SECTION 8:
FILE I/O

ENGR 112 – Introduction to Engineering Computing

File I/O

- As engineers, we often generate large amounts of data
  - Simulation – in MATLAB or other simulation tools
  - Measurements
- Often need to process and analyze these data
  - Export data from simulator to a file
  - Read data into MATLAB
  - Process data in MATLAB – analysis, display, etc.
  - Write the data generated in MATLAB to a file

MATLAB Data (.mat) Files

.mat-Files

- Often want to store data generated in MATLAB to be later read back into MATLAB
  - Not interfacing with any other tools

- Can use MATLAB-specific .mat files for data storage
  - Useful when generating lots of data in MATLAB
    - e.g. running many time-consuming simulations
    - Save data to, possibly many, .mat-files
    - Load later for processing and analysis

- Save data with MATLAB's `save.m` function
- Load data with `load.m`

`save.m` and `load.m`

- To ***save*** workspace variables to a .mat-file

```
save(filename,variables)
```

  - *`filename`*: data saved to `filename.mat` – a *string*
  - *`variables`*: *optional* – workspace variables to be saved – default is to save all variables – enclose each in single quotes and separate by commas

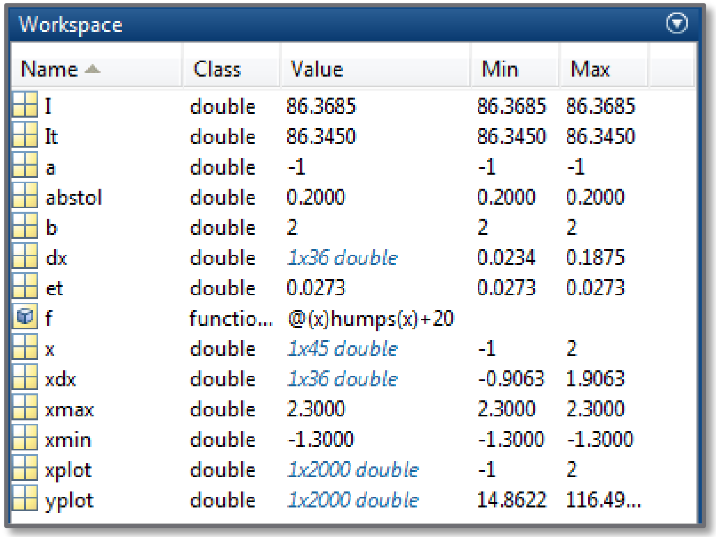
---

- To ***load*** workspace variables from a .mat-file

```
load(filename,variables)
```

  - *`filename`*: load data from `filename.mat` – a *string*
  - *`variables`*: *optional* – load only the specified variables to the workspace – enclose each in single quotes and separate by commas

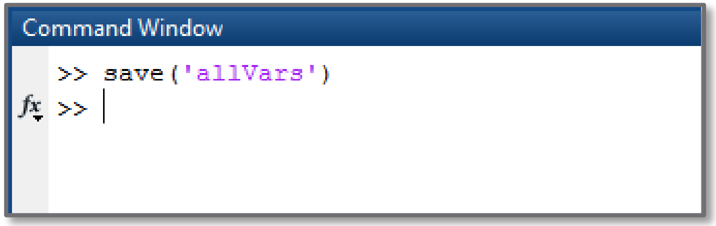
save.m – Example 1



Workspace

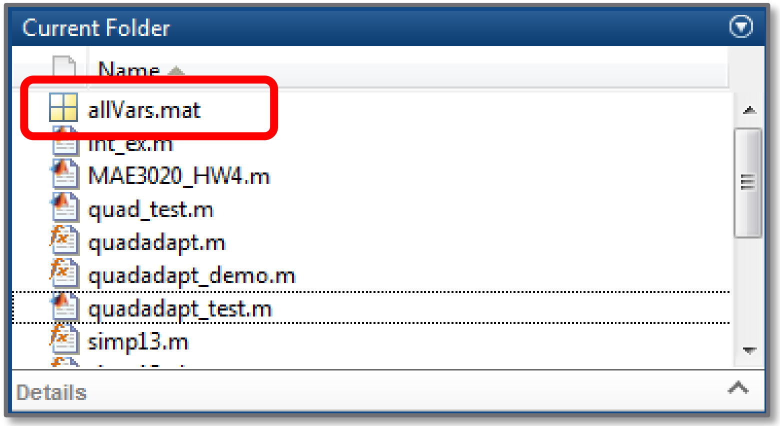
Name	Class	Value	Min	Max
I	double	86.3685	86.3685	86.3685
It	double	86.3450	86.3450	86.3450
a	double	-1	-1	-1
abstol	double	0.2000	0.2000	0.2000
b	double	2	2	2
dx	double	*1x36 double*	0.0234	0.1875
et	double	0.0273	0.0273	0.0273
f	functio...	@(x)humps(x)+20		
x	double	*1x45 double*	-1	2
xdx	double	*1x36 double*	-0.9063	1.9063
xmax	double	2.3000	2.3000	2.3000
xmin	double	-1.3000	-1.3000	-1.3000
xplot	double	*1x2000 double*	-1	2
yplot	double	*1x2000 double*	14.8622	116.49...

- Variable names not specified
  - All variables saved by default
- .mat extension appended to filename automatically



Command Window

```
>> save('allVars')
>> |
```



Current Folder

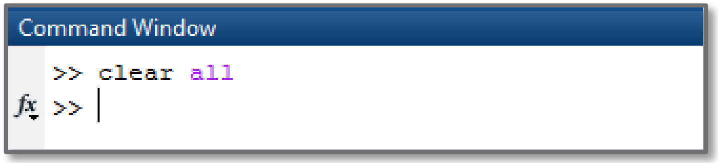
Name

- allVars.mat
- int_ex.m
- MAE3020_HW4.m
- quad_test.m
- quadadapt.m
- quadadapt_demo.m
- quadadapt_test.m
- simp13.m

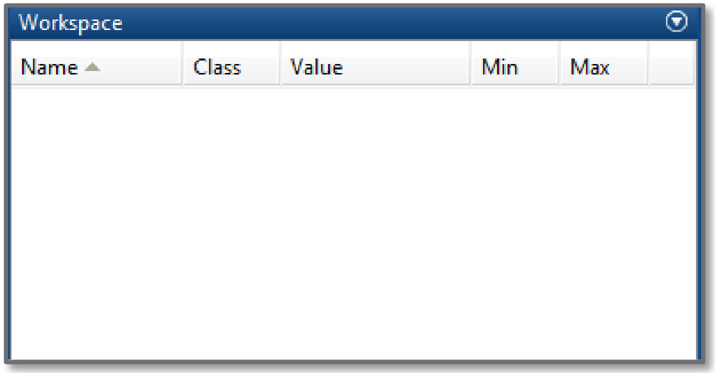
Details

`load.m` – Example 1

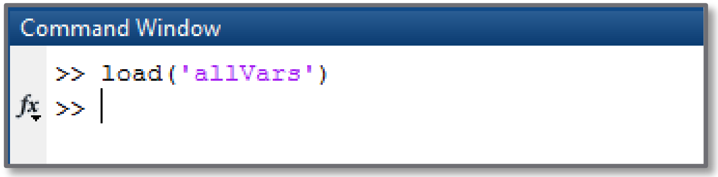
- □ All variables loaded by default
- □ Need not include .mat extension with filename



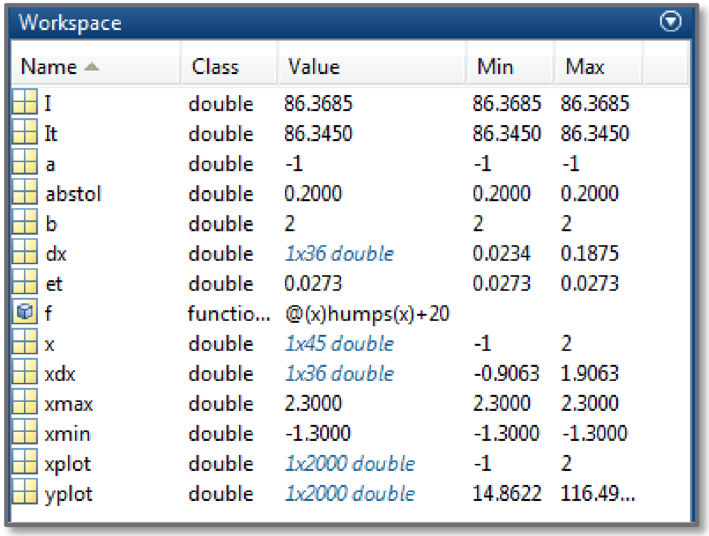
```
>> clear all
>> |
```



Name ▲	Class	Value	Min	Max

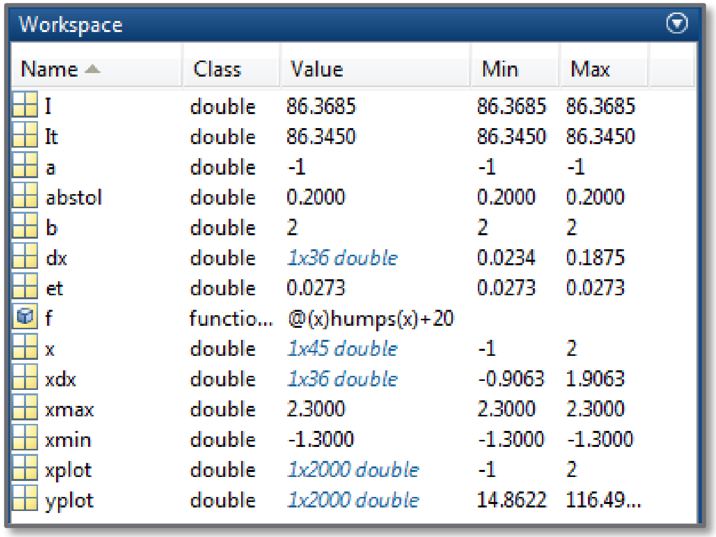


```
>> load('allVars')
>> |
```



Name ▲	Class	Value	Min	Max
I	double	86.3685	86.3685	86.3685
It	double	86.3450	86.3450	86.3450
a	double	-1	-1	-1
abstol	double	0.2000	0.2000	0.2000
b	double	2	2	2
dx	double	*1x36 double*	0.0234	0.1875
et	double	0.0273	0.0273	0.0273
f	functio...	@(x)humps(x)+20		
x	double	*1x45 double*	-1	2
xdx	double	*1x36 double*	-0.9063	1.9063
xmax	double	2.3000	2.3000	2.3000
xmin	double	-1.3000	-1.3000	-1.3000
xplot	double	*1x2000 double*	-1	2
yplot	double	*1x2000 double*	14.8622	116.49...

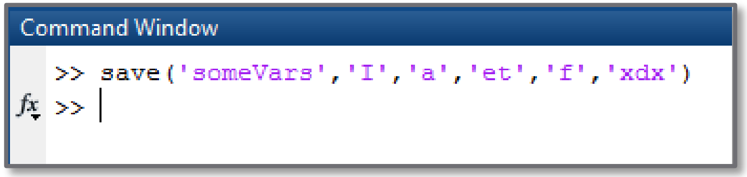
save.m – Example 2



Workspace

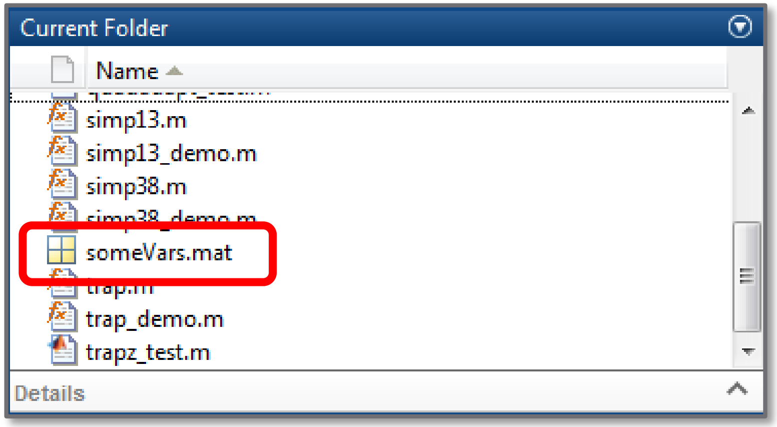
Name ▲	Class	Value	Min	Max
I	double	86.3685	86.3685	86.3685
It	double	86.3450	86.3450	86.3450
a	double	-1	-1	-1
abstol	double	0.2000	0.2000	0.2000
b	double	2	2	2
dx	double	*1x36 double*	0.0234	0.1875
et	double	0.0273	0.0273	0.0273
f	functio...	@(x)humps(x)+20		
x	double	*1x45 double*	-1	2
xdx	double	*1x36 double*	-0.9063	1.9063
xmax	double	2.3000	2.3000	2.3000
xmin	double	-1.3000	-1.3000	-1.3000
xplot	double	*1x2000 double*	-1	2
yplot	double	*1x2000 double*	14.8622	116.49...

- Save only specified variables
- Enclose each in single quotes
- Separate variables with commas



Command Window

```
>> save('someVars','I','a','et','f','xdx')
>> |
```



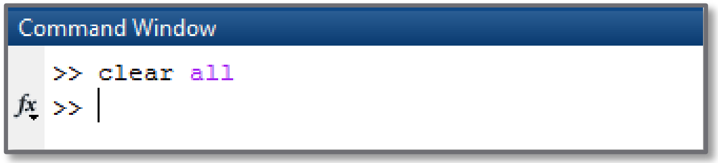
Current Folder

Name ▲

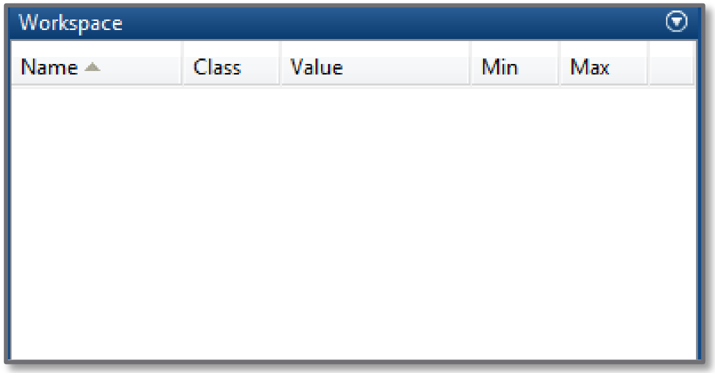
simp13.m
simp13_demo.m
simp38.m
simp38_demo.m
someVars.mat
trap.m
trap_demo.m
trapz_test.m

Details

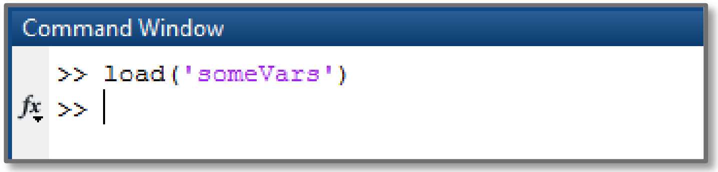
`load.m` – Example 2



```
>> clear all
>> 
```

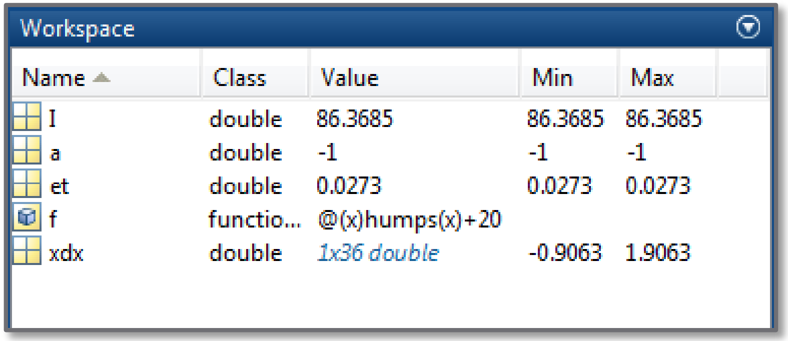


Name	Class	Value	Min	Max



```
>> load('someVars')
>> 
```

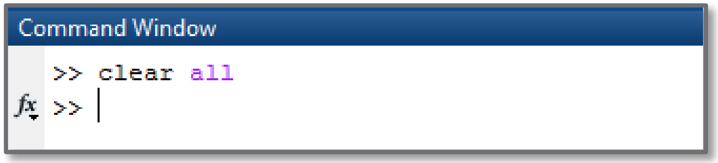
- □ `someVars.mat` file contains only a subset of the original workspace



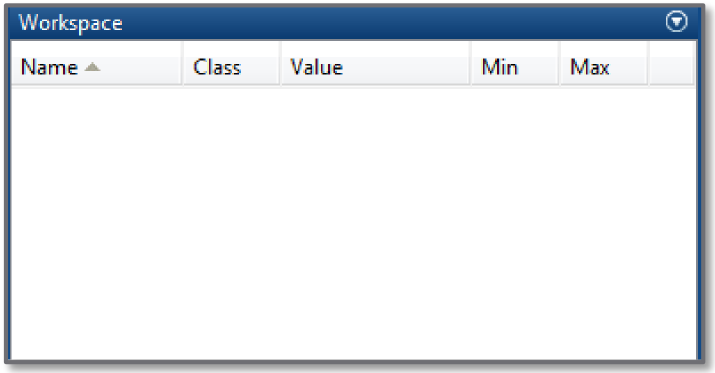
Name	Class	Value	Min	Max
I	double	86.3685	86.3685	86.3685
a	double	-1	-1	-1
et	double	0.0273	0.0273	0.0273
f	functio...	@(x)humps(x)+20		
xdx	double	1x36 double	-0.9063	1.9063

`load.m` – Example 3

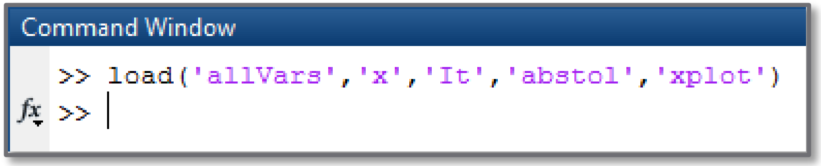
- Load only a subset of the variables in a .mat file
- Enclose each in single quotes
- Separate variables with commas



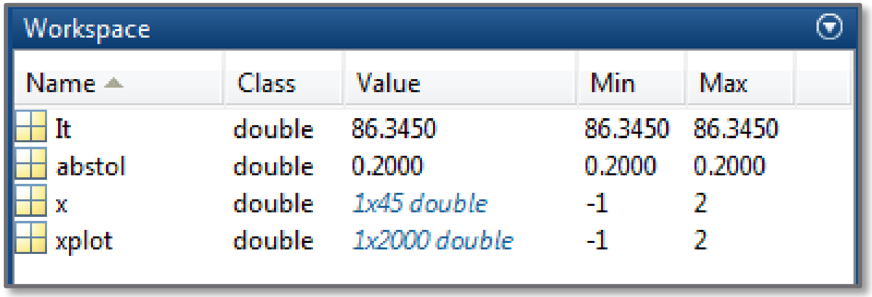
```
>> clear all
>> 
```



Name ▲	Class	Value	Min	Max



```
>> load('allVars','x','It','abstol','xplot')
>> 
```



Name ▲	Class	Value	Min	Max
It	double	86.3450	86.3450	86.3450
abstol	double	0.2000	0.2000	0.2000
x	double	*1x45 double*	-1	2
xplot	double	*1x2000 double*	-1	2

String and Number Formatting

String Formatting

- Often want to ***create strings that include variable values*** – numeric or strings
  - `sprintf.m` – write formatted data to an output string
  - `fprintf.m` – write formatted data to a text file or to the command window
- Can control the formatting of the variable values that are inserted into the string, e.g.:
  - integer
  - fixed point format
  - string
  - scientific notation
  - # of decimal places
  - etc.

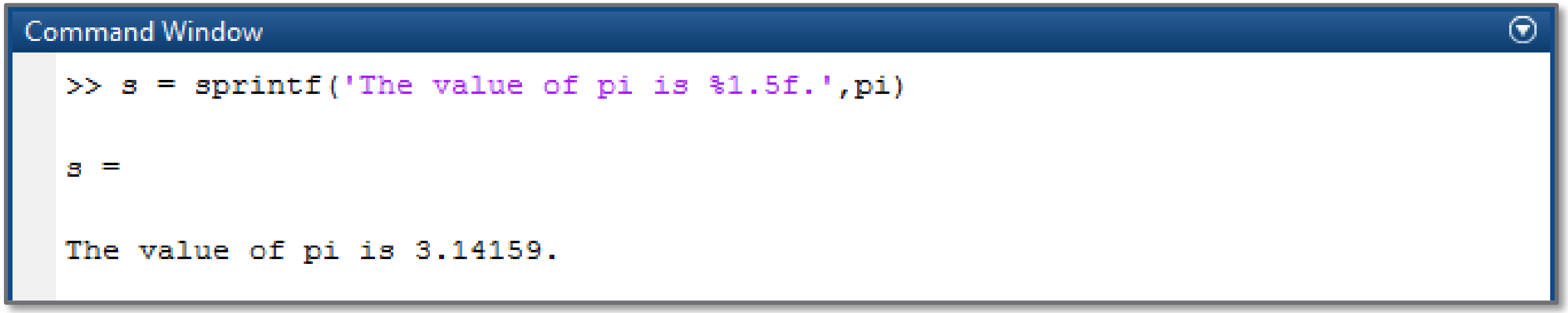
sprintf.m

- Write formatted data to an output string

```
str = sprintf(formatSpec,A1,A2,…,An)
```

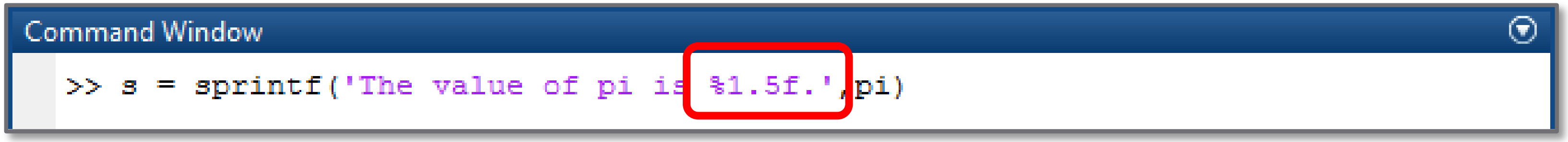
  - `formatSpec`: a *string* – may contain ***formatting sequences*** for insertion of variable values
  - `A1,A2,…,An`: variables whose values are to be inserted into the string – one for each formatting sequence in `formatSpec`
  - `str`: variable to which the created string is stored

- For example:



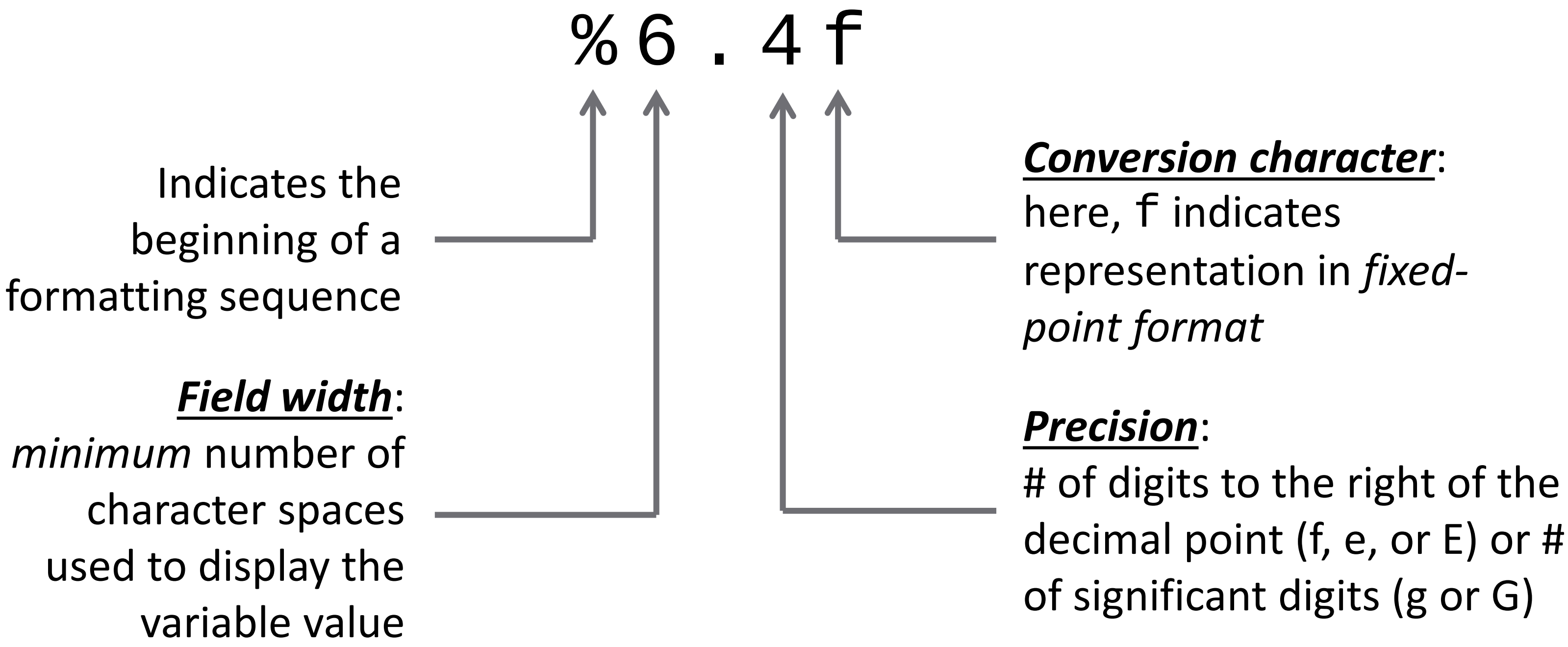
```
>> s = sprintf('The value of pi is %1.5f.',pi)

s =

The value of pi is 3.14159.
```

Formatting Sequences



```
>> s = sprintf('The value of pi is %1.5f.',pi)
```

- String may contain number ***formatting sequences***
  - Percent character (%) followed by conversion sequence



%6.4f

Indicates the beginning of a formatting sequence

Field width: *minimum* number of character spaces used to display the variable value

Conversion character: here, `f` indicates representation in *fixed-point format*

Precision: # of digits to the right of the decimal point (f, e, or E) or # of significant digits (g or G)

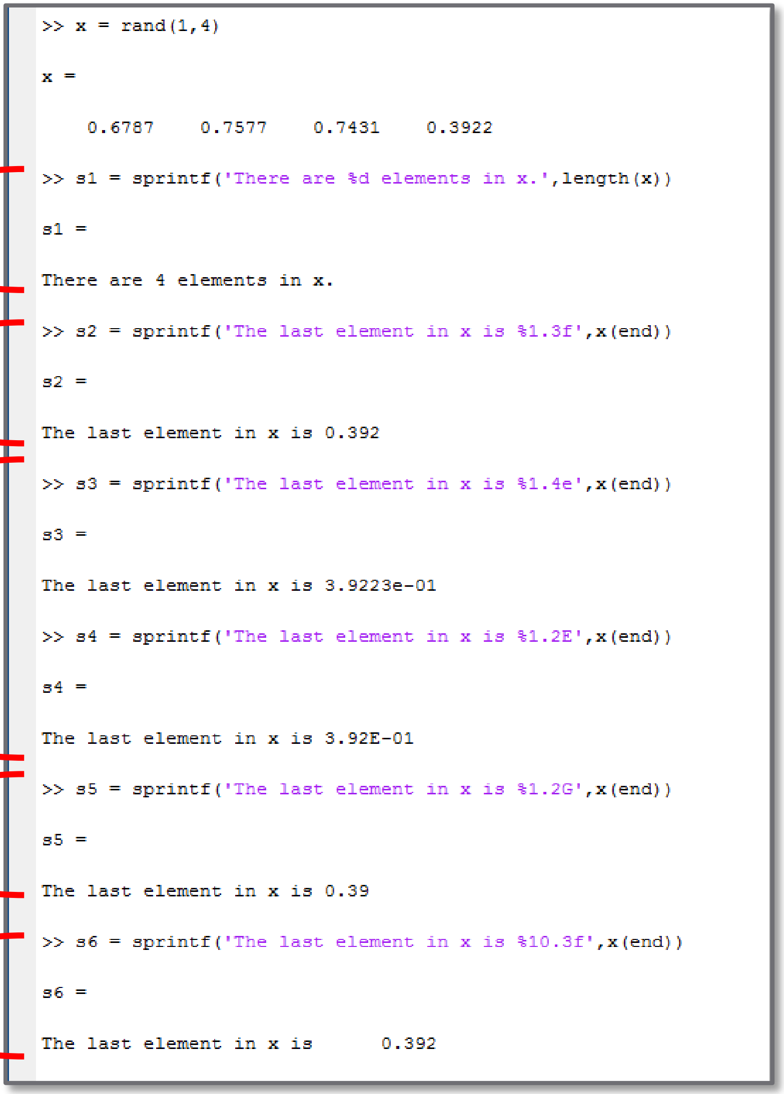
Conversion Characters

- Conversion characters specify how to format variable values within a string

Value Type	Conversion Character
Signed integer	%d
Unsigned integer	%u
Fixed-point notation	%f
Exponential notation (e.g., 1.6e-19)	%e
Exponential notation (e.g., 1.6E-19)	%E
More compact of %e or %f	%g
More compact of %E or%f	%G
Single character	%c
String	%s

Formatting Sequences – Examples

- □ Integer: %d
- □ Fixed-point: %f
- □ Exponential notation
- □ Compact format
- □ Field-width control



```
>> x = rand(1,4)
x =
    0.6787    0.7577    0.7431    0.3922
>> s1 = sprintf('There are %d elements in x.',length(x))
s1 =
There are 4 elements in x.
>> s2 = sprintf('The last element in x is %1.3f',x(end))
s2 =
The last element in x is 0.392
>> s3 = sprintf('The last element in x is %1.4e',x(end))
s3 =
The last element in x is 3.9223e-01
>> s4 = sprintf('The last element in x is %1.2E',x(end))
s4 =
The last element in x is 3.92E-01
>> s5 = sprintf('The last element in x is %1.2G',x(end))
s5 =
The last element in x is 0.39
>> s6 = sprintf('The last element in x is %10.3f',x(end))
s6 =
The last element in x is      0.392
```

Low-Level File I/O

Low-Level File I/O

- MATLAB includes many ***high-level functions*** for easily importing data from text files
  - Usually use these – very easy to use
  - We'll cover these later in the notes

- MATLAB also includes ***low-level functions*** for reading from and writing to files
  - More of a ***manual operation*** – line-by-line operation
  - ***Similar to other computer languages*** (e.g. C), which may not include simple high-level file I/O functions

Opening a Text File – `fopen.m`

- Prior to reading from or writing to a text file, we must first ***open the file***

```
fileID = fopen(filename,permission)
```

  - *`filename`*: name of the file to open – need not exist yet – a *string*
  - *`permission`*: *optional* – a string specifying file access type, e.g. read-only, write access, etc. – default is read-only
  - *`fileID`*: an integer file identifier – can be passed as input to functions, such as fscanf.m and fprintf.m

File Permissions

- □ Optional permission sequences indicate the type of file access when opening a file

Permission String	Description
`'r'`	Open file for reading (default)
`'w'`	Open or create new file for writing – discard existing contents
`'a'`	Open or create new file for writing – append data to the end of the file
`'r+'`	Open file for reading and writing
`'w+'`	Open or create new file for reading and writing – discard existing contents
`'a+'`	Open or create new file for reading and writing – append data to the end of the file

Closing a text file – `fclose.m`

- After opening and writing to or reading from a text file, that file must be closed

```
fclose(fid)
```

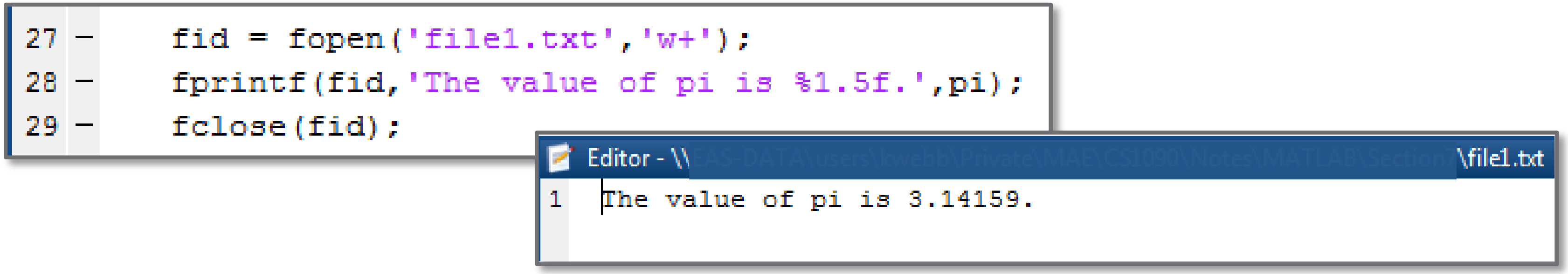
  - `fid` is the file identifier obtained from execution of the fopen command

Output Data to Text Files – `fprintf.m`

```
fprintf(fileID,formatSpec,A1,A2,…,An)
```

- `fileID`: *optional* file identifier – an integer obtained from an `fopen` command – if not specified, data is output to the command window
- `formatSpec`: a *string* – may contain ***formatting sequences*** for insertion of variable values
- `A1,A2,…,An`: variables whose values are to be inserted into the string – one for each formatting sequence in `formatSpec`

- For example:



```
fid = fopen('file1.txt','w+');
fprintf(fid,'The value of pi is %1.5f.',pi);
fclose(fid);
```

Editor - \\[redacted]\file1.txt

```
The value of pi is 3.14159.
```

Control Characters

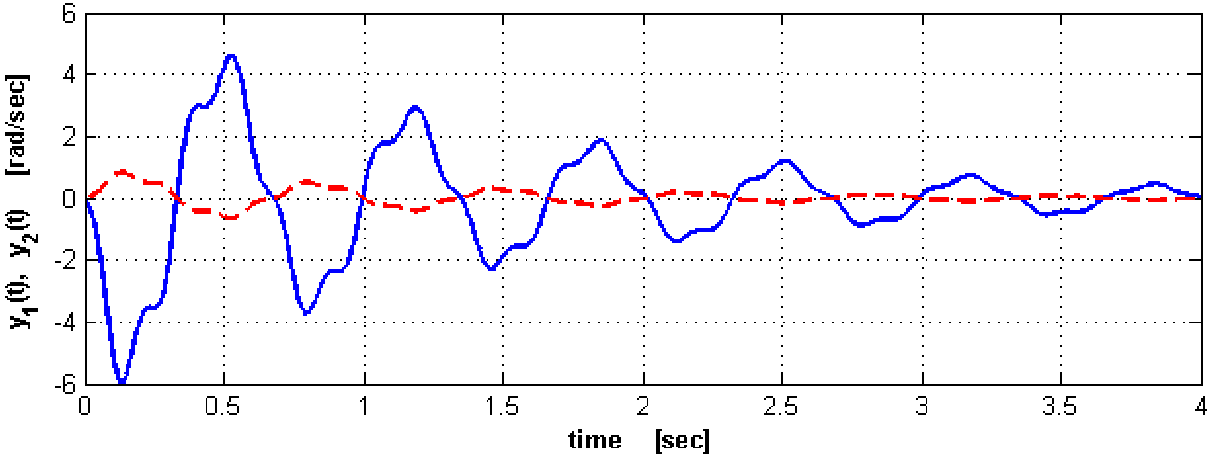
- ***Control characters*** are available for inserting things like ***tabs, new lines, and special characters***

Control Character	Description
%%	Percent character
\\	Backslash
\t	Horizontal tab
\n	New line

- These are a few of the more common control characters
  - See MATLAB documentation for more

Writing to a Text File – Example

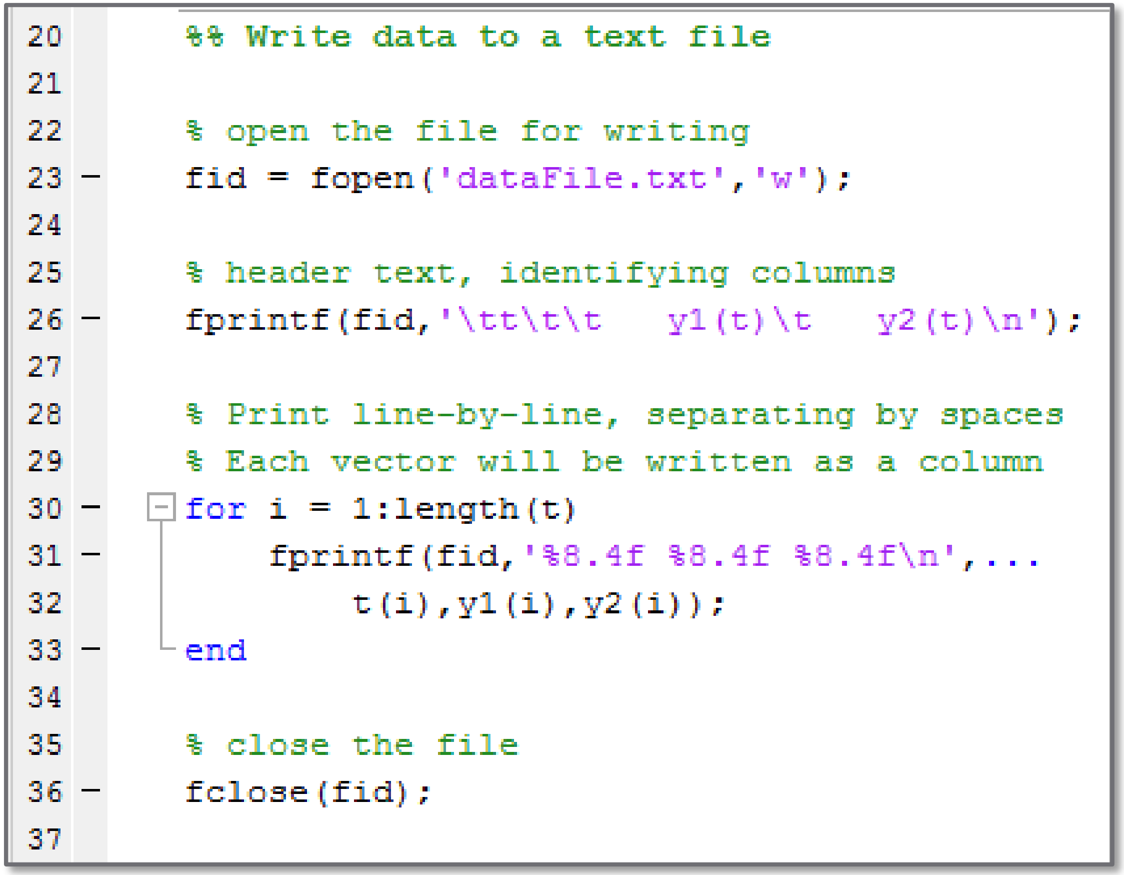
- Let's say you generated data from a simulation in MATLAB
  - Time vector and two corresponding output vectors
- Want to save these data to a text file for processing and analysis at a later time



- Save the data to a text file as three columns
  - t, $y_1(t)$, and $y_2(t)$

Writing to a Text File – Example

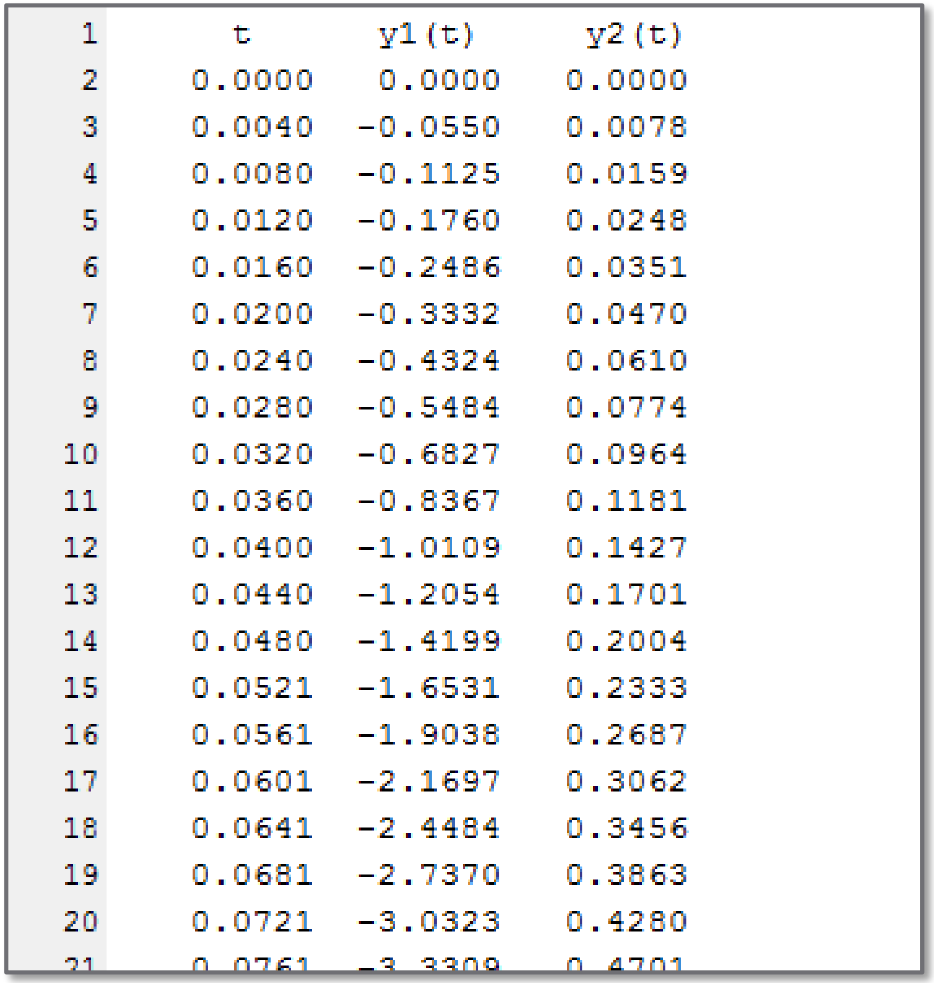
- Write vectors as columns
- Write data line-by-line
- Here, columns are separated by spaces
  - Could be tabs or commas, or …



```
%% Write data to a text file

% open the file for writing
fid = fopen('dataFile.txt','w');

% header text, identifying columns
fprintf(fid,'\tt\t\t   y1(t)\t   y2(t)\n');

% Print line-by-line, separating by spaces
% Each vector will be written as a column
for i = 1:length(t)
    fprintf(fid,'%8.4f %8.4f %8.4f\n',...
        t(i),y1(i),y2(i));
end

% close the file
fclose(fid);
```

Writing to a Text File – Example

- The resulting text file:



```
      t         y1(t)      y2(t)
   0.0000     0.0000     0.0000
   0.0040    -0.0550     0.0078
   0.0080    -0.1125     0.0159
   0.0120    -0.1760     0.0248
   0.0160    -0.2486     0.0351
   0.0200    -0.3332     0.0470
   0.0240    -0.4324     0.0610
   0.0280    -0.5484     0.0774
   0.0320    -0.6827     0.0964
   0.0360    -0.8367     0.1181
   0.0400    -1.0109     0.1427
   0.0440    -1.2054     0.1701
   0.0480    -1.4199     0.2004
   0.0521    -1.6531     0.2333
   0.0561    -1.9038     0.2687
   0.0601    -2.1697     0.3062
   0.0641    -2.4484     0.3456
   0.0681    -2.7370     0.3863
   0.0721    -3.0323     0.4280
   0.0761    -3.3309     0.4701
```

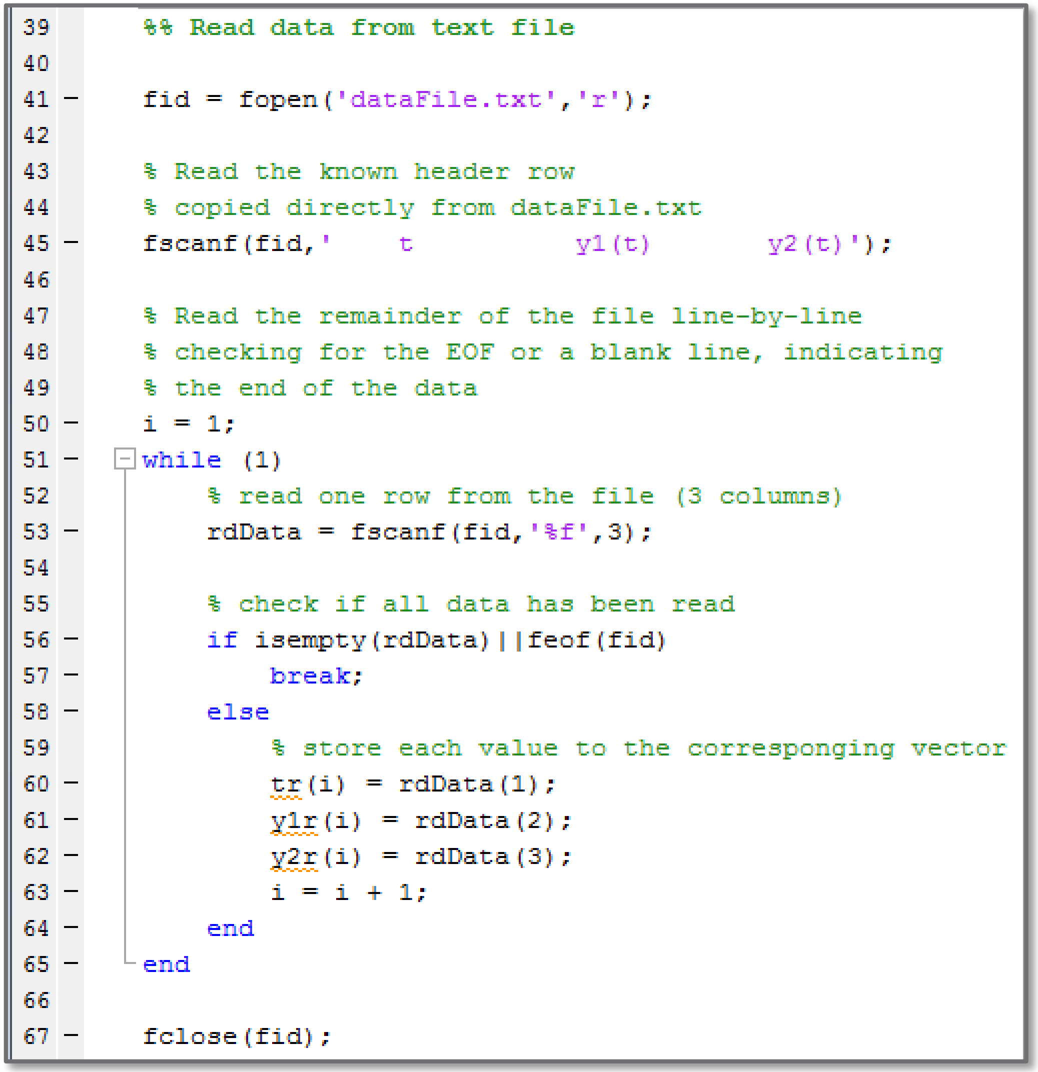
Reading Data from Text Files – `fscanf.m`

`A = fscanf(fileID,format,sizeA)`

- *`fileID`*: file identifier –obtained from `fopen`
- *`format`*: a *string* enclosed in single quotes, describing the contents of each field to be read – conversion characters
- *`sizeA`*: *optional* – dimension of the output matrix, specified as:
    - `inf`: read to end of file (default) and store as a column vector
    - `n`: read n elements and store as an `n×1` column vector
    - `[m,n]`: read `m*n` elements, row-by-row, and store in *column order* as an `m×n` matrix
- *A*: output matrix – stored in *column* order, even though data is read line-by-line (row-by-row)

Reading from a Text File – Example

- First read header line
  - File pointer advances to start of data on the following line
- Read line-by-line
  - Three elements at a time – one from each column
  - Store each element to its corresponding vector
- Continue reading data until EOF or a blank line is reached



```
%% Read data from text file

fid = fopen('dataFile.txt','r');

% Read the known header row
% copied directly from dataFile.txt
fscanf(fid,'    t           y1(t)        y2(t)');

% Read the remainder of the file line-by-line
% checking for the EOF or a blank line, indicating
% the end of the data
i = 1;
while (1)
    % read one row from the file (3 columns)
    rdData = fscanf(fid,'%f',3);

    % check if all data has been read
    if isempty(rdData)||feof(fid)
        break;
    else
        % store each value to the corresponging vector
        tr(i) = rdData(1);
        y1r(i) = rdData(2);
        y2r(i) = rdData(3);
        i = i + 1;
    end
end

fclose(fid);
```

High-Level File I/O Functions

importdata.m

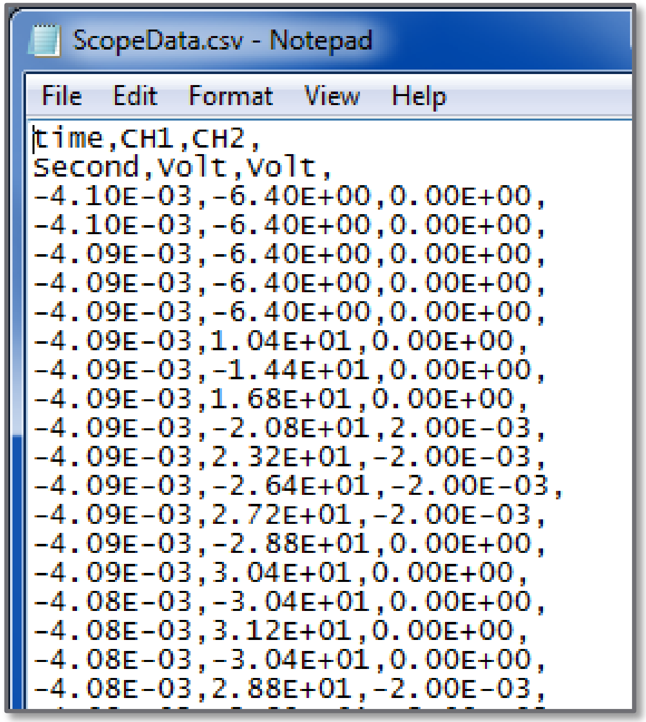
- Load column-oriented data from a text file

```
A = importdata(filename,delim,nheaderlines)
```

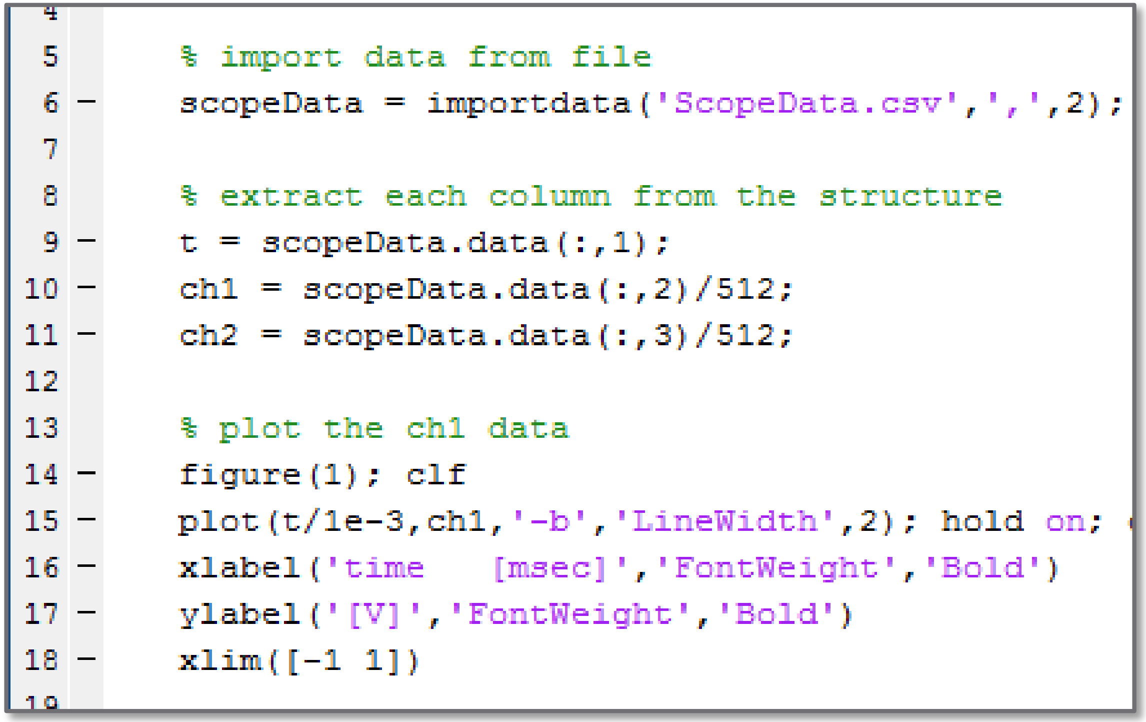
  - `filename`: name of file from which to read – a string
  - `delim`: type of delimiter between columns – a string, e.g., `'\t'` or `','` or `' '`, etc.
  - `nheaderlines`: number of non-data header lines in the file – data is read starting at `nheaderlines + 1`
  - A: data stored as either a *matrix*, *multi-dimensional array*, or a *structure*, depending on file format

importdata.m – Example

- Oscilloscope data
  - A comma-separated-variable, .csv, file
  - Three sets of data: time, channel 1 data, and Channel 2 data
  - Two header lines at the top of the file



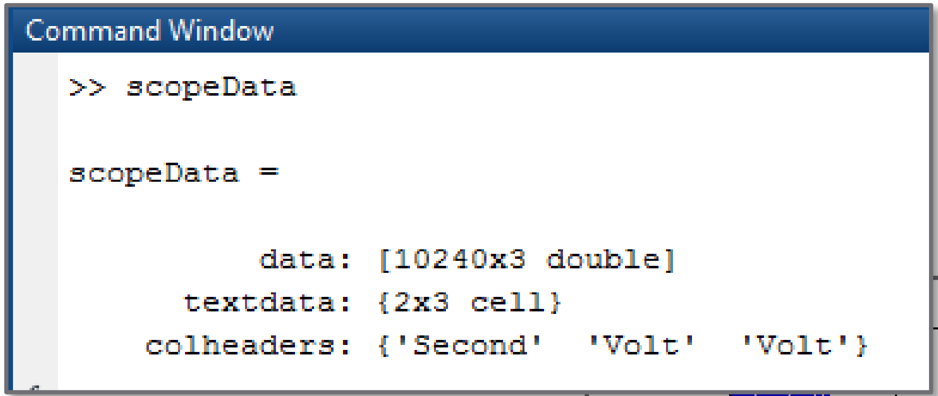
```
time,CH1,CH2,
Second,Volt,Volt,
-4.10E-03,-6.40E+00,0.00E+00,
-4.10E-03,-6.40E+00,0.00E+00,
-4.09E-03,-6.40E+00,0.00E+00,
-4.09E-03,-6.40E+00,0.00E+00,
-4.09E-03,-6.40E+00,0.00E+00,
-4.09E-03,1.04E+01,0.00E+00,
-4.09E-03,-1.44E+01,0.00E+00,
-4.09E-03,1.68E+01,0.00E+00,
-4.09E-03,-2.08E+01,2.00E-03,
-4.09E-03,2.32E+01,-2.00E-03,
-4.09E-03,-2.64E+01,-2.00E-03,
-4.09E-03,2.72E+01,-2.00E-03,
-4.09E-03,-2.88E+01,0.00E+00,
-4.09E-03,3.04E+01,0.00E+00,
-4.08E-03,-3.04E+01,0.00E+00,
-4.08E-03,3.12E+01,0.00E+00,
-4.08E-03,-3.04E+01,0.00E+00,
-4.08E-03,2.88E+01,-2.00E-03,
```



```
% import data from file
scopeData = importdata('ScopeData.csv',',',2);

% extract each column from the structure
t = scopeData.data(:,1);
ch1 = scopeData.data(:,2)/512;
ch2 = scopeData.data(:,3)/512;

% plot the ch1 data
figure(1); clf
plot(t/1e-3,ch1,'-b','LineWidth',2); hold on;
xlabel('time   [msec]','FontWeight','Bold')
ylabel('[V]','FontWeight','Bold')
xlim([-1 1])
```

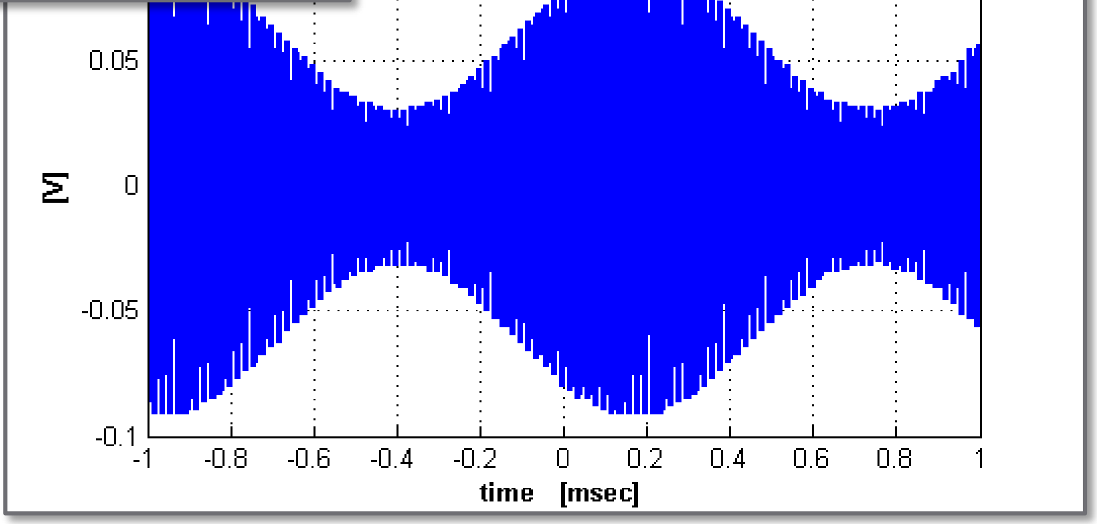
importdata.m – Example

- In this case the data is read in as a ***structure***:



```
>> scopeData

scopeData =

           data: [10240x3 double]
       textdata: {2x3 cell}
     colheaders: {'Second'   'Volt'   'Volt'}
```

- Plotting the channel 1 data:



`xlsread.m`

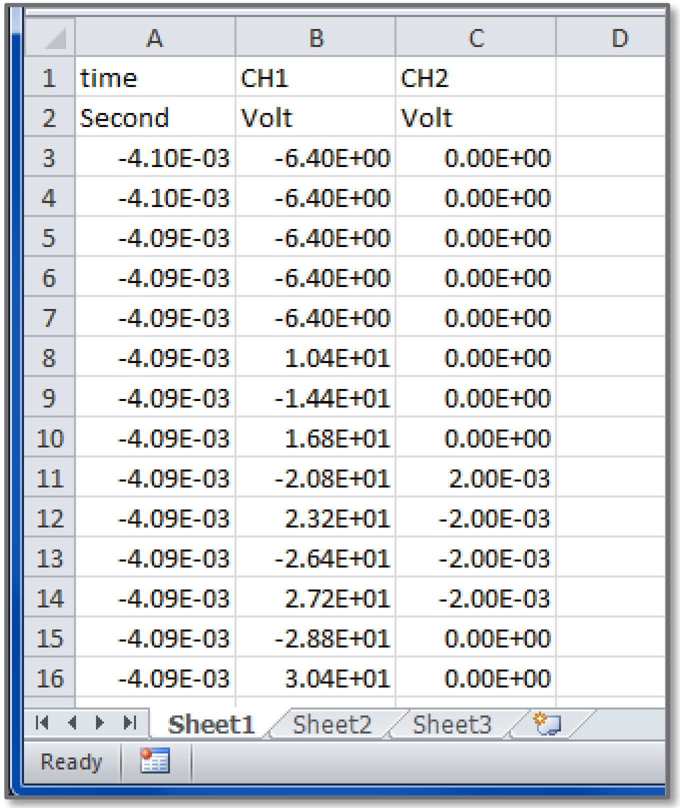
- Read data from a Microsoft Excel spreadsheet file

```
A = xlsread(filename,sheet,range)
```

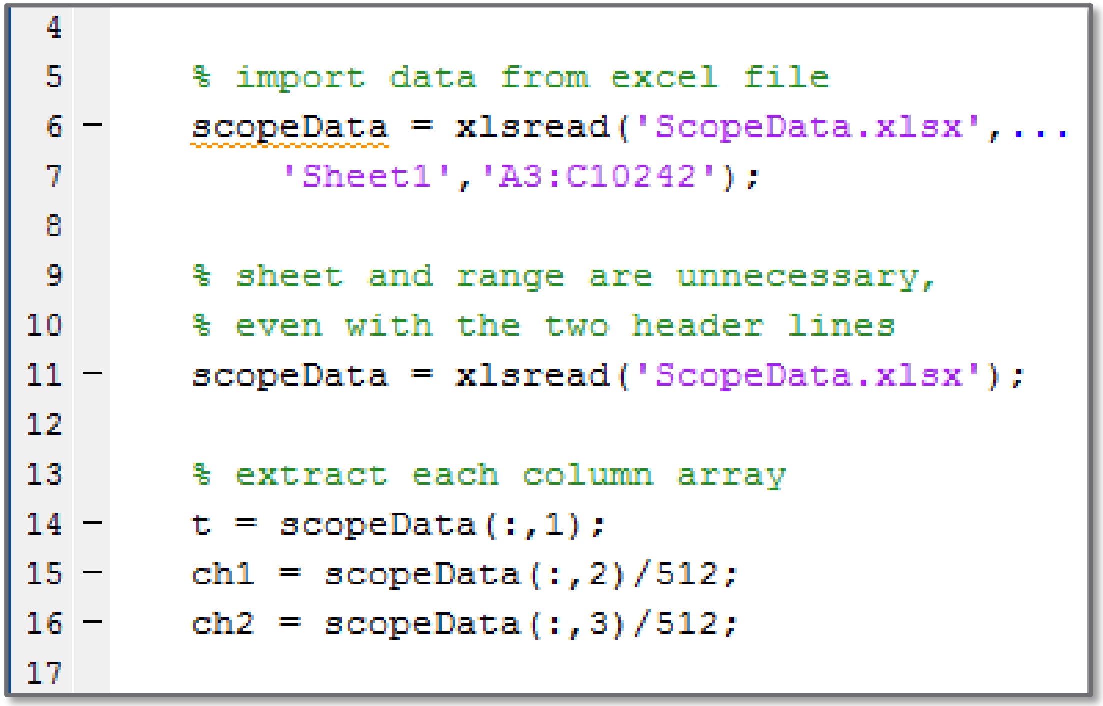
  - `filename`: name of Excel file – a string
  - `sheet`: *optional* name of worksheet within the workbook – a string, e.g., `'sheet1'` – default is the first sheet
  - `range`: *optional* rectangular cell range to read – a string, e.g., `'B2:D43'` – default is to read all data
  - A: matrix of imported data

xlsread.m – Example

- Now, read the same data from an Excel spreadsheet
  - Data is on first sheet – need not specify sheet or range
  - Text column labels are skipped automatically



	A	B	C	D
1	time	CH1	CH2	
2	Second	Volt	Volt	
3	-4.10E-03	-6.40E+00	0.00E+00	
4	-4.10E-03	-6.40E+00	0.00E+00	
5	-4.09E-03	-6.40E+00	0.00E+00	
6	-4.09E-03	-6.40E+00	0.00E+00	
7	-4.09E-03	-6.40E+00	0.00E+00	
8	-4.09E-03	1.04E+01	0.00E+00	
9	-4.09E-03	-1.44E+01	0.00E+00	
10	-4.09E-03	1.68E+01	0.00E+00	
11	-4.09E-03	-2.08E+01	2.00E-03	
12	-4.09E-03	2.32E+01	-2.00E-03	
13	-4.09E-03	-2.64E+01	-2.00E-03	
14	-4.09E-03	2.72E+01	-2.00E-03	
15	-4.09E-03	-2.88E+01	0.00E+00	
16	-4.09E-03	3.04E+01	0.00E+00	



```

% import data from excel file
scopeData = xlsread('ScopeData.xlsx',...
    'Sheet1','A3:C10242');

% sheet and range are unnecessary,
% even with the two header lines
scopeData = xlsread('ScopeData.xlsx');

% extract each column array
t = scopeData(:,1);
ch1 = scopeData(:,2)/512;
ch2 = scopeData(:,3)/512;

```

xlswrite.m

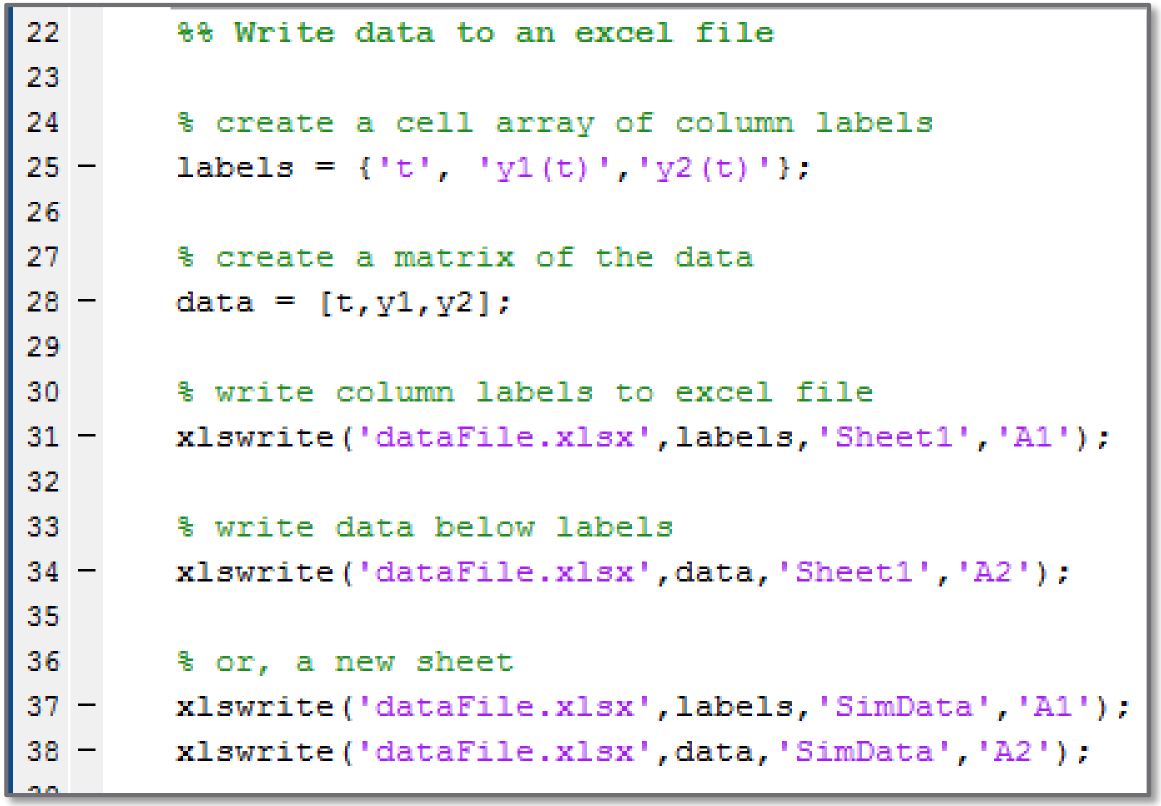
- Write MATLAB data to an Excel spreadsheet

```
xlswrite(filename,A,sheet,range)
```

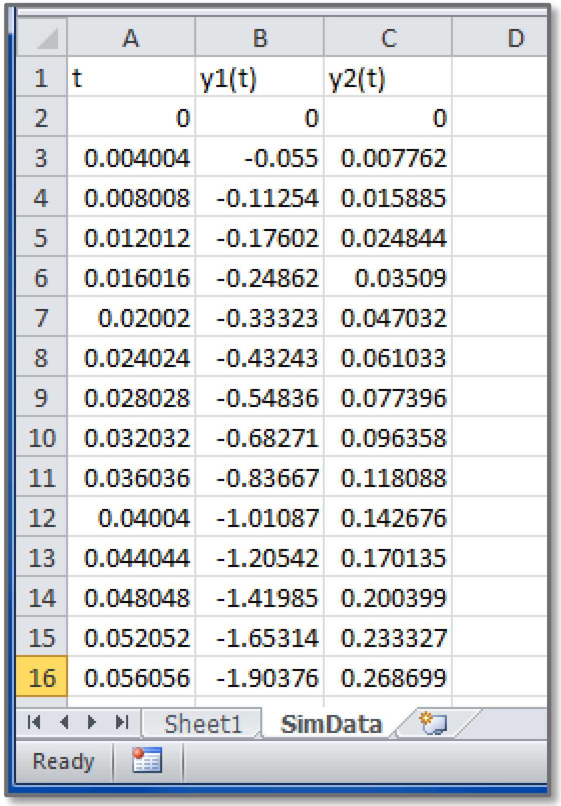
  - *filename*: name of Excel file – a string – if file does not exist, it will be created
  - A: matrix of data to export
  - *sheet*: *optional* name of worksheet within the workbook – a string, e.g., ‘sheet1’ – default is the first sheet
  - *range*: *optional* rectangular cell range – if sheet is specified then only the upper left-hand cell need be specified, e.g., ‘C2’, if not rectangular range is required, e.g., ‘C2:E18’

xlswrite.m – Example

- Write MATLAB simulation data to an Excel file



```
%% Write data to an excel file

% create a cell array of column labels
labels = {'t', 'y1(t)','y2(t)'};

% create a matrix of the data
data = [t,y1,y2];

% write column labels to excel file
xlswrite('dataFile.xlsx',labels,'Sheet1','A1');

% write data below labels
xlswrite('dataFile.xlsx',data,'Sheet1','A2');

% or, a new sheet
xlswrite('dataFile.xlsx',labels,'SimData','A1');
xlswrite('dataFile.xlsx',data,'SimData','A2');
```



	A	B	C	D
1	t	y1(t)	y2(t)	
2	0	0	0	
3	0.004004	-0.055	0.007762	
4	0.008008	-0.11254	0.015885	
5	0.012012	-0.17602	0.024844	
6	0.016016	-0.24862	0.03509	
7	0.02002	-0.33323	0.047032	
8	0.024024	-0.43243	0.061033	
9	0.028028	-0.54836	0.077396	
10	0.032032	-0.68271	0.096358	
11	0.036036	-0.83667	0.118088	
12	0.04004	-1.01087	0.142676	
13	0.044044	-1.20542	0.170135	
14	0.048048	-1.41985	0.200399	
15	0.052052	-1.65314	0.233327	
16	0.056056	-1.90376	0.268699	