KU LEUVEN

MATLAB

IO : high level functions

input / output

- File IO
  - high level io
  - more info:
    - `help iofun`

save/load

save/load mat-files

- To export workspace variables to a binary or ASCII file, use the `save` function. (easiest way)
- Save all variables from the workspace in a single operation (default file matlab.mat):
  `save(filename)`
- Save the variables that you specify:
  `save(filename, var1, var2, ... varN)`
- Use of wildcard character (*) in the variable name is allowed
  `save (filename, str*)`
- whos -file examines contents of the MAT-file:
  `whos -file filename`

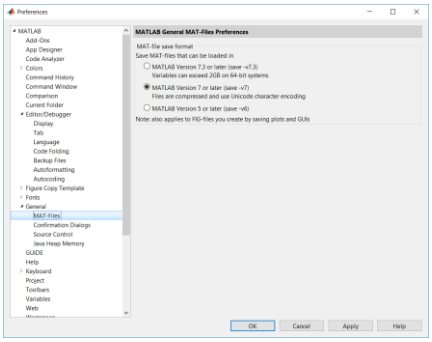
save/load mat-files - append

- add new variables to those already stored in an existing MAT-file with `save(filename, var1, var2, ... varN, '-append')`
  - For each variable that already exists in the MAT-file, MATLAB overwrites its saved value with the new value taken from the workspace.
  - For each variable not found in the MAT-file, MATLAB adds that variable to the file and stores its value from the workspace.

save/load mat-files - compression

- MATLAB compresses the data that you save to a MAT-file.
- can save a significant amount of storage space
- caution! version dependent!, check documentation

- **Preferences** dialog, select **General MAT-Files**



save/load mat-files - ascii

```
save(filename, var1, var2, ... varN, '-ascii')
```

Each variable to be saved must be either a two-dimensional double array or a two-dimensional character array.

- Saving a complex double array causes the imaginary part of the data to be lost
- Each MATLAB character in a character array is converted to a floating-point number equal to its internal ASCII code and written out as a floating-point number string. There is no information in the saved file that indicates whether the value was originally a number or a character.
- Advice: be careful with the –ascii option (check documentation)
- *File: io_load_save_1.m*

save/load mat-files

Using the load Function

- import variables from a binary or ASCII file on your disk to the workspace, use the `load` function. (inverse of save)
- load all variables from the workspace in a single operation (default filename: matlab.mat):
  `load(filename)`
- load specified:
  `load(filename, var1, var2, ..., varN)`
- wildcard character (*) in the variable name to load those variables that match a specific pattern. (This works for MAT-files only.)
  `load(filename, str*)`

save/load mat-files

Loading ASCII Data

- ASCII files must be organized as a rectangular table of numbers, with each number in a row separated by a blank or tab character, and with an equal number of elements in each row.
- In the workspace, MATLAB assigns the array to a variable named after the file being loaded
  `load mydata.dat`
  reads all of the data from mydata.dat into the workspace as a single array mydata

save/load mat-files

Advice:

- if data are to be exchanged between MATLAB and other programs, use the ASCII format.
  If data is to be exchanged within the MATLAB environment, use the MAT-file format
- use .dat extension for ASCII-files, .mat for MAT-files
- MAT-format contains more info, getting lost in the ascii-option

IO legacy functions

File import/export functions (before R2019a)

BEFORE R2019a		Data type	Delimiter
csvread	Read a comma separated value f	Numeric data	Comma
dlmread	Read ASCII delimited file	Numeric data	Any character
csvwrite	Write a comma separated value file		
dlmwrite	Write ASCII delimited file		
xlsread	reads the first worksheet in the Microsoft® Excel® spreadsheet	Mix numeric + text	
xlswrite	writes matrix A to the first worksheet in he Microsoft® Excel® spreadsheet	Mix numeric + text	

csvread / csvwrite

- `csvread / csvwrite` is a subset of dlmread/dlmwrite (separator is ‘,’)
- Syntax:
  - `a = csvread('filename')`
  - `a = csvread('filename',row,col)`
  - `a = csvread('filename',row,col,range)`
- Note
  - csvread does not like to read in text!
  - will work with all numeric
- *File: io_csvread.m*
- *File: io_csvwrite.m*

dlmread / dlmwrite

- `dlmread` function reads formatted ASCII data without using low level routines. (1 line command!)
- `M = dlmread('filename', delimiter, R, C)` reads numeric data from the ASCII-delimited file filename, using the specified delimiter.
  R and C specify the row and column where the upper left corner of the data lies in the file.
- advice: use for numerical data with a specific separator
- data is read into 1 matrix, without separator
- *File: io_dlmread.m*
  File: io_dlmwrite.m

xlsread/xlswrite

- Will read Excel's .xls files directly into Matlab.
- Read in the first sheet in the xls file(the default), or pick the sheet you want to read into Matlab.
- Very handy if you have any data stored in Excel spreadsheets you want to read into Matlab.
- Using `xlsread` saves you from having to export the excel file as an ascii file. The format of the xlsread function is: `xlsread(filename, sheetname)`
- `xlsread(filename, -1)` allows interactive selection of the data
- ex.: *io_xlsread.m*

xlsfinfo

- Use the `xlsfinfo` to determine if a file contains a readable Microsoft Excel spreadsheet.
- Inputs to xlsfinfo are
  - Name of the spreadsheet file
- Outputs from xlsfinfo are
  - String 'Microsoft Excel Spreadsheet' if the file contains an Excel worksheet readable with the xlsread function.
  - Cell array of strings containing the names of each worksheet in the file.

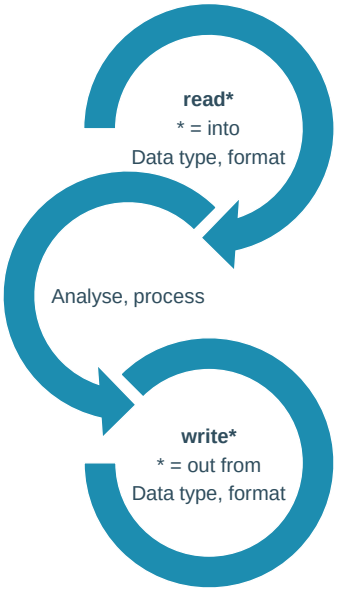
xlswrite

- `xlswrite('filename', M)`
  writes matrix M to the Excel file filename.
- The maximum size of array M depends on the associated Excel version. For more information on Excel specifications and limits, see the Excel help.
- `xlswrite('filename', M, sheet)` writes matrix M to the specified worksheet sheet in the file filename. The sheet argument can be either a positive, double scalar value representing the worksheet index, or a quoted string containing
- ex.: *io_xlswrite.m*

KU LEUVEN

Basic IO (since R2019)

KU LEUVEN



readmatrix
readcell
readtable
…

writematrix
writecell
writetable
…

KU LEUVEN

File import (read…) functions

R2019a	
`A=readmatrix(filename,... [opt,name,value])`	Read 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
`T=readtable(filename),… [opt,name,value])`	Read column-oriented data from filename into a table T
`[v1,…,vn]=readvars(filename),... [opt,name,value])`	Read column-oriented data from a file into variables v1,…,vn
`C=readcell(filename),... [opt,name,value])`	Create a cell array C by reading column-oriented data from a file

KU LEUVEN

readmatrix

- *File: import_using_readmatrix.mlx*
- Creates an array by reading column-oriented data from a file.
  - `a = readmatrix('csvlist_65_empty.dat')`
  - `x = readmatrix('test_alltext.csv')`
- Basic form: imports numerical data only, non numerical data are imported as NaN
- Limited to returning one type of data in the output array as the 'OutputType' named parameter is limited to a scalar string/cell string.
- Performs automatic detection of import parameters for your file. It determines the file format from the file extension:
  - .txt, .dat, or .csv for delimited text files
  - .xls, .xlsb, .xlsm, .xlsx, .xltm, .xltx, or .ods for spreadsheet files

readmatrix(filename,opts)

- Create import options based on file content
  - `opts = detectImportOptions(filename)`
- Preview the data from a file and import numerical data
  - `preview(filename,opts)`
- Changing the options is possible
  - `Opts.Delimiter = {',' ':'}`
  - `Opts.VariableNamesLine = 2`
- Use the opts object to import the data.

readmatrix(filename, Name, Value, Name, Value, …)

- Use Name/value pairs to set the value (check documentation)

Name	Used with
OutputType	Text & spreadsheet
FileType	Text & spreadsheet
Range	Text & spreadsheet
NumHeaderLines	Text & spreadsheet
TreatAsMissing	Text & spreadsheet
ExpectedNumVariables	Text & spreadsheet
Sheet	Spreadsheet
UseExcel	Spreadsheet

Name	Used with
Delimiter	Text
Commentstyle	Text
LineEnding	Text
DateLocale	Text
Encoding	Text
Whitespace	Text
DecimalSeparator	Text
ThousandsSeparator	Text
ConsecutiveDelimitersRule	Text
LeadingDelimitersRule	Text

readtable

- *File: import_using_readtable.mlx*
- `readtable` works the same way as `readmatrix`, the resulting output is stored in a table. Is used to store mixed-type data in a rectangular column-oriented container

readvars

- *File: import_using_readvars.mlx*
- Very similar to `readtable`
  - Specify the output variables
  - Skipping a (column)variable can be done with ~
- Output is a set of column vectors, that can have a different class (data type)

readcell

- *File: import_using_readcell.mlx*
- `readcell` works the same way as `readmatrix`, but the resulting output is stored in a cell array. This allows for importing both numerical and alaphanumerical into a single container. This function allows for the most general import.
- Instead of `NaN`, `missing` is used.
- Works fine for spreadsheets
  - Spreadsheets are easy to import
    - A grid of rows and columns
    - Multiple sheets: consider it as a 3-dimensional array

Writing with write…

- *File: export_using_writecell*
- Write cell array to file
- Check the resulting file! There are some instances where the `writecell` function creates a file that does not represent the input data exactly.

- `writematrix, writetable` work the same way; the elements are written with a default separator (,)

More: reading arbitrary formatted files

- Mixed data: numerical + text: `textscan`
  - https://nl.mathworks.com/matlabcentral/answers/312599-how-do-i-parse-this-complex-text-file-with-textscan

- Low level functions: C-like functions
  - `fscanf`
  - `fgetl`
  - `fread`
  - `fwrite`