MATLAB Tutorial

Chapter 5. File input/output
5.1. Saving/reading binary files and making calls to the operating system

When using MATLAB, either when running a m-file or performing calculations interactively, there is a master memory structure that MATLAB uses to keep track of the values of all of the variables. This memory space can be written in a binary format to a file for storing the results of your calculations for later use. This is often useful when you have to interrupt a MATLAB session. The following commands demonstrate how to use this storage option to make binary .mat files.

First, let us define some variables that we want to save.

```
num_pts =10;
Afull=zeros(num_pts,num_pts);
Afull(1,1) = 1;
Afull(num_pts,num_pts) = 1;
for i=2:(num_pts-1) sum over interior points
Afull(i,i) = 2;
Afull(i,i-1) = -1;
Afull(i,i+1) = -1;
end
b = linspace(0,1,num_pts)';
x = Afull\b;
```

whos; display contents of memory

The "save" command saves the data in the memory space to the named binary file.

```
save mem_store1.mat;
```

```
clear all;
```

whos; no variables are stored in memory

ls *.mat display all .mat files in directory

The "load" command loads the data stored in the named binary file into memory.

```
load mem_store1.mat;
```

whos; we see that the data has been loaded again

If we want to get rid of this file, we can use the "delete" command.

```
delete mem_store1.mat;
ls *.mat
```

In the commands above, I have used path names to specify the directory. We can view our current default directory using the command "pwd".

pwd displays the current directory

We can then change to another directory using the "cd" command.

cd .. move up one directory
pwd
ls list files in directory
cd MATLAB_tutorial; directory name may differ for you
pwd; ls

We can also use the "save" command to save only selected variables to a binary file.
save mem_store2.mat Afull;

clear all
whos

load mem_store2.mat
whos

delete mem_store2.mat

clear all

5.2. Input/output of data to/from an ASCII file

First, let use define some variables that we want to save.
num_pts =10;

Afull=zeros(num_pts,num_pts);
Afull(1,1) = 1;
Afull(num_pts,num_pts) = 1;
for i=2:(num_pts-1) sum over interior points
Afull(i,i) = 2;
Afull(i,i-1) = -1;
Afull(i,i+1) = -1;
end

b = linspace(0,1,num_pts)';
x = Afull\b;

whos; display contents of memory

Now, let us write out the contents of Afull into a file that we can read.

One option is to use the "save" command with the option -ascii, that writes to a file using the ASCII format.
save store1.dat Afull -ascii;
type store1.dat view contents of file

We can also load a file in this manner. The contents of the ASCII file filename.dat are stored in the MATLAB variable filename. This is a good way to import data from experiments or other programs into MATLAB.
load store1.dat;

If we add the option -double, the data is printed out with double the amount of digits for higher precision.
delete store1.dat;
save store1.dat Afull -ascii -double;
type store1.dat

We can use this command with multiple variables, but we see that no spaces are added.
delete store1.dat;
save store1.dat Afull b x -ascii;
type store1.dat view contents of file
delete store1.dat get rid of file

MATLAB also allows more complex formatted file input/output of data using commands that are similar to those in C.

First, we list all of the files in the directory.
ls

Next, we see create the output file and assign a label to it
with the "fopen" command that has the syntax
FID = fopen(FILENAME,PERMISSION)
where PERMISSION is usually one of :
'r' = read only
'w' = write (create if needed)
'a' = append (create if needed)
'r+' = read and write (do not create)
'w+' = create for read and write
'a+' = read and append (create if needed)

```
FID_out = fopen('test_io.dat','w');
ls
```

Now, we print the b vector to the output file as a column vector using the "fprintf" command. In the FORMAT string '\n' signifies a carriage return, and 10.5f specifies a floating point decimal output with 5 numbers after the decimal point and a total field width of 10.

```
for i=1:length(b)
fprintf(FID_out,'10.5f \n',b(i));
end
```

We now close the file and show the results.

```
fclose(FID_out);
disp('Contents of test_io.dat : ');
type test_io.dat;
```

MATLAB's "fprintf" can also be loaded to avoid the need of
using a for loop

```
FID_out = fopen('test_io.dat','a');
fprintf(FID_out,'\n');
fprintf(FID_out,'10.5f \n',x);
fclose(FID_out);

disp('Contents of test_io.dat : ');
type test_io.dat;
```

We can also use "fprintf" to print out a matrix.

```
C = [1 2 3; 4 5 6; 7 8 9; 10 11 12];
FID_out = fopen('test_io.dat','a');
fprintf(FID_out,'\n');
for i = 1:size(C,1)
fprintf(FID_out,'5.0f 5.0f 5.0f \n',C(i,:));
end
fclose(FID_out);

disp('Contents of test_io.dat : ');
type test_io.dat;
```

We can read in the data from the formatted file using
"fscanf", which works similarly to "fprintf".

First, we open the file for read-only.

```
FID_in = fopen('test_io.dat');
```

We now read the b vector into the variable b_new. First, we allocate space for the vector, and then we read in the values one by one.

```
b_new = linspace(0,0,num_pts)';
for i=1:num_pts
b_new(i) = fscanf(FID_in,'f',1);
end
b_new
```

Now read in x to x_new, using the overloading possible in MATLAB.

```
x_new = linspace(0,0,num_pts)';
x_new = fscanf(FID_in,'f',num_pts);
x_new
```

Finally, we read in the matrix C to C_new.

```
C_new = zeros(4,3);
for i=1:size(C,1)
for j=1:size(C,2)
C_new(i,j) = fscanf(FID_in,'f',1);
end
end
C_new

fclose(FID_in);

clear all
```