Session – 28 : Some more about octave

25-10-2019, 08:00 – 09:00 Hrs, RJN 302

1. Loading data into an array:

One can load an array of data stored in a file directly in octave.

Commands to achieve this when the data file is "mydata.dat"
`A = load('mydata.dat');`

One can only load numeric data as arrays into array objects in octave. The numbers can be integers like "`21`" as well as real numbers like "`1.023e-21`" for example.

Images can be loaded as arrays directly. The following lines read an image called "`test.png`" and display the same. Try this out and compare the size of the array with the output of "`file test.png`" command. Gray scale images are 1 layer arrays, rgb images are 3 layer arrays – one each for red, green and blue channels and cmyk images are 4 layer arrays – one each for cyan, magenta, yellow and black channels. You can take slice of the array or a specific layer, copy to another array and perform operations on them just like on any array.

```
T = imread("test.png");
imshow(T)
```

One cannot do that with array of strings easily. The procedure is a little longer. We take up a task as described below to learn about that.

2. Loading a list of functions from a file.

The task we have is as follows. A file called "`f.txt`" has lines that contain definition of a function with `x` as variable, the range of values of x over which we need to plot the function and visualize. The function definition, xmin and xmax are separated using a comma. For our convenience, the function definitions are given in a syntax that is compatible with octave or Matlab.

Contents of the file "`f.txt`"

`sin(x),0,3.14`
`cos(2*x),0,3.14`
`exp(x),0,5`
`sin(x).*exp(-x),0,5`
`sin(2*x)+cos(2*x),0,3.14`
`x+2*x.*x+3*x.*x.*x,0,5`

The above list can be loaded in to octave and plotted individually with the corresponding ranges using the following code.

Contents of "getlines.m"

```
infile = fopen("f.txt");
n_lines = fskipl(infile, Inf);
frewind(infile);
lines=cell(n_lines,1);
for i=1:n_lines,
  lines{i} = fscanf(infile,'%s',1);
end

for i=1:n_lines,
  fields=strsplit(lines{i},",");
  fstring = sprintf("function y=myf(x); y=%s;",fields{1});
  eval(fstring);
  xmin = str2num(fields{2});
  xmax = str2num(fields{3});
  delta = (xmax-xmin)/10.0;
  x=[xmin:delta:xmax];
  y=myf(x);
  figure(i)
  p = plot(x,y)
  set(p,'linewidth',[2]);
  xlabel('value of x');
  ylabel('f(x)');
  tstr = sprintf("f(x)=%s",fields{1});
  title(tstr);
  fstr = sprintf("print -dpng f-%d.png",i);
  eval(fstr);
end
```

Concepts that we learn from the above code are as follows:

- File handle returned by calling `fopen()` is similar to file pointer in C language.
- The function `fskipl()` gives the number of lines in the file pointed by the handle. While reading the lines, the file pointer is keeps traversing from the beginning of the file and at the end it points to the end of the file.
- The function `frewind()` brings the file pointer back to the beginning of the file.
- The function `cell()` creates an empty array to load the strings we are planning to read from the file.
- The function `fscanf()` is similar to its counterpart in C language – it reads a string from the file and returns the same.
- The function `strsplit()` uses the field separator provided and splits the array of characters into separate strings in a list. The number of items in this list equals the number of fields separated by comma in each line read.

- The function `sprintf()` has the same role as in C language. By creating a string that contains the definition of a function, we aim to pass it on to "`eval`" to evaluate it and create a function definition *on the fly*.
- The function `str2num()` converts a string to a number. Visually, a string of characters may look like a number but the internal representation (aka type[1]) could be character and could be incompatible with numeric functions. This function str2num() helps convert strings to numbers to pass on to numeric functions. If the string does not contain a number, it will return a null array.
- In the above code, we choose to plot the function using `10` intervals. Thus we create an array `x` that has 11 points by dividing the interval `xmin` to `xmax` with `10`.
- The name of the function created *on the fly* is `myf()`. Thus we use it for all the lines to evaluate the function and store the output in the array `y`.
- We can then `plot` the functions, increase the `linewidth` to make the plot thicker, set the labels for x-axis and y-axis and make the `title` more useful by using a string that contains the function definition.
- We can save the plots using names that are created dynamically using `sprintf()` to create the print command the using `eval` to execute it..

Comment lines in the above code using `%` character at the beginning of the line, delete the semicolon `;` at the end of lines that you want to inspect.

1 The so called "type safety" is a big deal in programming languages. Converting one type to another type may appear to be a pain but actually it helps a lot when doing serious computing by avoiding errors due to inappropriate conversions.