Basic Unix and Matlab 101

1 Logging in from another Unix machine, *e.g.* ECS lab Dells

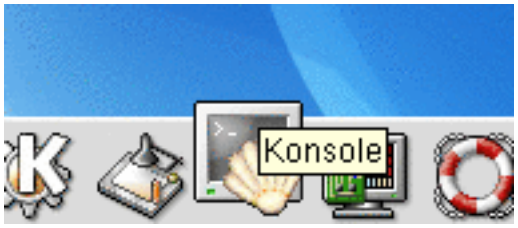
The computer we will be using for our assignments is called

`malkhut.engr.umbc.edu`

which is a Unix/Linux machine that is used for simulations by members of the Photonics Research Group in TRC. You all have accounts and passwords on malkhut which were handed out in class. (The passwords are the same as for the class webpage.) We will be making assignments that support using the computers in the labs in the ECS building that run Linux. All Dell workstations that run Windows 2000 in the labs also run Linux if you reboot them. You can then log into them using your gl username and password.

Suppose you walk into one of the ECS labs and you want to log in to malkhut to start to use it. Follow the following steps to do so:

1. Choose one of the Dell machines in the ECS labs.
2. Assuming the machine you choose is running Windows 2000, it needs to be rebooted into Linux for your work for this class. Do this by pressing Ctrl+Alt+Del, then click on "Yes", then click "Shutdown", and then choose "Restart" from the menu. When you click OK, the machine will shutdown and reboot.
3. After the Dell splash screen, you will have a choice of Windows or Linux. Press the down-arrow to highlight Linux and press Enter.
4. You may see a prompt for "LILO Boot:". Press Enter or wait a few seconds, and the machine will boot into Linux.
5. Choose KDE from the pull-down menu and try logging in with your username and password. KDE is a windowing environment on Unix machines that resembles Microsoft Windows and is convenient to use. You may go through a series of menus to set up KDE. The default choices for setting up KDE are probably fine.
6. Click on the console icon at the bottom of the screen:

   A new shell window should appear.
7. Type "`hostname`" at the prompt and take note of what it says. The `hostname` command gives the name of the machine you are working on. Example:

   ```
   [marks@ecs021pc07-lx ~]$ hostname
   ecs021pc07-lx.ucslab.umbc.edu
   ```

8. On Unix machines, you can run a program (*e.g.* Matlab) on one machine, and have the results displayed on another. For example, you can sit at the computer in the ECS lab, log in remotely to malkhut, and run Matlab so that the plots display on your computer in the ECS lab. Type the following to allow windows from malkhut to appear on your computer:

   ```
   [marks@ecs021pc07-lx ~]$ xhost +malkhut.engr.umbc.edu
   malkhut.engr.umbc.edu being added to access control list
   ```

This tells the local (lab) machine to allow malkhut to contact it to display windows.

9. To log into malkhut, type

```
[marks@ecs021pc07-lx ~]$ ssh marks@malkhut.engr.umbc.edu
marks@malkhut.engr.umbc.edu's password:
Last login: Mon Jan 27 16:40:09 2003 from 130.85.112.174

[marks@malkhut ~]$
```

   However, of course, you would substitute "`marks`" for your username.

10. Now that you're logged into malkhut, you need to tell malkhut (the remote machine) to display windows on the local (lab) machine. Type the following to do this:

```
[marks@malkhut ~]$ setenv DISPLAY ecs021pc07-lx.ucslab.umbc.edu:0
```

    Note: The name of the machine you are on in the lab is whatever the response was from the "`hostname`" command — `ecs021pc07-lx.ucslab.umbc.edu` is just an example. **Note #2:** The `:0` at the end of the hostname is **important and meaningful**. The windowing program (X-windows) puts the windows on the screen associated with the display *¡hostname¿:¡screen number¿*, in which the screen number 0 is the default screen. If a computer has more than one monitor, one can use the associated screen number greater than 0. The bottom line is: your windows will not be displayed properly if you forget the `:0`!

11. Now you can type

```
[marks@malkhut ~]$ matlab
```

    on malkhut, and you should see the Matlab splash window pop up on your lab machine. Type "quit" to get out of Matlab.

2 Common Unix commands

Helpful first command to know:

- **logout** or **exit** : Let me out of here! Logs off the machine.

File listing commands:

- **ls** : Show me a list of the current directory.
    1. **ls -lsap** : Detailed listing of files including file permissions, sizes, and file types.
    2. **ls -ltr** : Similarly detailed listing, but it shows the most recently modified files last.

Directory commands:

- **pwd** : Print working directory, *i.e.*, what directory are you in now?
- **mkdir** : Make a subdirectory in the current directory
- **rmdir** : Remove (delete) a subdirectory, which must be empty first
- **cd** : Change directory. Ex: `cd programs` puts you into a subdirectory named "programs"
    1. **cd ..** : Go up one level in the directory tree

File commands:

- **cp** : Copy a file from one place to another

- **mv** : Rename or move a file from one place or name to another
- **rm** : Remove (delete) a file. Use with caution.
- **wc** : Word count. This returns three columns for the number of lines, words, and characters in a file.
- **more** : View a listing of the file page-by-page
- **head** : View the first few lines of a file
- **tail** : View the last few lines of a file
- **gzip**, **gunzip** : Compress, uncompress a file, for space conservation
- **tar** : "Glue" files together into a single archive file
    1. **tar cvf file.tar file1 file2 ...** : Create a tar file containing file1 file2 ...
    2. **tar xvf file.tar** : Un-tar (extract) a tar file

Text editors:

- **xemacs** : Graphical user interface (GUI) based version of emacs, which is a very powerful text editor for programmers. xemacs is not too difficult for newcomers to learn to use.
- **nedit** : GUI-based powerful text editor that's even simpler for newcomers than xemacs.
- **pico** : Easy-to-use text editor for a terminal window
- **emacs** : Terminal window version of emacs. Not quite as easy to use as the GUI version.
- **vi** : Another terminal window text editor. An old stand-by for some, but not for the faint-of-heart.

HELP!!:

- **man** : Manual pages, which give information about all Unix commands as well as common C and C++ functions. Use this frequently if Unix seems complicated to you!
- **man -k** : Manual pages too, but it looks for a "keyword" rather than a command itself. This is useful if you don't know quite what you're looking for.

3 Basic Matlab

Simply put, everything in Matlab is a matrix. Hence the name, which is derived from MATrix LABoratory. Matlab is a powerful computational tool that can be used for anything from simply plotting data to performing complicated numerical simulations.

Once you have logged into malkhut and set the display properly, as described in Section 1, you can run Matlab by typing

```
matlab &
```

at the command line prompt on malkhut. After a short wait, you should see the Matlab window appear on your screen.

As a good first cut, you should read and go through the *Matlab Primer* that is linked on the class webpage. It is only 39 pages and has all the basics of Matlab in it. It does not take long to go through and will be extremely helpful. In particular, I highly encourage going through the Introduction, Sections 2–12, 16, 18, and 20. More (and newer) information can be found in the book *Mastering Matlab 6* by Hanselman and Littlefield from Prentice Hall Publishing. We have this book on reserve.

Once Matlab is launched on your screen, you can enter commands directly, or you can edit Matlab scripts (M-files or .m-files) using your favorite Unix text editor or by using the Matlab editor itself using the command `edit`. Matlab's editor is actually quite good for .m-files.

3.1 Simple example #1: Plotting a function

In order to plot a function in Matlab, you need an x and you need a y. The Matlab `linspace` command comes in handy when defining these quantities. At the Matlab command prompt, type

```
>>  x = linspace(0, 6*pi, 200);
>>  y = sin(x);
```

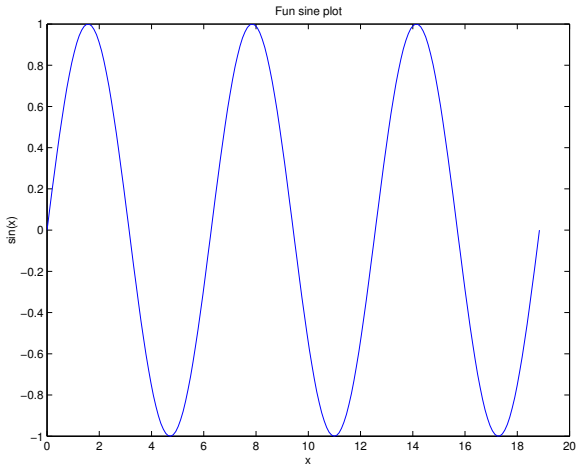
The first line generates a vector `x` with 200 elements whose first element is 0, whose last element is 6π, and whose other elements are evenly spaced between the two. The second line generates a vector `y` with 200 elements whose values are the sine of the elements of `x`. Now we can plot the function $\sin(x)$ easily by simply typing

```
>>  plot(x,y)
```

By adding the commands

```
>>  xlabel('x');
>>  ylabel('sin(x)');
>>  title('Fun sine plot');
```

we get the following plot:



3.2 Simple example #2: Plotting data from a file

In a similar way, we can load data from a file and plot it. If we have an ASCII data file called with two columns, we can load this data into Matlab and plot it using the following simple commands:

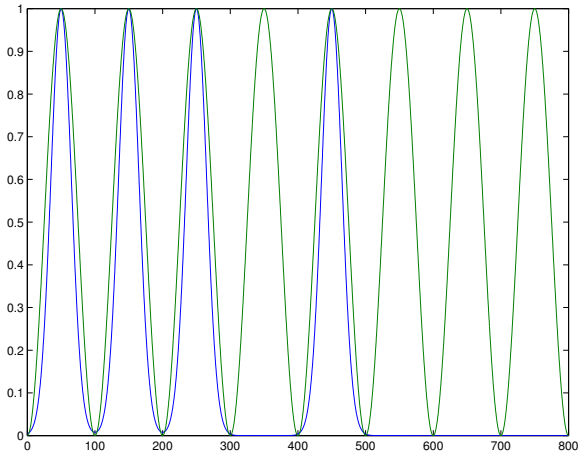
```
>>  dum = load('bit_string.txt');
>>  x = dum(:,1);
>>  y = dum(:,2);
>>  plot(x,y);
```

Since the file is in two-column format, the `load` command in Matlab loads it in as an $N \times 2$ matrix. The second and third commands above assign the first and second columns of the matrix `dum` to `x` and `y`.

One can also plot more than one column of data by using similar commands. Suppose you have a file with three columns of data, and that the first column represents time, and the other two represent some function of time. We would like to plot both functions of time on the same plot with different colors. Then we would use the commands

```
>>  dum = load('bit_string3.txt');
>>  x = dum(:,1);
>>  y = [dum(:,2) dum(:,3)];
>>  plot(x,y);
```

and get one curve in blue and one curve in green:



3.3 M-files and programming

M-files are Matlab scripts — files that give Matlab a number of commands. They are simply files that end with a `.m` extension that reside in Matlab's working directory. They can consist of a sequence of normal Matlab commands, or they take the form of a function that accepts arguments and returns a set of values. M-files can access other M-files in the same directory or path.

```
%  This m-file is a script that plots data from a file called infile.dat
%  that is in column format.  It assumes the first column is the x-axis
%  and that all other columns define the dependent variables y1...yN.

data = load('infile.dat');

% Assuming infile.dat is in column format, the matrix 'data' is M x N.
% We can find out how many columns were read in by doing a 'size' command:
[M N] = size(data);


%  The first column vector is assigned to the variable x.  The rest
%  of the columns form a matrix that we assign to y.
x = data(:,1);
y = data(:,2:N);

%  Plot all the data:
plot(x,y);

%%  Questions:  What are the pitfalls of this program?  When does it work
%%  and when might it fail?  How flexible is it?
```

```
function [xx, yy] = less_simple_plot(fname,cols_to_plot,plot_type)
% Function  [xx, yy] = less_simple_plot(fname, cols_to_plot, plot_type)
%
% This m-file is a less simple plotting utility than really_simple_plot.m.
% It accepts the following arguments:
%   fname        :  The name of the input file of column data to read in
%   cols_to_plot :  A vector indicating which columns of fname to plot
%                   (optional)
%   plot_type    :  A string indicating if the plot should be a log plot
%                   (optional).  Possibilities are 'lin', 'logx', 'logy',
%                   and 'logxy'.
% Its outputs are the following:
%   xx           :  The first column of fname, assumed to be the independent
%                   variable
%   yy           :  A matrix comprised of the columns of fname given by the
%                   vector cols_to_plot


% As before, read in the data from the file fname and determine its size.
data = load(fname);
[m n] = size(data);

% In Matlab, nargin is a function that determines how many input arguments
% were passed to the function.
% We can use nargin to set up default values for the optional arguments:
if nargin == 1
  cols_to_plot = 2:n;
  plot_type = 'lin';
elseif nargin == 2
  plot_type = 'lin';
```

```
end

% As before, the first column is assumed to define the x-axis and
% the remaining columns, defined now by cols_to_plot,
% are to be plotted against this.
x = data(:,1);
y = data(:,cols_to_plot);

% Test to see what type of plot it is and use the appropriate plotting
% command:
switch plot_type
  case {'lin'}
    plot(x,y);
  case {'logx'}
    semilogx(x,y);
  case {'logy'}
    semilogy(x,y);
  case {'logxy'}
    loglog(x,y);
  otherwise
    error('Unknown plot_type argument.  Quitting.');
end

% Also note that in this code, the output [x y] is the x and y data
% defined by the first column of fname and the columns defined
% by cols_to_plot, respectively.
% So, if there are two output arguments present,
% assign x and y to them:
if nargout == 2
  xx = x;
```

```
  yy = y;
end

%% Question:  What are the pitfalls and limitations of this code?
```

```
function [x, y, z] = simple_plot_3d(fname, stride, plot_type)
%  function [x, y, z] = simple_plot_3d(fname, stride, plot_type)
%  Reads in a three-column file and plots a surface plot of the data.
%  The input file is of the form:
%     z1  t1  p11
%     z1  t2  p12
%       ...
%     z1  tN  p1N
%     z2  t1  p21
%       ...
%  which is useful if you want to plot data of the form
%  distance, time, and power, as is useful with OCS.
%  In this demo, stride = N = number of time points in
%  the discretization.

% As always, read in data and find its size:
data = load(fname);
[m n] = size(data);

% As before, do some checking for number of arguments:
if nargin == 1
  error('Not enough arguments!');
elseif nargin == 2
  plot_type = 'lin';
end

% We next have to reformulate the data into something matlab understands.
% Matlab's surf(X,Y,Z) command plots a surface of data where
% X, Y, and Z are MxN matrices describing the X, Y, and Z data.
% Typically, the X matrix would have every row being the same, and
```

```
% the Y matrix would have every column being the same for a plot
% like ours.  The following sets up the X, Y, and Z matrices as
% the variables time, distance, and power which are numZs x stride big.
numZs = m/stride;
time = ones(numZs,1)*data(1:stride,2)';          % Form a numZs x stride mat
rix of time
distance = data(1:stride:end,1)*ones(1,stride);  % Form a numZs x stride mat
rix of dist
power = reshape(data(:,3),stride,numZs)';        % Form a numZs x stride mat
rix of power

switch plot_type
  case {'lin'}
    surf(time,distance,power,'linestyle','none');
  case ('logz')
    surf(time,distance,log10(power),'linestyle','none');
  otherwise
    error('Unknown plot type');
end

% As we did in less_simple_plot, output the data into x, y, and z
% variables for convenience.
if nargout == 3
  x = time;
  y = distance;
  z = power;
end

%% Questions: What are the pitfalls and deficiencies of this code?
```

```
function [xx, yy] = eye_plot(fname,col_to_plot,N_bits,Eye_width)
% Function  [xx, yy] = eye_plot(fname, col_to_plot, N_bits, Eye_width)
%
% This function takes in time data from the file fname and plots it as
% an eye diagram.  The user must input how many bits are in the time
% sequence in the input file, and how many bits wide the eye diagram
% should be.  The first column of fname is assumed to be the time mesh.
%  Input arguments:
%     fname           : Input file name
%     col_to_plot     : Which column of fname will be used to plot the eye
%     N_bits          : How many bits of data are in the time mesh in fname
%     Eye_width       : How many bits wide should the eye diagram be
%  Output arguments:
%     xx              : Vector of time data
%     yy              : Matrix consisting of all the traces in the eye
%                       diagram


% As before, read in the data from the file fname and determine its size.
data = load(fname);
[m n] = size(data);

% As before, the first column is assumed to define the x-axis and
% the second column is the time domain data of the signal,
% to be plotted as an eye diagram.
x = data(:,1);
y = data(:,col_to_plot);

% First determine how many time mesh points there are per bit.
% This is needed so we know how much to time shift the data to plot
```

```
% the eye.
Pts_per_bit = m/N_bits;

% One trace in the eye diagram is the time data itself.  Then
% all possible time-shifted copies are considered as well.  This
% for-loop concatenates all the possible bit shifts in one big
% matrix.  The circular (periodic) shifts are done with the
% Matlab command wshift.
yplot = y;
for j = 1:N_bits
  yplot = [yplot wshift(1,y,Pts_per_bit*j)];
end

% We then plot the data in black, and use the xlim command to
% restrict the view of the eye diagram to a certain number of
% bits.
plot(x,yplot,'k'); xlim([0 x(end)*Eye_width/N_bits]);

% Also note that in this code, the output [x y] is the x and y data
% defined by the first column of fname and the columns defined
% by cols_to_plot, respectively.
% So, if there are two output arguments present,
% assign x and y to them:
if nargout == 2
  xx = x;
  yy = yplot;
end
```