MatLab Introduction – work in groups of 1, 2, or 3

To use Matlab:

You have 5 options for using Matlab:

1. Go to a campus lab where it is already installed and setup. This includes the Pearson lab and the lab in the basement of Smith Hall. **THIS IS BY FAR THE EASIEST OPTION**. If you choose this option, ignore options 2 and 3 and jump down to the Pre-Lab

2. Download Octave, a Matlab clone, for your computer.

3. Use ssh on your own machine to connect to Strauss. Then at the Strauss prompt you can type

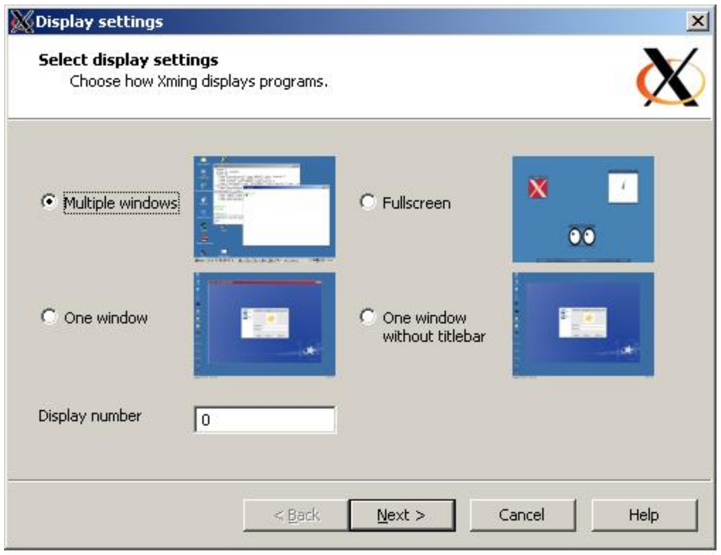
```
> matlab –nojvm
```

to use the Matlab interpreter without the convenient (but very slow) GUI user interface. You can write files in one ssh window using an editor like emacs, and use the matlab interpreter in the other window.

4. Buy a student / academic version of Matlab to install on your own computer. The basic setup costs $49/99 and is good for at least a year (see details at the site). http://www.mathworks.com/academia/student_version/faq/prodinfo.html for information.

4. Run Matlab remotely from your machine. This is not trivial. You will also need to run a web browser (firefox) remotely (on strauss) to submit the code to Sakai. If you cannot get this to work you should go with Option 1. ***You are responsible for setting it up on your own computer if you decide to use this option.***

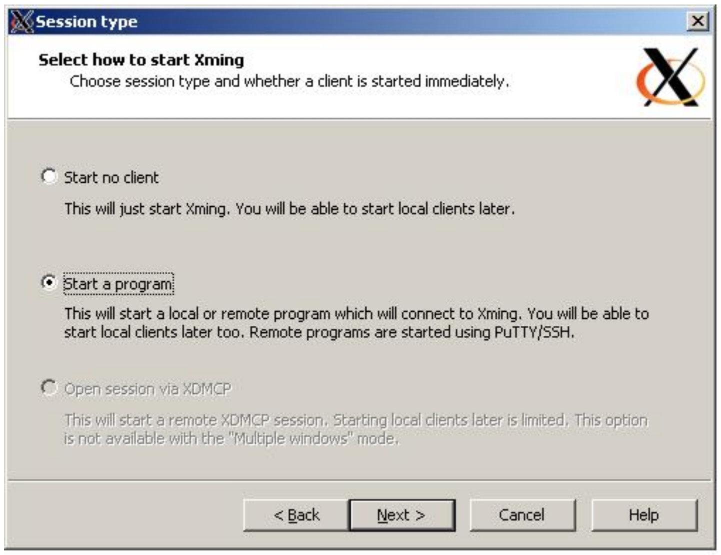
- For Mac: make sure you have X11 installed. You can find it under Applications->Utilities->X11. If you do not have X11, you can download from the Apple site for your OS. Launching this application will give you a terminal window from which you can type: (note the option '-YC' is typed there)
  ssh -YC username@strauss.udel.edu
  Where username is your UDSIS username. Once at the strauss> prompt, simply run matlab and firefox by typing them into the prompt:
  ```
  strauss> matlab &
  strauss> firefox &
  ```
- For PC: you will need to install the Xming X11 environment (http://sourceforge.net/project/downloading.php?group_id=156984&filename=Xming-6-9-0-31-setup.exe). When installing DO NOT check the option at the end to launch Xming or Xlaunch. Use all defaults during the installation.
  Once installed, go to your Program menu and find the Xming->Xlaunch option. Setup a new Xlaunch configuration as shown in these pictures:


Display settings

Select display settings
Choose how Xming displays programs.

[✓ Multiple windows] Fullscreen
One window
One window without titlebar

Display number: 0

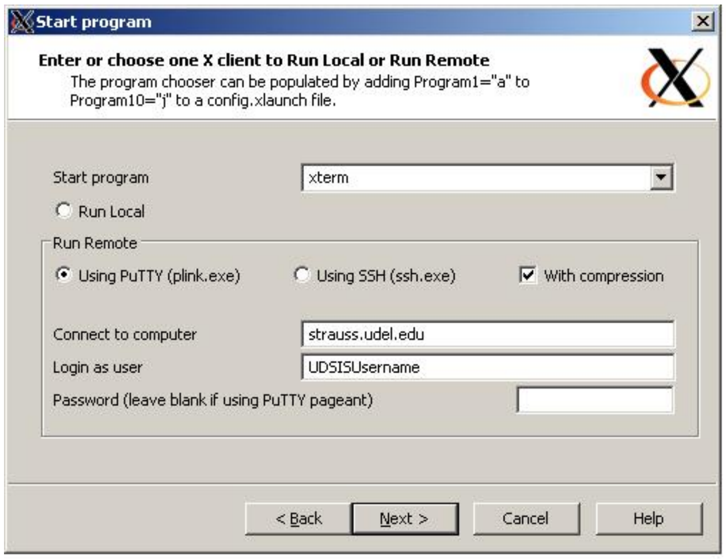

Session type

Select how to start Xming
Choose session type and whether a client is started immediately.

Start no client
This will just start Xming. You will be able to start local clients later.

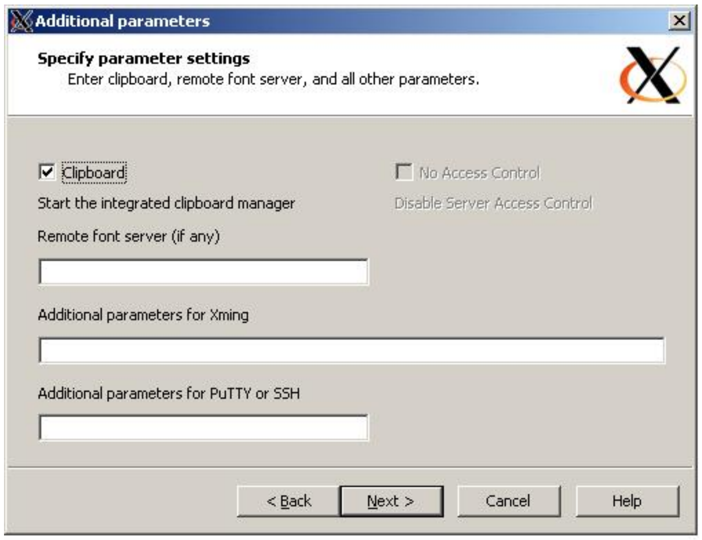
[✓ Start a program]
This will start a local or remote program which will connect to Xming. You will be able to start local clients later too. Remote programs are started using PuTTY/SSH.

Open session via XDMCP
This will start a remote XDMCP session. Starting local clients later is limited. This option is not available with the "Multiple windows" mode.


Start program

Enter or choose one X client to Run Local or Run Remote
The program chooser can be populated by adding Program1="a" to Program10="j" to a config.xlaunch file.

Start program: xterm
Run Local
Run Remote
[✓ Using PuTTY (plink.exe)] Using SSH (ssh.exe) [✓ With compression]
Connect to computer: strauss.udel.edu
Login as user: UDSISUsername
Password (leave blank if using PuTTY pageant):



Additional parameters

Specify parameter settings
Enter clipboard, remote font server, and all other parameters.

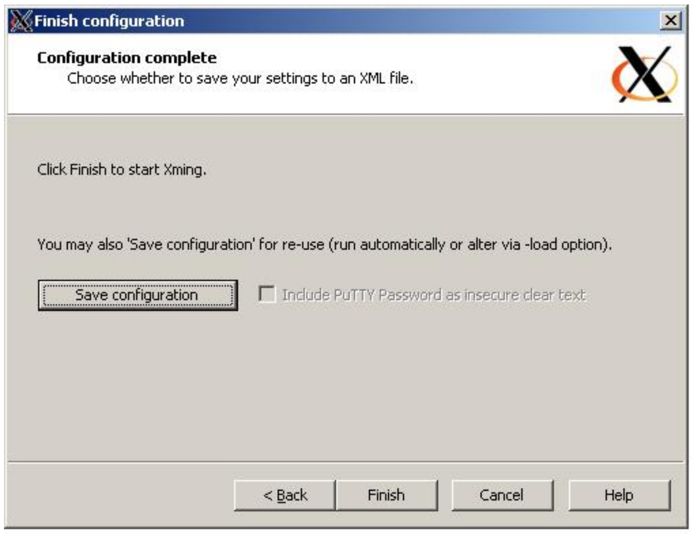
[✓] Clipboard
Start the integrated clipboard manager

[ ] No Access Control
Disable Server Access Control

Remote font server (if any)

Additional parameters for Xming

Additional parameters for PuTTY or SSH



Finish configuration

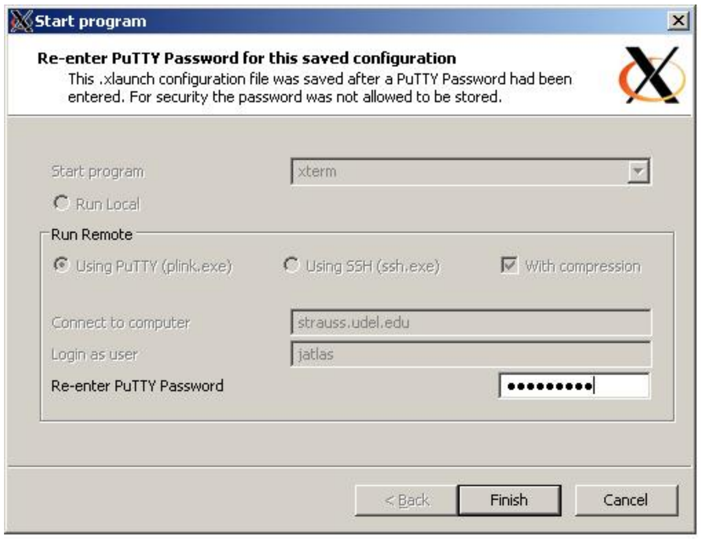
Configuration complete
Choose whether to save your settings to an XML file.

Click Finish to start Xming.

You may also 'Save configuration' for re-use (run automatically or alter via -load option).

Save configuration   [ ] Include PuTTY Password as insecure clear text

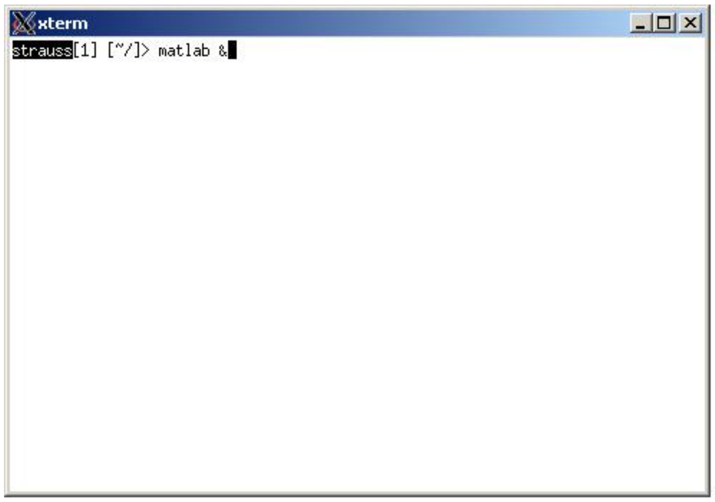
Make sure to Save Configuration here. After Finishing, open the configuration you just saved and you should get:



Start program

Re-enter PuTTY Password for this saved configuration
This .xlaunch configuration file was saved after a PuTTY Password had been entered. For security the password was not allowed to be stored.

Start program: xterm
( ) Run Local
Run Remote
(•) Using PuTTY (plink.exe)   ( ) Using SSH (ssh.exe)   [✓] With compression
Connect to computer: strauss.udel.edu
Login as user: jatlas
Re-enter PuTTY Password: ••••••••••



```
strauss[1] [~/]> matlab &
```

Running your Xlaunch configuration should give you a terminal window with a prompt on strauss. Now simply run matlab and firefox by typing them into the prompt:

strauss> matlab &

strauss> firefox &

NOTE: It may be slow, but eventually a matlab window will show up your display.

Pre-lab:

The linked PDF is available in web form at www.mathworks.com/help/techdoc/learn_matlab/bqr_2pl.html

Instructions

Step 1: Start up both MATLAB and a web browser (e.g. Firefox)

Step 2**:** Type some expressions into MATLAB and see their effect

The following exercises are designed to get you comfortable with the MATLAB environment. For now, we'll do all our work in the Command Window part of the MATLAB environment., which is just an interpreter window like the one we've been using in IDLE for the Python language.

Try

```
>> 2^4
```

You should get the following result:

```
 ans =
                     16
```

Try

```
>> area = pi * 4^2
```

Try

```
>> x = sin(pi/2)
```

Try

```
>> y = log(2)
```

Take a minute and play with the command window.

Step 3: Write a MATLAB function into a file (a so-called "M-file" file)

As in python, you can work in a separate editor window and save your python code in a file, ending with a .m for matlab:

- `myLab1.m`
- `calculateAnswer.m`
- etc...

To create a new .m file, choose "File->New->Blank M-file". This will open an Editor window where you can begin to type commands. Be patient—it may take several seconds for the new window to open!

Comments:

Comments in Matlab are signified by putting % at the beginning of the line (as opposed to python, which uses #). The header comments in your file should follow the commenting style we used in class.

Example:

```
% Description: calculates the area of a circle given the radius by
%              multiplying the radius squared and pi
% Input: a number (for the radius)
% output: a number  (representing the area of a circle)
% Examples: circleArea(5) = 78.5
function outputValue = circleArea(radius)
  outputValue = pi * radius^2
```

Here we're creating a function. The function's name is circleArea. The function takes, as an input parameter, the radius. In the function, we calculate the area of the circle by multiplying the radius variable squared (^2) by pi. pi is predefined for us by matlab to 3.1415… Next, we put the value calculated into the variable called outputValue. So outputValue is the variable holding the value we want returned from the function.

Notice how we return it: we say outputValue = circleArea(radius). That means that the value in outputValue is the value being returned from the matlab function circleArea.

> ***Note:*** ***<u>DO NOT</u>*** *copy and paste this text. Type it into the MATLAB editor. MS Word places special characters into its text that mess with MATLAB's brain.*

When you are finished, use the Save As command from the File menu to save the file with the name `circleArea.m`

Step 4: Test your function in the command window

In the command window, type

```
>>circleArea(5)
```

Note: You should get 78.5398 as the output. If you didn't, check your code.

Step 5: Now write the function cylinderVolume. cylinderVolume can be written by calling the function circleArea within your new function. cylinderVolume should take as input the height and the radius of the cylinder and should return the volume of the cylinder.

Save this to turn in later

Step 6: Summing using loops:

Matlab has loops, just like python. It has both while loops and for loops (and can do recursion as well). The while loops are quite similar to what you've seen in python. However, the for loops use a bit of a different syntax (although they accomplish pretty much the same thing.)

An example of a for loop (and a function) would be:

```
% Description: This function calculates the sum of a series
%     of integers from 1 to finish and returns that sum.
% Input: last integer in series of ints to be summed (scalar value)
% Output: sum of integers from 1 to finish (scalar value)
% Examples:
%     sumInts(5) -> 15
%     sumInts(10) -> 55
%     sumInts(12) -> 78

function sum = sumInts(finish)
     sum = 0;
     for current = 1:finish
           sum = sum + current
     end
```

Both Python and Matlab for loops iterate over an ordered set (lists in Python, a matrix in Matlab). In this example, Matlab for a parameter of 4, Matlab sees the expression 1:4 and produces the matrix [1 2 3 4]. In this case, the for loop starts by initializing the variable *current* to hold 1 and will end when the variable current is equal to the value inside of the ***finish*** variable. It increments by the default value of 1.

This would be equivalent to the Python for loop:

```
for current in range(1,4):

    sum += current
```

We signify the end of the Matlab loop with the word ***end.*** In Python we end the loop by ending the indented section.

So if the value in the variable *finish* was 4, current would first hold 1, then 2, 3, and finally 4. It stops when it is equal to the value inside of *finish*.

Note that matrix indices in Matlab start at 1, while Python list indices (like most languages) start at 0.

for loops are flexible. If nothing is specified, the variable automatically increases by 1 each time through the loop. However, you can change the amount we increase the loop by. For instance,

```
function printbyfives(endnum)
        for count = 1:5:endnum
                disp (count)
        end
```

This should count by fives. (Note that the increment value is in the center, unlike python, for which you'd say,

```
        for count in range(1,endnum+1,5):
```

One other useful tidbit of information: you can use the mod function to get the remainder. So, for instance,

```
>>mod(13,5)

        ans = 3

>>mod(13,2)

        ans = 1
```

Step 7: Write the functions sumOddInts.

sumOddInts will use a for loop to sum all the odd integers between two parameters called "start" and "finish" and return that value, e.g. sumOddInts(5,9) returns 21

Save this to turn in later.

Plotting Introduction:

The basic 2-D plot command is: plot (x,y) where **x** is a vector (an array of one dimension), and **y** is a vector (an array). Both vectors **must** have the same number of elements.

- The plot command creates a single curve with the **x** values on the abscissa (horizontal axis) and the **y** values on the ordinate (vertical axis).
- The curve is made from segments of lines that connect the points that are defined by the **x** and **y** coordinates of the elements in the two vectors.

Try it: Create two arrays (vectors) of numbers (note the lack of commas between elements in your array, which is different from python):

```
>> x=[1 2 3 5 7 7.5 8 10];

>>  y=[2 6.5 7 7 5.5 4 6 8];
```

Then plot the two vectors:

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

Now that you've got a line plotted, add a few things to make it more interesting. We can start with line specifiers. Line specifiers specify information about how the line will appear. So, for instance: plot(x, y, '- r +') will plot the above line with a solid line (-), the line will be red (r) and the markers on the line will be + (+). Read about line markers, axes, and more at:

http://www.mathworks.com/help/matlab/ref/linespec.html

Read about the use of "hold" in the "Getting Started" doc – it allows you to plot multiple items in one graph.

Try the following (and feel free to try any other combination you'd like:

```
>> plot(x,y)

>> plot(x,y,'r')

>> plot(x,y,'--y')   A

>> plot(x,y,'*')

>> plot(x,y,'g:d')
```

Okay, our plot is looking pretty colorful. But what now? We probably want to save this, and maybe we want to save it in a way we can use it in, say, a MS Word document. We can do this using the print option:

```
>> print <options> <filename>
```

Print options include:

- -dtiff – prints current figure to .tiff format (into filename.tiff)
- -deps – eps file
- -depsc – color eps file
- -djpeg – jpeg image
- -dpng – png image

So, for instance, if we wanted to save the current plot as a png image, we'd do:

```
>> print –dpng ourPlot.png.
```

We'll save ours as a jpeg file.

```
>> print –djpeg linePlot.jpg
```

Now open up **linePlot.jpg** (outside of Matlab – use the browser or other) to make sure it looks like your Matlab plot.

Save this to turn in later.

Part 2: Plotting a function

What if you wanted to plot a function? Say we wanted to plot:

$$y = 3.5^{-0.5x}\cos(6x) \text{ for } -2<=x<=4$$

First we'd create a vector, running from -2 to 4, with spacing of 0.01:

```
>>x = [-2:0.01:4]  %how does this differ from using range in Python?
```

Then we calculate a value of y for each value in x:

```
>>y=3.5.^(-0.5*x).*cos(6*x)
```

And then we'd plot it (adding the line specifiers of your choice):

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

Add a title, a legend, and an x and y label. Then save this to a jpg file as **plottingCos.jpg**.

Save this to turn in.

To Turn In:

- Matlab files:
  - CircleArea
  - SumOddInts
  - linePlot
  - plottingCos.jpg