MATLAB
User-defined functions, Data Input/Output

Edited by Péter Vass

User-defined functions

Although, MATLAB provides a wide range of built-in functions it may often be necessary to create and use an own, user-defined function.
Similarly to a script, a user-defined function is stored in a file of .m extension.
It can be created in the script editor window.
The name of the file must be the same as the name of the function.
Unlike a script, a function can accept input parameters (or arguments) and return output parameters.
A function operates on variables within its own *local workspace*.
The local workspace is independent of the *base workspace* of the MATLAB command line.
This is the reason why a function cannot reach the variables of the base workspace during its running.

User-defined functions

A function definition has two main parts:

- header,
- body.

The information about the name and parameters of the function is specified in the header of the function.

The body of the function contains the sequence of instructions by which a specific problem can be solved.

The general format of a function definition:

function *[out1,out2, ..., outN]* = funcname*(in1,in2,in3, ..., inM)*

instructions

After defining and saving the function, it can be called either in the command line or a script.

Moreover, it can also be called in the body of another user-defined function.

So, more complex functions can be implemented by the utilization of simpler ones.

User-defined functions

Example

Let us write a simple, user-defined functions for computing the root mean square error (RMSE) and mean absolute error (MAE) between two input vectors of the same size.

```
function [rmse, mae]=calcerrors(x, y)
%The function computes the RMSE and MAE between two input
% vectors of the same time

rmse=sqrt(sum((x-y).^2)/length(x));
mae=sum(abs(x-y))/length(x);
```

User-defined functions

Example of calling the function

```
>> t=0:0.1:5;
>> u=exp(-t).*cos(2*pi*t);
>> plot(t,u)
>> noise=0.05*randn(size(t));
>> un=u+noise;
>> hold on;
>> plot(t,un,'r');
>> [err1, err2]=calcerrors(u,un)
```

Data input/ output

MATLAB provides three different techniques to read data from files and save data into files:

- using the load and save commands,
- using other commands for loading and saving plain ASCII files,
- using low-level input/ output functions for other file formats.

Load and save commands

The save command is used for saving all the variables of the base (command line) workspace into a file.

The *filename* must be specified after the save command:

>>save *filename*

MATLAB creates a binary file of .mat extension in the current folder (the general format of .mat files is described in the documentation).

Data input/ output

The variables stored in a MATLAB binary format file can be read in by using the load command:

>>load *filename*

If not all the variables are needed to save or load, the names of the required variables must be listed after the name of the file in the command line.

>>save *filename var1 var2 .. varN*

>>load *filename var1 var 2*

This type of data input and output is suggested when the actual MATLAB workflow has to be finished temporarily, but later it is planned to continue with the variables created in the previous MATLAB session.

The file of MATLAB binary format, however, cannot be processed by other software environments (e.g. spreadsheet or word-processing software).

Data input/ output

Loading and saving plain ASCII files

The MATLAB provides some useful built-in functions by which tabular format data sets can be saved in plain ASCII coded files and data can also be loaded in the workspace from such files.

>>csvwrite('filename.ext', M)

writes the matrix M into the file specified between single quotes. The numbers are separated by commas and arranged in a tabular format.

>>M = csvread('filename.ext')

reads the data from a comma-separated file whose name and extension are specified between single quotes.
The file can only contain numeric values in a tabular format.
The data are loaded in the matrix M.

Data input/ output

>>M = csvread('filename.ext', R, C)
reads the data from a comma-separated file starting at row *R* and column *C*. The numbering of rows and columns starts with 0.
This way of calling csvread can be useful when the first row in the file is a header.

>>dlmwrite('filename.ext',M,'dlm')
writes the matrix M into the file specified between single quotes.
The numbers are separated by using the character *dlm* as the delimiter of data columns.
A frequently applied delimiter:
horizontal tabulator '\t'

Data input/ output

>>M= dlmread('filename.ext','dlm')
reads the data from the file specified between single quotes.
The file can only contain numeric values in a tabular format, and the numbers are separated by using the character *dlm* as the delimiter of data columns.
The data are loaded in the matrix M.

>>M= dlmread('filename.ext','dlm', R, C)
reads the data from the file starting at row R and column C. The numbering of rows and columns starts with 0.

Data input/ output

Examples

```
>> data=[t',u',un'];
>> size(data)
>>save data
>> csvwrite('datacsv.csv',data)
>> M=csvread('datacsv.csv');
>> dlmwrite('datatab.txt',data,'\t');
>> N=dlmread('datatab.txt','\t');
```

Data input/ output

Loading data from MS Excel spreadsheet

xlsread

>> help xlsread

Writing data into MS Excel spreadsheet

xlswrite

>> help xlsread

Examples

```
>> xlswrite('dataxls.xls',data);
>> XLS=xlsread('dataxls.xls');
```

Data input/ output

Low-level input/ output functions

The so-called low level input/output functions provide the most flexible way of file input and output.

However, their proper application for reading or writing files requires the detailed knowledge of their documentation and some programming skill.

File opening and closing:	fclose	fopen	
Unformatted I/O:	fread	fwrite	
Formatted I/O:	fgetl	fprintf	
	fgets	fscanf	
File positioning:	feof	fseek	
	ferror	ftell	frewind
String conversion:	sprintf	sscanf	

Data input/ output

Example of the script file *writedata.m*

```
% open a file for writing
fid = fopen('datafile.txt','w');
% writing a header and a unit line
fprintf(fid,'time   noiseless   noisy\n');
fprintf(fid,'[s]   [mV]   [mV]\n');
% print values in column order
% three values of float-type appear on each row of the file
fprintf(fid,'%f   %f   %f\n', data');
fclose(fid);
```

Data input/ output

Example of the script file *readdata.m*

```
% open a file for reading
fid = fopen('datafile.txt');
% reading the header
header=fgetl(fid);
%reading the unit line
units=fgetl(fid);
%  Read the values in column order, and transpose to match the
%appearance of the file
DATA=fscanf(fid,'%f',[3, inf])';
fclose(fid);
```