Files

Reading and writing files in MatLab and ASCII format.

Matlab format: binary format, file can contain any number of variables of various types.

ASCII format – text file (can be opened with notepad).
The only permissible format for ASCII files: file must contain the same number of elements in each row

save - saves file.
If no arguments are supplied saves file 'matlab.mat' with all current workspace variables in the current directory.
Filename, names of variables and options can be supplied.

save filename
stores all workspace variables in the current directory in filename.mat. To save to another directory, use the full pathname for the filename.

save filename variable1 variable2 ...
saves listed variables with appropriate names in file *filename* with extension *.mat*.
File name might contain folder name that indicates where to store the file.
File name can contain an extension and extension does not have to be '.mat'.

save can be used as a regular function that accepts input arguments. In this case *filename* and names or variables must be in a strings format. Use the * wildcard to save only those variables that match the specified pattern. For example, *save('filename','A*')* saves all variables that start with A.

Options parameter that controls how the information is saved can be supplied. Usage: *save ... option*

Option	Action
-append	The specified existed MAT-file, appended to the end
-ascii	8-digit ASCII format
-ascii -double	16-digit ASCII format
-ascii -tabs	delimits with tabs
-ascii -double -tabs	16-digit ASCII format, tab delimited

Examples: save 'dat0545.prn' zz -ascii -tabs saves file with name dat0545.prn in current folder in ASCII format with elements separated by tab symbols.
save dat x y z saves file dat.mat in Matlab format containing variables x, y and z.
save('D:\DATA\dat.dta','x','y','z') saves file dat.dta in Matlab format containing variables x, y and z in folder D:\DATA.

load - loads file.
If no filename is supplied, loads file 'matlab.mat' from the current directory or from the first place in the path. Filename can be used to specify the name of file to load.

load filename
loads all the variables from filename given a full pathname. If filename has no extension, load looks for file named filename or filename.mat and treats it as a binary MAT-file. If filename has an extension other than .mat, load treats the file as ASCII data.

load filename X Y Z ...
loads just the specified variables from the MAT-file. The wildcard '*' loads variables that match a pattern (MAT-file only).

load -ascii filename or *load -mat filename*
forces load to treat the file as either an ASCII file or a MAT-file, regardless of file extension.
With -ascii, load returns an error if the file is not numeric text.
With -mat, load returns an error if the file is not a MAT-file. *load -ascii filename* returns all the data in the file as a single two dimensional double array with its name taken from the filename (minus any extension).
The number of rows is equal to the number of lines in the file and the number of columns is equal to the number of values on a line. An error occurs if the number of values differs between any two rows.
load filename.ext reads ASCII files that contain rows of space-separated values. The resulting data is placed into a variable with the same name as the file (without the extension). *load* replaces leading underscores or digits in filename with an X and replaces other non-alphabetic character with underscores. ASCII files may contain MATLAB comments (lines that begin with %).

S = load(...)
returns the contents of a MAT-file in the variable S. If the file is a MAT-file, S is a struct containing fields that match the variables in retrieved. When the file contains ASCII data, S is a double-precision array.

Simple examples:

```
» who
Your variables are:
DH   F    F1   NN   Na   T    T1   Tn   Tn1  ans  b    bF   cf   dbF  g    k    kB

» save currentdata
» clear all
» who
» load currentdata
» who
Your variables are:
DH   F    F1   NN   Na   T    T1   Tn   Tn1  ans  b    bF   cf   dbF  g    k    kB

»  v=[T1',F1'];
»  size(v)
ans =
    50     2
»  save('v.dat','v','-tabs','-ascii')
»  !notepad v.dat&
»  clear v
»  load v.dat
»  size(v)
ans =
    50     2
» v1=load('v.dat');
» size(v1)
ans =
    50     2
```

Other standard file formats

Comma separated values (CSV) files

csvread Read a comma-separated value file

Syntax

```
M = csvread('filename')
M = csvread('filename',row,col)
M = csvread('filename',row,col,range)
```

csvwrite Write a comma-separated value file

Syntax

```
csvwrite('filename',M)
csvwrite('filename',M,row,col)
```

see help for more information

Delimeted file

dlmread Read an ASCII delimited file into a matrix

Syntax

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

dlmwrite Write a matrix to an ASCII delimited file

Syntax

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

see help for more information

Text files of other formats

textread Read formatted data from text file

Example: Read the file fft.m into cell array of strings.

```
file = textread('fft.m','%s','delimiter','\n','whitespace','');
```

Spreadsheets

wk1read Read Lotus123 spreadsheet file (.wk1)

wk1write Write a matrix to a Lotus123 WK1 spreadsheet file

xlsread Read Microsoft Excel spreadsheet file (.xls)

xlsfinfo Determine if file contains Microsoft Excel (.xls) spreadsheet

Images

imread - Read image from graphics file.

Syntax:

```
A = imread(filename,fmt)
[X,map] = imread(filename,fmt)
[...] = imread(filename)
```

A = imread(filename,fmt)

reads a greyscale or color image from the file specified by the string filename, where the string fmt specifies the format of the file. If the file is not in the current directory or in a directory in the MATLAB path, specify the full pathname of the location on your system.

Supported image formats (partial list):

JPEG Any baseline JPEG image; JPEG images with some commonly used extensions; 8-bit and 12-bit lossy compressed RGB and grayscale images; 8-bit and 12-bit lossless compressed RGB images; 8-bit, 12-bit, and 16-bit lossless compressed grayscale images

TIFF Any baseline TIFF image, including 1-bit, 8-bit, and 24-bit uncompressed images; 1-bit, 8-bit, and 24-bit images with packbits compression; 1-bit images with CCITT compression; also, 16-bit grayscale, 16-bit indexed, and 48-bit RGB images

GIF Any 1-bit to 8-bit GIF image

BMP 1-bit, 4-bit, 8-bit, 16-bit, 24-bit, and 32-bit uncompressed images; 4-bit and 8-bit run-length encoded (RLE) images

PCX 1-bit, 8-bit, and 24-bit images

ICO 1-bit, 4-bit, and 8-bit uncompressed images

imread returns the image data in the array A. If the file contains a grayscale image, A is a two-dimensional (M-by-N) array. If the file contains a color image, A is a three-dimensional (M-by-N-by-3) array. The class of the returned array depends on the data type used by the file format.

For most file formats, the color image data returned uses the RGB color space. For TIFF files, however, *imread* can return color data that uses the RGB, CIELAB, ICCLAB, or CMYK color spaces. If the color image uses the CMYK color space, A is an M-by-N-by-4 array.

[X,map] = imread(filename,fmt)

reads the indexed image in filename into X and its associated colormap into map. The colormap values are rescaled to the range [0,1].

[...] = imread(filename)

attempts to infer the format of the file from its content.

[...] = imread(URL,...)

reads the image from an Internet URL. The URL must include the protocol type (e.g., http://).

imwrite Write image to graphics file

Syntax

```
imwrite(A,filename,fmt)
imwrite(X,map,filename,fmt)
imwrite(...,filename)
imwrite(...,Param1,Val1,Param2,Val2...)
```

imwrite(A,filename,fmt)

writes the image A to the file specified by filename in the format specified by fmt. A can be an M-by-N (greyscale image) or M-by-N-by-3 (color image) array. A cannot be an empty array. If the format specified is TIFF, imwrite can also accept

an M-by-N-by-4 arrray containing color data that uses the CMYK color space. filename is a string that specifies the name of the output file. fmt can be any of the text strings of supported formats.

imwrite(...,filename)

writes the image to filename, inferring the format to use from the filename's extension.

imwrite(X,map,filename,fmt)

writes the indexed image in X and its associated colormap map to filename in the format specified by fmt. If X is of class uint8 or uint16, imwrite writes the actual values in the array to the file. If X is of class double, the imwrite function offsets the values in the array before writing, using uint8(X-1). The map parameter must be a valid MATLAB colormap. Note that most image file formats do not support colormaps with more than 256 entries.

imwrite(...,Param1,Val1,Param2,Val2...)

specifies parameters that control various characteristics of the output file for HDF, JPEG, PBM, PGM, PNG, PPM, and TIFF files. For example, if you are writing a JPEG file, you can specify the quality of the output image.

See MATLAB help for more information on image file formats.

saveas Save figure using specified format

Syntax

```
saveas(h,'filename.ext')
saveas(h,'filename','format')
```

saveas(h,'filename.ext')

saves the figure or model with the handle h to the file filename.ext.
The format of the file is determined by the extension, ext. Possible values (partial list)

ai	Adobe Illustrator `88
bmp	Windows bitmap
emf	Enhanced metafile
eps	EPS Level 1
fig	MATLAB figure
jpg	JPEG image
pcx	Paintbrush 24-bit
tif	TIFF image, compressed

saveas(h,'filename','format')

saves the figure or model with the handle h to the file called filename using the specified format. The filename can have

an extension but the extension is not used to define the file format. If no extension is specified, the standard extension corresponding to the specified format is automatically appended to the filename.

Image files can be also saved by using *print* command. See help for *print*.

Sound files

wavread Read Microsoft WAVE (.wav) sound file

Syntax

```
y = wavread('filename')
[y,Fs,bits] = wavread('filename')
```

wavwrite Write Microsoft WAVE (.wav) sound file

Syntax

```
wavwrite(y,'filename')
wavwrite(y,Fs,'filename')
wavwrite(y,Fs,N,'filename')
```

Movies

aviread Read an Audio Video Interleaved (AVI) file.

Syntax

```
mov = aviread(filename)
mov = aviread(filename,index)
```

aviinfo Return information about an Audio Video Interleaved (AVI) file

movie2avi Create an Audio Video Interleaved (AVI) movie from MATLAB movie

Syntax

```
movie2avi(mov,filename)
movie2avi(mov,filename,param,value,param,value...)
```

For many other file formats, low-level input-output file operations are necessary to read or write files.

Low-level file I/O functions.

File opening and closing.

fopen — Open file. FID = *fopen*('filename',permission) opens the specified file with the specified permission. Permission is one of the strings:

'r'	read
'w'	write (create if necessary)
'a'	append (create if necessary)
'r+'	read and write (do not create)
'w+'	truncate or create for read and write
'a+'	read and append (create if necessary)
'W'	write without automatic flushing
'A'	append without automatic flushing

By default, files are opened in binary mode. To open a text file, add 't' to the permission string, for example 'rt' and 'wt+'. (On Unix systems, text and binary files are the same so this has no effect. But on PC, Macintosh, and VMS systems this is critical.)

FID = *fopen*('filename') assumes a permission of 'r'.

If the open is successful, FID gets a scalar MATLAB integer, thefile identifier, to be used as the first argument to other FileIO routines. If the open was not successful, -1 is returned for FID.

Three file identifiers are automatically available and need not be opened. They are fid=0 (standard input), fid=1 (standard output), and fid=2 (standard error).

[FID, MESSAGE] = *fopen*('filename',permission) returns a system dependent error message if the open is not successful.

fopen('all') returns a row vector, the file identifiers for all thefiles currently opened by the user. (But not 0, 1, and 2.)

[FILENAME,PERMISSION] = *fopen*(FID) returns the filename and permission associated with the given file identifier.

If the file is opened in 'r' mode and it is not found in the current working directory, *fopen* searches down MATLAB's search path.

fclose — Close file. *fclose*(FID) closes the file with file identifier FID, which is an integer obtained from an earlier *fopen*(). *fclose*('all') closes all open files, except 0, 1 and 2. *fclose*() returns 0 if successful, -1 if not.

Unformatted I/O.

fread Read binary data from file.

[A, COUNT] = *fread*(FID,SIZE,PRECISION) reads binary data from the specified file and writes it into matrix A. If COUNT is used, *fread* returns the number of elements successfully read. FID is an integer file identifier obtained from FOPEN, or 0 for standard input. The size argument is optional; if not specified, the entire file is read; if specified, valid entries are:

N read N elements into a column vector.

inf read to the end of the file.

[M,N] read elements to fill an M-by-N matrix, in column order.

The precision argument controls the number of bits read for each value and the interpretation of those bits as character, integer or floating point values. Any of the following strings, either the MATLAB versions, or their C or Fortran equivalents, may be used. If not specified, the default is 'uchar'. In all cases, the resulting matrix elements are stored as long floating point values, as is all other MATLAB data.

MATLAB	C or Fortran	Description
'char'	'char'	character, 8 bits
'schar'	'signed char'	signed character, 8 bits
'short'	'short'	integer, 16 bits
'int'	'int'	integer, 16 or 32 bits
'long'	'long'	integer, 32 bits
'float'	'float'	floating point, 32 bits
'double'	'double'	long floating point, 64 bits
'uchar'	'unsigned char'	unsigned character, 8 bits
'ushort'	'unsigned short'	unsigned integer, 16 bits
'uint'	'unsigned int'	unsigned integer, 16 or 32 bits
'ulong'	'unsigned long'	unsigned integer, 32 bits
'char'	'char*1'	character, 8 bits
'float32'	'real*4'	32 bit floating point
'float64'	'real*8'	64 bit floating point
'int8'	'integer*1'	integer, 8 bits.
'int16'	'integer*2'	integer, 16 bits.

'int32'	'integer*4'	integer, 32 bits.
'intN'		signed integer, N bits wide
'uintN'		unsigned integer, N bits wide

where N represents any value between 1 and 32.

fwrite Write binary data to file.

COUNT = *fwrite*(FID,A,PRECISION) writes the elements of matrix A to the specified file, translating MATLAB values to the specified precision. The data are written in column order. COUNT is the number of elements successfully written.

FID is an integer file identifier obtained from FOPEN, or 1 for standard output or 2 for standard error.

PRECISION controls the form and size of the result. See the list of allowed precisions under *fread*.

Formatted I/O.

fscanf Read formatted data from file.

[A,COUNT] = *fscanf*(FID,FORMAT,SIZE) reads data from the file specified by file identifier FID, converts it according to the specified FORMAT string, and returns it in matrix A. COUNT is an optional output argument that returns the number of elements successfully read.

FID is an integer file identifier obtained from FOPEN.

SIZE is optional; it puts a limit on the number of elements that can be read from the file; if not specified, the entire file is considered; if specified, valid entires are: N read at most N elements into a column vector.

inf read at most to the end of the file.

[M,N] read at most M * N elements filling at least an M-by-N matrix, in column order. N can be inf, but not M.

FORMAT is a string containing C language conversion specifications. Conversion specifications involve the character %, optional assignment-suppressing asterisk and width field, and conversion characters d, i, o, u, x, e, f, g, s, c, and [. . .] (scanset). Complete ANSI C support for these conversion characters is provided consistent with 'expected' MATLAB behavior.

If a conversion character s is used an element read may cause several MATLAB matrix elements to be used, each holding one character. Mixing character and numeric conversion specifications will cause the resulting matrix to be numeric and any characters read to show up as their ASCII values one character per MATLAB matrix element.

fscanf differs from its C language namesake in an important respect - it is "vectorized" in order to return a matrix argument. The format string is recycled through the file until an end-of-file is reached or the amount of data specified by SIZE is read in.

fprintf Write formatted data to file.

COUNT = *fprintf*(FID,FORMAT,A,...) formats the data in matrix A (and in any additional matrix arguments), under control of the specified FORMAT string, and writes it to the file associated with file identifier FID. COUNT is an optional output argument that returns the number of bytes successfully written.

FID is an integer file identifier obtained from FOPEN. It can also be 1 for standard output (the screen) or 2 for standard error.

FORMAT is a string containing C language conversion specifications.

fprintf behaves like ANSI C with certain exceptions and extensions. These include:

1. If the MATLAB double doesn't convert exactly to the datatype associated with the conversion specifier then e format is used. You must explicitly convert non-integral MATLAB values to integral values if you plan to use an integral conversion specifier like d and get the expected ANSI C behavior.

2. The following non-standard subtype specifiers are supported for conversion characters o, u, x, and X.

t - The underlying C datatype is a float rather than an unsigned integer.

b - The underlying C datatype is a double rather than an unsigned integer.

For example, to print out in hex a double value use a format like '%bx'.

For more information see *fscanf* function above and format C specification below.

Format specification from MS *C++* help:

A format specification, which consists of optional and required fields, has the following form:

%[flags] [width] [.precision] [{ h | l | L}]type

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a type character (for example, %s). The optional fields, which appear before the type character, control other aspects of the formatting. The fields in a printf format specification are described in the following list:

Field Description

type Required character that determines whether the associated argument is interpreted as a character, a string, or a number. (See below)

flags Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.

width Optional number that specifies minimum number of characters output.

precision Optional number that specifies maximum number of characters printed for all or part of the output field, or minimum number of digits printed for integer values.

h, l, L Optional prefixes that determine the size of the argument expected, as shown below:

h Used with the integer types d, i, o, x, and X to specify that the argument is short int, or with u to specify short unsigned int. If used with %p, it indicates a 16-bit pointer.

l Used with d, i, o, x, and X types to specify that the argument is long int, or with u to specify long unsigned int; also used with e, E, f, g, and G types to specify double rather than float. If used with %p, it indicates a 32-bit pointer.

L Used with e, E, f, g, and G types to specify long double.

If a percent sign is followed by a character that has no meaning as a format field, the character is copied to stdout. For example, to print a percent-sign character, use %%.

The *type character* is the only required format field for the printf function; it appears after any optional format fields. The type character determines whether the associated argument is interpreted as a character, string, or number (as shown below).

Character	Type	Output Format
d	int	Signed decimal integer.
i	int	Signed decimal integer.
u	int	Unsigned decimal integer.
o	int	Unsigned octal integer.
x	int	Unsigned hexadecimal integer, using "abcdef."
X	int	Unsigned hexadecimal integer, using "ABCDEF."

f double Signed value having the form [-]dddd.dddd, where dddd is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.

e double Signed value having the form [-]d.dddd e [sign]ddd, where d is a single decimal digit, dddd is one or more decimal digits, ddd is exactly three decimal digits, and sign is + or -.

E double Identical to the e format, except that E, rather than e, introduces the exponent.

g double Signed value printed in f or e format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than -4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.

G double Identical to the g format, except that G, rather than g, introduces the exponent (where appropriate).

c int Single character.

s String Characters printed up to the first null character ('\0') or until the precision value is reached.

n Pointer to integer Number of characters successfully written so far to the stream or buffer; this value is stored in the integer whose address is given as the argument.

p Far pointer to void Prints the address pointed to by the argument in the form xxxx:yyyy, where xxxx is the segment and yyyy is the offset, and the digits x and y are uppercase hexadecimal digits; %hp indicates a near pointer and prints only the offset of the address.

fgetl Read line from file, discard newline character.

fgetl(FID) returns the next line of a file associated with file identifier FID as a MATLAB string. The newline and a carriage return if it preceeds it are NOT included. Use *fgets*() to get the next line with those characters INCLUDED. If just an end-of-file is encountered then -1 is returned. This is intended to be used ONLY with text files.

Example: (equivalent to 'type fgetl.m')

```
fid=fopen('fgetl.m');
while 1
   line = fgetl(fid);
   if ~isstr(line), break, end
   disp(line)
end
fclose(fid);
```

fgets Read line from file, keep newline character.

fgets(FID) returns the next line of a file associated with file identifier FID as a MATLAB string. The newline IS included. For more information see *fgetl* above.

File positioning.

ferror — Inquire file I/O error status. MESSAGE = *ferror*(FID,'clear') returns the error message for the most recent file I/O operation associated with the specified file. The string 'clear' is optional. If present, it clears the error indicator for the specified file. [MESSAGE, ERRNUM] = *ferror*(FID) also returns the error number.

feof — Test for end-of-file. *feof*(FID) tests whether the end-of-file indicator for the file with file identifier FID has been set. *feof*(FID) returns 1 if the end-of-file indicator is set, or 0 if it is not set.

fseek — Set file position indicator. STATUS = *fseek*(FID, OFFSET, ORIGIN) repositions the file position indicator in the specified file to the specified byte offset with respect to the specified origin.

FID is an integer file identifier obtained from FOPEN.

OFFSET values are interpreted as follows:

- > 0 Move toward the end of the file.
- = 0 Do not change position.
- < 0 Move toward the beginning of the file.

ORIGIN values are interpreted as follows:

- 'bof' or -1 Beginning of file
- 'cof' or 0 Current position in file
- 'eof' or 1 End of file

STATUS is 0 on success and -1 on failure. Example: `fseek(fid,0,-1)` "rewinds" the file.

ftell — Get file position indicator. POSITION = *ftell*(FID) returns the location of the file position indicator in the specified file. Position is indicated in bytes from the beginning of the file. If -1 is returned, it indicates that the query was unsuccessful; use *ferror* to determine the nature of the error. FID is an integer file identifier obtained from FOPEN, or 0 for standard input, 1 for standard output or 2 for standard error.

frewind — Rewind file. *frewind*(FID) sets the file pointer to the beginning of the file associated with file identifier fid.

Two Examples:

Formatted I/O

```
» JKR_FIT_P(1:5,:)
ans =
  Columns 1 through 7
   17.0000   16.0000    0.0264    0.0329    0.5000    0.0063  111.0000
   56.0000    6.0000    0.0267    0.0190    0.5000    0.0052  111.0000
   26.0000   41.0000    0.0428    0.0080    0.5000    0.0032   91.5713
   25.0000   39.0000    0.0348    0.0287    0.5000    0.0060   95.7821
   24.0000   57.0000    0.0159    0.0189    0.5000    0.0059  100.0000
  Columns 8 through 10
    0.1416   13.0000    1.0000
    0.1485   13.6000    1.0000
    0.1450   15.5000    1.0000
    0.1450   13.6000    1.0000
    0.1506   13.6060    1.0000
```

Use formatting to make it look better:

```
» format_str='%3.0f  %3.0f  %6.3f  %8.3g  %6.3f  %8.3g  %6.1f  %6.3f  %6.2f  %6.2f\n';
» title_str=' X    Y    error    Young    Poisson  Dupre  Radius  Spring  Surface  Factor';
» disp(title_str),  fprintf(1,format_str,JKR_FIT_P(1:5,:)')
 X    Y    error    Young    Poisson  Dupre  Radius  Spring  Surface  Factor
 17   16   0.026    0.0329   0.500    0.00626  111.0   0.142   13.00    1.00
 56    6   0.027    0.019    0.500    0.00523  111.0   0.148   13.60    1.00
 26   41   0.043    0.00798  0.500    0.00323   91.6   0.145   15.50    1.00
 25   39   0.035    0.0287   0.500    0.00601   95.8   0.145   13.60    1.00
 24   57   0.016    0.0189   0.500    0.00588  100.0   0.151   13.61    1.00
```

Unformatted I/O

Example for reading data file stored by Digital Instruments AFM.

Obtain Handle

```
[fid, MESSAGE]=fopen(fname);
if fid==-1, disp(MESSAGE), return, end
```

Read the header:

```
header_size=8192;
[header, count]=fread(fid,header_size,'uchar');
```

Read the rest:

```
[hi, count]=fread(fid,[hi_siz,hi_siz],'short');
hi=hi'*hi_sc/65536;
fv_all=fread(fid,[fv_siz,2*FP_P/fv_siz],'short');
ind=1:2:(2*FP_P/fv_siz-1);
fvr=fv_all(:,ind); fve=fv_all(:,ind+1);
fclose(fid);
```

Analyze read_spa.m or read_spc.m function as examples on using low level file IO.