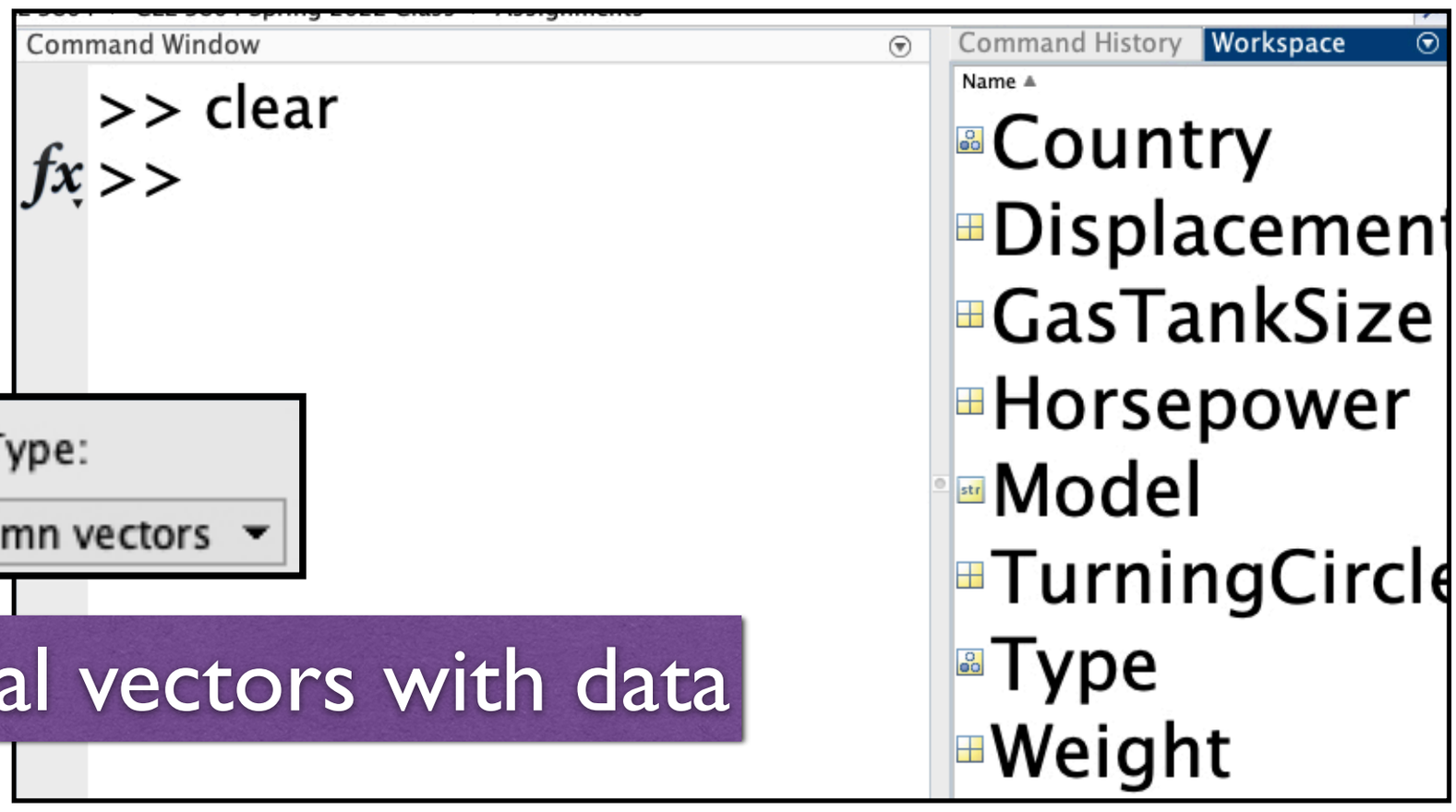

carData

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| | Model | Country | Type | Weight | TurningC... | Displace... | Horsepo... | GasTank... |
| | Text | Categori... | Categori... | Number | Number | Number | Number | Number |
| 1 | Model | Country | Type | Weight | Turning ... | Displace... | Horsepo... | Gas Tank... |
| 2 | Acura Int... | Japan | Small | 2700 | 37 | 112 | 130 | 13.2 |
| 3 | Acura Le... | Japan | Medium | 3265 | 42 | 163 | 160 | 18.0 |
| 4 | Audi 100 | Other | Medium | 2935 | 39 | 141 | 130 | 21.1 |
| 5 | Audi 80 | Other | Compact | 2670 | 35 | 121 | 108 | 15.9 |
| 6 | Audi 90 | Other | Compact | 2790 | 35 | 141 | 130 | 15.9 |
| 7 | BMW 325i | Other | Compact | 2895 | 35 | 152 | 168 | 16.4 |
| 8 | BMW 535i | Other | Medium | 3640 | 39 | 209 | 208 | 21.1 |
| 9 | Buick Ce... | USA | Medium | 2880 | 41 | 151 | 110 | 15.7 |
| 10 | Buick Ele... | USA | Large | 3350 | 43 | 231 | 165 | 18.0 |
| 11 | Buick Le ... | USA | Large | 3325 | 42 | 231 | 165 | 18.0 |
| 12 | Buick Riv... | USA | Medium | 3465 | 41 | 231 | 165 | 18.8 |

# Matlab Import Wizard
# (Built-in Interactive Procedure)

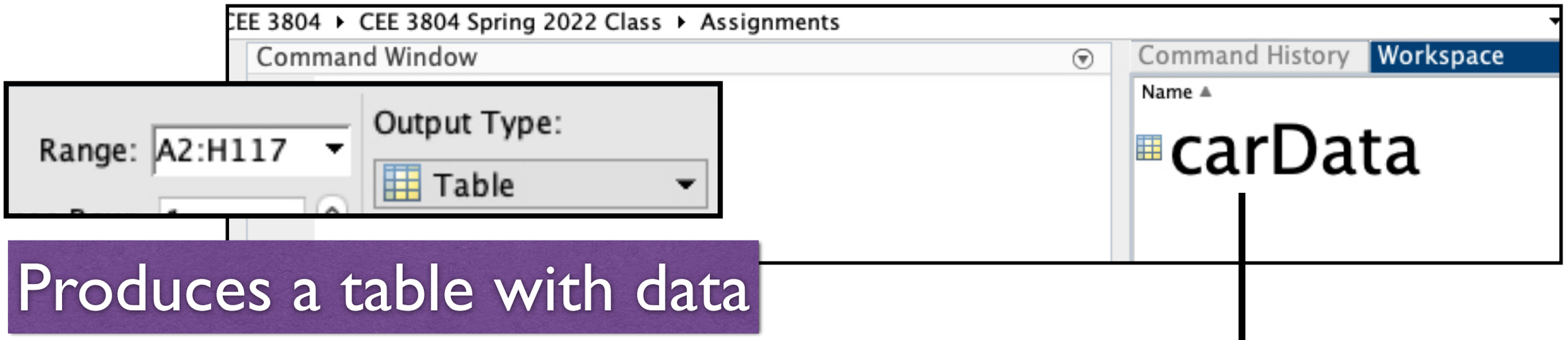



Produces a table with data




Produces individual vectors with data

# Matlab Import Wizard
# Handling Data Stored In Tables




Produces a table with data

carData

116x8 table

| | 1 Model | 2 Country | 3 Type | 4 Weight_lbs | 5 TurningCircle_ft |
|---|---|---|---|---|---|
| 1 | 'Acura Integra' | 'Japan' | 'Small' | 2700 | 37 |
| 2 | 'Acura Legen... | 'Japan' | 'Medium' | 3265 | 42 |
| 3 | 'Audi 100' | 'Other' | 'Medium' | 2935 | 39 |
| 4 | 'Audi 80' | 'Other' | 'Compact' | 2670 | 35 |
| 5 | 'Audi 90' | 'Other' | 'Compact' | 2790 | 35 |
| 6 | 'BMW 325i' | 'Other' | 'Compact' | 2895 | 35 |
| 7 | 'BMW 535i' | 'Other' | 'Medium' | 3640 | 39 |
| 8 | 'Buick Century' | 'USA' | 'Medium' | 2880 | 41 |
| 9 | 'Buick Electra ... | 'USA' | 'Large' | 3350 | 43 |
| 10 | 'Buick Le Sabr... | 'USA' | 'Large' | 3325 | 42 |

# Referencing to Data Stored In Tables

Example: reference all elements in two columns of the table carData

carData(:,1)

carData(:,4)

carData

116x8 table

| | 1 Model | 2 Country | 3 Type | 4 Weight_lbs | 5 TurningCircle_ft |
|---|---|---|---|---|---|
| 1 | 'Acura Integra' | 'Japan' | 'Small' | 2700 | 37 |
| 2 | 'Acura Legen... | 'Japan' | 'Medium' | 3265 | 42 |
| 3 | 'Audi 100' | 'Other' | 'Medium' | 2935 | 39 |
| 4 | 'Audi 80' | 'Other' | 'Compact' | 2670 | 35 |
| 5 | 'Audi 90' | 'Other' | 'Compact' | 2790 | 35 |
| 6 | 'BMW 325i' | 'Other' | 'Compact' | 2895 | 35 |
| 7 | 'BMW 535i' | 'Other' | 'Medium' | 3640 | 39 |
| 8 | 'Buick Century' | 'USA' | 'Medium' | 2880 | 41 |
| 9 | 'Buick Electra ... | 'USA' | 'Large' | 3350 | 43 |
| 10 | 'Buick Le Sabr... | 'USA' | 'Large' | 3325 | 42 |

# Define New Variables from Table Data

Creating a new variable using all the elements of the first column in carData

Produces a one-column table

```
carName = carData(:,1)
```

carData

116x8 table

| | 1 Model | 2 Country | 3 Type | 4 Weight_lbs | 5 TurningCirc |
|---|---|---|---|---|---|
| 1 | 'Acura Integra' | 'Japan' | 'Small' | 2700 | |
| 2 | 'Acura Legen... | 'Japan' | 'Medium' | 3265 | |
| 3 | 'Audi 100' | 'Other' | 'Medium' | 2935 | |
| 4 | 'Audi 80' | 'Other' | 'Compact' | 2670 | |
| 5 | 'Audi 90' | 'Other' | 'Compact' | 2790 | |
| 6 | 'BMW 325i' | 'Other' | 'Compact' | 2895 | |
| 7 | 'BMW 535i' | 'Other' | 'Medium' | 3640 | |
| 8 | 'Buick Century' | 'USA' | 'Medium' | 2880 | |
| 9 | 'Buick Electra ... | 'USA' | 'Large' | 3350 | |
| 10 | 'Buick Le Sabr... | 'USA' | 'Large' | 3325 | |

carName

116x1 table

| | 1 Model |
|---|---|
| 1 | 'Acura Integra' |
| 2 | 'Acura Legen... |
| 3 | 'Audi 100' |
| 4 | 'Audi 80' |
| 5 | 'Audi 90' |
| 6 | 'BMW 325i' |
| 7 | 'BMW 535i' |
| 8 | 'Buick Century' |
| 9 | 'Buick Electra ... |
| 10 | 'Buick Le Sabr... |
| 11 | 'Buick Riviera ... |
| 12 | 'Buick Skylark' |
| 13 | 'Cadillac Brou... |
| 14 | 'Cadillac De V... |

# Matlab can Create a Script to Import the Data

IMPORT | VIEW

Range: A2:H117 | Variable Names Row: 1 | SELECTION

Output Type: Table | Text Options | IMPORTED DATA

Replace | unimportable cells with | NaN | UNIMPORTABLE CELLS

Import Selection

- Import Data
- Generate Live Script
- Generate Script
- Generate Function

carData_excel.xlsx

carDataexcel

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| | Model | Country | Type | Weight_lbs | TurningC... | Displace... | Horsepo... | GasTank... |
| | Text | Text | Text | Number | Number | Number | Number | Number |
| 1 | Model | Country | Type | Weight_lbs | Turning ... | Displace... | Horsepo... | Gas Tank... |
| 2 | Acura Int... | Japan | Small | 2700 | 37 | 112 | 130 | 13.2000 |
| 3 | Acura Le... | Japan | Medium | 3265 | 42 | 163 | 160 | 18 |
| 4 | Audi 100 | Other | | | | | | |
| 5 | Audi 80 | Other | | | | | | |
| 6 | Audi 90 | Other | | | | | | |

EDITOR | PUBLISH | VIEW

New | Open | Save | Compare | Print | FILE

Go To | Find | Bookmark | NAVIGATE

Refactor | CODE

Profiler | Analyze | ANALYZE

Run Section | Section Break | Run and Advance | Run to End | SECTION

Run | Step | Stop | RUN

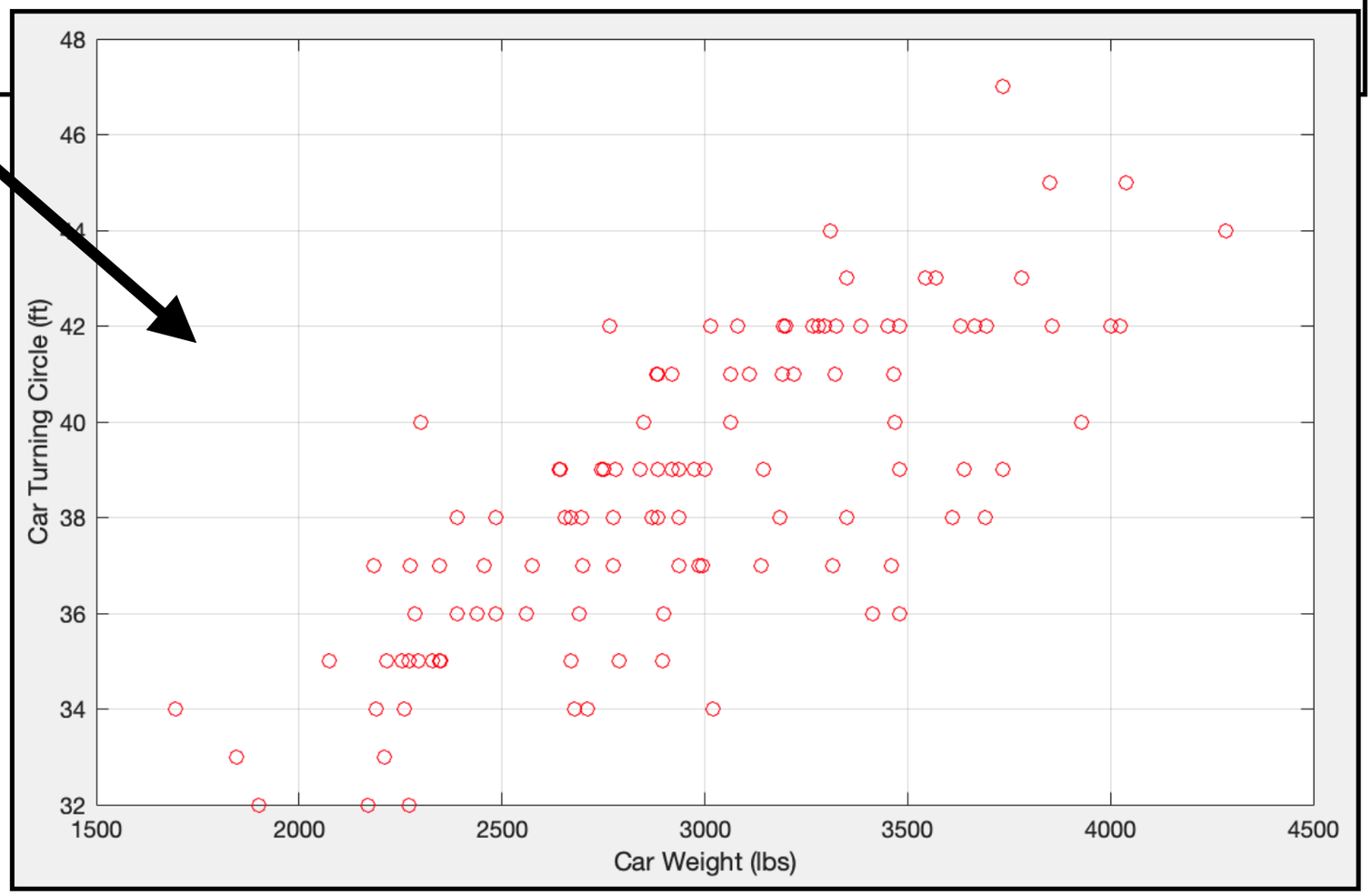
cruiseCalculator.m | thrust_calculationNoLoss.m | TTBW_class.m | UnrestrictedClimb.m | UnrestrictedDescent.m | untitled *

```
7   % Auto-generated by MATLAB on 28-Feb-2024 08:31:55
8
9   %% Set up the Import Options and import the data
10  opts = spreadsheetImportOptions("NumVariables", 8);
11
12  % Specify sheet and range
13  opts.Sheet = "carData";
14  opts.DataRange = "A2:H117";
15
16  % Specify column names and types
17  opts.VariableNames = ["Model", "Country", "Type", "Weight_lbs", "T
```

# Working with Table Data

## Make a plot of car weight versus turning circle

```
Command Window
>> plot(carData.Weight_lbs,carData.TurningCircle_ft,'or')
>> xlabel('Car Weight )lbs)')
>> xlabel('Car Weight (lbs)')
>> ylabel('Car Turning Circle (ft)')
>> grid
```



# Reading the Sample Data File

**Method 3 -** Use MATLAB fopen and fscan functions

The following script will read the text file 'ohare_schedule' using 'fopen' and 'fread' functions.

```
% Format for data input is a 4-column data file

format long
fid = fopen ('ohare_schedule','rt') % 'rt' = read text file
y = fscanf(fid, '%g', [4,inf]);     % reads in 4 columns
y = y';

[nrow,ncol] = size(y);              % extracts array size
```

# Manipulating Array Data with MATLAB

- Suppose we would like to maintain the results from the data file ‘ohare_schedule’ in four one-dimensional arrays called ‘hour’,’arrivals’,’departures’, and ‘total_ops’.
- Here we use an explicit for-loop to insert values of array ‘y’ into column vectors’hour’,’arrivals’,’departures’, and ‘total_ops’

```
% read data in vector form for each variable
for i=1:1:nrow;
    hour(i)         = y(i,1);
    arrivals(i)     = y(i,2);
    departures(i)   = y(i,3);
    total_ops(i)    = y(i,4);
end
```

# Manipulating Array Data with MATLAB (II)

- An easier procedure to assign and create four 1-D arrays is to use an implicit declaration in MATLAB
- Here we use a vector operation (takes less time)

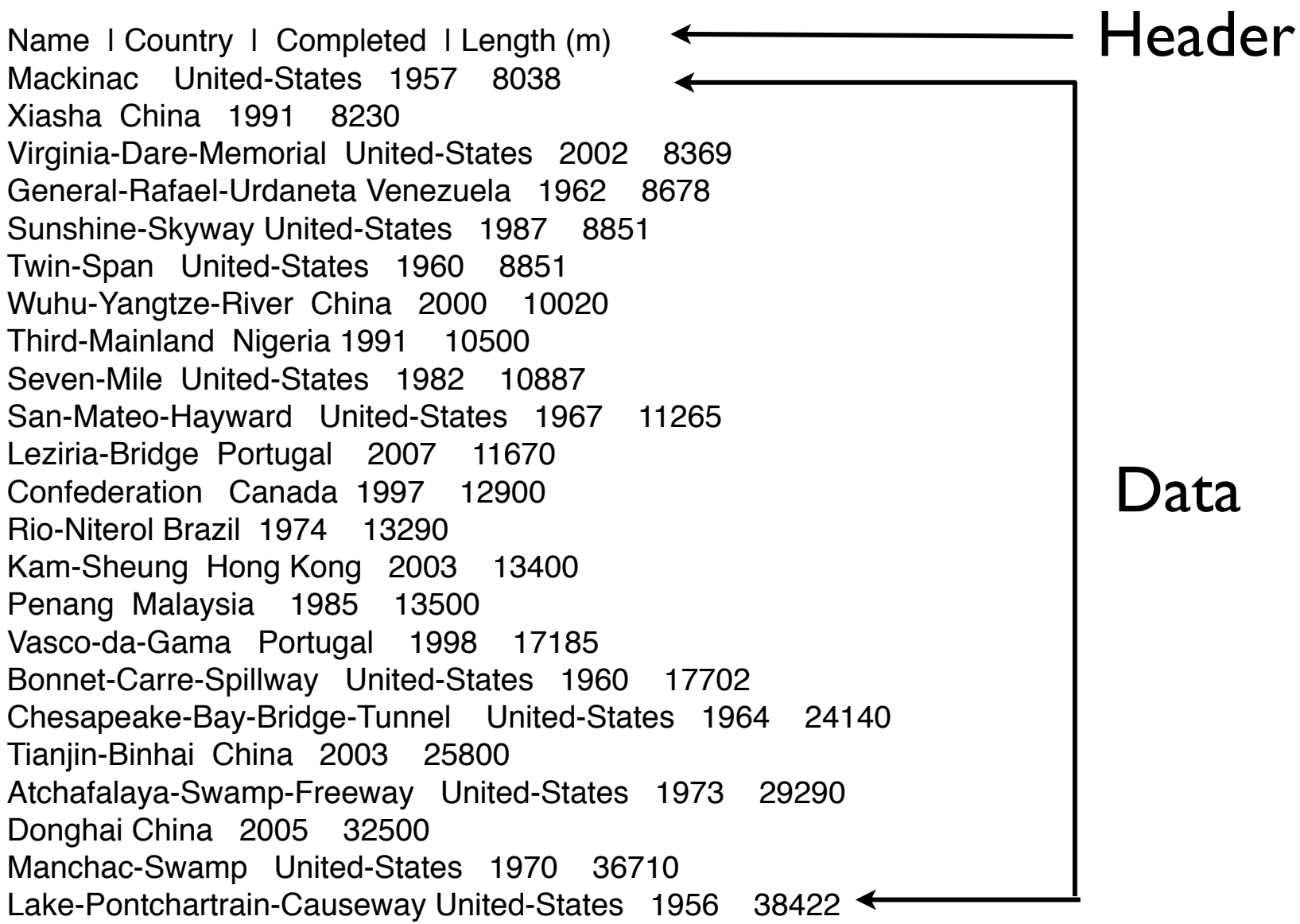
```
% implicit assignment form

hour       = y(: , 1);
arrivals   = y(: , 2);
departures = y(: , 3);
total_ops  = y(: , 4);
```

# Reading Data Files

- **Method 4** - Using the **Textscan** Command
- Here is a sample script to read a text file containing data on bridges of the world

```
fid = fopen('bridges_of_the_world')

readHeader = textscan(fid, '%s', 4, 'delimiter', '|');

readData = textscan(fid, '%s %s %f %f');

fclose(fid);
```

# Data File (bridges_of_the_world)



# Explanations of the Matlab Script

fid = **fopen**('bridges_of_the_world')

- fid - file ID assigned by Matlab
- fopen - "opens" (or reads) the text file called 'bridges_of_the_world'

readHeader = **textscan**(fid, '%s', 4, 'delimiter', '|');

- variable readHeader will store the contents of the first row in the file ('bridges_of_the_world')
- textscan reads the first row of the file using '%s',4 (four string variables) with 'delimiter' = '|'

```
Name  I Country  I  Completed  I Length (m)
Mackinac    United-States   1957    8038
Xiasha  China   1991    8230
Virginia-Dare-Memorial  United-States   2002    8369
```

# Explanations of the Matlab Script

readData = **textscan**(fid, '%s %s %f %f');

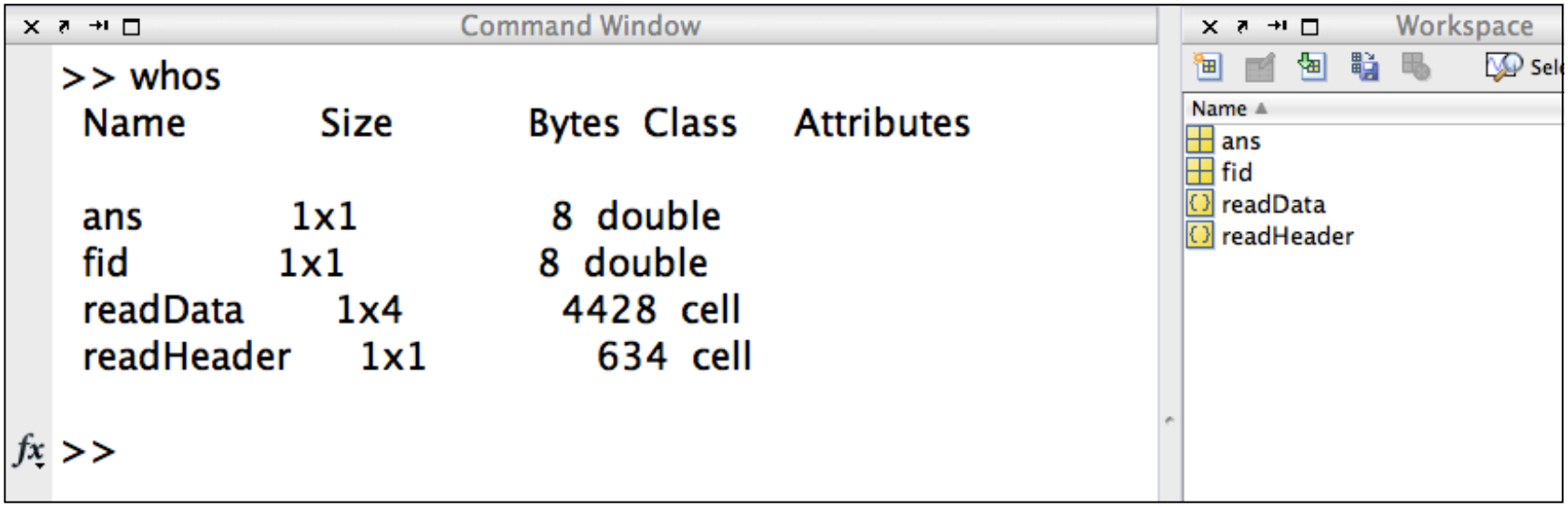
- variable readData will store the contents of the information starting in the second row (until the end) in the file ('bridges_of_the_world')
- textscan reads the row data using '%s %s' two string variables and two '%f %f' numerical variables (f stands for floating point)

fclose(fid);

- fclose(fid) closes the file (fid) opened at the beginning of the script

```
Name  I Country  I  Completed  I Length (m)
Mackinac    United-States   1957    8038
Xiasha  China   1991    8230
Virginia-Dare-Memorial  United-States   2002    8369
```

# What is Produced by the Matlab Script?

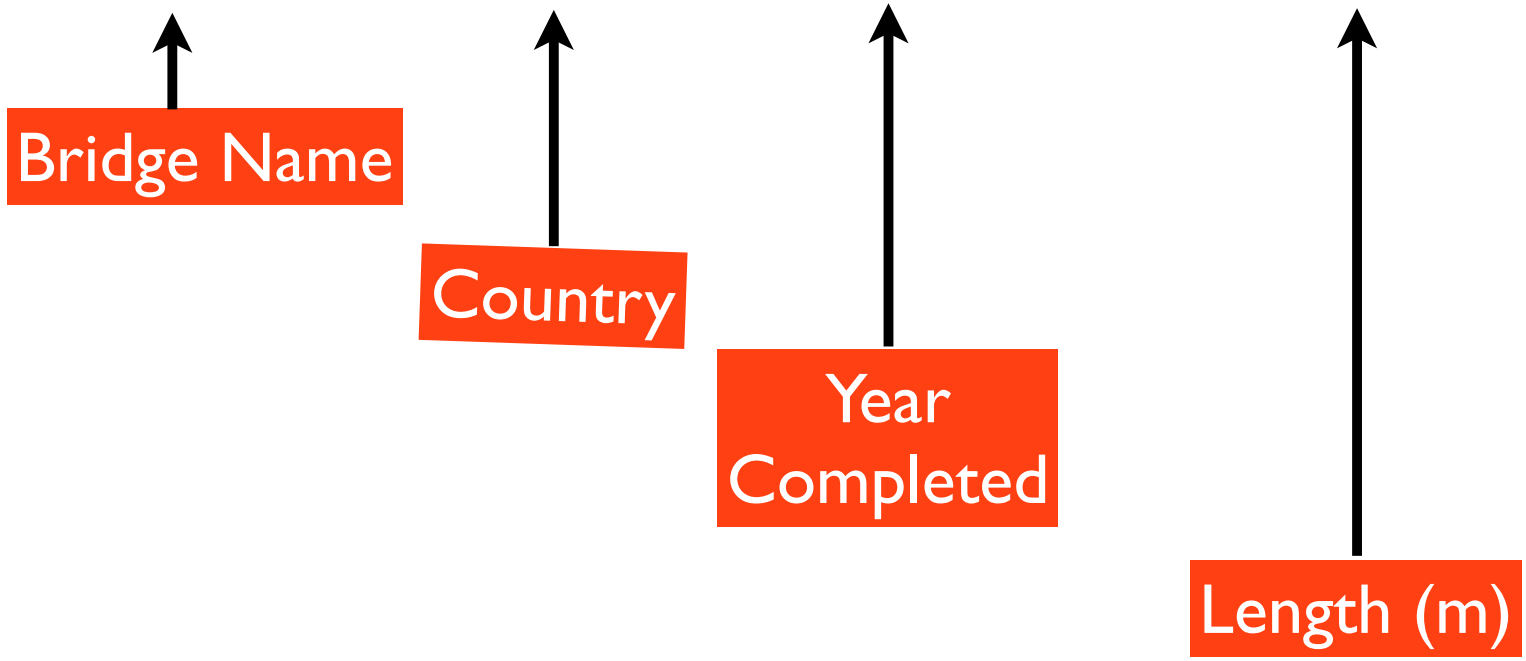



- Four variables (2 are temporary - "ans" and "fid")
- Two variables with the information in the file (***readHeader*** and ***readData***)
- Both variables are **cell arrays (more on this)**

# What is a Cell Array?

- A special structure in Matlab to store dissimilar data types (i.e., strings and numeric data)

```
>> readData

readData = {23x1 cell}    {23x1 cell}    [23x1 double]    [23x1 double]
```



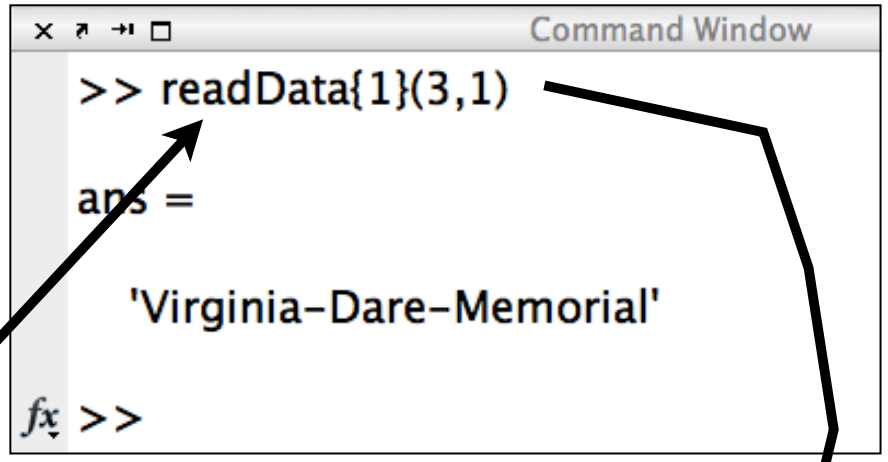
# Addressing the Contents of a Cell Array

- Cell arrays are referenced using curly brackets (first) then using standard brackets - to address individual elements of the cell array
- readData{1} references the first column of the array

```
>> readData{1}

ans =

    'Mackinac'
    'Xiasha'
    'Virginia-Dare-Memorial'
    'General-Rafael-Urdaneta'
    'Sunshine-Skyway'
    'Twin-Span'
    'Wuhu-Yangtze-River'
    'Third-Mainland'
    'Seven-Mile'
    'San-Mateo-Hayward'
    'Leziria-Bridge'
    'Confederation'
    'Rio-Niterol'
    'Kam-Sheung'
```

# Addressing the Contents of a Cell Array

- Cell arrays are referenced using curly brackets (first) then using standard brackets - to address individual elements of the cell array
- readData{1}(3,1) references the third row element of the cell array




```
>> readData{1}

ans =

   'Mackinac'
   'Xiasha'
   'Virginia-Dare-Memorial'
   'General-Rafael-Urdaneta'
   'Sunshine-Skyway'
   'Twin-Span'
   'Wuhu-Yangtze-River'
   'Third-Mainland'
   'Seven-Mile'
   'San-Mateo-Hayward'
   'Leziria-Bridge'
   'Confederation'
   'Rio-Niterol'
   'Kam-Sheung'
```

# Addressing the Contents of a Cell Array

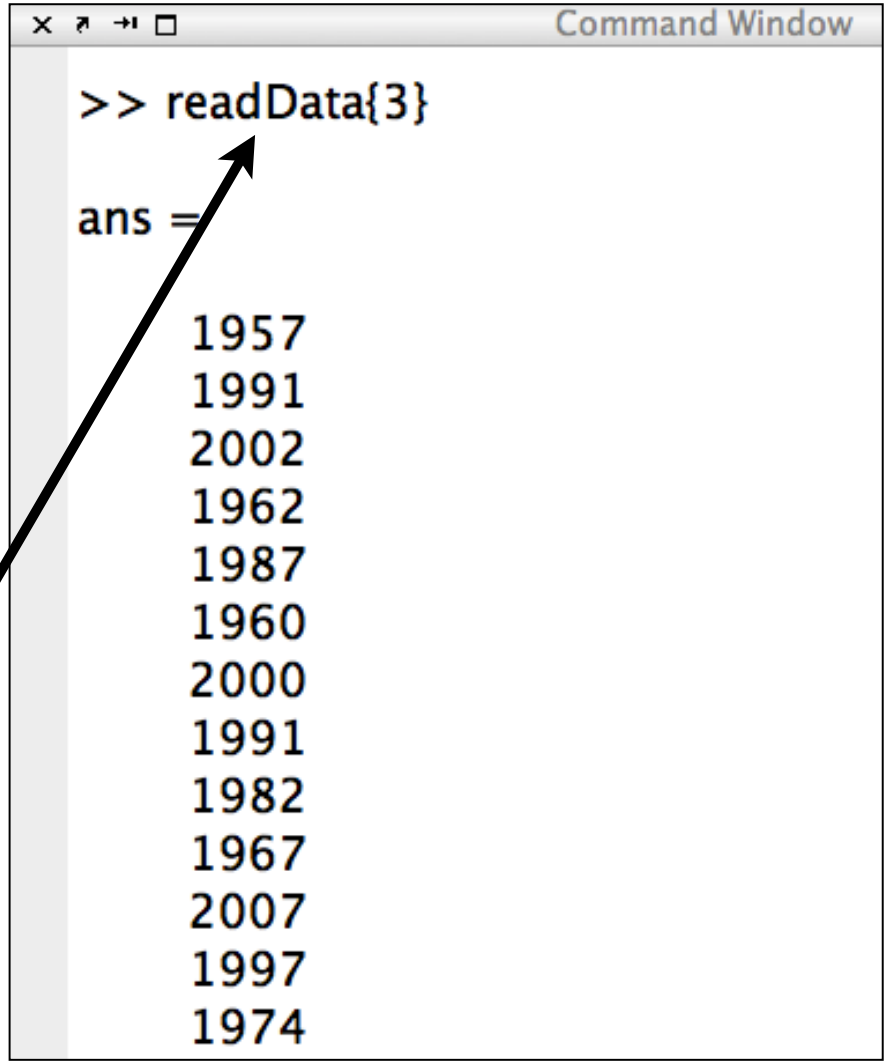
- Cell arrays are referenced using curly brackets (first) then using standard brackets - to address individual elements of the cell array
- readData{1}(3:5,1) references the third, fourth and fifth row elements of the cell array

```
>> readData{1}(3:5,1)

ans =

    'Virginia-Dare-Memorial'
    'General-Rafael-Urdaneta'
    'Sunshine-Skyway'

fx >>
```

```
>> readData{1}

ans =

    'Mackinac'
    'Xiasha'
    'Virginia-Dare-Memorial'
    'General-Rafael-Urdaneta'
    'Sunshine-Skyway'
    'Twin-Span'
    'Wuhu-Yangtze-River'
    'Third-Mainland'
    'Seven-Mile'
    'San-Mateo-Hayward'
    'Leziria-Bridge'
    'Confederation'
    'Rio-Niterol'
    'Kam-Sheung'
```

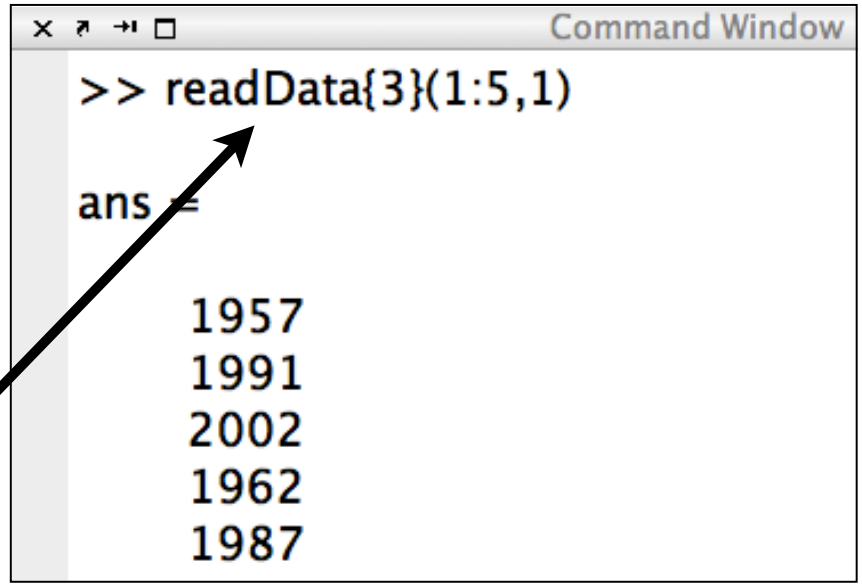
# Addressing the Contents of a Cell Array

- Cell arrays are referenced using curly brackets (first) then using standard brackets - to address individual elements of the cell array
- readData{3} references all the elements of the third column of the cell array



# Addressing the Contents of a Cell Array

- Cell arrays are referenced using curly brackets (first) then using standard brackets - to address individual elements of the cell array
- readData{3}(1:5,1) references the first five row elements of the third column of the cell array



# Reading Excel Data Files with Matlab

- **Method 5** - Using the **xlsread** Command
- Here is a sample script to read a data file containing data on bridges of the world

```
[num,txt,raw] = xlsread
('bridges_of_the_world_short.xls','Bridge data');
```

- Reads the Excel worksheet named 'Bridge data' contained in file called 'bridges_of_the_world_short.xls'
- Assigns all numeric data to variable **'num'**
- Assigns all text data to variable called **'txt'**
- All other unassigned data is stored in variable **'raw'**

# Excel File to be Read

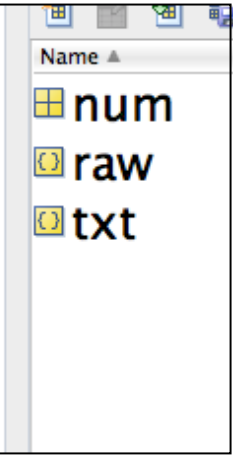
| | A | B | C | D |
|---|---|---|---|---|
| 1 | **Name** | **Country** | **Completed** | **Length (m)** |
| 2 | Mackinac | United States | 1957 | 8038 |
| 3 | Xiasha | China | 1991 | 8230 |
| 4 | Virginia-Dare-Memorial | United States | 2002 | 8369 |
| 5 | General-Rafael-Urdaneta | Venezuela | 1962 | 8678 |
| 6 | Sunshine-Skyway | United States | 1987 | 8851 |
| 7 | Twin-Span | United States | 1960 | 8851 |
| 8 | Wuhu-Yangtze-River | China | 2000 | 10020 |
| 9 | Third-Mainland | Nigeria | 1991 | 10500 |
| 10 | Seven-Mile | United States | 1982 | 10887 |
| 11 | San-Mateo-Hayward | United States | 1967 | 11265 |
| 12 | Leziria-Bridge | Portugal | 2007 | 11670 |
| 13 | Confederation | Canada | 1997 | 12900 |
| 14 | Rio-Niterol | Brazil | 1974 | 13290 |
| 15 | Kam-Sheung | Hong Kong | 2003 | 13400 |
| 16 | Penang | Malaysia | 1985 | 13500 |
| 17 | Vasco-da-Gama | Portugal | 1998 | 17185 |
| 18 | Bonnet-Carre-Spillway | United States | 1960 | 17702 |
| 19 | Chesapeake-Bay-Bridge-Tunnel | United States | 1964 | 24140 |
| 20 | Tianjin-Binhai | China | 2003 | 25800 |
| 21 | Atchafalaya-Swamp-Freeway | United States | 1973 | 29290 |
| 22 | Donghai | China | 2005 | 32500 |
| 23 | Manchac-Swamp | United States | 1970 | 36710 |
| 24 | Lake-Pontchartrain-Causeway | United States | 1956 | 38422 |

Bridges_of_the_world_short.xls

# What Happens after Executing the One Line Script?

- Three arrays are created using the previous script
- Array '**num**" is a standard matrix with size (23 x 2)
- Arrays '**raw**' and '**txt**' are cell arrays (24 x 4) each

```
>> whos
 Name      Size              Bytes  Class     Attributes

 num      23x2                368  double
 raw      24x4              12328  cell
 txt      24x4              11960  cell
```



# Observations

- 'num' is a standard numeric array as shown
- Elements of 'num" can be referenced in the usual (row,column) format
- **num(2,2)=8230**

```
>> num

num =

     1957     8038
     1991     8230
     2002     8369
     1962     8678
     1987     8851
     1960     8851
     2000    10020
     1991    10500
     1982    10887
     1967    11265
```

# Observations (2)

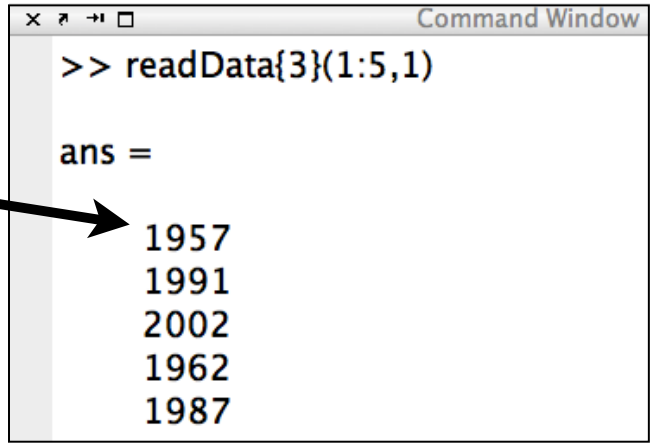
- 'txt' is a cell array containing **string** data as shown
- Elements of 'txt" can be referenced using the cell array nomenclature cell{i}(row,column)
- **txt{1,2}=Country**

txt <24x4 cell>

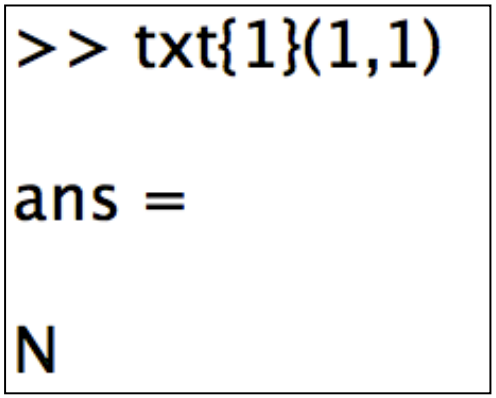
| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | Name | Country | Completed | Length (m) |
| 2 | Mackinac | United States | | |
| 3 | Xiasha | China | | |
| 4 | Virginia-Da... | United States | | |
| 5 | General-Raf... | Venezuela | | |
| 6 | Sunshine-S... | United States | | |
| 7 | Twin-Span | United States | | |
| 8 | Wuhu-Yang... | China | | |
| 9 | Third-Mainl... | Nigeria | | |
| 10 | Seven-Mile | United States | | |
| 11 | San-Mateo-... | United States | | |
| 12 | Leziria-Bridge | Portugal | | |
| 13 | Confederation | Canada | | |
| 14 | Rio-Niterol | Brazil | | |
| 15 | Kam-Sheung | Hong Kong | | |
| 16 | Penang | Malaysia | | |
| 17 | Vasco-da-... | Portugal | | |
| 18 | Bonnet-Car... | United States | | |
| 19 | Chesapeake... | United States | | |
| 20 | Tianjin-Binhai | China | | |
| 21 | Atchafalaya... | United States | | |
| 22 | Donghai | China | | |
| 23 | Manchac-S... | United States | | |
| 24 | Lake-Pontc... | United States | | |

# Note Differences in How Cell Arrays Store Information

- In previous case, a cell array storing numerical data can be referenced
- **readData{3}(1:5,1)**




- In this case, the cell array contains string information
- **txt{1}(1,2)=N**



# Matlab **xlsread** can Read a Range in an Excel

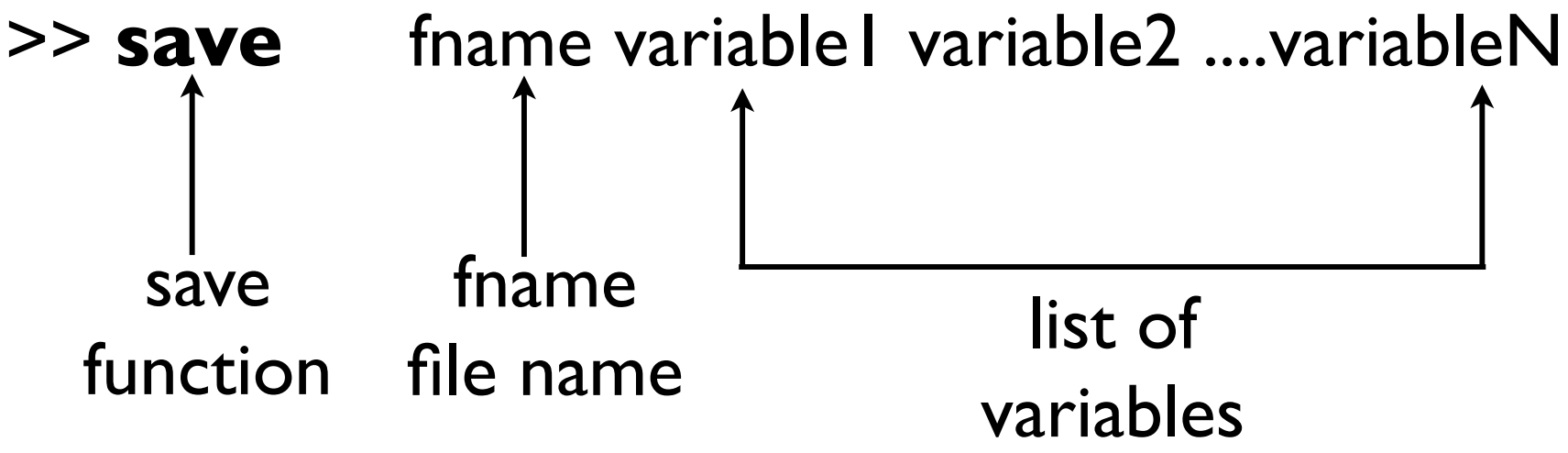
- The Matlab statement:
- [num,txt,raw] = xlsread ('bridges_of_the_world_short.xls','Bridge data (A2:D24)');
- Reads the Excel file but only across the range specified (A2:D24)
- This is useful if you know the data structure of the file you are reading (however, does not work in Macintosh computers)

# Exporting Data in Matlab

- Five methods are presented here:
- Some have been discussed in previous pages of the handout:
  - Save data to Matlab binary files (**save** function)
  - Save data from Matlab to ASCII delimited file (**dlmwrite** function)
  - Save data to a text file (**fprintf** function)
  - Save data to an Excel spreadsheet (**xlswrite** function)
  - Write results to Command Window (**disp** function)

# Save Function in Matlab

- Saves data in binary format (**.mat** files)
- Fast retrieval and fast saving
- **.mat files** can only be opened with Matlab




Remember: files save using the save command can be loaded using the **"load"** function

# Example: Saving Data to Binary File

```
16      % Read the complete Excel file (previous exercise)
17
18 -    [num,txt,raw] = xlsread('bridges_of_the_world_short.xls','Bridge data');
19
20      % Create individual variables with names: NOTE: convert cell array to
21      % string array (equal length)
22
23 -    noRows = length(raw);          % 23 entries for this case
24
25 -    bridgeName      = char(raw{2:noRows,1});       % converts cell array into a string array in Matlab -
26                                                     % saves the result into variable bridgeName
27 -    bridgeCountry   = char(raw{2:noRows,2});      % saves result into string array bridgeCountry
28 -    bridgeYear       = num(:,1);                   % no need to convert - values are all numeric
29 -    bridgeLength     = num(:,2);                   % no need to convert - values are all numeric
30
31      % Save all 4 variables created in this script to a Matlab binary format
32
33 -    save bridgesOfTheWord   bridgeName bridgeCountry  bridgeYear  bridgeLength
```

# Delimited ASCII File (dlmwrite)

- **dlmwrite('FILENAME',M,'DLM')**
- Writes matrix M into FILENAME using the character DLM as the delimiter
- ASCII stands for **American Standard Code for Information Interchange**
- ASCII is a character-encoding scheme used to represent text in computers
- Generally writes numerical data to an ASCII file

# Example Delimited ASCII File (dlmwrite)

- Writes matrix M into FILENAME using the character DLM as the delimiter

```
16      % Read the complete Excel file (previous exercise)
17
18 -    [num,txt,raw] = xlsread('bridges_of_the_world_short.xls','Bridge data');
19
20      % Create individual variables with names: NOTE: convert cell array to
21      % string array (equal length)
22
23 -    noRows = length(raw);          % 23 entries for this case
24
25 -    bridgeYear      = num(:,1);                % no need to convert - values are a
26 -    bridgeLength    = num(:,2);                % no need to convert - values are a
27
28      % collectAllVariables = [bridgeName, bridgeCountry,  bridgeYear,  bridgeLength];
29
30      % Create a text file (called myOutout2.txt) and write the numerical data
31      % variables (year completed and length) to an ASCII file
32
33 -    dlmwrite('myOutput2.txt',[bridgeYear bridgeLength],'delimiter','\t')
```

```
1957    8038
1991    8230
2002    8369
1962    8678
1987    8851
1960    8851
2000    10020
1991    10500
1982    10887
1967    11265
2007    11670
1997    12900
1974    13290
2003    13400
1985    13500
1998    17185
1960    17702
1964    24140
2003    25800
1973    29290
2005    32500
1970    36710
1956    38422
```

tab delimited ASCII file

# Another Example of ASCII File (dlmwrite)

- This time we use the "comma" as the delimiter between data

```
16      % Read the complete Excel file (previous exercise)
17
18 -    [num,txt,raw] = xlsread('bridges_of_the_world_short.xls','Bridge data');
19
20      % Create individual variables with names: NOTE: convert cell array to
21      % string array (equal length)
22
23 -    noRows = length(raw);          % 23 entries for this case
24
25 -    bridgeYear       = num(:,1);                    % no need to convert - values ar
26 -    bridgeLength     = num(:,2);                    % no need to convert - values ar
27
28      % collectAllVariables = [bridgeName, bridgeCountry,  bridgeYear,  bridgeLength
29
30      % Create a text file (called myOutout2.txt) and write the numerical data
31      % variables (year completed and length) to an ASCII file
32
33 -    dlmwrite('myOutput2.txt',[bridgeYear bridgeLength],'delimiter',',')
```

```
1957,8038
1991,8230
2002,8369
1962,8678
1987,8851
1960,8851
2000,10020
1991,10500
1982,10887
1967,11265
2007,11670
1997,12900
1974,13290
2003,13400
1985,13500
1998,17185
1960,17702
1964,24140
2003,25800
1973,29290
2005,32500
1970,36710
1956,38422
```

comma delimited ASCII file

# Using the **fprintf** Function

- **fprintf (fid,format,A)**
- Applies the FORMAT to all elements of array A and any additional array arguments in column order, and writes the data to a text file
- "fid" is an integer file identifier designated automatically by Matlab
- You need to get an "fid" using the "fopen" statement (see next example)

# Example of using the **fprintf** Function

```
16      % Read the complete Excel file (previous exercise)
17
18 -    [num,txt,raw] = xlsread('bridges_of_the_world_short.xls','Bridge dat
19
20      % Create individual variables with names: NOTE: convert cell array to
21      % string array (equal length)
22
23 -    noRows = length(raw);          % 23 entries for this case
24
25 -    bridgeName      = char(raw{2:noRows,1});      % converts cell array into a string array in Matlab -
26                                                    % saves the result into variable bridgeName
27 -    bridgeCountry   = char(raw{2:noRows,2});      % saves result into string array bridgeCountry
28 -    bridgeYear       = num(:,1);                   % no need to convert - values are all numeric
29 -    bridgeLength     = num(:,2);                   % no need to convert - values are all numeric
30
31      % Create a text file (called myOutout.txt) and write all four output
32      % variables (bridge name, country, year completed and length)
33
34 -    for i=1:noRows-1                              % loop to write output
35 -        fid = fopen ('myOutput.txt','a');         % create a text file
36 -        fprintf(fid, '%s %s %f %f \n',bridgeName(i,:), bridgeCountry(i,:),  bridgeYear(i),  bridgeLength(i));
37 -        status = fclose(fid);
38 -    end
```

create 4 new variables to store values of bridge name, country, year and length

# Comments on the Matlab Code

- The data read using xlsread contains three variables: num, txt and raw
- Because txt and raw contain information in Cell Arrays, before exporting the information to a file, we are required to change the data type from **Cell** array to **String array**
- Use the **char** function to do the conversion from cell arrays to string arrays
- Once bridgeName and bridgeCountry are string arrays, they can be exported using the **fprintf** function

# More Explanations

```
31      % Create a text file (called myOutout.txt) and write all four output
32      % variables (bridge name, country, year completed and length)
33
34 -    for i=1:noRows-1                                    % loop to write output
35 -        fid = fopen ('myOutput.txt','a');               % create a text file
36 -        fprintf(fid, '%s %s %f %f \n',bridgeName(i,:), bridgeCountry(i,:),  bridgeYear(i),  bridgeLength(i));
37 -        status = fclose(fid);
38 -    end
```

- A simple **FOR-Loop** is used to iterate through the complete string array and save variables
- An output file called **myOutput.txt** is created to save the values of four variables: bridgeName, bridgeCountry, bridgeYear and bridgeLength (line 36)
- Output format is controlled by **%s** or **%f** statement in line 36
- the format **'a'** in line 35 **"appends"** (i.e., adds a new line) a line every time the loop executes

# Example Output File Produced (myOutput.txt)

```
31      % Create a text file (called myOutout.txt) and write all four output
32      % variables (bridge name, country, year completed and length)
33
34 -    for i=1:noRows-1                                    % loop to write output
35 -          fid = fopen ('myOutput.txt','a');             % create a text file
36 -          fprintf(fid, '%s %s %f %f \n',bridgeName(i,:), bridgeCountry(i,:),  bridgeYear(i),  bridgeLength(i));
37 -          status = fclose(fid);
38 -    end
```

myOutput.txt ASCII text file produced

```
Mackinac                        United States 1957.000000 8038.000000
Xiasha                          China         1991.000000 8230.000000
Virginia-Dare-Memorial          United States 2002.000000 8369.000000
General-Rafael-Urdaneta         Venezuela     1962.000000 8678.000000
Sunshine-Skyway                 United States 1987.000000 8851.000000
Twin-Span                       United States 1960.000000 8851.000000
Wuhu-Yangtze-River              China         2000.000000 10020.000000
Third-Mainland                  Nigeria       1991.000000 10500.000000
Seven-Mile                      United States 1982.000000 10887.000000
San-Mateo-Hayward               United States 1967.000000 11265.000000
Leziria-Bridge                  Portugal      2007.000000 11670.000000
Confederation                   Canada        1997.000000 12900.000000
Rio-Niterol                     Brazil        1974.000000 13290.000000
Kam-Sheung                      Hong Kong     2003.000000 13400.000000
Penang                          Malaysia      1985.000000 13500.000000
Vasco-da-Gama                   Portugal      1998.000000 17185.000000
Bonnet-Carre-Spillway           United States 1960.000000 17702.000000
Chesapeake-Bay-Bridge-Tunnel    United States 1964.000000 24140.000000
Tianjin-Binhai                  China         2003.000000 25800.000000
Atchafalaya-Swamp-Freeway       United States 1973.000000 29290.000000
Donghai                         China         2005.000000 32500.000000
Manchac-Swamp                   United States 1970.000000 36710.000000
Lake-Pontchartrain-Causeway     United States 1956.000000 38422.000000
```

# Exporting Data to Excel

- Use the Matlab "xlswrite" function to write data to Excel spreadsheets

writeToExcelData.m*

```
% Sample file to import Excel data into Matlab using the
% xlsread and xlswrite commands
% Line 22 illustrates the procedure to export from Matlab to Excel
% NOTE: Mac users: Line 22 will not work if the data exported is a cell
% array. You can export numerical data using the xlswrite function.

clear

% Sample Excel data file

% Name      Country     Completed    Length (m)
% Mackinac  United States   1957    8038
% Xiasha    China   1991    8230
% Virginia-Dare-Memorial    United States   2002    8369

% Read the complete Excel file (previous exercise)

[num,txt,raw] = xlsread('bridges_of_the_world_short.xls','Bridge data');

% Read the Excel file specifying columns and rows to read

xlswrite('exportToExcel.xls',raw,'Sheet1','A1')    % writes to Excel starting at cell A1
```

Exports data contained in variable "raw" to Excel (Sheet1 and starting in cell A1)

# Exporting Data to Excel
# Note to Mac Users

- The xlswrite function has a problem for Mac users
- Mathworks states:
  - "If your system does not have Excel for Windows, or if the COM server (part of the typical installation of Excel) is unavailable, xlswrite: Writes ARRAY to a text file in comma-separated value (CSV) format
  - Ignores SHEET and RANGE (the last two arguments) arguments
  - Generates an error when ARRAY is a cell array".

# Exporting Data to Excel

- This example works on both Mac and Windows because the data is all numeric

- xlswrite('exportToExcel.xls',num)
- Writes to Excel starting at cell A1

Produced file is:
.xls in Windows
.csv in the Mac OS

| | A | B |
|---|---|---|
| 1 | 1957 | 8038 |
| 2 | 1991 | 8230 |
| 3 | 2002 | 8369 |
| 4 | 1962 | 8678 |
| 5 | 1987 | 8851 |
| 6 | 1960 | 8851 |
| 7 | 2000 | 10020 |
| 8 | 1991 | 10500 |
| 9 | 1982 | 10887 |
| 10 | 1967 | 11265 |
| 11 | 2007 | 11670 |
| 12 | 1997 | 12900 |
| 13 | 1974 | 13290 |
| 14 | 2003 | 13400 |
| 15 | 1985 | 13500 |
| 16 | 1998 | 17185 |
| 17 | 1960 | 17702 |
| 18 | 1964 | 24140 |
| 19 | 2003 | 25800 |
| 20 | 1973 | 29290 |
| 21 | 2005 | 32500 |
| 22 | 1970 | 36710 |
| 23 | 1956 | 38422 |

# Displaying Output on the Command Window

- Use function ‘disp’ to display output to the screen.
- Typically used in conjunction with ‘num2str’ to convert numerical to string variables

Example:

```
x = 35
displ([‘This is a test to display  ’, num2str(x), ‘   here’])
```

Results:

This is a test to display  35  here

# Matlab Introduction
# Handling Basic Searches in Matrices and Arrays

Dr. Antonio A. Trani
Professor
Dept. of Civil and Environmental Engineering

# Using Index Variables in Searches performed in Matlab Matrices

- Manipulation of data requires “parsing” information from data sets
- Parsing is the process of selecting individual data elements from a larger dataset for further analysis
- Matlab provides some simple functions to find information contained inside arrays or cell arrays
- Once the information is found we can use **index variables** to locate data we want

# Data File (bridges_of_the_world.txt)

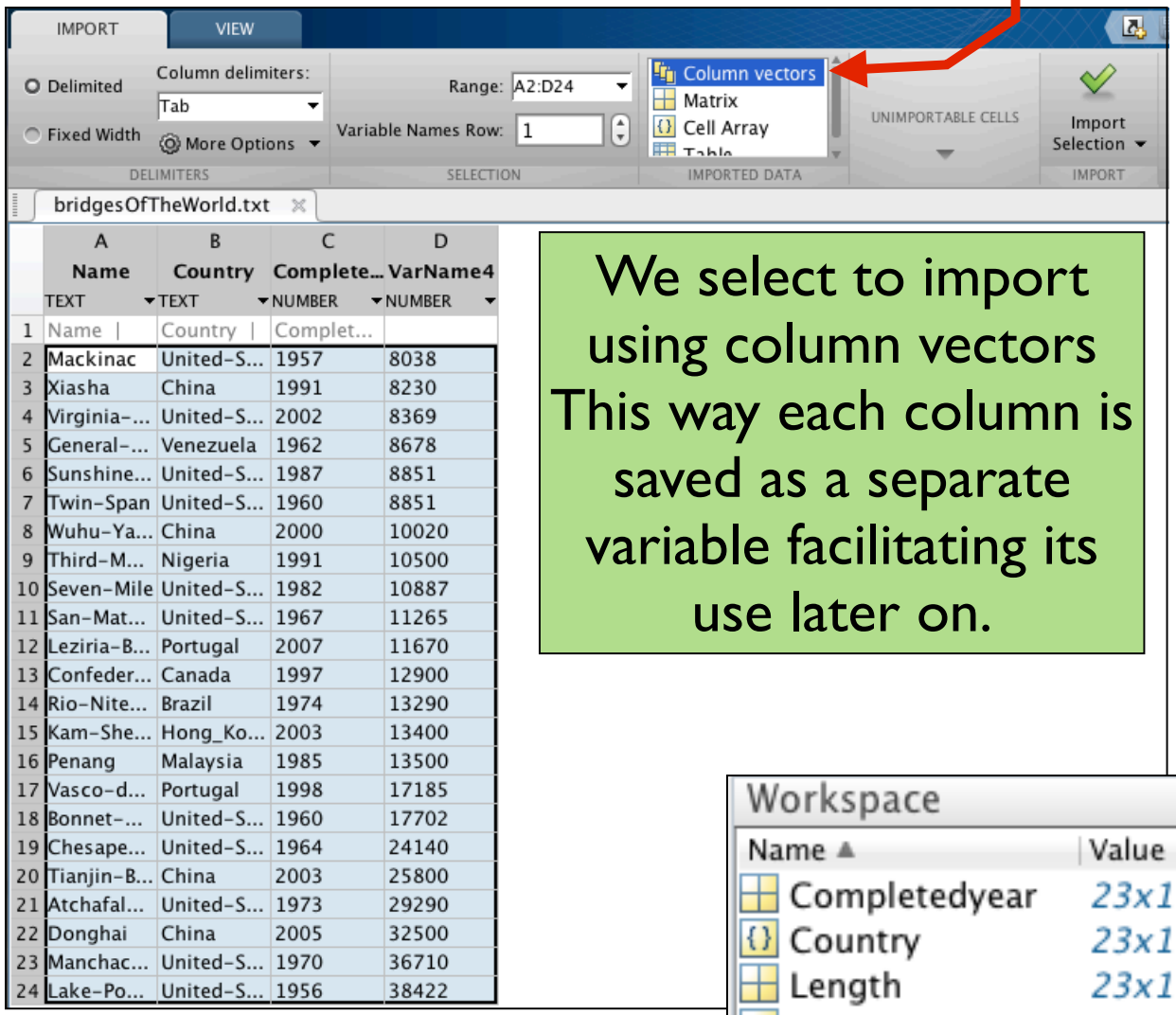
| Name | Country | Completed | Length (m) |
|---|---|---|---|
| Mackinac | United States | 1957 | 8038 |
| Xiasha | China | 1991 | 8230 |
| Virginia-Dare-Memorial | United States | 2002 | 8369 |
| General-Rafael-Urdaneta | Venezuela | 1962 | 8678 |
| Sunshine-Skyway | United States | 1987 | 8851 |
| Twin-Span | United States | 1960 | 8851 |
| Wuhu-Yangtze-River | China | 2000 | 10020 |
| Third-Mainland | Nigeria | 1991 | 10500 |
| Seven-Mile | United States | 1982 | 10887 |
| San-Mateo-Hayward | United States | 1967 | 11265 |
| Leziria-Bridge | Portugal | 2007 | 11670 |
| Confederation | Canada | 1997 | 12900 |
| Rio-Niterol | Brazil | 1974 | 13290 |
| Kam-Sheung | Hong Kong | 2003 | 13400 |
| Penang | Malaysia | 1985 | 13500 |
| Vasco-da-Gama | Portugal | 1998 | 17185 |
| Bonnet-Carre-Spillway | United States | 1960 | 17702 |
| Chesapeake-Bay-Bridge-Tunnel | United States | 1964 | 24140 |
| Tianjin-Binhai | China | 2003 | 25800 |
| Atchafalaya-Swamp-Freeway | United States | 1973 | 29290 |
| Donghai | China | 2005 | 32500 |
| Manchac-Swamp | United States | 1970 | 36710 |
| Lake-Pontchartrain-Causeway | United States | 1956 | 38422 |

Suppose we would like to find all the bridges that were build after the year 2000

Assume the variables are imported into Matlab using the Import Wizard method

# Import Procedure

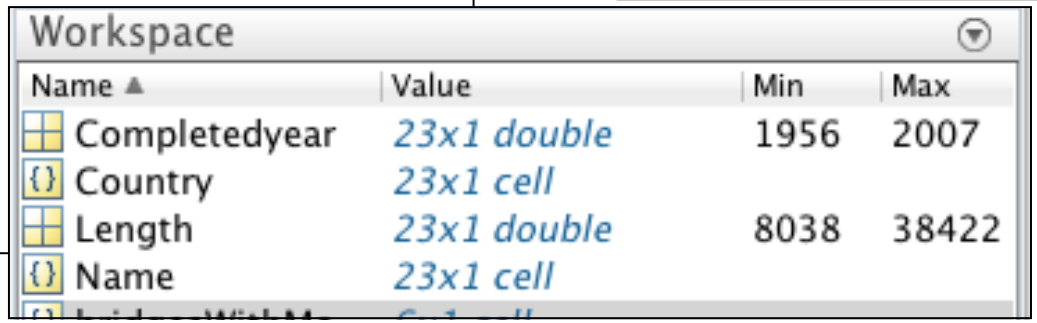
- Saving as column vectors



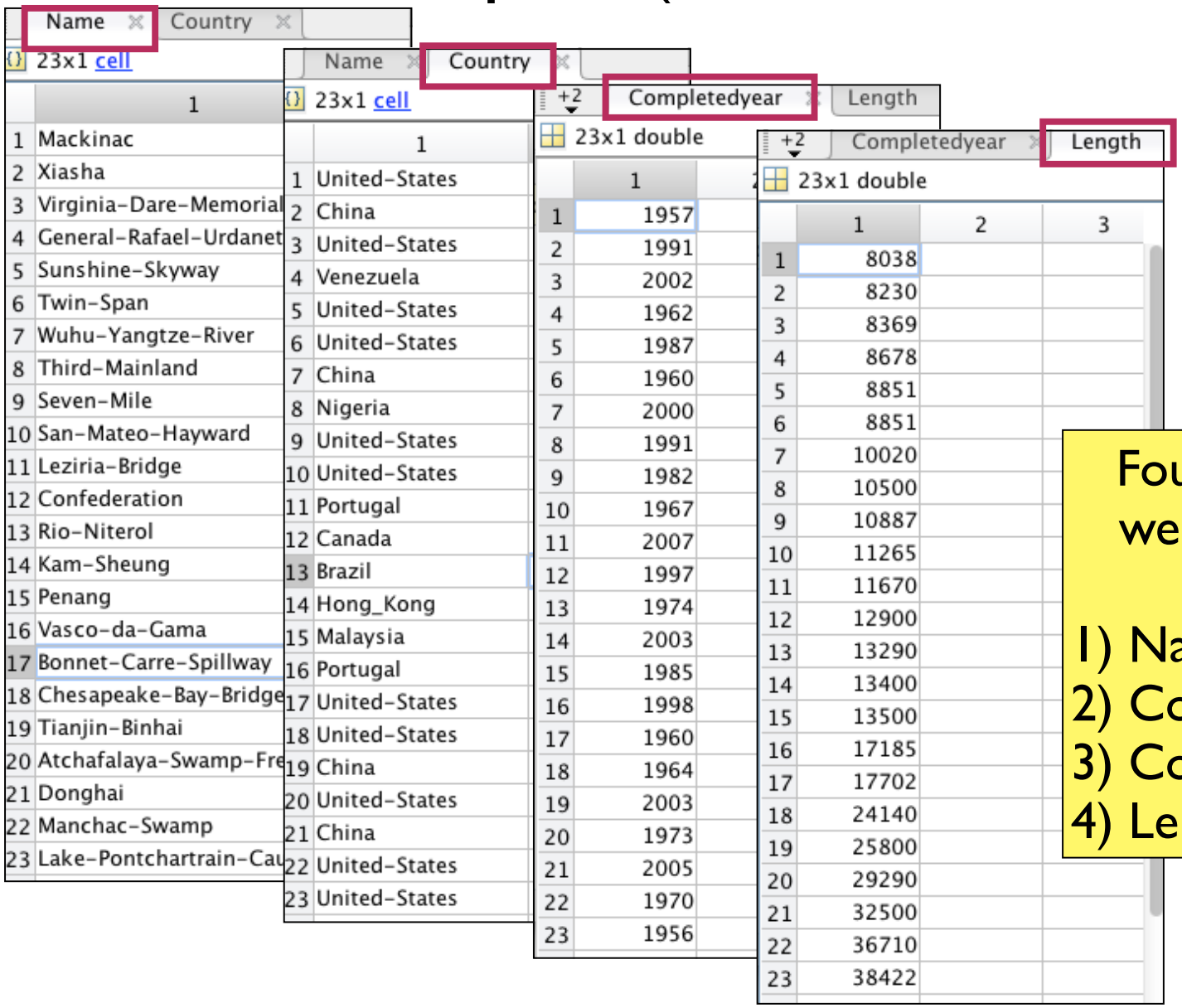

| | A | B | C | D |
|---|---|---|---|---|
| | Name | Country | Complete... | VarName4 |
| | TEXT | TEXT | NUMBER | NUMBER |
| 1 | Name | Country | Complet... | |
| 2 | Mackinac | United-S... | 1957 | 8038 |
| 3 | Xiasha | China | 1991 | 8230 |
| 4 | Virginia-... | United-S... | 2002 | 8369 |
| 5 | General-... | Venezuela | 1962 | 8678 |
| 6 | Sunshine... | United-S... | 1987 | 8851 |
| 7 | Twin-Span | United-S... | 1960 | 8851 |
| 8 | Wuhu-Ya... | China | 2000 | 10020 |
| 9 | Third-M... | Nigeria | 1991 | 10500 |
| 10 | Seven-Mile | United-S... | 1982 | 10887 |
| 11 | San-Mat... | United-S... | 1967 | 11265 |
| 12 | Leziria-B... | Portugal | 2007 | 11670 |
| 13 | Confeder... | Canada | 1997 | 12900 |
| 14 | Rio-Nite... | Brazil | 1974 | 13290 |
| 15 | Kam-She... | Hong_Ko... | 2003 | 13400 |
| 16 | Penang | Malaysia | 1985 | 13500 |
| 17 | Vasco-d... | Portugal | 1998 | 17185 |
| 18 | Bonnet-... | United-S... | 1960 | 17702 |
| 19 | Chesape... | United-S... | 1964 | 24140 |
| 20 | Tianjin-B... | China | 2003 | 25800 |
| 21 | Atchafal... | United-S... | 1973 | 29290 |
| 22 | Donghai | China | 2005 | 32500 |
| 23 | Manchac... | United-S... | 1970 | 36710 |
| 24 | Lake-Po... | United-S... | 1956 | 38422 |

We select to import using column vectors This way each column is saved as a separate variable facilitating its use later on.

After importing



# After Import (4 Column Vectors)



| | 1 |
|---|---|
| 1 | Mackinac |
| 2 | Xiasha |
| 3 | Virginia-Dare-Memorial |
| 4 | General-Rafael-Urdanet |
| 5 | Sunshine-Skyway |
| 6 | Twin-Span |
| 7 | Wuhu-Yangtze-River |
| 8 | Third-Mainland |
| 9 | Seven-Mile |
| 10 | San-Mateo-Hayward |
| 11 | Leziria-Bridge |
| 12 | Confederation |
| 13 | Rio-Niterol |
| 14 | Kam-Sheung |
| 15 | Penang |
| 16 | Vasco-da-Gama |
| 17 | Bonnet-Carre-Spillway |
| 18 | Chesapeake-Bay-Bridge |
| 19 | Tianjin-Binhai |
| 20 | Atchafalaya-Swamp-Fre |
| 21 | Donghai |
| 22 | Manchac-Swamp |
| 23 | Lake-Pontchartrain-Cau |

| | 1 |
|---|---|
| 1 | United-States |
| 2 | China |
| 3 | United-States |
| 4 | Venezuela |
| 5 | United-States |
| 6 | United-States |
| 7 | China |
| 8 | Nigeria |
| 9 | United-States |
| 10 | United-States |
| 11 | Portugal |
| 12 | Canada |
| 13 | Brazil |
| 14 | Hong_Kong |
| 15 | Malaysia |
| 16 | Portugal |
| 17 | United-States |
| 18 | United-States |
| 19 | China |
| 20 | United-States |
| 21 | China |
| 22 | United-States |
| 23 | United-States |

| | 1 |
|---|---|
| 1 | 1957 |
| 2 | 1991 |
| 3 | 2002 |
| 4 | 1962 |
| 5 | 1987 |
| 6 | 1960 |
| 7 | 2000 |
| 8 | 1991 |
| 9 | 1982 |
| 10 | 1967 |
| 11 | 2007 |
| 12 | 1997 |
| 13 | 1974 |
| 14 | 2003 |
| 15 | 1985 |
| 16 | 1998 |
| 17 | 1960 |
| 18 | 1964 |
| 19 | 2003 |
| 20 | 1973 |
| 21 | 2005 |
| 22 | 1970 |
| 23 | 1956 |

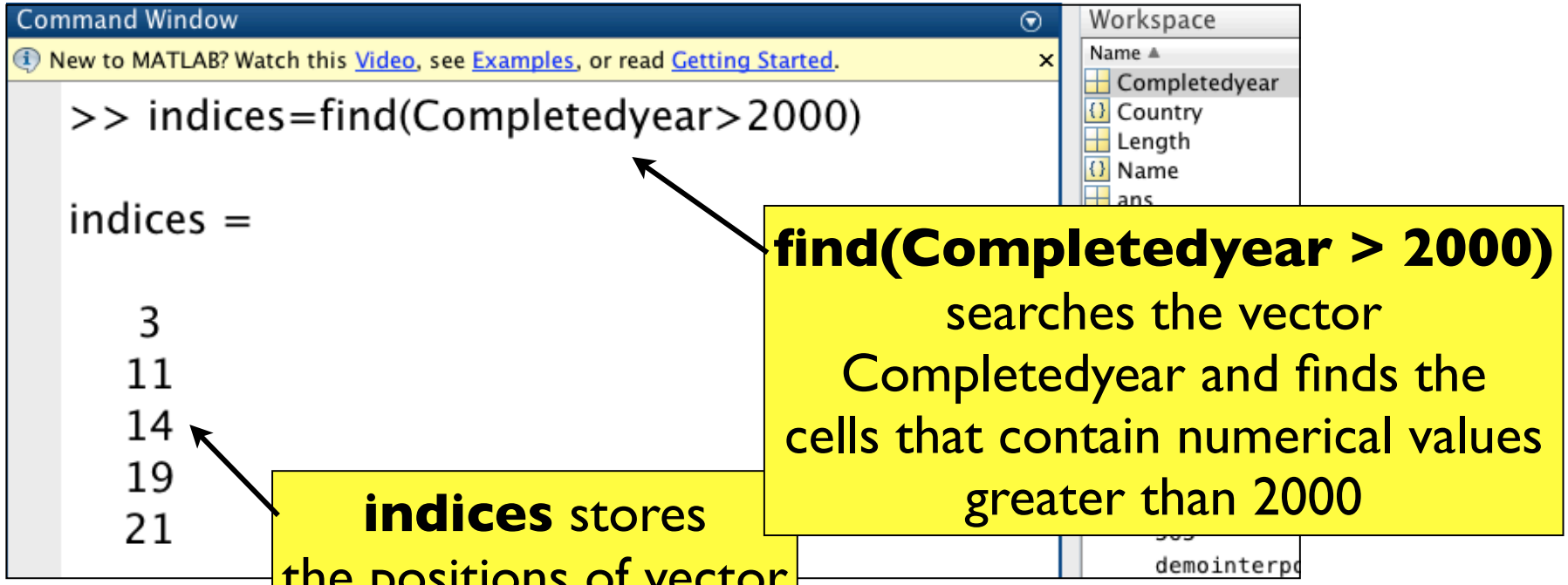
| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 8038 | | |
| 2 | 8230 | | |
| 3 | 8369 | | |
| 4 | 8678 | | |
| 5 | 8851 | | |
| 6 | 8851 | | |
| 7 | 10020 | | |
| 8 | 10500 | | |
| 9 | 10887 | | |
| 10 | 11265 | | |
| 11 | 11670 | | |
| 12 | 12900 | | |
| 13 | 13290 | | |
| 14 | 13400 | | |
| 15 | 13500 | | |
| 16 | 17185 | | |
| 17 | 17702 | | |
| 18 | 24140 | | |
| 19 | 25800 | | |
| 20 | 29290 | | |
| 21 | 32500 | | |
| 22 | 36710 | | |
| 23 | 38422 | | |

Four variables were created:

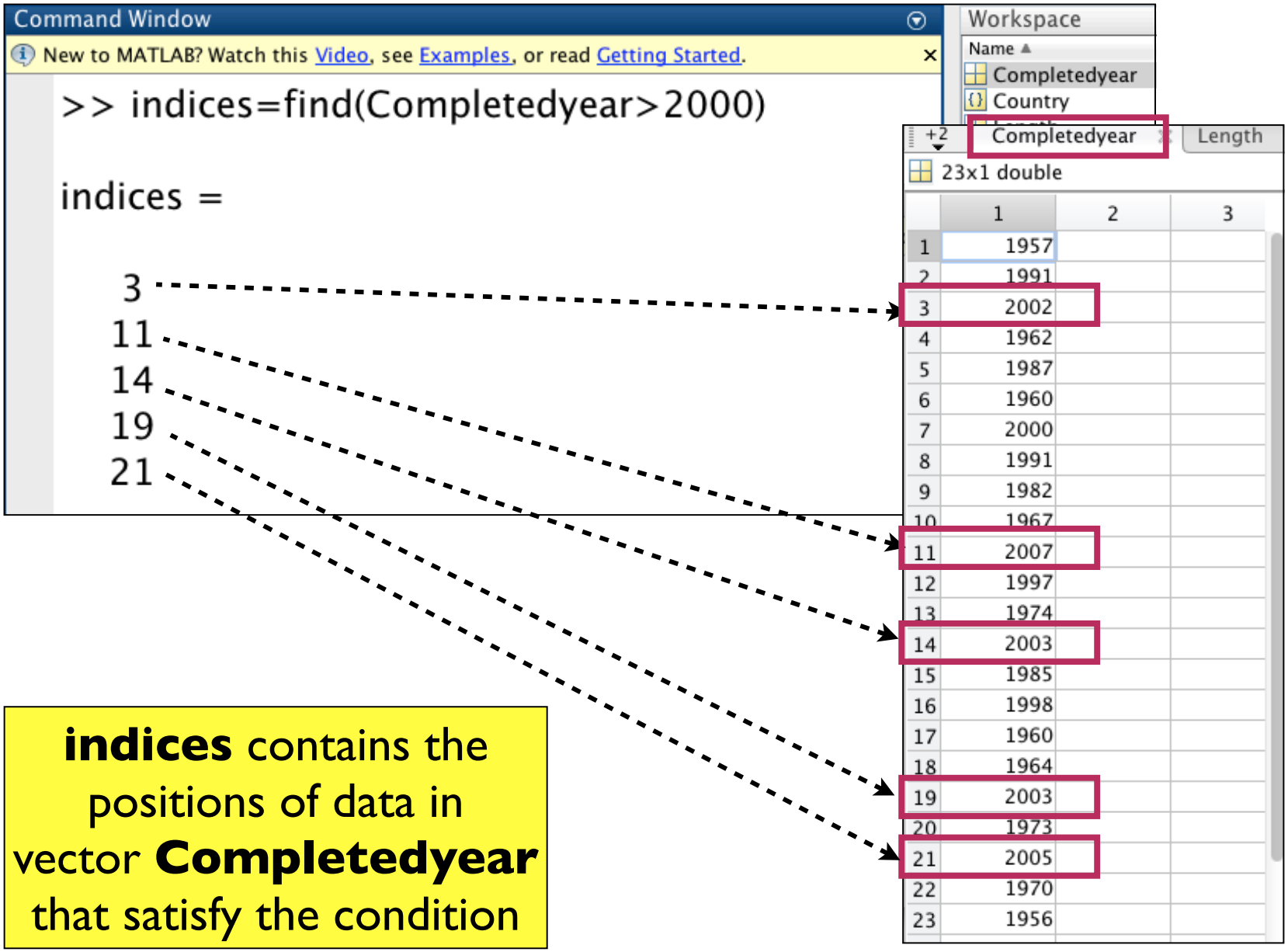
1) Name
2) Country
3) Completedyear
4) Length

# Bridges Built After the Year 2000

- Define an **index variable** called "indices"
- Use the Matlab "find" function to find all bridges built after the year 2000



# Bridges Built After the Year 2000




**indices** contains the positions of data in vector **Completedyear** that satisfy the condition

# Bridges Built After the Year 2000

```
>> bridgesBuiltAfter2000 = Name(indices)

bridgesBuiltAfter2000 =

    'Virginia-Dare-Memorial'
    'Leziria-Bridge'
    'Kam-Sheung'
    'Tianjin-Binhai'
    'Donghai'
```

Workspace: Name, Completedyear, Country, Length

| | Name |
|---|---|
| 1 | Mackinac |
| 2 | Xiasha |
| 3 | Virginia-Dare-Memorial |
| 4 | General-Rafael-Urdaneta |
| 5 | Sunshine-Skyway |
| 6 | Twin-Span |
| 7 | Wuhu-Yangtze-River |
| 8 | Third-Mainland |
| 9 | Seven-Mile |
| 10 | San-Mateo-Hayward |
| 11 | Leziria-Bridge |
| 12 | Confederation |
| 13 | Rio-Niterol |
| 14 | Kam-Sheung |
| 15 | Penang |
| 16 | Vasco-da-Gama |
| 17 | Bonnet-Carre-Spillway |
| 18 | Chesapeake-Bay-Bridge-Tu... |
| 19 | Tianjin-Binhai |
| 20 | Atchafalaya-Swamp-Freeway |
| 21 | Donghai |
| 22 | Manchac-Swamp |
| 23 | Lake-Pontchartrain-Causew... |

| | Completedyear |
|---|---|
| 1 | 1957 |
| 2 | 1991 |
| 3 | 2002 |
| 4 | 1962 |
| 5 | 1987 |
| 6 | 1960 |
| 7 | 2000 |
| 8 | 1991 |
| 9 | 1982 |
| 10 | 1967 |
| 11 | 2007 |
| 12 | 1997 |
| 13 | 1974 |
| 14 | 2003 |
| 15 | 1985 |
| 16 | 1998 |
| 17 | 1960 |
| 18 | 1964 |
| 19 | 2003 |
| 20 | 1973 |
| 21 | 2005 |
| 22 | 1970 |
| 23 | 1956 |

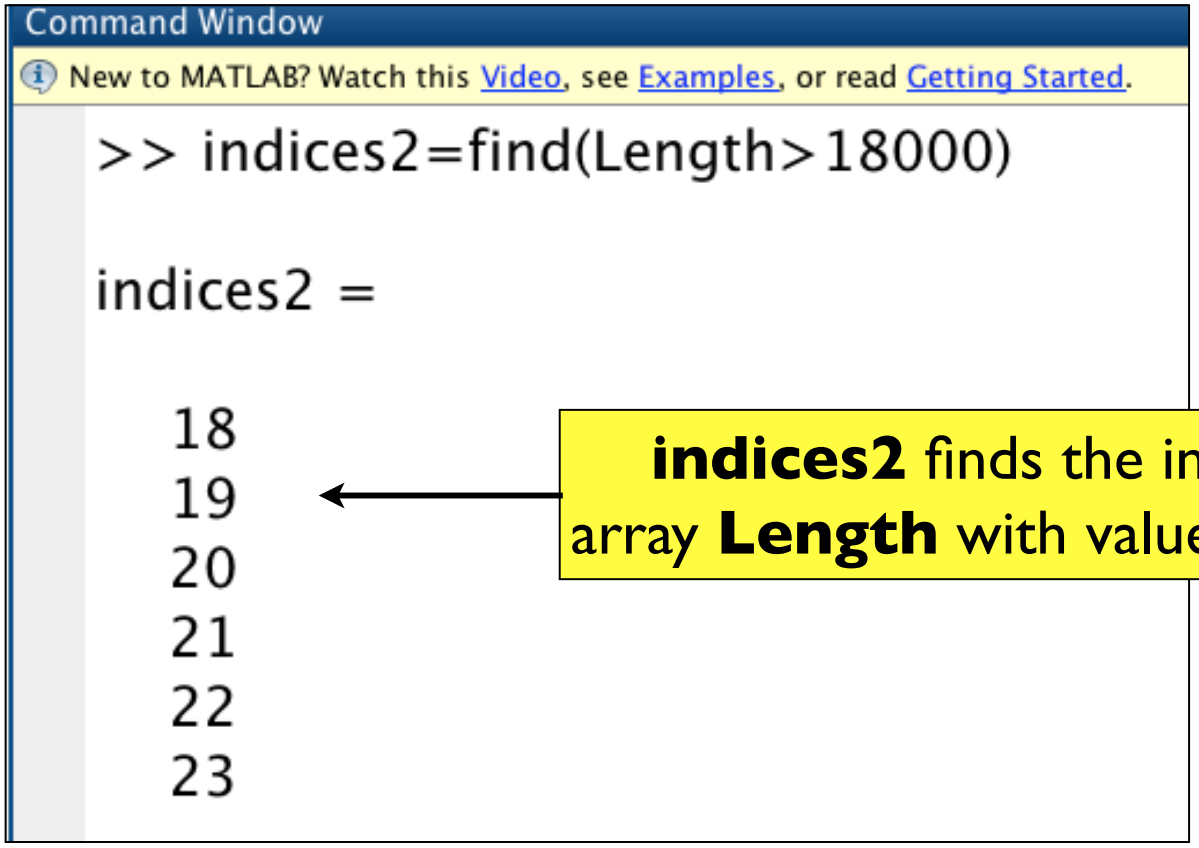
indices is now used as **index variable** to extract the names of the bridges built after the year 2000

# Another Example: Find Bridges whose Length is Greater than 14,000 meters

| Name | Country | Completed | Length (m) |
|---|---|---|---|
| Mackinac | United States | 1957 | 8038 |
| Xiasha | China | 1991 | 8230 |
| Virginia-Dare-Memorial | United States | 2002 | 8369 |
| General-Rafael-Urdaneta | Venezuela | 1962 | 8678 |
| Sunshine-Skyway | United States | 1987 | 8851 |
| Twin-Span | United States | 1960 | 8851 |
| Wuhu-Yangtze-River | China | 2000 | 10020 |
| Third-Mainland | Nigeria | 1991 | 10500 |
| Seven-Mile | United States | 1982 | 10887 |
| San-Mateo-Hayward | United States | 1967 | 11265 |
| Leziria-Bridge | Portugal | 2007 | 11670 |
| Confederation | Canada | 1997 | 12900 |
| Rio-Niterol | Brazil | 1974 | 13290 |
| Kam-Sheung | Hong Kong | 2003 | 13400 |
| Penang | Malaysia | 1985 | 13500 |
| Vasco-da-Gama | Portugal | 1998 | 17185 |
| Bonnet-Carre-Spillway | United States | 1960 | 17702 |
| Chesapeake-Bay-Bridge-Tunnel | United States | 1964 | 24140 |
| Tianjin-Binhai | China | 2003 | 25800 |
| Atchafalaya-Swamp-Freeway | United States | 1973 | 29290 |
| Donghai | China | 2005 | 32500 |
| Manchac-Swamp | United States | 1970 | 36710 |
| Lake-Pontchartrain-Causeway | United States | 1956 | 38422 |

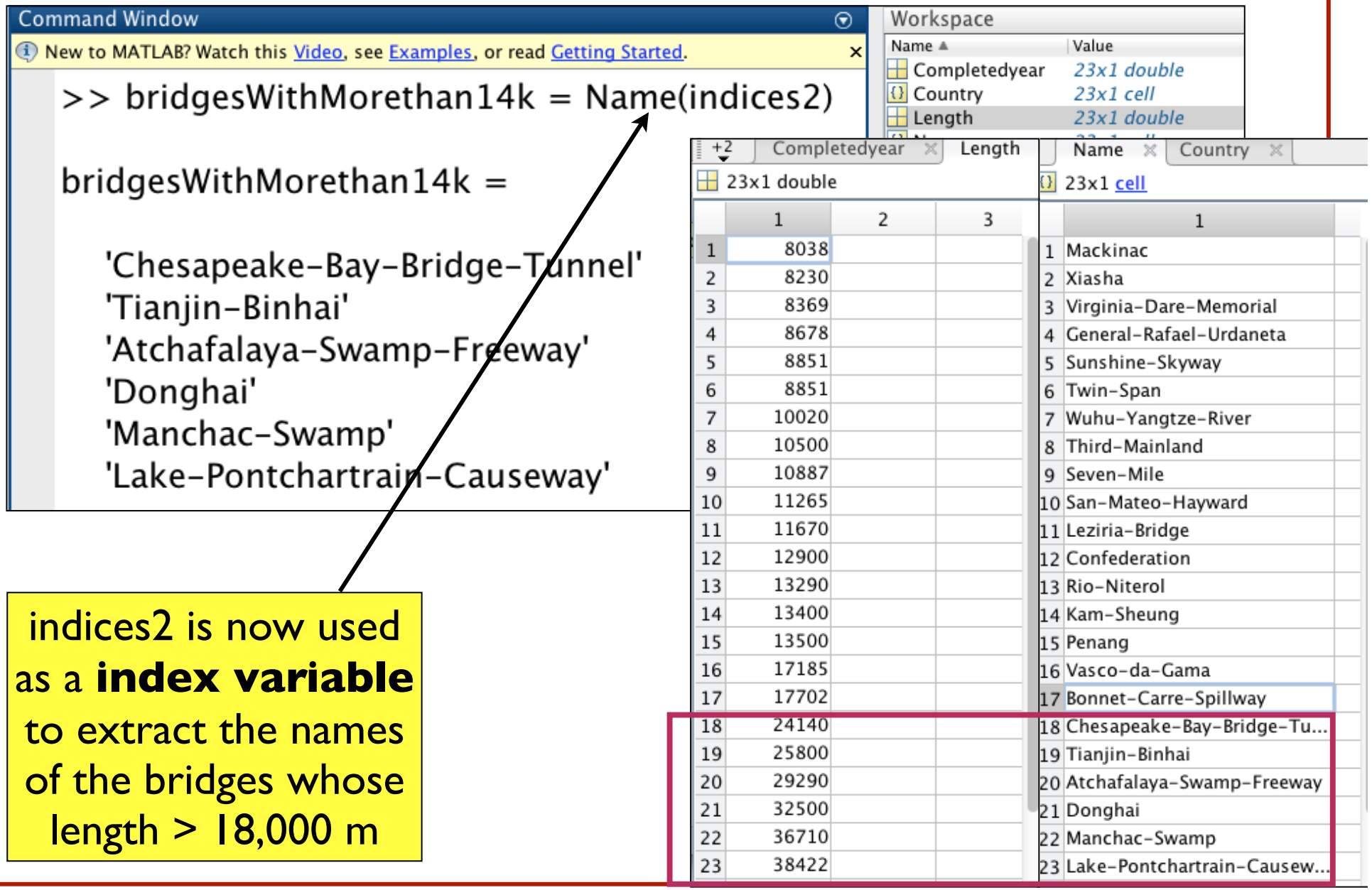
# Bridges with Length > 18,000 m

- Define a new **pointer variable** "indices2"
- Use the **Matlab "find" function** to find all bridges with bridge length > 18,000 m




**indices2** finds the indices of array **Length** with values > 18,000

# Bridges with Length > 18,000 m




indices2 is now used as a **index variable** to extract the names of the bridges whose length > 18,000 m

# Another Example: Find Bridges Built in United States

| Name | Country | Completed | Length (m) |
|---|---|---|---|
| Mackinac | United States | 1957 | 8038 |
| Xiasha | China | 1991 | 8230 |
| Virginia-Dare-Memorial | United States | 2002 | 8369 |
| General-Rafael-Urdaneta | Venezuela | 1962 | 8678 |
| Sunshine-Skyway | United States | 1987 | 8851 |
| Twin-Span | United States | 1960 | 8851 |
| Wuhu-Yangtze-River | China | 2000 | 10020 |
| Third-Mainland | Nigeria | 1991 | 10500 |
| Seven-Mile | United States | 1982 | 10887 |
| San-Mateo-Hayward | United States | 1967 | 11265 |
| Leziria-Bridge | Portugal | 2007 | 11670 |
| Confederation | Canada | 1997 | 12900 |
| Rio-Niterol | Brazil | 1974 | 13290 |
| Kam-Sheung | Hong Kong | 2003 | 13400 |
| Penang | Malaysia | 1985 | 13500 |
| Vasco-da-Gama | Portugal | 1998 | 17185 |
| Bonnet-Carre-Spillway | United States | 1960 | 17702 |
| Chesapeake-Bay-Bridge-Tunnel | United States | 1964 | 24140 |
| Tianjin-Binhai | China | 2003 | 25800 |
| Atchafalaya-Swamp-Freeway | United States | 1973 | 29290 |
| Donghai | China | 2005 | 32500 |
| Manchac-Swamp | United States | 1970 | 36710 |
| Lake-Pontchartrain-Causeway | United States | 1956 | 38422 |

# Bridges Built in the United States

- Define a new **index variable** "indices3"
- Use the **Matlab "strcmp" function** to compare string arrays to extract information needed

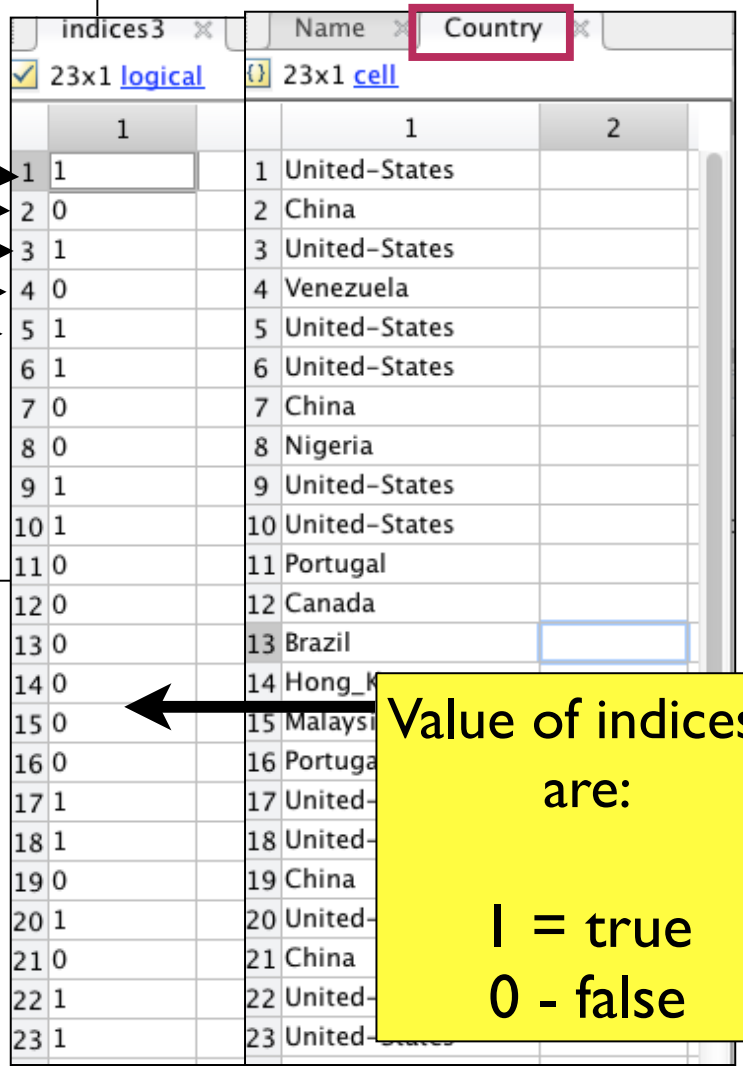
```
>> indices3=strcmp(Country,'United-States')

indices3 =

     1
     0
     1
     0
     1
     1
     0
     0
```

indices of variable Country that meet the criteria

1 = true
0 - false

# Bridges in the United States

```
>> indices3=strcmp(Country,'United-States')

indices3 =

     1
     0
     1
     0
     1
     1
     0
     0
```

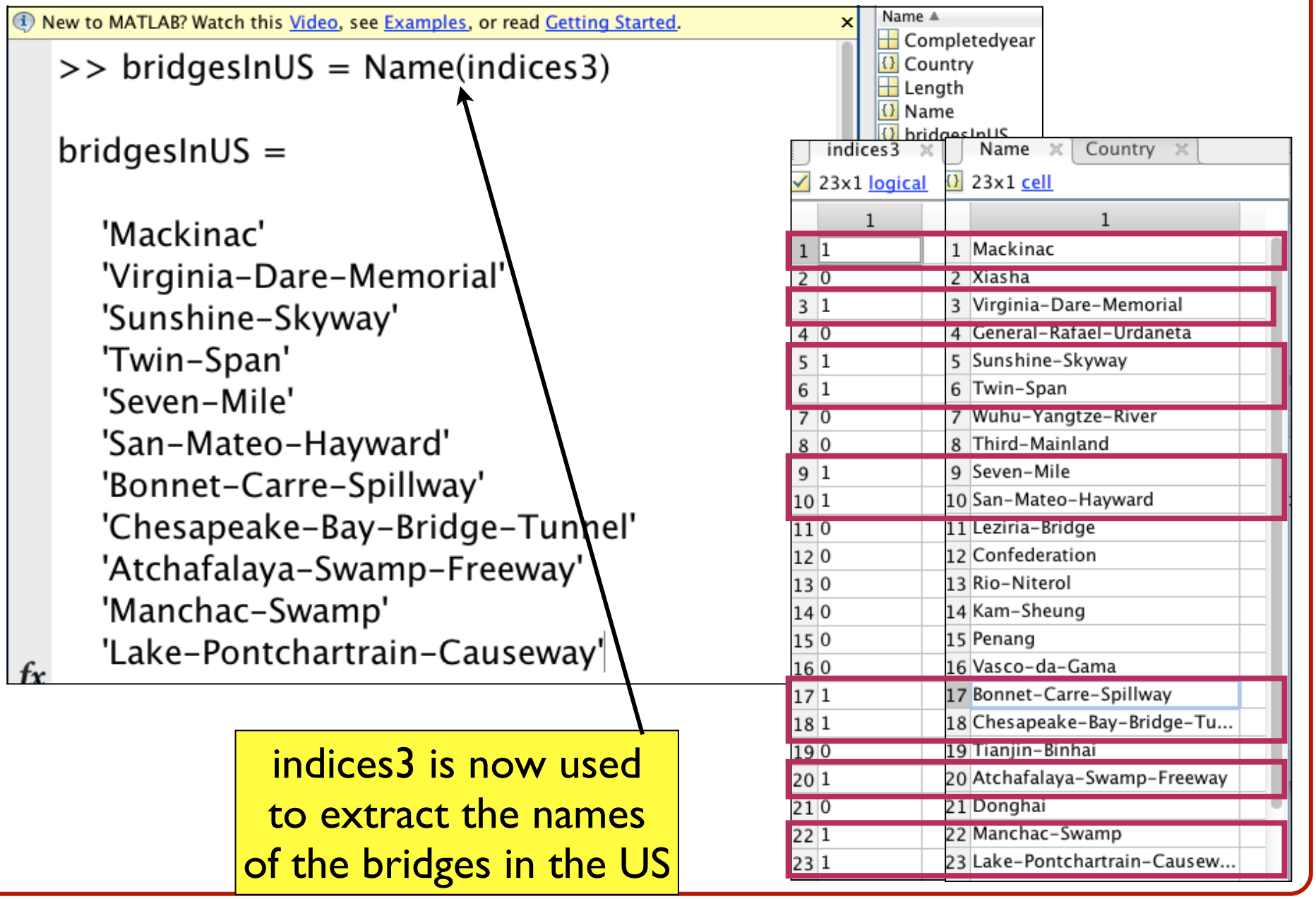



indices3 is a vector of 0-1 (called a **logical** variable) that contains information on what values of vector Country match the condition tested

Value of indices3 are:

1 = true
0 - false

# Bridges in the United States



# Summary

- Parsing data from a large set is relatively easy with Matlab
  - **"find"** - function is useful to find numeric data contained in large datasets
  - **"strcmp"** - function is useful to find string data contained in large datasets
- These functions can be used in regular Matlab scripts to parse or filter data and perform analyses