Chapter 4. Mixed-Type Data
“Containers” and Input/Output Management
• 4.1 Introduction
• 4.2 Data Types
• 4.3 Cell Arrays
• 4.4 Structure Arrays
• 4.5 Tables
• 4.6 Types of MATLAB Files
• 4.7 Recording MATLAB Sessions and Storing Workspace Variables
• 4.8 Importing and Exporting Data
• 4.9 Writing and Displaying Formatted Data
• 4.10 Interactive Input and Output
• 4.11 Datetime, duration, and calendar duration Arrays
• 4.12 Timetables
• 4.13 categorical Arrays
Different toolboxes might have/use additional data classes. For example, the Symbolic Math Toolbox utilizes the `sym` data type, Statistics and Machine Learning Toolbox utilizes the `dataset` arrays (introduced in R2007a and somewhat similar to the `table` class), etc.
## Numeric Data Classes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td><code>w=2.5*10^300, h=2-i</code></td>
<td>The default numeric type in MATLAB</td>
</tr>
<tr>
<td></td>
<td><code>w = 2.5000e+300</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>h = 2.0000 - 1.0000i</code></td>
<td></td>
</tr>
<tr>
<td>single</td>
<td><code>single(3*10^300)</code></td>
<td>Requires less storage space than double but has less precision and a smaller range</td>
</tr>
<tr>
<td></td>
<td><code>ans = Inf</code></td>
<td></td>
</tr>
<tr>
<td>int8, int16, int32, int64</td>
<td><code>a=int8(500)</code></td>
<td>Enables more efficient use of memory for signed and unsigned integers</td>
</tr>
<tr>
<td></td>
<td><code>a = 127</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>b=uint8(500)</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>b = 255</code></td>
<td></td>
</tr>
</tbody>
</table>
Non-Numeric Data Classes to Store Heterogeneous Data

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
</table>
| **cell**  | `d{1,1}=12; d{1,2}='Red';`  
`d{1,3}=magic(4)`  
`d = {[12]} {'Red'} {4x4 double}` | Is used to store mixed-type data in a manner similar (in terms of indexing) to numeric or logical arrays |
| **structure** | `u.day=12; u.color='Red';`  
`u.mat=magic(3)`  
`u = day: 12`  
`color: 'Red'`  
`mat: [3x3 double]` | Is used to store mixed-type data in a C-like structures composed of the named fields |
| **table**  | `T=table([1934;1923],...`  
`{'R';'A'},'VariableNames',...`  
`{'Year' 'State'},...`  
`'RowNames',{'Yuri';'Alan'})`  
`T = Year State`  
`____ ____`  
`Yuri  34 {'R'}`  
`Alan  23 {'A'} | Is used to store mixed-type data in a rectangular column-oriented container |

Statistics and Machine Learning Toolbox has another data type for storing heterogeneous data - *dataset* arrays, introduced in R2007a. It is expected, however, to be replaced with the *table* class in the future releases.
### Other Non-Numeric Data Classes

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical</td>
<td><code>&gt;&gt; f=rand(2)&gt;0.5</code></td>
<td>Is used to represent the logical values of 1 or 0 to represent true and false, respectively.</td>
</tr>
<tr>
<td></td>
<td><code>f = 1 1</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>0 1</code></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td><code>&gt;&gt; c='Summer'</code></td>
<td>Is used to store a sequence of characters</td>
</tr>
<tr>
<td></td>
<td><code>c = Summer</code></td>
<td></td>
</tr>
<tr>
<td>string</td>
<td><code>&gt;&gt; s=['WA' 'OR' 'CA']</code></td>
<td>Is used to store sequences of characters of the different length</td>
</tr>
<tr>
<td></td>
<td><code>s = 'WA' 'OR' 'CA'</code></td>
<td></td>
</tr>
<tr>
<td>function handle</td>
<td><code>&gt;&gt; p=@sin</code></td>
<td>A pointer to a function (handle) that can be saved or passed as an input argument to another function</td>
</tr>
<tr>
<td></td>
<td><code>p = @sin</code></td>
<td></td>
</tr>
</tbody>
</table>
### Type Conversion Functions

#### Valid Combinations of Unlike Classes

<table>
<thead>
<tr>
<th>Type</th>
<th>numeric</th>
<th>math expression</th>
<th>character</th>
<th>string</th>
<th>symbolic expression</th>
<th>cell</th>
<th>structure</th>
<th>table</th>
<th>function handle</th>
<th>inline function</th>
</tr>
</thead>
<tbody>
<tr>
<td>numeric</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>math expression</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>character</td>
<td></td>
<td></td>
<td>eval, str2num, str2double</td>
<td>eval, str2num</td>
<td>string</td>
<td>str2sym</td>
<td>str2func</td>
<td>inline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string</td>
<td></td>
<td></td>
<td>double, str2num, str2double</td>
<td>eval, str2num</td>
<td>char, str2mat</td>
<td>str2sym</td>
<td>str2func</td>
<td>inline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>symbolic expression</td>
<td>double, vpa</td>
<td>char</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>matlabFunction</td>
</tr>
<tr>
<td>cell</td>
<td></td>
<td></td>
<td>cell2mat</td>
<td>cell2str</td>
<td>cell2sym</td>
<td>cell2struct</td>
<td>cell2table</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>structure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>function handle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inline function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Creating Cell Arrays

\[
c{1,1} = "2\text{-by-2}"; \\
c{1,2} = 'eigenvalues of eye(2)'; \\
c{2,1} = \text{eye}(2); \\
c{2,2} = \text{eig}(\text{eye}(2)); \\
\]

\[
\begin{array}{c|c}
\text{Cell 1,1} & \text{Cell 1,2} \\
\hline
"2\text{-by-2}" & 'eigenvalues of eye(2)'
\end{array}
\]

\[
\begin{array}{c|c}
\text{Cell 2,1} & \text{Cell 2,2} \\
\hline
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix} & \begin{bmatrix}
1 \\
1
\end{bmatrix}
\end{array}
\]

\[
b = \text{eye}(3); \\
A(1,1) = \{[2,4,5;3,1,-1;6,0,4]\}; \\
A(1,2) = \{b\}; \\
A(2,1) = 'MATLAB is cool!'; \\
A(2,2) = [1,2,3];
\]

\[
\begin{array}{c|c}
\text{Cell 1,1} & \text{Cell 1,2} \\
\hline
\begin{bmatrix}2 & 4 & 5 \\3 & 1 & -1 \\6 & 0 & 4\end{bmatrix} & \begin{bmatrix}1 & 0 & 0 \\0 & 1 & 0 \\0 & 0 & 1\end{bmatrix}
\end{array}
\]

\[
\begin{array}{c|c}
\text{Cell 2,1} & \text{Cell 2,2} \\
\hline
'MATLAB is cool!' & \begin{bmatrix}1 & 2 & 3\end{bmatrix}
\end{array}
\]
Displaying Cell Array Content

**Cell array c**

```matlab
>> c

2x2 cell array

{["2-by-2"]} {'eigenvalues of eye(2)'}
{[2x2 double]} {2x1 double}

>> celldisp(c)

c(1,1) =

2-by-2

1  0
0  1

c(2,1) =

1  0
0  1

c(1,2) =

eigenvalues of eye(2)

c(2,2) =

1
1
```

**Cell array A**

```matlab
>> A

2x2 cell array

{[3x3 double]} {3x3 double}
{MATLAB is cool!} {1x3 double}

>> celldisp(A)

A(1,1) =

2  4  5
3  1  6
6  0  4

A(2,1) =

MATLAB is cool!

A(1,2) =

1  0  0
0  1  0
0  0  1

A(2,2) =

1  2  3
```
Visualization of Cell Arrays

Cell array c

```
Cell 1,1
'2-by-2'
```
```
Cell 2,1
\[ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \]
```
```
Cell 1,2
'eigenvalues of eye(2)'
```
```
Cell 2,2
\[ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \]
```

Cell array A

```
Cell 1,1
'MATLAB is cool!'
```
```
Cell 2,1
\[ \begin{bmatrix} 2 & 4 & 5 \\ 3 & 1 & -1 \\ 6 & 0 & 4 \end{bmatrix} \]
```
```
Cell 1,2
\[ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \]
```
```
Cell 2,2
\[ \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \]
```
Cell Array Indexing

Cell indexing

```matlab
c{1,1} = "2-by-2";
c{1,2} = 'eigenvalues of eye(2)';
c{2,1} = eye(2);
c{2,2} = eig(eye(2));
```

```matlab
c(2,1) returns
ans =
   2x2 double
   2  1
   1  2
```

```matlab
A{2,1}='MATLAB is cool!';
A{2,2}=[1,2,3];

A(2,1) returns
ans =
   cell
   'MATLAB is cool!'
```

```matlab
A{2,1}(4:6) returns
ans =
   cell
   'LAB'
```

Cell content indexing

```matlab
>> c{2,1} returns
ans =
   1     0
   0     1
double
```

```matlab
>> A{2,2}(2) returns
ans =
   2
double
```

```matlab
>> A{2,1}(4:6) returns
ans =
   char
   'LAB'
```
Suppose you have a cell array containing some numerical data

```matlab
Gen{1,1,1}='Ramjet';       % type of engine
Gen{2,1,1}=[0.3 1; 0.8 6]; % thrust to weight ratio, CL_max, M_max, nz_max
Gen{1,2,1}=[12000 900];    % ceiling (m), range (km)
Gen{2,2,1}=[0.1; 4000];    % price (million dollars), number manufactured
Gen(1,1,2)={'Turbojet'};
Gen(2,1,2)={[0.4 0.6; 0.9 7]};
Gen(1,2,2)={14000 1100};
Gen(2,2,2)={0.2; 34000};
Gen(1,1,3:5)={'Turbojet w/AB' 'Turbojet w/AB' 'LBRTurbofan'};
Gen(2,1,3:5)={[0.6 0.8; 2.2 8] [0.73 1.6; 2 9] [0.6 1.1; 1.6 7]};
Gen(1,2,3:5)={16000 1700}; [17000 2200]; [16000 2000]);
Gen(2,2,3:5)={1.4; 43000}, [30; 1000], [150; 1000]);

>> format bank
>> Data=[Gen{2,1,:}]
>> [a,b]=max([Data(3:4:end)])
>> format
```

```
Data =
Columns 1 through 5
0.30 1.00 0.40 0.60 0.60
0.80 6.00 0.90 7.00 2.20
Columns 6 through 10
0.80 0.73 1.60 0.60 1.10
8.00 2.00 9.00 1.60 7.00
a =
1.60
b =
4.00
```

The trick to process these data is to assign the content of multiple cells to some variable, in this case Data, and then treat this new variable in a usual way.
The Cell Array Functions

- `cell`: creates a cell array of empty matrices
- `cellstr`: converts a character array to a cell array of strings
- `num2cell`: converts a numeric array into a cell array
- `mat2cell`: breaks a matrix up into a cell array of matrices
- `struct2cell`: converts a structure array into a cell array
- `table2cell`: converts a table into a cell array
- `celldisp`: displays the contents of a cell array
- `cellplot`: displays a graphical depiction of cell array
- `iscell`: returns true if an argument happens to be a cell array
- `iscellstr`: returns true if an argument happens to be a cell array of strings
- `deal`: deals the inputs a cell array to outputs
- `cell2mat`: converts the contents of a cell array into a single matrix
- `cell2struct`: converts cell array into a structure array
- `cell2table`: converts cell array into a table
- `cellfun`: applies a function to each cell in a cell array

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Creating Structures

student = struct('name', 'Indiana Jones', 'ssn', '123-45-6789', 'email', 'ijones@arkraiders.com', 'tests', [100,95,70]);
student(2) = struct('name', 'Luke Skywalker', 'ssn', '987-65-4321', 'email', [ ], 'tests', [80,85,100]);

student.name = 'Indiana Jones'
student.ssn = '123-45-6789'
student.email = 'ijones@arkraiders.com'
student.tests = [100,95,70];
student(2).name = 'Luke Skywalker'
student(2).ssn = '987-65-4321'
student(2).tests = [80,85,100];
Accessing Structure Content

>> getfield(student,'name')
ans =
    'Indiana Jones'

>> student(1).email
ans =
    'ijones@arkraiders.com'

>> student(1).email(8:10)
ans =
    'ark'

>> student.tests
ans =
    100  95  70
ans =
    80   85  100

>> [student.tests]
ans =
    100  95  70  80  85  100

>> mean(student(2).tests)
ans =
    88.3333
The Structure Functions

- **struct**: creates a structure array with the specified fields and values
- **cell2struct**: converts a cell array into a structure array
- **table2struct**: converts a table into a structure array
- **fieldnames**: returns the structure field names in a cell array of strings
- **getfield**: returns the structure field contents
- **setfield**: sets the structure field contents
- **rmfield**: removes the specified field(s) from a structure array
- **orderfields**: orders the fields of a structure array
- **isstruct**: returns true for structures
- **isfield**: returns true if the field is in structure array
- **struct2cell**: converts a structure array into a cell array
- **struct2table**: converts a structure array into a table
- **structfun**: applies a function to each field of a scalar structure
Tables (introduced in R2013b)

```matlab
T1 = readtable('TestsData.xls'); % creating a table from the Excel file
```

<table>
<thead>
<tr>
<th>Var1</th>
<th>Test1</th>
<th>Test2</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Student 2</td>
<td>4</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>Student 3</td>
<td>3</td>
<td>4</td>
<td>3.5</td>
</tr>
<tr>
<td>Student 4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Student 5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Student 6</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Student 7</td>
<td>3</td>
<td>4</td>
<td>3.5</td>
</tr>
</tbody>
</table>

A table (exploiting concepts of object-oriented programming) is a MATLAB container for storing heterogeneous column-oriented variables that have the same number of rows. The table entries can be addressed using row and column names or numbers.
Table Properties

T1.Properties

ans =

TableProperties with properties:

Description: ''
UserData: []
DimensionNames: {'Row' 'Variables'}
VariableNames: {'Var1' 'Test1' 'Test2' 'Average'}
VariableDescriptions: {{'' 'Original column heading: 'Test 1''
                          'Original column heading: 'Test 2'' ''}}
VariableUnits: {}
VariableContinuity: []
RowNames: {}
CustomProperties: No custom properties are set.
Use addprop and rmprop to modify CustomProperties.
Manipulating Table Data

Student = T1{4:10,1};  \textcircled{1} \text{Cell-array-like indexing into a table}
Test1 = T1{4:10,2};
Test2 = T1.Test1(4:10);
Av = num2str(T1{4:10,4});

T2 = table(Test1,Test2,Av,'RowNames',Student)

T2 =

<table>
<thead>
<tr>
<th></th>
<th>Test1</th>
<th>Test2</th>
<th>Av</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 4</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Student 5</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Student 6</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Student 7</td>
<td>3</td>
<td>3</td>
<td>3.5</td>
</tr>
<tr>
<td>Student 8</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Student 9</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Student 10</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

\text{Ans} = 3

\textcircled{2} \text{Structure-like indexing}
\textcircled{3} \text{Indexing into a table by the row name}
The Table Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table</code></td>
<td>creates a table from workspace variables</td>
</tr>
<tr>
<td><code>array2table</code></td>
<td>converts a homogeneous array to a table</td>
</tr>
<tr>
<td><code>cell2table</code></td>
<td>converts a cell array to a table</td>
</tr>
<tr>
<td><code>struct2table</code></td>
<td>converts a structure array to a table</td>
</tr>
<tr>
<td><code>table2array</code></td>
<td>converts a table to a homogeneous array</td>
</tr>
<tr>
<td><code>table2cell</code></td>
<td>converts a table to a cell array</td>
</tr>
<tr>
<td><code>table2struct</code></td>
<td>converts a table to a structure array</td>
</tr>
<tr>
<td><code>readtable</code></td>
<td>creates a table from a file</td>
</tr>
<tr>
<td><code>writetable</code></td>
<td>writes a table to a file</td>
</tr>
<tr>
<td><code>istable</code></td>
<td>determines whether an input is a table</td>
</tr>
<tr>
<td><code>height</code></td>
<td>returns the number of rows</td>
</tr>
<tr>
<td><code>width</code></td>
<td>returns the number of table variables</td>
</tr>
<tr>
<td><code>summary</code></td>
<td>prints a summary of a table</td>
</tr>
<tr>
<td><code>head</code></td>
<td>gets the first 8 (by default) rows of a table</td>
</tr>
<tr>
<td><code>tail</code></td>
<td>gets the last 8 (by default) rows of a table</td>
</tr>
<tr>
<td><code>addvars</code></td>
<td>adds new variables (columns)</td>
</tr>
<tr>
<td><code>movevars</code></td>
<td>changes the columns order</td>
</tr>
<tr>
<td><code>rows2vars</code></td>
<td>reorients a table so that rows become variables</td>
</tr>
<tr>
<td><code>intersect</code></td>
<td>sets an intersection of two arrays</td>
</tr>
<tr>
<td><code>ismember</code></td>
<td>checks whether array elements are members of a set array</td>
</tr>
<tr>
<td><code>setdiff</code></td>
<td>sets a difference of two arrays</td>
</tr>
<tr>
<td><code>setxor</code></td>
<td>sets an exclusive OR of two arrays</td>
</tr>
<tr>
<td><code>sortrows</code></td>
<td>sorts array rows</td>
</tr>
<tr>
<td><code>unique</code></td>
<td>finds the unique values in an array</td>
</tr>
<tr>
<td><code>union</code></td>
<td>sets a union of two arrays</td>
</tr>
<tr>
<td><code>join</code></td>
<td>merges two tables by matching up rows using key variables</td>
</tr>
<tr>
<td><code>innerjoin</code></td>
<td>inner join between two tables</td>
</tr>
<tr>
<td><code>outerjoin</code></td>
<td>outer join between two tables</td>
</tr>
<tr>
<td><code>stack</code></td>
<td>stacks data from multiple variables into a single variable</td>
</tr>
<tr>
<td><code>unstack</code></td>
<td>unstacks data from single variable into multiple variables</td>
</tr>
<tr>
<td><code>ismissing</code></td>
<td>finds table elements with the missing values</td>
</tr>
<tr>
<td><code>fillmissing</code></td>
<td>fills missing values</td>
</tr>
<tr>
<td><code>standardizeMissing</code></td>
<td>inserts missing value indicators into a table</td>
</tr>
<tr>
<td><code>varfun</code></td>
<td>applies a function to table variables</td>
</tr>
<tr>
<td><code>rowfun</code></td>
<td>applies a function to table rows</td>
</tr>
</tbody>
</table>

Applicable to the general arrays as well (see Ch. 3)
Displaying / Writing Data

`disp('test')`

`fprintf(fid,formSp,a,b,...)`

Displaying text or array
Displaying / writing formatted data (`fid` is optional)

`formSp` is character vector or string scalar that consists of formatting operators (format specifiers). It can also include ordinary text and special characters. Each format specifier has the following pattern:

```
%[-][number1.number2]Y
```

- `%` starts the conversion specification
- `[-]` specifies the alignment code (`-` left-justifies, `+` always prints a sign character, `0` pads with zeros rather than spaces)
- `number1` specifies the field width
- `number2` specifies the number of digits to the right of the decimal point (precision)
- `Y` specifies the notation of the output (format code)

```
%+12.5e
```

- `%` marker (required)
- `-` flag (optional)
- `12` field width (optional)
- `.5` precision (optional)
- `e` conversion character (required)

```
str=sprintf(formSp,a,b,...)
```

Formatting data into a character or string vector

```
str=compose(formSp,a,b,...)
```

Converting data into a cell array of character vectors or string array (introduced in R2016b)

All rights reserved. No part of this publication may be reprinted or stored in a database or retrieval system, without the prior permission of the publisher. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Conversion and Special Characters

### Conversion characters

- `%d` and `%i`: signed decimal format
- `%u`: unsigned decimal format
- `%o`: unsigned octal format
- `%x`: hexadecimal format (using lowercase letters a-f)
- `%X`: hexadecimal format (using uppercase letters A-F)
- `%f`: fixed-point decimal format
- `%e`: exponential (scientific) format with lowercase e (as in `3.1415e+00`)
- `%E`: exponential format with uppercase E (as in `3.1415E+00`)
- `%g` or `%f`: whichever is shorter (insignificant zeros do not print)
- `%G`: same as `%g`, but using an uppercase E
- `%c`: single character
- `%s`: character vector or string array (the output type is defined by the `formSp` type)

### Nonprinted and special characters

- `\n`: start a new line
- `\r`: carriage return
- `\t`: horizontal tab
- `\f`: form feed
- `\\`: backslash
- `'`: apostrophe (single quotation mark)
- `%`: percent character
- `\xN`: character with numeric hexadecimal Unicode number N

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Displaying Formatted Data

```matlab
>> fprintf('pi        pi^2 \n')
  pi        pi^2
>> fprintf('%15.4e  %8.5f \n',[pi pi^2])
  3.1416e+00  9.86960
>> fprintf('pi = %g;  pi^2 = %5.0f \n',pi,pi^2)
  pi = 3.141593e+00;  pi^2 =    10
>> fprintf("pi = %g;  pi^2 = %g \n",pi,pi^2)
  pi = 3.142;  pi^2 = +9.8696
>> x=[70:15:110]'; y=sind(x);
>> fprintf('beta = %3d deg; \t sin(beta) = %f \n',[x,y])
  beta =  70  deg;  sin(beta) = 0.939693
  beta =  85  deg;  sin(beta) = 0.996195
  beta = 100 deg;  sin(beta) = 0.984808
>> fprintf('1st number is %3u | 2nd number is %3o\n',1:3:15)
  1st number is   1 | 2nd number is   4
  1st number is   7 | 2nd number is  12
  1st number is  13 | 2nd number is >>
```
Displaying Formatted Data

```matlab
>> s = sprintf(['2*pi = ' num2str(2*pi)])
s =
    '2*pi = 6.2832'
>> disp(['2*pi = ' num2str(2*pi, 3)])
    ''2*pi = ''    ''6.28''
>> s = compose('pi/3=%7.4f; sqrt(pi)=%08.5g', pi/3, sqrt(pi))
s =
    ''pi/3= 1.0472; sqrt(pi)=001.7725''
>> b = compose('pi/5=%7.4f; ''sqrt(pi)\==%0.5g', pi/5, sqrt(pi))
b =
    1×1 cell array
    {'pi/5= 0.6283; 'sqrt(pi)\=1.7725'}

>> warning('This is a warning message')
Warning: This is a warning message
>> error('This is an error message')
This is an error message
```
Writing Data to a Text File

Saving to file

```matlab
>> x = 0:.1:1;
>> y = [x; exp(x)];
>> fid = fopen('expon.txt','w');
>> fprintf(fid,'%6.2f %12.8f
',y);
>> fclose(fid)
```

Note that while the first two commands create the 2-by-11 vector `y`, the `fprintf` function saves its transpose, i.e. a 11-by-2 vector, meaning that the format specification is applied to all elements of array `y` in column order!

```
0.00    1.00000000
0.10    1.10517092
...    ...
1.00    2.71828183
```

Reading back

```matlab
>> fid = fopen('exp.txt');
>> a = fscanf(fid,'%g %g',[2 inf])
>> a = a';
>> fclose(fid)
```

Reads file data into an array, `a`, with dimensions, defined by this vector

```
A=sscanf(str,formSp) Reads formatted data from a character or string vector
```

A =

```
0.00    1.00000000
0.10    1.10517092
...    ...
1.00    2.71828183
```
Recognizing Mathworks Files

Since R2012b the default file format for the Simulink models is **SLX** (with the default file extension .slx)

Define two polynomials, a and b, by their roots.
Major MATLAB/Simulink Files

- **m files**: ASCII files containing code in the MATLAB language. The two kinds of M-files are scripts, which do not accept input arguments or return output arguments (they usually operate on data in the workspace), and (user-defined) functions, which can accept input arguments and return output arguments;
- **mlx files**: Live Script files that use Open Packaging Conventions technology (extension of the zip file format). They contain both code and formatted content (stored in an XML document) and output using the Office Open XML (ECMA-376) format;
- **mlapp files**: App Designer files containing description of GUI component properties and callback code;
- **mat files**: Double-precision binary MATLAB format files created by the `save` command and readable by the `load` command. They can be created on one machine and later read by MATLAB on another machine with a different floating-point format, retaining as much accuracy and range as the disparate formats allow. They can also be manipulated by other programs, external to MATLAB (the Application Program Interface Libraries contain C- and Fortran-callable routines to read and write MAT-files from external programs);
- **fig files**: containing data for saved figures;
- **slx files**: containing Simulink models (`mdl` in older versions);
- **mex files**: platform-dependent compiled libraries from C, C++, or Fortran source code executed in MATLAB.
### Opening Files

Double-click or type “open filename”

<table>
<thead>
<tr>
<th>File Type</th>
<th>Extension</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure file</td>
<td>fig</td>
<td>Opens a figure name.fig in a figure window</td>
</tr>
<tr>
<td>HTML file</td>
<td>html</td>
<td>Opens an HTML file name.html in the MATLAB Web browser</td>
</tr>
<tr>
<td>M-file</td>
<td>m</td>
<td>Opens an M-file name.m in the Editor/Debugger</td>
</tr>
<tr>
<td>MLX file</td>
<td>mlx</td>
<td>Opens an MLX-file name.mlx in the Editor/Debugger as a live script</td>
</tr>
<tr>
<td>MAT file</td>
<td>mat</td>
<td>Opens an MAT-file name.mat in the Import Wizard</td>
</tr>
<tr>
<td>MLAPP file</td>
<td>mlapp</td>
<td>Opens an application file name.mlapp in the App Designer</td>
</tr>
<tr>
<td>Simulink model</td>
<td>slx (mdl)</td>
<td>Opens a model name.slx (name.mdl) in the Simulink development environment</td>
</tr>
<tr>
<td>Project file</td>
<td>prj</td>
<td>Opens a project file name.prj in the MATLAB Compiler Deployment Tool</td>
</tr>
<tr>
<td>Other</td>
<td>custom</td>
<td>Opens name.custom by calling the helper function opencustom, where</td>
</tr>
<tr>
<td></td>
<td></td>
<td>opencustom is a user-defined function</td>
</tr>
<tr>
<td>Variable</td>
<td>none</td>
<td>Opens a selected array in the workspace in the Variables Editor; (using</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the openvar function)</td>
</tr>
</tbody>
</table>
The save Function

```matlab
>> hmatrix=magic(3); kmatrix=rand(6,2); kk1=linspace(1,3);
>> save trial
>> whos('-file','trial')
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>hmatrix</td>
<td>3x3</td>
<td>72</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>kk1</td>
<td>1x100</td>
<td>800</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>kmatrix</td>
<td>6x2</td>
<td>96</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

**Function Syntax**
- `save` saves all workspace variables in the default file called `matlab.mat` (saved variables are in the binary format).
- `save filename` saves all workspace variables in a binary file labeled `filename.mat`.
- `save filename var1 var2` saves only two variables, `var1` and `var2`.
- `save filename -ascii` saves variables in ASCII readable format (8 digits), without `.mat` extension.
- `save filename -double` saves variables in ASCII double precision format (16 digits).
- `save filename -tabs` saves variables in tab delimited format.
- `save(filename,v3,'-append')` appends a variable `v3` to the existing `filename` mat-file (note the use of the function syntax as opposed to command syntax).

**Command Syntax**
- `filename` is either a character vector or string scalar.

Specifies to explore a mat file rather than a global workspace.
The **load** Function

- **load**: retrieves variables from the file `matlab.mat` and loads them into the workspace.
- **load filename**: retrieves variables from the `filename.mat` file and loads them into the workspace.
- **load filename.ext**: loads the ASCII file `filename.ext` into the matrix `filename` (the ASCII file must be space-delimited).
- **load filename var1 var2**: retrieves only two specified variables, `var1` and `var2`, from the `filename.mat` file.
- **load –ascii filename**: retrieves variables from the space-delimited ASCII file.
- **load(filename,'-regexp','^h')**: retrieves variables with the names staring from `h`.
- **S=load(filename)**: returns the contents of `filename` (defined using a character vector or string scalar) in variable `S`. If `filename` is a `mat`-file, `S` is a structure containing the fields matching variables retrieved. If `filename` is an ASCII file, `S` is a double precision array.

```matlab
>> load trial -regexp 'k'
>> who
Name      Size      Bytes  Class Attributes
kk1      1x100     800 double
kmatrix  6x2       96   double
```

A regular expression is a string of characters that defines a certain pattern.
The **diary** Function

**diary** toggles the switch to record the session on and off

**diary on** turns the diary switch on

**diary off** turns the diary switch off

**diary filename** or **diary('filename')** (command syntax or function syntax) records the session in the file with the specified name (if the filename is not specified MATLAB creates a file named **diary** in the current directory)

**get(0,'Diary')** checks the status of the diary switch (on/off)

**get(0,'DiaryFile')** returns the name of the diary file
## Importing and Exporting Digital and Textual Data

<table>
<thead>
<tr>
<th>File Format</th>
<th>File Content</th>
<th>Extension</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB formatted</td>
<td>Saved MATLAB workspace</td>
<td>.mat</td>
<td>load, save</td>
</tr>
<tr>
<td>Text/data</td>
<td>Text</td>
<td>any</td>
<td>textread, textscan</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fscanf, sscanf</td>
</tr>
<tr>
<td></td>
<td>Delimited numbers/text</td>
<td>any</td>
<td>dlmread, dlmwrite</td>
</tr>
<tr>
<td></td>
<td>Comma-separated numbers/text</td>
<td>.csv</td>
<td>csvread, csvwrite</td>
</tr>
<tr>
<td>Spreadsheet Data</td>
<td>Excel worksheet</td>
<td>.xls, .xlsx</td>
<td>xlsread, xlswrite</td>
</tr>
<tr>
<td></td>
<td>Lotus 123 worksheet</td>
<td>.wk1</td>
<td>wklread, wklwrite</td>
</tr>
<tr>
<td>Webpage Data</td>
<td>HTML content</td>
<td>URL</td>
<td>webread, webwrite</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(urlread, urlwrite)</td>
</tr>
<tr>
<td>Extensible Markup</td>
<td>XML-formatted text</td>
<td>.xml</td>
<td>xmlread, xmlwrite</td>
</tr>
<tr>
<td>Language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scientific data</td>
<td>Data in Common Data Format</td>
<td>.cdf</td>
<td>cdfread, cdfwrite</td>
</tr>
<tr>
<td></td>
<td>Flexible Image Transport System data</td>
<td>.fits</td>
<td>fitsread</td>
</tr>
<tr>
<td></td>
<td>Data in Hierarchical Data Format</td>
<td>.hdf</td>
<td>hdfread</td>
</tr>
</tbody>
</table>
The **xlsread** and **xlswrite** Functions

- **xlsread**
  - `num = xlsread(filename)`
  - `num = xlsread(filename, -1)`
  - `num = xlsread(filename, sheet)`
  - `num = xlsread(filename, 'range')`
  - `[num, txt]= xlsread(filename, ...)`
  - `[num, txt, raw] = xlsread(filename, ...)`

- **xlswrite**
  - `xlswrite(filename, A)`
  - `xlswrite(filename, A, sheet)`
  - `xlswrite(filename, A, xlRange)`
  - `xlswrite(filename, A, sheet, xlRange)`

- Interactive reading
  - E.g., 'C1:D8'

- Additionally returns text fields (headers) in a cell array `txt`

- Returns unprocessed data (numbers and text) in cell array `raw`

- `filename` can be specified as a character vector or string scalar

- Example:
  ```matlab```
  ```
  filename = 'Test';
  t=1:10; d=0.5*9.81*t.^2;
  xlswrite(filename,[t', d'],'Data','A2')
  xlswrite(filename,{'Time','Depth'},2,'A1')
  ```

**Use '' (the empty string) as a placeholder (for 'sheet')**
The `dlmread` and `dlmwrite` Functions

```
M = dlmread('filename')
M = dlmread('filename', delimiter)
M = dlmread('filename', delimiter, R, C)
M = dlmread('filename', delimiter, range)
```

```
dlmwrite('filename', M)
dlmwrite('filename', M, delimiter)
dlmwrite('filename', M, delimiter, R, C)
dlmwrite('filename', M, delimiter, range)
```

`R, C` define the upper-left corner of the data
(starts from 0,0, not from 1,1)

`range` defines `[R1 C1 R2 C2]`
The `textread` Function

\[
[A, B, C, \ldots] = \text{textread}('filename', 'formSp', N)
\]

```matlab
>> [name, rank, GPA, age, degree] = textread('mydata.dat', '%s %s %f %d %s', 1)
```

- name = 'John'
- rank = 'LT'
- GPA = 3.34
- age = 25
- degree = 'BS'

- `name`, `rank`, GPA, age, degree
- `textread` function
- `mydata.dat` file
- `format specification`
- `number of lines`

`textread` Function

\[
[A, ~, B, \ldots] = \text{textread}(\ldots, \text{param}, \text{value}, \ldots)
\]

- `Delimiter` - one or more characters (act as delimiters between elements)
- `Endofline` - single character or `\r\n` (character that denotes the end of a line)
- `Headerlines` - positive integer (ignores the specified number of lines at the beginning of the file)
The `textscan` Function

Reads data from an open text file `fileID` into cell array, `C`.

Reads data from a text string `'str'` into cell array, `C`.

`C = textscan(fileID, 'formSp')`

`C = textscan(fileID, 'formSp', N)`

`C = textscan('str', 'formSp')`

`C = textscan('str', 'formSp', N)`

`C = textscan(..., param, value)`

```matlab
>> s='John  LT 3.34 25 BS';
>> C=textscan(s,'%s %s %f %d %s')
C =
    {1x1 cell}    {1x1 cell}    [3.3400]    [25]    {1x1 cell}
>> C=textscan(s,'%s %s')
C =
    {3x1 cell}    {2x1 cell}
>> File=fopen('mydata.dat','r')
>> C=textscan(File,'%3c %s')
File =
    3
C =
    [3x3 char]    {3x1 cell}
```
## Format Specifiers

<table>
<thead>
<tr>
<th>Format Code</th>
<th>Action</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literals (ordinary characters)</td>
<td>Ignore the matching characters. For example, in a file that has MAE followed by a number (for course number), to skip the MAE and read only the number, use 'MAE' in the formSp string</td>
<td>None</td>
</tr>
<tr>
<td>%d</td>
<td>Read a signed integer value</td>
<td>Double array</td>
</tr>
<tr>
<td>%u</td>
<td>Read an unsigned integer value</td>
<td>Double array</td>
</tr>
<tr>
<td>%f</td>
<td>Read a floating-point value</td>
<td>Double array</td>
</tr>
<tr>
<td>%s</td>
<td>Read a white-space or delimiter-separated string</td>
<td>Cell array of strings</td>
</tr>
<tr>
<td>%q</td>
<td>Read a string, which could be in double quotes</td>
<td>Cell array of strings (doesn’t include the double quotes)</td>
</tr>
<tr>
<td>%c</td>
<td>Read characters, including white space</td>
<td>Character array</td>
</tr>
<tr>
<td>%[ ]</td>
<td>Read the longest string containing characters specified in the brackets</td>
<td>Cell array of strings</td>
</tr>
<tr>
<td>%[^ ]</td>
<td>Read the longest nonempty string containing characters that are not specified in the brackets</td>
<td>Cell array of strings</td>
</tr>
<tr>
<td>%* (instead of %)</td>
<td>Ignore the matching characters specified by *</td>
<td>No output</td>
</tr>
<tr>
<td>%w... (instead of %)</td>
<td>Read field width specified by w. The %f format supports %w.pf, where w is the field width and p is the precision</td>
<td>None</td>
</tr>
</tbody>
</table>
D = dlmread('Data.log', '', 10, 1);

D =

1.0000  -250.0000  -0.8700  0
2.0000  -250.0000  -0.8667  0
3.0000  -250.0000  -0.8633  0

>> [a, b, c, d]=textread('Data.log','c %f %f %f %f',-1,'headerlines',10);

>> D=[a,b,c,d];

Data.log

D =

1.0000  -250.0000  -0.8700  0
2.0000  -250.0000  -0.8667  0
3.0000  -250.0000  -0.8633  0

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
v ATLAS
#SSF 2.0 generic
######################################################################
# Copyright © 1984–2008 #
# Silvaco Data Systems, Inc. All rights reserved #
# KEY=SVC86hn7562asdx PROD=345 #
######################################################################
j  4  1  2  94  95
k  2  2  0  342109921  0  670000017

>> fID = fopen('Data.log');
>> C=textscan(fID,...
>> '%*s %*s %*s %*s %*s %s %s %f %f : %f : %f %f',1,'headerlines',1);
>> flose(fID);

>> [a,b,c,d,e,f,g]=textread('Data.log',... 
>> '%*s %*s %*s %*s %*s %s %s %f %f : %f : %f %f',1,'headerlines',1);
>> C=cell([a,b,c,d,e,f,g])];

>> cellplot(C);
Practical Examples

```matlab
37 2
2008-12-16 00Z, DZID00 i, j = (15.9, 15.6), lat, long = (54.832, 83.104)
SFALT, SFPRES
381 968.3
LINE AGL(m) T(C) RH(%) WSPD(m/s) WDD P(mb)
1 2.0 11.76 50.93 9.33 182.93 968.04
2 15.2 11.64 51.16 10.97 182.91 966.53
3 56.2 11.26 52.30 13.47 182.37 961.80
4 111.2 10.73 54.07 14.80 181.77 955.47
5 173.0 10.15 56.25 15.57 180.91 948.40
6 242.0 9.53 58.50 16.02 179.44 940.56
7 319.0 8.85 60.97 16.48 178.28 931.86

>> F = fopen('Winds.txt');
>> aux1 = textscan(F, '%s', 2);
>> date = textscan(F, '%s', 1, 'delimiter', '_');
>> aux2 = textscan(F, '%s', 4, 'delimiter', '\n');
>> data = textscan(F, '%d %f %f %f %f %f %f', 5);
>> fclose(F);
```

```matlab
>> subplot(2,2,1), cellplot(aux1)
>> subplot(2,2,2), cellplot(date)
>> subplot(2,2,3), cellplot(aux2)
>> subplot(2,2,4), cellplot(data)
```
<HTML>
<TITLE>University of Wyoming - Radiosonde Data</TITLE>
<LINK REL="StyleSheet" HREF="/resources/select.css" TYPE="text/css">
<BODY BGCOLOR="white">
<H2>72493 OAK Oakland Int Obs</H2>
<PRES>
<table>
<thead>
<tr>
<th>PRES</th>
<th>HGHT</th>
<th>TEMP</th>
<th>DWPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1013.0</td>
<td>3</td>
<td>14.2</td>
<td>12.2</td>
</tr>
<tr>
<td>1012.0</td>
<td>11</td>
<td>14.2</td>
<td>10.6</td>
</tr>
<tr>
<td>1004.0</td>
<td>77</td>
<td>13.6</td>
<td>11.9</td>
</tr>
<tr>
<td>1000.0</td>
<td>110</td>
<td>13.2</td>
<td>11.6</td>
</tr>
</tbody>
</table>

... 8.4 32553 -44.5 -80.5 1013.0 3 14.2 12.2 1012.0 11 14.2 10.6 1004.0 77 13.6 11.9 1000.0 110 13.2 11.6

</PRES><H3>Station information and sounding indices</H3>

Station identifier: OAK  Station number: 72493  Observation time: 180901/1200

fid = fopen('AtmData.txt');
while ~feof(fid) % finding a horizontal dashed line
    tline=fgetl(fid);
    if numel(tline)==0 && tline(1)=='-';
        disp(tline)
        break
    end
    for i=1:3; % reading three more lines
        tline=fgetl(fid);
        end
    i=1;
    while ~feof(fid) % reading numeric data in
        tline=fgetl(fid);
        if numel(tline)==0 && tline(1:6)="</PRE">
        A(i,:)=sscanf(tline,'%f %f %f %f %f %f %f %f %f %f')
        i=i+1;
    else
        break
    end
    fclose(fid)
## read... Family of Functions

Introduced in R2019a

<table>
<thead>
<tr>
<th>Function</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>A=readmatrix(filename,... [opt,name,value])</code></td>
<td>Reads homogeneous numeric or text data from filename into a matrix A. The file format is determined from the file extension. Optional import options can be specified in opts object and by one or more name-value pair arguments.</td>
</tr>
<tr>
<td><code>C=readcell(filename,... [opt,name,value])</code></td>
<td>Reads mixed numeric and text data from filename into a cell array C.</td>
</tr>
<tr>
<td><code>T=readtable(filename,... [opt,name,value])</code></td>
<td>Reads column-oriented data from filename into a table T (introduced earlier, in R2013b).</td>
</tr>
<tr>
<td><code>TT=readtimetable(filename,... [opt,name,value])</code></td>
<td>Reads column-oriented data from a file into a timetable TT.</td>
</tr>
<tr>
<td><code>[v1,...,vn]=readvars(filename,... [opt,name,value])</code></td>
<td>Reads column-oriented data from a file into variables v1,...,vn.</td>
</tr>
</tbody>
</table>

The reciprocal writing functions are `writematrix`, `writecell`, `writetable`, and `writetimetable`.

As far as reading tabular data, the `detectImportOptions` function, introduced in R2016b, may help locating a table in a file and returns the import options for importing the table:

```matlab```
opts=detectImportOptions(filename,[Name,Value])
```

Alternatively, the import options object `opts` for the tables and timetables can be created manually, using `DelimitedTextImportOptions`, `FixedWidthImportOptions`, and `SpreadsheetImportOptions`. The correctness of defining tabular data can be checked using the `preview` function:

```matlab```
T=preview(filename,[opts])
```

returning a table containing the first eight rows of data in `filename`, using the import options `opts`.

You might want to try using `readmatrix` instead of `dlmread`, `csvread`, and `xlsread` (in the case you intend to read homogeneous numeric or text data), and `readcell` instead of `xlsread` in the case you are reading mixed numeric and text data.
**Low-Level File I/O Functions**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fclose</code></td>
<td>Closes one or all open files</td>
</tr>
<tr>
<td><code>feof</code></td>
<td>Tests for the end of file</td>
</tr>
<tr>
<td><code>ferror</code></td>
<td>Provides file I/O error information</td>
</tr>
<tr>
<td><code>fgetl</code></td>
<td>Reads a line from file, removing newline characters</td>
</tr>
<tr>
<td><code>fgets</code></td>
<td>Reads a line from file, keeping newline characters</td>
</tr>
<tr>
<td><code>fileread</code></td>
<td>Reads contents of file as text</td>
</tr>
<tr>
<td><code>fopen</code></td>
<td>Opens a file, or obtain information about open files</td>
</tr>
<tr>
<td><code>fprintf</code></td>
<td>Writes data to a text file</td>
</tr>
<tr>
<td><code>fread</code></td>
<td>Reads data from a binary file</td>
</tr>
<tr>
<td><code>frewind</code></td>
<td>Moves a file position indicator to beginning of an open file</td>
</tr>
<tr>
<td><code>fscanf</code></td>
<td>Reads data from a text file</td>
</tr>
<tr>
<td><code>fseek</code></td>
<td>Moves to a specified position in file</td>
</tr>
<tr>
<td><code>ftell</code></td>
<td>Returns a current position</td>
</tr>
<tr>
<td><code>fwrite</code></td>
<td>Writes data to a binary file</td>
</tr>
</tbody>
</table>

- `fgetl(fileID)` reads the next line of the specified file `fileID`, including the newline characters
- `fgets(fileID)` reads the next line of the specified file `fileID`, including the newline characters
- `fgets(fileID, nchar)` returns up to `nchar` characters of the next line
- `frewind(fileID)` sets the file position indicator to the beginning of an open file
- `ftell(fileID)` returns the current location of the position pointer in the specified file `fileID`
- `fseek(fileID, offset, origin)` sets the file position indicator `offset` bytes from origin (beginning of file `bof`, current position in file, `cof`, or end of file `eof`) in the specified file `fileID`
- `feof(fid)` returns the status of the end-of-file indicator

*All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.*
Import Wizard - Reading ASCII Data

![Notepad and MATLAB Import Wizard](image)

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Import Wizard (before R2012b) — Reading Excel Spreadsheet

### Microsoft Excel - Test_Scores

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Student 1</td>
<td>Test 1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Student 2</td>
<td>Test 2</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>Student 3</td>
<td>Test 3</td>
<td>3</td>
<td>4</td>
<td>3.5</td>
</tr>
<tr>
<td>4</td>
<td>Student 4</td>
<td></td>
<td>4</td>
<td>5</td>
<td>4.5</td>
</tr>
<tr>
<td>5</td>
<td>Student 5</td>
<td></td>
<td>4</td>
<td>5</td>
<td>4.5</td>
</tr>
<tr>
<td>6</td>
<td>Student 6</td>
<td></td>
<td>4</td>
<td>5</td>
<td>4.5</td>
</tr>
<tr>
<td>7</td>
<td>Student 7</td>
<td></td>
<td>4</td>
<td>5</td>
<td>4.5</td>
</tr>
<tr>
<td>8</td>
<td>Student 8</td>
<td></td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>Student 9</td>
<td></td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>Student 10</td>
<td></td>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

#### Import Wizard

Select variables to import using checkboxes:
- Create variables matching preview.
- Create vectors from each column using column names.
- Create vectors from each row using row names.

Variables in D:\AA & ME Courses\AE2440-2007\M-Scripts\Topic 4 - Data ImportExport\Test_Scores.xls

<table>
<thead>
<tr>
<th>Import</th>
<th>Name</th>
<th>Size</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>data</td>
<td>10x3</td>
<td>double</td>
</tr>
<tr>
<td></td>
<td>textdata</td>
<td>11x4</td>
<td>cell</td>
</tr>
</tbody>
</table>

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Import Wizard – Reading Excel Spreadsheet

Yellow color is used to highlight a portion of data that cannot be imported as the selected data type.
sta_num='72493';
Y='2018'; M='09'; D='1';

%% Create URL address
url=['http://weather.uwyo.edu/cgi-bin/sounding?region=naconf&TYPE=TEXT', ...
    '%3ALIST&YEAR=',Y,'&MONTH=',M,'&FROM=',D,'12&TO=0212&STNM=',sta_num];

%% Read source (HTML) code
SourceFile=webread(url);
dlmwrite('AtmData.txt',SourceFile,'delimiter','
');

%% Find the beginning and end of table data
Pattern='-----'; K=strfind(SourceFile,Pattern);
[ind]=find(diff(K)>1); start=K(ind(2))+length(Pattern);
pos=strfind(SourceFile,'</PRE>'); ending=pos(1)-1;

%% Read data in
C=textscan(SourceFile(start:ending),'%f %f %f %f %f %f %f %f %f %f %f');

%% Plot data
plot(C{:,3},C{:,2}/1000,'.-.'), grid, xlabel('T, ^oC'), ylabel('H, km')

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
## Dealing with Media Data

<table>
<thead>
<tr>
<th>Type of Media File</th>
<th>Supported Formats</th>
<th>Extension</th>
<th>MATLAB Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphics</strong></td>
<td>BMP (Windows Bitmap)</td>
<td>.bmp</td>
<td>imread</td>
</tr>
<tr>
<td></td>
<td>CUR (Cursor File)</td>
<td>.cur</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIF (Graphics Interchange Format)</td>
<td>.gif</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HDF4 (Hierarchical Data Format)</td>
<td>.hdf4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ICO (Icon File)</td>
<td>.ico</td>
<td>imwrite</td>
</tr>
<tr>
<td></td>
<td>JPEG (Joint Photographic Experts Group) and JPEG 2000</td>
<td>.jpeg, .jp2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PBM (Portable Bitmap)</td>
<td>.pbm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PCX (Windows Paintbrush)</td>
<td>.pcx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PGM (Portable Graymap)</td>
<td>.pgm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PNG (Portable Network Graphics)</td>
<td>.png</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PPM (Portable Pixmap)</td>
<td>.ppm</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RAS (Sun Raster)</td>
<td>.ras</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TIFF (Tagged Image File Format)</td>
<td>.tiff</td>
<td></td>
</tr>
<tr>
<td></td>
<td>XWD (X Window Dump)</td>
<td>.xwd</td>
<td></td>
</tr>
<tr>
<td><strong>Video</strong></td>
<td>Motion JPEG and Motion JPEG 2000 encoded video</td>
<td>.avi, .mj2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MPEG-1</td>
<td>.mpg</td>
<td>VideoReader</td>
</tr>
<tr>
<td></td>
<td>MPEG-4, including H.264 encoded video</td>
<td>.wmv, .asf, .asx</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Windows Media video</td>
<td>.mp4, .m4v</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Apple QuickTime Movie video</td>
<td>.mov</td>
<td></td>
</tr>
<tr>
<td><strong>Audio</strong></td>
<td>WAVE, OGG, FLAC, AU, MP3, MPEG-4, AAC files</td>
<td>.wav, .ogg, .flac, .au, .mp3, .m4a, .mp4</td>
<td></td>
</tr>
</tbody>
</table>

*All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.*
A = imread('2221989.jpg');

1. figure, image(A)

2. figure, imagesc(A(:,:,3)), colorbar

B = rgb2gray(A);

3. figure, imshow(B)

4. figure, imshow(255-B)

---

Red, Green and Blue layers

Values spread from 0 to 255
Importing Video

Vobj = VideoReader('MVI_2674.mov');

% Read and show 1 frame at a time

while hasFrame(Vobj)
    video = readFrame(Vobj);
    image(video);
    pause(1/Vobj.FrameRate);
end

Vobj = VideoReader('MVI_2674.mov');

nFrames = fix(Vobj.Duration*Vobj.FrameRate);

% Read and show 1 frame at a time

for k = 1 : nFrames
    currentframe = read(Vobj,k);
    imshow(currentframe)
end

Vobj = VideoReader with properties:
General Properties:
  Name: 'MVI_2652.MOV'
  Path: 'C:\Users\Oleg\Desktop\TestVideo'
  Duration: 3.2115
  CurrentTime: 0
  Tag: ''
  UserData: []

Video Properties:
  Width: 1920
  Height: 1080
  FrameRate: 23.9760
  BitsPerPixel: 24
  VideoFormat: 'RGB24'
Dealing with the Audio Files

### Reading audio file

- ```File='Magic_Fly.mp3';```
- ```audioinfo(File)```
- ```[y,Fs]=audioread(File);``` 
- ```sound(y,Fs)``` 
- ```plobj=audioplayer(y,Fs);``` 
- ```play(plobj)``` 

Creating the audioplayer object
(allowing more flexibility during playback, including the ability to pause(plobj), resume(plobj))

### Recording audio file via microphone

- ```recObj = audiorecorder(10000,16,2)``` 
- ```disp('Start speaking now')``` 
- ```recordblocking(recObj,5);``` 
- ```disp('End of recording')``` 
- ```play(recObj);```
Using the `input` function

```matlab
d1 = input('Input your birthday DD-MM-YYYY: ','s');
d2 = input('Input the current date DD-MM-YYYY: ','s');
D = datenum(d2, 'dd-mm-yyyy') - datenum(d1, 'dd-mm-yyyy');
fprintf('As of today you lived %8g days\n',D)
```

Using the `menu` function

```matlab
k = 0;
while k ~= 3
    h = randi(2);  % 1 or 2
    k = menu('Heads or Tails?','Heads','Tails','Quit');
    if k == h,
        disp('You won')
    elseif k == 3,
        disp('Thanks for trying')
    else
        disp('You loose')
    end
end
```
### Predefined Dialog Boxes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialog</td>
<td>creates a dialog box</td>
<td>questdlg</td>
<td>creates a question dialog box</td>
</tr>
<tr>
<td>errordlg</td>
<td>creates an error dialog box</td>
<td>uigetdir</td>
<td>displays a dialog box to retrieve name of directory</td>
</tr>
<tr>
<td>exportsetupdlg</td>
<td>displays the figure export settings dialog box</td>
<td>uigetfile</td>
<td>displays a dialog box to retrieve name of file for reading</td>
</tr>
<tr>
<td>export2wsdlg</td>
<td>creates a dialog for exporting variables to workspace</td>
<td>uigetpref</td>
<td>conditionally opens a dialog box according to user preference</td>
</tr>
<tr>
<td>helpdlg</td>
<td>displays a help dialog box</td>
<td>uiopen</td>
<td>opens a dialog box for selecting a file to load into workspace / saving variables to MAT-file</td>
</tr>
<tr>
<td>inputdlg</td>
<td>creates an input dialog box</td>
<td>uisave</td>
<td></td>
</tr>
<tr>
<td>listdlg</td>
<td>creates a list selection dialog box</td>
<td>uisetcolor</td>
<td>opens a color selection dialog box</td>
</tr>
<tr>
<td>msgbox</td>
<td>creates a message dialog box</td>
<td>uisefont</td>
<td>opens a font selection dialog box</td>
</tr>
<tr>
<td>printpreview</td>
<td>displays a dialog box showing the figure it will print</td>
<td>waitbar</td>
<td>displays a wait bar</td>
</tr>
<tr>
<td>printdlq</td>
<td>displays a print dialog box</td>
<td>warndlg</td>
<td>creates a warning dialog box</td>
</tr>
</tbody>
</table>

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
Examples of Menus

```matlab
k = questdlg('You want to continue?', ... 'Premature Exit','Yes',... 'No','Yes');

h = waitbar(0,'Please wait...');
for i = 1:100
    % computations appear here
    waitbar(i/100)
    pause(0.2)
end
close(h)

warndlg('Pressing OK will clear memory','!! Warning!!')
```
Examples of Menus

```matlab
d = dir
str = {d.name}
[s, v] = listdlg('PromptString', 'Select a file:', ...
    'SelectionMode', 'single', 'ListString', str)
str(s)
```

```matlab
d = struct
5x1 struct array with fields:
    name
    date
    bytes
    isdir
    datenum

str = cell
    '.', '..', 'Plot_KTM_TSPI.m' 'TSPI Analysis of...' 'UAV8Cam3.mat'

s = 5
v = 1
ans = char
    'UAV8Cam3.mat'
```

button pushed (1 – OK, 0 - Cancel)
Examples of Menus

```matlab
prompt = {'Enter the first number','Enter the second number'};
dlg_title = 'Enter two numbers';
num_lines = 1;
def = {'10','10'};
answer = inputdlg(prompt,dlg_title,num_lines,def,'on');
a = answer{1};
b = answer{2};
str2num(a) + str2num(b);
```

![Menu Example](image)
>> guide

Allows you to create your own GUIs

See Appendix C
App Designer

>> appdesigner

Differ from GUIDE in the code structure and callback syntax
Prior to 2014b date and time data were stored in one of the three formats:

- **Date string**, a character vector, e.g. 'Thursday, December 31, 2020 11:45:44.123 PM'
- **Date vector**, a 1-by-6 numeric vector containing the year, month, day, hour, minute, and second, e.g. [2020 12 31 23 45 44.123]
- **Serial date number**, a single number equal to the number of days since January 0, 0000 in the proleptic Gregorian calendar (calendar produced backward to dates preceding its official introduction in 1582), e.g. 7.3816e+05

<table>
<thead>
<tr>
<th>Date string</th>
<th>Date vector</th>
<th>Serial date number</th>
<th>Datetime scalar</th>
</tr>
</thead>
<tbody>
<tr>
<td>datevec(DateStr)</td>
<td>datenumber(DateStr)</td>
<td>datetime(DateStr)</td>
<td></td>
</tr>
<tr>
<td>datetotr(DateVec)</td>
<td>datenumber(DateVec)</td>
<td>datetime(DateVec)</td>
<td></td>
</tr>
<tr>
<td>datetotr(DateNum)</td>
<td>datevec(DateNum)</td>
<td>datenumber(DateNum)</td>
<td></td>
</tr>
<tr>
<td>datetime scalar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(introduced in R2014b)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char(DateTime)</td>
<td>datevec(DateTime)</td>
<td>datenumber(DateTime)</td>
<td></td>
</tr>
<tr>
<td>cellstr(DateTime)</td>
<td>datenumber(DateTime)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>string(DateTime)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#datetime (datestr(DateNum)) or datetime(datevec(DateNum)) or datetime(DateNum,'ConvertFrom','datenumber')

```matlab
>> t = datetime(2022,6,22,0:23,0,0);
>> disp([t(1),t(end)])
22-Jun-2022 00:00:00   22-Jun-2022 23:00:00

>> Tb = linspace(t1,t2,60); disp(Tb(1:2))
22-Jun-2022 00:00:00   22-Jun-2022 12:12:12

>> Tc = t1:caldays(3):t2; disp(Tc(1:2))
22-Jun-2022 00:00:00   25-Jun-2022 00:00:00

>> Td = Tc(1):hours(4):Tc(2); disp(Td(1:2))
22-Jun-2022 00:00:00   22-Jun-2022 04:00:00

>> d1 = duration(32,45,7)
d1 = 32:45:07

>> d2 = 0:seconds(15):minutes(1); disp(d2)
0 sec 15 sec 30 sec 45 sec 60 sec

>> cd = calendarDuration(3,5,13) cd = 3y 5mo 13d```

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
datetime, duration & calendarDuration

For advanced users

February 3, 10:45pm

March 3, 10:45pm

February 3, 10:45pm

March 5, 10:45pm

>> disp(datetime('2021-2-3 10:45 pm')+duration(24*30,0,0))
05-Mar-2021 22:45:00

>> disp(datetime('2021-2-3 10:45 pm')+calendarDuration(0,1,0))
03-Mar-2021 22:45:00

>> disp(datetime('2021-2-3 10:45 pm')+30)
03-Mar-2021 22:45:00

>> dt=t(end)-t(1)
dt = 23:00:00

>> dtc=caldiff(Tb(1:2))
dtc = 12h 12m 12.203s

>> minutes(dt)
ans = 1380

>> calmonths(cd)
ans = 41

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
For advanced users

### datetime Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>'yyyy-MM-dd'</td>
<td>2014-04-19</td>
<td>Use y, ..., yyyy – for year;</td>
</tr>
<tr>
<td>'dd/MM/yy'</td>
<td>19/04/14</td>
<td>u, ..., uuuu – for ISO year*;</td>
</tr>
<tr>
<td>'dd.M.yyyy'</td>
<td>19.4.2014</td>
<td>M, ..., MMMM – for month;</td>
</tr>
<tr>
<td>'eeee, MMM d, uuuu'</td>
<td>Saturday, Apr 19, 2014</td>
<td>d, dd – for day of month;</td>
</tr>
<tr>
<td>'MMMM d, yyyy h:mm a'</td>
<td>April 19, 2014 9:41 PM</td>
<td>e, ..., eeeee – for day of week;</td>
</tr>
<tr>
<td>'MMMM d, yyyy HH:mm z'</td>
<td>May 19, 2014 21:41 PST</td>
<td>h, hh – for hour (12-hour notation);</td>
</tr>
<tr>
<td>'yyyy-MM-dd'' T ''HH:mm:SS'</td>
<td>1962-04-14 T 13:55:00</td>
<td>H, HH – for hour (24-hour notation);</td>
</tr>
<tr>
<td>'yyyy-MM-dd HH:mm:ss.SSS'</td>
<td>2014-04-19 21:41:06.123</td>
<td>m, mm – for minute;</td>
</tr>
<tr>
<td>'yyyy-MM-dd HH:mm XXX'</td>
<td>2014-04-19 21:41 -08:00</td>
<td>s, ss – for second;</td>
</tr>
<tr>
<td>'''FY'''yy, QQQQ'</td>
<td>FY'19, 2nd quarter</td>
<td>x, X, ..., xxxxx – for time zone offset;</td>
</tr>
<tr>
<td>'yyyy G'</td>
<td>2019 CE</td>
<td>z, Z, ..., ZZZZ – for time zone offset;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q, ..., QQQQ – for quarter;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a is used for a day period, G – Era, W – week of the month</td>
</tr>
</tbody>
</table>

* The International Organization for Standardization (ISO) date format, ISO 8601, that eliminates ambiguity in expressing a numeric calendar date including BC dates.

```matlab
>> t.Format='''FY'''yy, QQQQ'; disp(t(end))
FY'22, 2nd quarter
```

In a similar manner, you can request / change other properties as well. They are Year, Month, Day, Hour, Minute, Second, TimeZone, and SystemTimeZone.
>> t=now+caldays(1:365);
(Instead of caldays(1:365), you could have used calendarDuration(0,0,1:365)). Let us also create the same-size y vector
>> y= linspace(10,40,365)+10*rand(1,365);
>> subplot(311), scatter(t,y), grid, Y=xlim; xtickformat( 'MM-dd-yy' )
>> subplot(312), plot(t,y,'d-.');
>> grid, xlim( [Y(1)+150 Y(2)-200] )
>> subplot(313), stem(t,y)
>> xlim(datetime(2019,[7,8],[15,7])), xtickformat( 'd MMMM' )

<table>
<thead>
<tr>
<th>Function</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>now</td>
<td>Returns the current date and time as a serial date number</td>
</tr>
<tr>
<td>clock</td>
<td>Returns the current date and time as a six-element date vector</td>
</tr>
<tr>
<td>date</td>
<td>Returns the current date as a character vector</td>
</tr>
<tr>
<td>calendar</td>
<td>Returns a 6-by-7 matrix containing a calendar (which runs Sunday through Saturday) for the current month, the month that includes a specific date (specified as a serial date number or character vector), or the specified month M of the specified year Y</td>
</tr>
<tr>
<td>eomday(Y,M)</td>
<td>Returns the last day of the year and month given by corresponding elements of the numeric arrays Y and M</td>
</tr>
<tr>
<td>[N,Name]=weekday(t)</td>
<td>Returns the day of week, e.g. [~,k]=weekday('2023-1-22') returns k='Sun'</td>
</tr>
</tbody>
</table>

All rights reserved. No part of this publication may be reproduced, distributed, or transmitted, unless for course participation, in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Publisher and/or Author. Contact the American Institute of Aeronautics and Astronautics, Professional Development Programs, 12700 Sunrise Valley Drive, Suite 200, Reston, VA 20191-5807.
>> TA = readtable('AirDropData.xlsx');
>> disp(head(TA,4))

<table>
<thead>
<tr>
<th>Date</th>
<th>System</th>
<th>DropAlt_ft_</th>
<th>Weight_lbs_</th>
<th>ExitTime_Zulu_</th>
<th>Miss_m_</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/20/2009</td>
<td>'10K LCADS HALO'</td>
<td>17500</td>
<td>'9900'</td>
<td>'21:52:54'</td>
<td>'447.7'</td>
</tr>
<tr>
<td>10/20/2009</td>
<td>'10k Screamer'</td>
<td>14000</td>
<td>'9000'</td>
<td>'17:25:27'</td>
<td>'84.7'</td>
</tr>
<tr>
<td>10/20/2009</td>
<td>'10k Screamer'</td>
<td>14000</td>
<td>'8200'</td>
<td>'17:25:29'</td>
<td>'48'</td>
</tr>
<tr>
<td>10/20/2009</td>
<td>'2K G-12 HALO'</td>
<td>9999</td>
<td>'2170'</td>
<td>'18:04:15'</td>
<td>'364.7'</td>
</tr>
</tbody>
</table>

>> TA.Weight_lbs_ = str2double(TA.Weight_lbs_);
>> TA.Miss_m_ = str2double(TA.Miss_m_);
>> TT = table2timetable(TA);

>> TT tod = TT.Date + timeofday(datetime(TT.ExitTime_Zulu_));
>> TT tod.Format = ['dd.MM.uuu HH:mm:ss']; disp(TT tod(1:3));
>> TT.Date = TT tod; TT.ExitTime_Zulu_ = []; disp(head(TT,4));

<table>
<thead>
<tr>
<th>Date</th>
<th>System</th>
<th>DropAlt_ft_</th>
<th>Weight_lbs_</th>
<th>Miss_m_</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.10.2009 21:52:54</td>
<td>'10K LCADS HALO'</td>
<td>17500</td>
<td>9900</td>
<td>447.7</td>
</tr>
<tr>
<td>20.10.2009 17:25:27</td>
<td>'10k Screamer'</td>
<td>14000</td>
<td>9000</td>
<td>84.7</td>
</tr>
<tr>
<td>20.10.2009 17:25:29</td>
<td>'10k Screamer'</td>
<td>14000</td>
<td>8200</td>
<td>48</td>
</tr>
<tr>
<td>20.10.2009 18:04:15</td>
<td>'2K G-12 HALO'</td>
<td>9999</td>
<td>2170</td>
<td>364.7</td>
</tr>
</tbody>
</table>

>> TT('20.10.2009 18:04',:)

<table>
<thead>
<tr>
<th>Date</th>
<th>System</th>
<th>DropAlt_ft_</th>
<th>Weight_lbs_</th>
<th>Miss_m_</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.10.2009 18:04:15</td>
<td>'2K G-12 HALO'</td>
<td>9999</td>
<td>2170</td>
<td>364.7</td>
</tr>
</tbody>
</table>
>> TT.Properties
ans =
TimetableProperties with properties:
    Description: ''
    UserData: []
    DimensionNames: {'Date' 'Variables'}
    VariableNames: {'System' 'DropAlt_ft_' 'Weight_lbs_' 'Miss_m_'}
    VariableDescriptions: {1x4 cell}
    VariableUnits: {}
    VariableContinuity: []
    RowTimes: [140x1 datetime]
    StartTime: 20.10.2009 21:52:54
    SampleRate: NaN
    TimeStep: NaN
    CustomProperties: No custom properties are set.
Use addprop and rmprop to modify CustomProperties.

>> L1=stackedplot(TT(:,2:end),'.'); grid

>> max(TT{:,vartype('numeric')},[],'omitnan')
ans = 17500
      31750
         9249

>> L2=parallelplot(TT)
Data Cleaning

For advanced users

**Function Brief Description**

<table>
<thead>
<tr>
<th>Function</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>standardizeMissing</code> (R2013b)</td>
<td>Inserts standard missing values in a numerical array, table, or timetable</td>
</tr>
<tr>
<td><code>movmean</code> (R2016a)</td>
<td>Computes moving averages for a numerical array</td>
</tr>
<tr>
<td><code>fillmissing</code> (R2016b)</td>
<td>Fills missing entries of data with the constant value</td>
</tr>
<tr>
<td><code>isoutlier</code> (R2017a)</td>
<td>Finds outliers in data</td>
</tr>
<tr>
<td><code>smoothdata</code> (R2017a)</td>
<td>Smooths noisy data</td>
</tr>
<tr>
<td><code>filloutliers</code> (R2017a)</td>
<td>Detects and remove outliers in data</td>
</tr>
<tr>
<td><code>ischange</code> (R2017b)</td>
<td>Finds abrupt changes in data</td>
</tr>
<tr>
<td><code>islocalmax</code> / <code>islocalmin</code> (R2017b)</td>
<td>Finds local maxima or minima in data</td>
</tr>
<tr>
<td><code>rmoutliers</code> (R2018b)</td>
<td>Detects and removes outliers from data</td>
</tr>
<tr>
<td><code>detrend</code> (R2018b)</td>
<td>Removes the mean value or linear trend from a vector or matrix (usually for FFT processing)</td>
</tr>
</tbody>
</table>

```
>> TA = readtable('SensorData.dat');
>> TT = table2timetable(TA);
>> TTs = synchronize(TT1, TT2);
>> TTs = synchronize(TT1, TT2, 'intersection');
>> TTs = synchronize(TT1, TT2, 'union', 'linear');
```
categorical (ordinal) Class

>> TT.System=categorical(TT.System);
>> summary(TT.System)
   10K Dragonfly          1
   10K LCADS              2
   10K LCADS HALO         1
   10K Sherpa             1
   10k Screamer           2
   2K G-12 HALO           6

>> catNames={'LA drop' 'MA drop' 'HA drop'};
>> binDataA=discretize(TT.DropAlt_ft_,[0 5000 10000 20000],'categorical',catNames);
>> summary(binDataA)
   LA drop        33
   MA drop        49
   HA drop        68

>> pie(binDataA)

% LA drop (22%)
% HA drop (45%)
% MA drop (33%)

>> isordinal(binDataA)
ans = 1
>> sum(binDataA <= 'MA drop')
ans = 82

% Convert to string array
>> S=string(Projects.Title);
% Join all string elements into a 1x1 string
>> S=join(S);
% Define punctuation characters
>> punctChar=({'(' ')',' ','-','.'});
% Replace them with the space characters
>> S=replace(S,punctChar,' ');
% Split on the space characters
>> S=strsplit(S); % Split the string
% Remove words with fewer than five characters
>> S(strlen(S)<5)=[ ];
% Convert words to a categorical array
>> C=categorical(S);
% Word cloud
>> wc=wordcloud(C);
The End of Chapter 4

Questions?