

Armstrong State University
Engineering Studies
MATLAB Marina – File Input and Output Primer

Prerequisites

The File Input and Output Primer assumes knowledge of the MATLAB IDE, MATLAB help, arithmetic operations, built in functions, scripts, variables, arrays, logic expressions, conditional structures, iteration, functions, debugging, characters and strings, cell arrays, and structures. Material on these topics is covered in the MATLAB Marina Introduction to MATLAB module, MATLAB Marina Variables module, MATLAB Marina Arrays module, MATLAB Marina Logic Expressions module, MATLAB Marina Conditional Structures module, MATLAB Marina Iteration module, MATLAB Marina Functions module, MATLAB Marina debugging module, MATLAB Marina Character and Strings module, MATLAB Marina Cell Arrays module, and MATLAB Marina Structures module.

Learning Objectives

1. Be able to save and restore MATLAB workspace variables.
2. Be able to write functions and programs that read data from delimited text files and write data to text files.
3. Be able to write functions and programs that read data from Microsoft Excel files and write data to Microsoft Excel files.

Terms

MATLAB (.mat) file, text file, Excel file, delimiter, file input, file output

MATLAB Functions, Keywords, and Operators

load, save, dlmread, dlmwrite, csvread, csvwrite, importdata, xlsread, xlsxwrite

File Formats

Data files are serial input/output (I/O) streams, i.e. the data is saved or retrieved sequentially. Data is stored as binary values (1s and 0s). How the data is interpreted depends upon the type or format of the data file. For example, reading an ASCII text file like it is an image file will result in garbage.

File formats are typically indicated by the file's extension. The file extension specifies the nature and organization of the data in the file. MATLAB has built in read and write operations for most common types of data files. General numeric data is typically stored in one of the following formats: Microsoft Excel file, plain text file, or text file with a delimiter such as a comma, space, tab, or pipe (|) separating values. Some of the more commonly encountered file types and extensions are shown in Figure 1. See MATLAB's help for more information on the various read and write operations.

File Type	Extension	MATLAB read/write operations
MATLAB data	.mat	save, load
Plain text	.txt, .dat, etc.	textscan, fprintf
Delimited text	varies	dlmread, dlmwrite
Comma separated	.csv	csvread, csvwrite
Excel	.xls, .xlsx	xlsread, xlswrite
Audio	.wav	wavread, wavwrite
image	.bmp, .jpg, etc.	imread, imwrite
XML	.xml	xmlread, xmlwrite
Movie	.avi	aviread

Figure 1, Table of Commonly Encountered File Types

Saving and Restoring MATLAB Workspace

The `save` and `load` functions allow one to save the workspace variables to a file and load previously saved workspaces. Saving the workspace saves the variables in the workspace not any code that generated the variables. The entire MATLAB workspace is saved by using the `save` function and providing a file name to save the workspace in. Specific variables in the workspace can be saved in a similar manner by providing which variables to save. The variable names are preserved when saving and restoring from `.mat` files. Previously saved workspaces or portions of a previously saved workspace can be loaded using the `load` function.

Figure 2a shows an example of using the `save` function to save the MATLAB workspace and Figure 2b shows an example of using the `load` function to restore a MATLAB workspace.

```
>> save workspacedata.mat;
>> save partialworkspacedata.mat t x y;
```

Figure 2a, Using `save` Function to Save MATLAB Workspace

```
>> load workspacedata.mat;
>> load workspacedata.mat t x;
```

Figure 2b, Using `load` Function to Restore MATLAB Workspace

In Figure 2a, the first `save` statement, saves the entire workspace (all variables) and the second `save` statement saves only the three specified workspace variables (`t`, `x`, and `y`). In Figure 2b, the first `load` statement, restores the entire workspace saved in the file `workspacedata.mat`, and the second `load` statement will restore only the two specified variables `t` and `x` saved in the file `workspacedata.mat`.

MATLAB workspace (`.mat`) files are binary files and not generally readable by software packages other than MATLAB.

Reading Numeric Data from Delimited Text Files

Text files and Microsoft Excel files are widely used for storing data. Text files and Microsoft Excel files can be read in many commonly used software packages and thus are generally good file type choices for distributing data.

Data in text files is commonly stored with each items' data values along columns (one per line). Multiple items are separated in the rows using a delimiter such as a space or comma. Numeric data in text files may have string column and row headers. Text files are coded in ASCII and can be read (opened) using software such as Notepad.

Consider loading automobile mileage data from a comma delimited text file and computing the gas mileage an automobile is getting. The mileage data from the text file is shown in Figure 2a and has the miles driven and fuel consumed separated by commas, one entry per line. The program of Figure 2b loads the data from the text file, computes the miles per gallon for each entry, and the average miles per gallon. The data is imported as a 7 by 2 numeric array and the miles and fuel variables are column vectors.

```
315.0, 16.4
306.8, 15.3
262.2, 12.0
241.5, 11.8
279.1, 14.1
288.0, 14.0
251.6, 12.9
```

Figure 2a, Mileage Data

```
% read mileage data from comma delimited text file
mileageData = dlmread('mileage.txt','');
% extract miles traveled and fuel consumed
miles = mileageData(:,1);
fuel = mileageData(:,2);
% compute mpg for each fill up and average mpg
mpg = miles./fuel;
avg_mpg = sum(miles)/sum(fuel);
```

Figure 2b, MATLAB Program for Importing and Processing Mileage Data

The MATLAB function `dlmread` reads numeric data from a delimited ASCII file. Typical delimiters are spaces, commas, tabs, and pipes (`|`). The MATLAB command `dlmwrite` can be used to write a numeric array of data to a file. For example, the MATLAB command `dlmwrite('mpg.txt',results,'/t')` would write the numeric data contained in the array `results` to the text file `mpg` using a tab as a delimiter to separate the data. Other versions of `dlmwrite` allow one to append data to an existing file, write data starting at a specific row and column in a text file, and allowing the precision of the data to be specified. MATLAB also has functions, `csvread` and `csvwrite` specifically for reading numeric data from a comma

delimited file and writing numeric data to a comma delimited file. See MATLAB's help on `dlmread` and `dlmwrite` for all the optional parameters.

One must be careful when appending data to files: one must ensure that the original data is not written over (destroyed) and that if the file will be used for input later that the new file format is taken into account when loading the data.

Reading Data from Delimited Text Files

The MATLAB functions `dlmread` and `csvread` only work with text files containing delimited numeric data. Many files have text strings for column and row headers. One could manually open the text file and remove any strings before working with file, but MATLAB's `importdata` and `textscan` functions allow one to read in data from text files containing numeric and string data.

The `importdata` function either tries to use one of the helper input functions if it can determine the type of file or treats the file as a delimited text file if it cannot determine the type of file. If the file helper function returns more than one nonempty result, the `importdata` function combines the results into a structure array. For text files, the `importdata` function expects the numeric data to be in a rectangular form and text strings to either be above or to the left of the numeric data. Numeric data is imported as a 2D array of real numbers and string data is imported as cell array of strings, one each for row headings, column headings, and all the text strings.

For example, a mileage data text file with row and column headings as shown in Figure 3a could be imported using the program of Figure 3b.

```
date, miles, gallons
1/14/2011, 315.0, 16.4
1/22/2011, 306.8, 15.3
2/3/2011, 262.2, 12.0
2/8/2011, 241.5, 11.8
2/15/2011, 279.1, 14.1
2/25/2011, 288.0, 14.0
3/2/2011, 251.6, 12.9
```

Figure 3a, Mileage Data

In the example of Figures 3a and 3b, the `importdata` function obtains two objects from the read operation: a 7 by 2 numeric 2D array and an 8 by 3 cell array of the text strings. These objects are combined into a structure with two fields: `data` for the numeric array and `textdata` for the text strings, which is returned and stored in the variable `data2`.

```

% read mileage data from text file with headings
data2 = importdata('mileage2.txt',' ');

% extract miles traveled and fuel consumed
miles2 = data2.data(:,1);
fuel2 = data2.data(:,2);

% compute mpg for each fill up and average mpg
mpg2 = miles2./fuel2;
avg_mpg2 = sum(miles2)/sum(fuel2);

```

Figure 3b, MATLAB Program for Importing and Processing Mileage Data

Along with the file name, the `importdata` function takes the text file delimiter as an argument. If the text file delimiter is not specified as one of the `importdata` function parameters, MATLAB tries to determine what the delimiter for the text file is. The number of header lines (lines containing strings) is an optional function argument of the `importdata` function. If the number of header lines is specified, numeric data is read starting at line header lines + 1. For example, numeric data would be read starting at line 2 for the statement `data2 = importdata('mileage2.txt',' ',1);` When the optional header lines function parameter is specified, the structure returned by the `importdata` function contains one to three cell arrays: one for the row headings, one for the column headings, and one containing all the text strings.

For text files with mixed string and numeric data, either the `textscan` function or low level text file i/o operations such as `fscanf`, `fgetl`, `fgets`, or `fread` should be used instead of `importdata`.

Writing Numeric Data to Delimited Text Files

The MATLAB functions `dlmwrite` and `csvwrite` allows one to save numeric data to a delimited text file (`csvwrite` for comma delimited files only). The MATLAB code segment of Figure 4a illustrates how to save the original mileage data plus an additional column of data for the miles per gallon of each fill up. The resulting file contents are shown in Figure 4b.

```

% save mileage and mpg numeric data in delimited text file
dlmwrite('mileageplusmpg.txt',[miles, fuel, mpg], ...
        'delimiter', ',', 'precision', 4, 'newline', 'pc');

```

Figure 4a, MATLAB Code Segment to Save Numeric Data to Delimited Text File

The `dmlwrite` function takes the file name to save the data in, the 2D numeric array of data, and an optional list of user configurable options as its function arguments. More information about the user configurable options for `dmlwrite` can be found in MATLAB's help documentation.

```
315,16.4,19.21
306.8,15.3,20.05
262.2,12,21.85
241.5,11.8,20.47
279.1,14.1,19.79
288,14,20.57
251.6,12.9,19.5
```

Figure 4b, Saved Mileage Plus mpg Data

Loading and Saving Data from Microsoft Excel Files

Microsoft Excel files have the extension `.xls` (versions of Office prior to 2007) or `.xlsx` (Office 2007/2010). Microsoft Excel files are binary files that are readable by most spreadsheet software and many other software packages including MATLAB. MATLAB's Microsoft Excel file functions work best when the computer also has the Microsoft Excel software installed on it. For systems that do not have Microsoft Excel software, the read and write functions operate in BASIC mode and are limited in their operation. See MATLAB's help for the full details of the BASIC mode limitations.

To read data from an Excel file, one of the following three formats of `xlsread` is typically used:

```
numbers = xlsread('grocerydatafile.xls');
[numbers, text] = xlsread('grocerydatafile.xls');
[numbers, text, raw] = xlsread('grocerydatafile.xls');
```

The first argument of the `xlsread` function is the filename to read the data from. For the first format, only numeric data is read and an array of real numbers is returned. For the second format, both numeric and text (string) data are read and an array of real numbers and a cell array of strings are returned. For the third format, both numeric and text (string) data are read and an array of real numbers a cell array of strings, and a cell array of the raw data are returned. The `xlsread` function also has optional arguments for specifying selected sheets and ranges of data.

MATLAB's `xlsread` function also supports interactive Microsoft Excel file input. The statement

```
data = xlsread(filename, -1);
```

opens the file `filename` in an Microsoft Excel window and allows the user to select the range of data in the worksheet to read in. One can import an entire worksheet or a selected region of a worksheet. This option can be used when the formatting of the data in the file is not known beforehand. Generally automatic importing of data is preferred though.

To save data to a Microsoft Excel file, the following format of `xlswrite` is typically used:

```
xlswrite('newgrocerydatafile.xls', filedata);
```

The first argument of `xlswrite` specifies the file to save the data to and the second argument contains the data to save. The data to save can be stored in an array for numeric data or a cell array for numeric and string data.

Consider loading grocery data from a Microsoft Excel file, populating a structure array of the grocery data, computing the cost of the items and total cost, and saving the results in a new Microsoft Excel file. Figure 5a shows the original grocery data. The MATLAB program of Figure 5b loads the grocery data from the file, extracts the grocery data and populates a structure array, computes the costs of each item and total cost of all items, and saves the results to a new Microsoft Excel file (Figure 5c).

	A	B	C	D
1	item	quantity	cost per item	
2	apple	4	0.50	
3	orange	2	0.75	
4	banana	6	0.25	
5	mango	1	1.00	
6				

Figure 5a, Excel File of Grocery Data

Since text data along with numeric data is being saved, the data can all be stored in a cell array or the numeric and text data can be written to the file separately. In this example, the numeric and text data are written separately. This requires the range where the data will be written to be specified. The text is stored in a cell array and the numeric data is stored in an array of the same sizes as the ranges they will be written to. The column headers in the new file are a combination of the original column headers obtained from the cell array of text data read in from the original file and an additional column heading for the total cost of each item. The numeric data in the new file consists of the original numeric data plus a column of data for the total cost of each item.

The downside to specifying exact ranges for the write operation is that if the size of the data changes, each range needs to be redone. If possible, specifying the upper left corner of where the data should be written to is better (just specifying the upper left corner will not work for reading data though). To write the data using one file write operation, all the data (numeric and text string) would need to be in a cell array of the appropriate size. Figure 5d shows a code segment that could be used to do this.

Data can also be read from ranges in Microsoft Excel files. For example, the statement

```
itemQuantities =
xlswrite('grocerydatafile.xlsx', 'Sheet1', 'B2:B5');
```

will read in only the item quantities from Sheet1, column V, rows 2 through 5.

```

% read grocery data from Microsoft Excel file
[numbers, text] = xlsread('grocerydatafile.xlsx');

% extract item names and column headings from text data
itemName = text(2:end,1);
columnHeadings = text(1,:);
% extract quantity and cost per item data from number data
itemQuantities = numbers(1:end,1);
costPerItem = numbers(1:end,2);

% store data in grocery structure array
numberItems = length(itemNames);
grocery = struct('item', cell(1,numberItems), 'quantity', ...
    zeros(1,numberItems), 'costperitem', zeros(1,numberItems));
for k = 1:1:numberItems
    grocery(k) =
creategrocery(itemNames(k),itemQuantities(k),costPerItem(k));
end

% compute cost of each item and total cost of all items
itemCosts = itemQuantities.*costPerItem;
totalCost = sum(itemCosts);

% save data to excel file
% column and row headings
xlswrite('newgrocerydatafile.xlsx',columnHeadings, 'Sheet1', 'A1:C1');
xlswrite('newgrocerydatafile.xlsx',itemName, 'Sheet1', 'A2:A5');
% original data
xlswrite('newgrocerydatafile.xlsx',itemQuantities, 'Sheet1', 'B2:B5');
xlswrite('newgrocerydatafile.xlsx',costPerItem, 'Sheet1', 'C2:C5');
% new data
xlswrite('newgrocerydatafile.xlsx',{'item costs'}, 'Sheet1', 'D1');
xlswrite('newgrocerydatafile.xlsx',itemCosts, 'Sheet1', 'D2:D5');
xlswrite('newgrocerydatafile.xlsx',{'Total cost'}, 'Sheet1', 'A8');
xlswrite('newgrocerydatafile.xlsx', totalCost, 'Sheet1', 'B8');

```

Figure 5b, MATLAB Program for Processing Grocery Data

The new grocery data could have been appended to the original grocery data file instead of saving it to a new file. Figure 5e shows a MATLAB code segment to do this. Like with text files, one must be careful when appending data to Microsoft Excel files: one must ensure that the original data is not written over (destroyed) and that if the file will be used for input later that the new file format is taken into account when loading the data.

	A	B	C	D	E
1	item	quantity	cost per it	item costs	
2	apple	4	0.5	2	
3	orange	2	0.75	1.5	
4	banana	6	0.25	1.5	
5	mango	1	1	1	
6					
7					
8	Total cost			6	

Figure 5c, Excel File of Saved Grocery Data

```

% save data to excel file
dataToWrite = cell(8,4);
% row and column headings
for k = 1:1:length(columnHeadings)
    dataToWrite(1, k) = columnHeadings(k);
end
dataToWrite(1,length(columnHeadings) + 1) = {'item costs'};
for k = 1:1:length(itemNames)
    dataToWrite(k+1, 1) = itemNames(k);
end
% numeric data
for k = 1:1:length(itemNames)
    dataToWrite(k+1, 2) = {itemQuantities(k)};
    dataToWrite(k+1, 3) = {costPerItem(k)};
    dataToWrite(k+1, 4) = {itemCosts(k)};
end
% total cost
dataToWrite(length(itemNames)+ 3, 1) = {'Total cost'};
dataToWrite(length(itemNames)+ 3, 2) = {totalCost};
% write data to file
xlswrite('newgrocerydatafile.xlsx', dataToWrite, 'Sheet1', 'A1');

```

Figure 5d, MATLAB Code Segment to Create Cell Array of Grocery Data for File Write

```

xlswrite('grocerydatafile.xlsx',{'Total Cost', totalCost},
'Sheet1','A8:B8');
xlswrite('grocerydatafile.xlsx',{'item costs'}, 'Sheet1','D1');
xlswrite('grocerydatafile.xlsx', itemCosts, 'Sheet1','D2:D5');

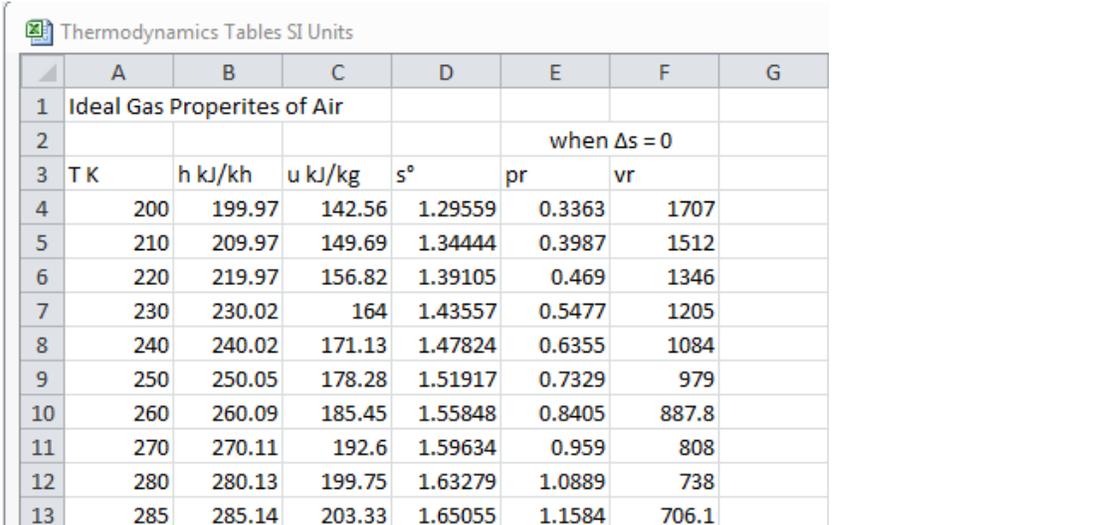
```

Figure 5e, Appending Data to Original File

File Input for Creating the Structure Array used in the Properties of Ideal Gas Table Loop Up

In the Structures primer, a function, `idealGasTableAirLookup`, was developed that looked up the ideal gas properties of air. The function arguments are a structure array containing the ideal gas table data, a field name, and a value. The structure array was assumed to already exist but typically it would need to be created from data read from a file.

The file format for the thermodynamics table for the ideal gas properties of air is shown in Figure 6a. Figure 6b shows the MATLAB program used to read the data from the Microsoft Excel file and create the structure array containing the data from the ideal gas properties table.



	A	B	C	D	E	F	G
1	Ideal Gas Properties of Air						
2					when $\Delta s = 0$		
3	T K	h kJ/kg	u kJ/kg	s°	pr	vr	
4	200	199.97	142.56	1.29559	0.3363	1707	
5	210	209.97	149.69	1.34444	0.3987	1512	
6	220	219.97	156.82	1.39105	0.469	1346	
7	230	230.02	164	1.43557	0.5477	1205	
8	240	240.02	171.13	1.47824	0.6355	1084	
9	250	250.05	178.28	1.51917	0.7329	979	
10	260	260.09	185.45	1.55848	0.8405	887.8	
11	270	270.11	192.6	1.59634	0.959	808	
12	280	280.13	199.75	1.63279	1.0889	738	
13	285	285.14	203.33	1.65055	1.1584	706.1	

Figure 6a, Portion of Thermodynamics Table for Ideal Gas Properties of Air (SI Units)

The data for this particular table is in sheet A22 of the file “Thermodynamics Tables SI Units.xlsx”. The file contains many other Thermodynamics tables. The table data starts at row 4, ends at row 125, and is spread over six columns, A through F. The first three rows include the table title and the table variables and units.

The file data is read in to two variables `data` and `headings`. The `data` variable holds a 2D array of the numeric table and the `headings` variable holds a 2D cell array of the string data from the first three rows. The numeric data is extracted from the 2D numeric array and the structure array is created using the data read from the file. The resulting structure array is saved in a MATLAB workspace file (.mat). This is more convenient than the Microsoft Excel file if the data is to be regularly used from MATLAB.

```

% import ideal gas table for air from Microsoft Excel file
[data, headings] = xlsread('Thermodynamics Tables SI
Units.xlsx', 'A22');
% Column 1 is T in degrees K
Ttemp = data(:,1)';
% Column 2 is h in kJ/kg
htemp = data(:,2)';
% Column 3 is u in kJ/kg
utemp = data(:,3)';
% Column 4 is s
stemp = data(:,4)';
% Column 5 is pr
prtemp = data(:,5)';
% Column 6 is vr
vrtemp = data(:,6)';
clear data;
clear headings;

% allocate space for ideal gas table for air table
numberEntries = length(Ttemp);
idealGasTableAir = struct('T', cell(1,numberEntries), ...
    'h', cell(1,numberEntries), 'u', cell(1,numberEntries), ...
    's', cell(1,numberEntries), 'pr', cell(1,numberEntries), ...
    'vr', cell(1,numberEntries));
% populate the ideal gas table for air structure array
for k = 1:1:numberEntries
    idealGasTableAir(k).T = Ttemp(k);
    idealGasTableAir(k).h = htemp(k);
    idealGasTableAir(k).u = utemp(k);
    idealGasTableAir(k).s = stemp(k);
    idealGasTableAir(k).pr = prtemp(k);
    idealGasTableAir(k).vr = vrtemp(k);
end

% save ideal gas data structure array to MATLAB workspace file
save('idealGasTableAir.mat', 'idealGasTableAir');

```

Figure 6b, MATLAB Program to Read Ideal Gas Table Data from a Microsoft Excel File

The data from each column could also be read in individually as shown in Figure 6c. In this case though, it is important to know the exact ranges of the data to read and if the file is altered, the ranges may need to be updated.

```

% import ideal gas table for air from Microsoft Excel file
% Column 1 is T in degrees K
Ttemp = xlsread('Thermodynamics Tables SI Units.xlsx','A22',
'A4:A125');
% Column 2 is h in kJ/kg
htemp = xlsread('Thermodynamics Tables SI Units.xlsx','A22',
'B4:B125');
% Column 3 is u in kJ/kg
utemp = xlsread('Thermodynamics Tables SI Units.xlsx','A22',
'C4:C125');
% Column 4 is s
stemp = xlsread('Thermodynamics Tables SI Units.xlsx','A22',
'D4:D125');
% Column 5 is pr
prtemp = xlsread('Thermodynamics Tables SI Units.xlsx','A22',
'E4:E125');
% Column 6 is vr
vrtemp = xlsread('Thermodynamics Tables SI Units.xlsx','A22',
'F4:F125');

```

Figure 6c, MATLAB Program to Read Ideal Gas Table Data from a Microsoft Excel File

Last modified Thursday, November 13, 2014



This work by Thomas Murphy is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License](https://creativecommons.org/licenses/by-nc-nd/3.0/).