PHYS 3300 WSU Spr 14 32762

Jump to Today

General Information

Class Times	SL 220, 10:30-11:20 am, MWF
Required Texts	**Required:** Introductory Computational Physics (Klein and Godunov) ISBN 0-521-82862-7 An Introduction to Computational Physics (Pang) ISBN 978-0-521-53276-1 **Recommended:** Computational Physics (Giordano and Nakanishi) ISBN 0-13-146990-8
Instructor	John Armstrong
Office Hours	SL 205, 11:30 AM - 12:30 PM MW or by appointment
Email	jcarmstrong@weber.edu (mailto:jcarmstrong@weber.edu)
Web	http://www.weber.edu/jcarmstrong (http://physics.weber.edu/armstrong)
Phone	801.626.6215

Course Description

This course extends the computational skills developed in PHYS 2300 to address a wider range of problems in modern physics. Students will explore the limits of computational methods and develop techniques suited to high-performance computing. Applications may be chosen from nonlinear dynamics, astrophysics, condensed matter physics, and quantum mechanics. ***Prerequisites:*** PHYS 2220 and PHYS 2300.

Course Goals

PHYS 3300 builds on the introductory programming and simulation skills you learned in PHYS 2300. This course is structured around four central themes: Tools, Numerical Methods, Classes of Problems, and Parallel Systems. We'll train you in some of the "tools of the trade" for computational physics, cover some more advanced numerical techniques, deal with some common classes of problems, and extend your experience into high-performance computing. After this course, you will:

- Have a strong foundation in a variety of computational and analysis tools
- Learn methods for designing and analyzing a host of numerical problems in physics
- Know how to evaluate and compare a range of numerical techniques applicable to different classes of problems
- Have experience using an industry standard high-performance computing environment

This is a hands-on course. Come prepared to work on problems in class.

Programming Language

There is no "official" language for this course. Everything we will be doing can be developed in python, java, C/C++, FORTRAN, and many other languages. However, you will be learning Unix, "shell" scripting, python for plotting, and possibly other programming tools. Also, you should be aware that if computing becomes part of your career, you will be required to learn languages as you go along. Other than specifics of syntax, they are really all the same anyway! I will be able to help you debug your code regardless of the language you choose, but I will rarely have you turn in your program code. In this course, the results are what matter. How you get there is up to you.

Grading Policy

Your course grade is computed based on your assignments and your final project, according to the following breakdown:

Assignments	60%
Project	40%

Emergency Closure

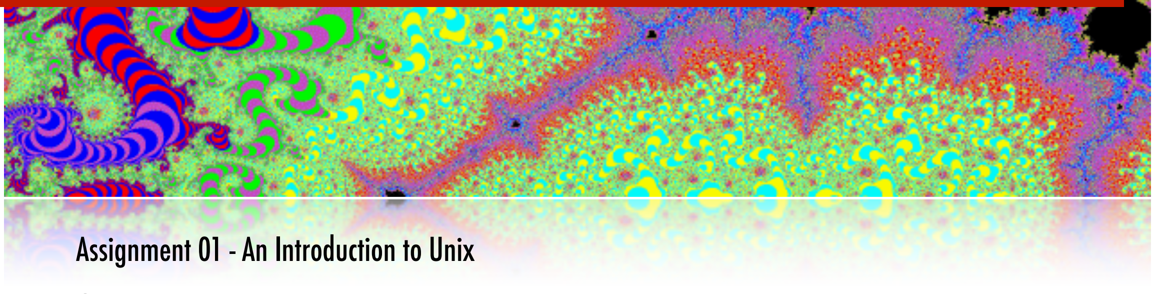
In the event of a campus emergency or campus closure, information will be posted to canvas or to the main University web site as soon as it is available. As always, your safety is the primary consideration. Regardless of the official closure instructions, *if you feel it is unsafe to travel to campus due to weather or other contingencies, follow your own best judgement.* Detailed instructions for any missed classes due to emergencies will be posted to canvas.

Special Accommodations

Any students requiring accommodations or services due to a disability must contact Services for Students with Disabilities (SSD) in room 181 of the Student Service Center. SSD can also arrange to provide course materials (including this syllabus) in alternative formats if necessary.

Date	Details	
Mon Jan 6, 2014	Topic 1 - Introduction to Unix (https://weber.instructure.com/courses/288302/assignments/1369545)	10:30am
Mon Jan 13, 2014	Topic 2 - Remote computing and job scheduling (https://weber.instructure.com/courses/288302/assignments/1369546)	10:30am
Fri Jan 17, 2014	Assignment 1 - Setting up your computing environment (https://weber.instructure.com/courses/288302/assignments/1370022)	11:30am
Mon Jan 20, 2014	Topic 3 - Data representation and visualization (https://weber.instructure.com/courses/288302/assignments/1369547)	10:30am
Mon Jan 27, 2014	Topic 4 - Interpolation Methods (https://weber.instructure.com/courses/288302/assignments/1369548)	10:30am
Wed Jan 29, 2014	Assignment 2 - Data representation and visualization (https://weber.instructure.com/courses/288302/assignments/1370023)	11:30am
Mon Feb 3, 2014	Topic 5 - Differentiation Techniques (https://weber.instructure.com/courses/288302/assignments/1369549)	10:30am
Wed Feb 5, 2014	Assignment 3 - Interpolation Methods (https://weber.instructure.com/courses/288302/assignments/1370024)	11:30am
Mon Feb 10, 2014	Topic 6 - Integration techniques (https://weber.instructure.com/courses/288302/assignments/1369551)	10:30am
Fri Feb 14, 2014	Assignment 4 - Differentiation and Integration (https://weber.instructure.com/courses/288302/assignments/1370026)	11:30am
Mon Feb 17, 2014	Topic 7 - Solving ordinary differential equations (https://weber.instructure.com/courses/288302/assignments/1369552)	10:30am
Mon Feb 24, 2014	Topic 8 - Damped Oscillator/Driven Oscillator (https://weber.instructure.com/courses/288302/assignments/1369553)	10:30am
Fri Feb 28, 2014	Assignment 5 - Oscillators and Chaos (https://weber.instructure.com/courses/288302/assignments/1370027)	11:30am
Mon Mar 3, 2014	Topic 9 - Boundary Value Problems (https://weber.instructure.com/courses/288302/assignments/1369558)	10:30am
Mon Mar 10, 2014	Spring Break (https://weber.instructure.com/courses/288302/assignments/1369560)	10:30am
Mon Mar 17, 2014	Topic 10 - Boundary Value Problems (https://weber.instructure.com/courses/288302/assignments/1369559)	10:30am
Fri Mar 21, 2014	Assignment 6 - Boundary Value Problems	11:30am

	(https://weber.instructure.com/courses/288302/assignments/1370028)	
Wed Apr 23, 2014	**Present projects (https://weber.instructure.com/courses/288302/assignments/1369586)**	11am



Assignment 01 - An Introduction to Unix

Our first task is to train you to use the tools typically ...

A Unix Tutorial

space.weber.edu/unixtut

“Shell” scripting

how to combine unix tasks

Remote computing

logging in, creating a work area, submitting a job

Your tasks

Log in, create your work area, copy test code, run code on cluster, plot results

You can access all of the tutorials at http://baldboybakery.com/courses/phys3300/index.html

Assignment 3 : Interpolation Methods
PHYS 3300

1. Write a program that implements the first order, linear interpolation method. Your program is designed to test this interpolation on the following functions:

$$\begin{aligned} f(x) &= sin(x^2) \\ f(x) &= e^{sin(x)} \\ f(x) &= \frac{0.2}{(x-3.2)^2+0.04} \end{aligned}$$

   Keep in mind the following information for your top-down design:

   - The user will select the equation to test
   - The user will decide at how many points will be evaluated, over what range, and at what value of x the function will be interpolated
   - The program will output the interpolated value of the function
   - The program will output the actual value of the function
   - The program will output the percent uncertainty between the two

2. Write a program that implements an n-point Lagrangian interpolation. In addition to the considerations above, the user will select the value of n.

3. Compare the results of these methods. Which method gives the best accuracy? What happens to the Lagrangian method when $n = 1$? When n is large?

4. Now we turn to “real data”. The text file *balloon_sounding.txt* (available on our web site) contains atmospheric sounding data (temperature, pressure, wind direction, and wind speed as a function of height). However, there are “missing” data, indicated by 99999. Modify your program to read input data from a file. Interpolate the missing values for temperature, wind speed, and direction using both methods, the linear and Lagrangian, that you prepared above. Then plot the results of the temperature, wind speed, and direction against height, comparing the data, the linear method, and the Lagrangian method. Some things to consider:

   (a) Typically, for atmospheric data, we plot the height on the y axis, and the other parameter on the x axis. However, for the interpolation, the parameter (i.e. temperature) is still the dependent variable.

   (b) You should end up with three plots, one for temperature, one for wind speed, and one for direction. Each plot should have the linear interpolation, the Lagrangian interpolation, and the original data plotted on it.

5. Comment on the two methods in this “real world” application. Which method worked best for which parameter?

Assignment 3 : Differentiation and Integration ***REVISED***

PHYS 3300 - Spring 2008, **Due Feb 21st, by 5pm**

1. **Numerical accuracy of differentiation and integration methods:** Write a program that calculates the derivative of $f(x)$ using the 2-point, 3-point, and 5-point formulas. Construct a plot of the 2-, 3-, and 5-point derivatives to compare the accuracy of the three using the function $f(x) = sin(x)$ with 101 uniformly spaced points from 0 to 2π. Using Simpson's rule, compute the integral of the result *for each case* and compare these to $\int_0^{2\pi} cos(x)dx$. What are the results for 51 points?

2. **Application of differentiation methods:** Planck's equation for a blackbody spectrum relates the spectral radiance, I, of a source (in units of $J\ m^{-2}\ s^{-1}$) to wavelength, λ, and temperature, T:

$$I(\lambda, T) = \frac{2hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda kT}} - 1} \tag{1}$$

   where h is Planck's constant and c is the speed of light. This means that each value of temperature results in a unique Planck spectrum. Using your 5-point method for differentiation, determine the relationship between the *peak* wavelength (that is, the wavelength where the Planck function is a maximum) and the temperature.

3. **Monte-Carlo integration:** When Simpson's Rule fails, it is possible to estimate the function, $f(x)$, from a sequence of random numbers and integrate the results. For instance, the integral is the width of the integration interval times the function's average value:

$$\int_a^b f(x)dx = (b-a)\langle f \rangle. \tag{2}$$

   We can estimate the integral by generating a set of random points over the interval to determine an estimate of the function's average value,

$$\int_a^b f(x)dx \approx (b-a)\langle f \rangle_N, \tag{3}$$

   where the estimate of the function's average is

$$\langle f \rangle_N = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{4}$$

   with x_i being a random sequence of values between a and b, and N the total number of points used to estimate the average.

Problem A: Using this approach, develop an algorithm to compute the integral

$$I = \int_0^1 e^x dx. \tag{5}$$

Your program should take only N as the input and return N and the integral's result. Do this for N = 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000. Make a plot of the integral's value on the y axis as a function of N on the x axis. Also, compare this to the function's exact integral $e - 1$.

Problem B: The beauty of *any* Monte Carlo method is its parallel nature: we can do additional estimates of the integral on separate machines at the same time. Using the *waradmiral* cluster, generate 100 estimates of Eq. (5) with N = 1000 simultaneously. Do this again for N = 10,000. What is the average value and standard deviation of the results in each case? How do these averages compare to the exact value of $e - 1$? Recall your grid submission tutorial. If N is the same for all cases, you should be able to submit the same script 100 times (for 100 sets of random numbers). However, you'll need to use some of your unix savvy to compile the results. For example, if your *script.csh.0xxxxx* files have the last line as the integral's value, you can use *tail* to compile the results:

```
$ tail -1 script.sh.o* > compiled_results.txt
```

and compute the averages and standard deviations from that file.

Problem *B* is for *Bonus*: Evaluate the 9-dimensional integral

$$I = \int_0^1 \cdots \int_0^1 \frac{da_x da_y da_z db_x db_y db_z dc_x dc_y dc_z}{\left(\vec{a} + \vec{b}\right) \cdot \vec{c}} \tag{6}$$

Submit at least 100 jobs to the cluster, evaluating the integral for N = 10,000, 100,000, and 1,000,000. What is the average and standard deviation of the results in each case?

Assignment 4 : Methods for solving ODEs

PHYS 3300 - Spring 2008, **Due Feb 28th, by 5pm**

In this problem, you will design a program to test various types of differential equation solvers. We want to test Euler, Leap Frog, and Runge-Kutta algorithms, so it would be easiest to design a code where one can specify which algorithm to use. Write a program that calls subroutines for these methods that can be used to solve a set of ordinary differential equations for an arbitrary number of dynamical variables. Use it to perform the following:

1. Consider the simple harmonic oscillator:

$$\ddot{x} + x = 0 \tag{1}$$

   where $x(0) = 1$, $\dot{x}(0) = 1$. Solve this equation numerically for $0 < x < 7$ using each of the methods described above with step sizes 1.6, 0.8, 0.4, 0.2, and 0.1.

2. Make a phase diagram (that is, plot x vs. $\dot{x}$) for each solver method for each step size. Each integrator has a maximum step size beyond which it is unstable. Roughly locate this step size in each case and label the graphs demonstrating where each method fails.

3. A damped, driven harmonic oscillator can be written as

$$\ddot{\theta} + q\dot{\theta} + sin(\theta) = bcos(\omega_0 t) \tag{2}$$

   Use your most accurate solver to reproduce the phase diagrams in Pang, Figures 3.2 and 3.3. Generate these plots by plotting one point out of every 10 (to get 1,000 points for 10,000 time steps).

Assignment 5 : Gravitational N-body Problem

PHYS 3300 - Spring 2008, **Due March 19th, by 5pm**

1. Write a program to solve the gravitational N-body problem for our solar system ($N = 10$, 8 planets, one dwarf planet, and the sun) using the N-squared, brute force method and leapfrog as the ODE solver. Download the initial conditions from our web site. The initial positions are given in astronomical units (1 AU $= 1.5 \times 10^{11}\ m$) and the velocities are given in AU/day. Treat the sun as a massive particle with initial position (0,0,0) and initial velocity (0,0,0) (but don't expect it to stay there!). Choose a time step short enough to resolve the orbit of Mercury and long enough to do at least one complete orbit of pluto. Generate plots of x vs. y and x vs. z to show the time evolution of your orbits.

2. Invent your own system to simulate. This could be a binary star system (with planets!), globular clusters in the galaxy, or a known extrasolar planetary system (see the Extrasolar Planet Encyclopedia at http://exoplanet.eu/). Make sure to choose a timestep short enough to resolve your fastest particle and long enough to explore the motion of the slowest. Again, make plots of x vs. y and x vs. z to show your system at work.

3. **Bonus:** Modify your code to allow for a large N in the case of test particles (that is, where you can assume their masses are equal to 0). Create a system of your choice to test the code and make a movie of the result using IDL.

Assignment 6 : Solving Systems of Linear Equations

PHYS 3300 - Spring 2008, **Due March 27th, by 5pm**

1. Write a program that can solve an arbitrary number of linear equations that can be expressed in the matrix form

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

Your algorithm should be as generic as possible. That is, it should allow the user to provide, as input, the coefficients of the linear equations and the values of the resulting vector, **b**, for an arbitrary number of equations. One way of doing this allows the user to provide an input file that includes the number of equations on the first line and the coefficients of the form

$$\begin{array}{cccc} 3 & & & \\ a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ a_{31} & a_{32} & a_{33} & b_3 \end{array} \tag{2}$$

2. Test your program on the following problem. A network of resistors and batteries in an electrical circuit is described by the following linear set of equations:

$$\begin{aligned} -9.2i_1 + 12.5i_2 &= 18 \\ 12.5i_2 - 2.8i_3 + 3.5i_4 &= 16 \\ 1.8i_3 - 16.4i_5 &= -16 \\ i_1 + i_2 + i_4 &= 0 \\ i_3 + i_4 - i_5 &= 0 \end{aligned} \tag{3}$$

Write a program to solve the system of equations and return the five currents i_1, i_2, i_3, i_4, and i_5. What results do you get? Note that it is *not* necessary that the currents all be positively valued; a negative value simply implies that the current goes in the opposite direction from what was assumed in developing the set of equations.

3. Now let's test the robustness of your algorithm. Based on the method you used, predict the following:

   (a) How will the time it takes for your algorithm to run scale with N, the number of equations?

   (b) Based on your allocation of array resources in your program, what is the maximum number of equations you can calculate?

4. Use your code to test your predictions. Make a graph of the time it takes your algorithm to complete (y-axis) vs. the number of equations. Do this for at least 10 different sets of linear equations with different N.

   HINT: To really have fun with this, you want to use large values of N. You can generate a test matrix that will have a viable solution if all of the coefficients a_{ij} and b_i have unique values. Also, to compute elapsed time, try something like this: **http://exampledepot.com/egs/java.lang/ElapsedTime.html**

5. List some ways you might improve your algorithm.

UNIX Tutorial Eleven

An introduction to *IDL* (Interactive Data Language)

One of the most important aspects of scientific computing is analyzing the results of your calculations. There are a number of analysis packages available, from spreadsheet programs like *Excel* and *Numbers*, to symbolic computing languages like *MATLAB* and *Mathematica*. One powerful, versatile, and popular package is the *Interactive Data Language* or *IDL*. *IDL* is commonly used in industry, academia, and is particularly popular with astronomers and planetary scientists. Most universities will have licenses for *IDL*, and a student edition is available for $89 from ITT. At the most basic, it is software that allows you to plot large amounts of data quickly. However, you can also program applications complete with GUI interfaces.

Setting up an *IDL* environment

IDL on our systems is run as a Unix program through *X11*. *X11* is another terminal emulation program that can be found in the /Applications/Utilities folder. Before you can use *IDL*, you need to set up your user environment by setting a few environment variables. On our systems, the information you need is located in the file **`/etc/bashrc`**. From your Unix prompt, copy this file to your home directory and rename is **`.bashrc`** (yes, it needs the leading .). For example,

```
% cp /etc/bashrc ~/.bashrc
```

Will copy the file to the correct location in your home directory. The leading . makes the file hidden, so it won't show up unless you use the **`-la`** option to **`ls`**, like

```
% ls -la
```

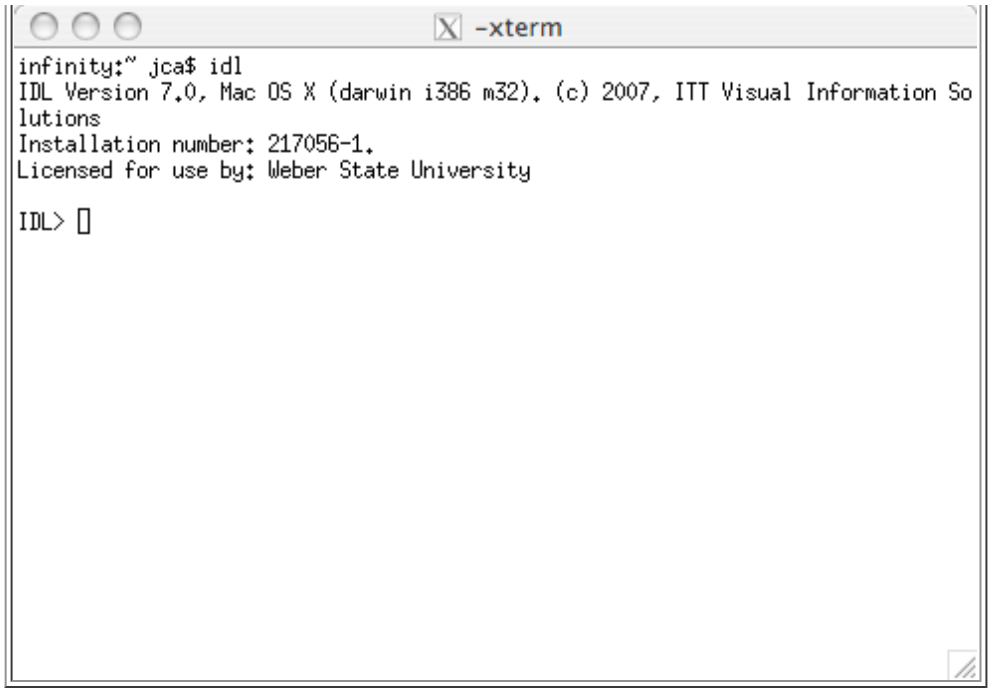
If you have already started *X11*, then quit the emulator and restart it. Alternatively, you can type

```
% source ~/.bashrc
```

which will load this file into memory. You can now start IDL. From the command prompt from within *X11*, type

```
% idl
```

This will launch the *IDL* package and give you the *IDL* prompt, something like this:



```
infinity:~ jca$ idl
IDL Version 7.0, Mac OS X (darwin i386 m32). (c) 2007, ITT Visual Information So
lutions
Installation number: 217056-1.
Licensed for use by: Weber State University

IDL> 
```

Simple calculations in IDL

The *IDL* command line operates much like the terminal, allowing you to execute commands and perform operations. For example, you can assign variables:

```
IDL> x = 1
IDL> print,x
```

Will print the value 1. With these, you can use *IDL* like a calculator:

```
IDL> y = 7
IDL> print,x+y
```

Will print the value 8. If you want to know the type of variable you are dealing with, you can use the `help` command:

```
IDL> help,x
```

Should give you output like this:

```
X               INT      =      1
```

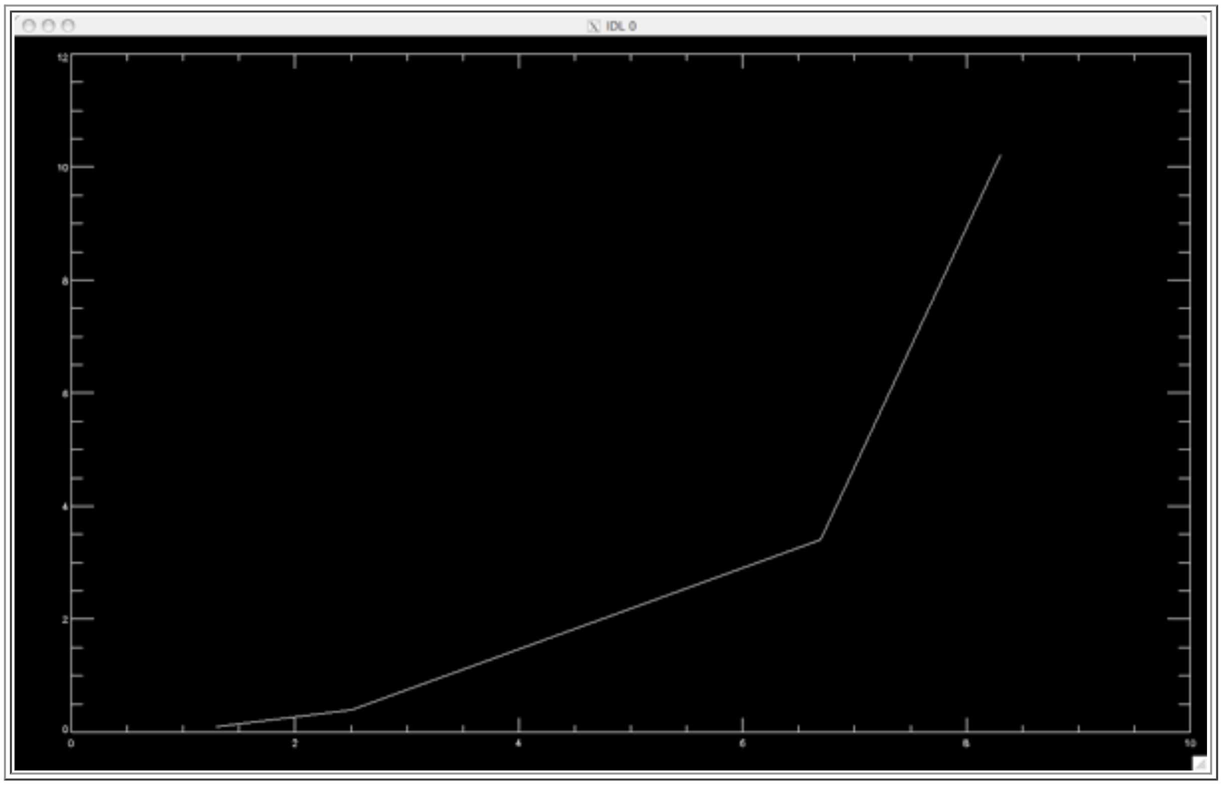
Telling you that X is an INT type with 1 element. For general help with IDL, type a `?` to get the help browser.

Plotting with *IDL*

IDL has standard 2-D and 3-D plotting packages, with PLOT being the most common. You can plot data as well as function. To plot some data, first generate arrays with some numbers to plot, specifying arrays with the [] notation:

```
IDL> x = [1.3, 2.5, 6.7, 8.3]
IDL> y = [0.1, 0.4, 3.4, 10.2]
IDL> plot,x,y
```

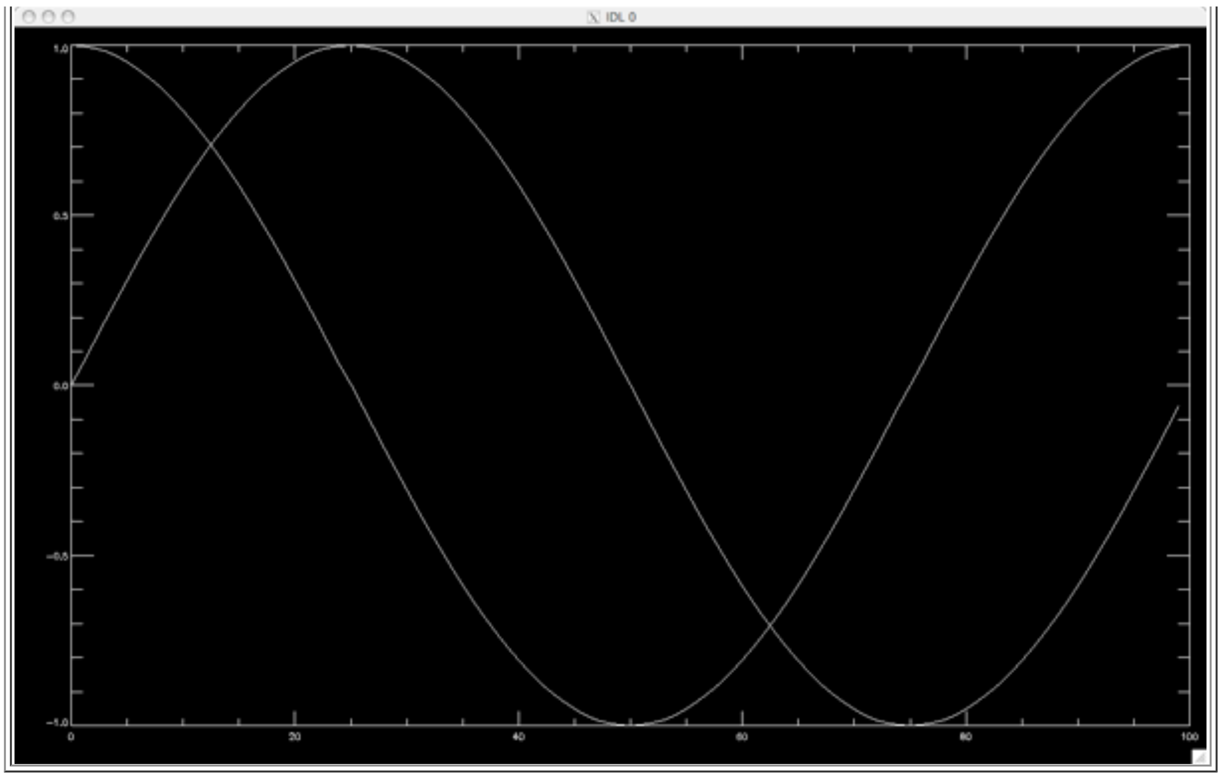
You should get a plot window that looks like



IDL also allows you to plot common functions like **SIN** and **COS**. First, generate a list of X values using the **FINDGEN()** routine, plot the **SIN** function, and then overlay the **COS** function, like this:

```
IDL> x = 2*!PI/100 * FINDGEN(100)
IDL> plot,x,sin(x)
IDL> oplot,x,cos(x)
```

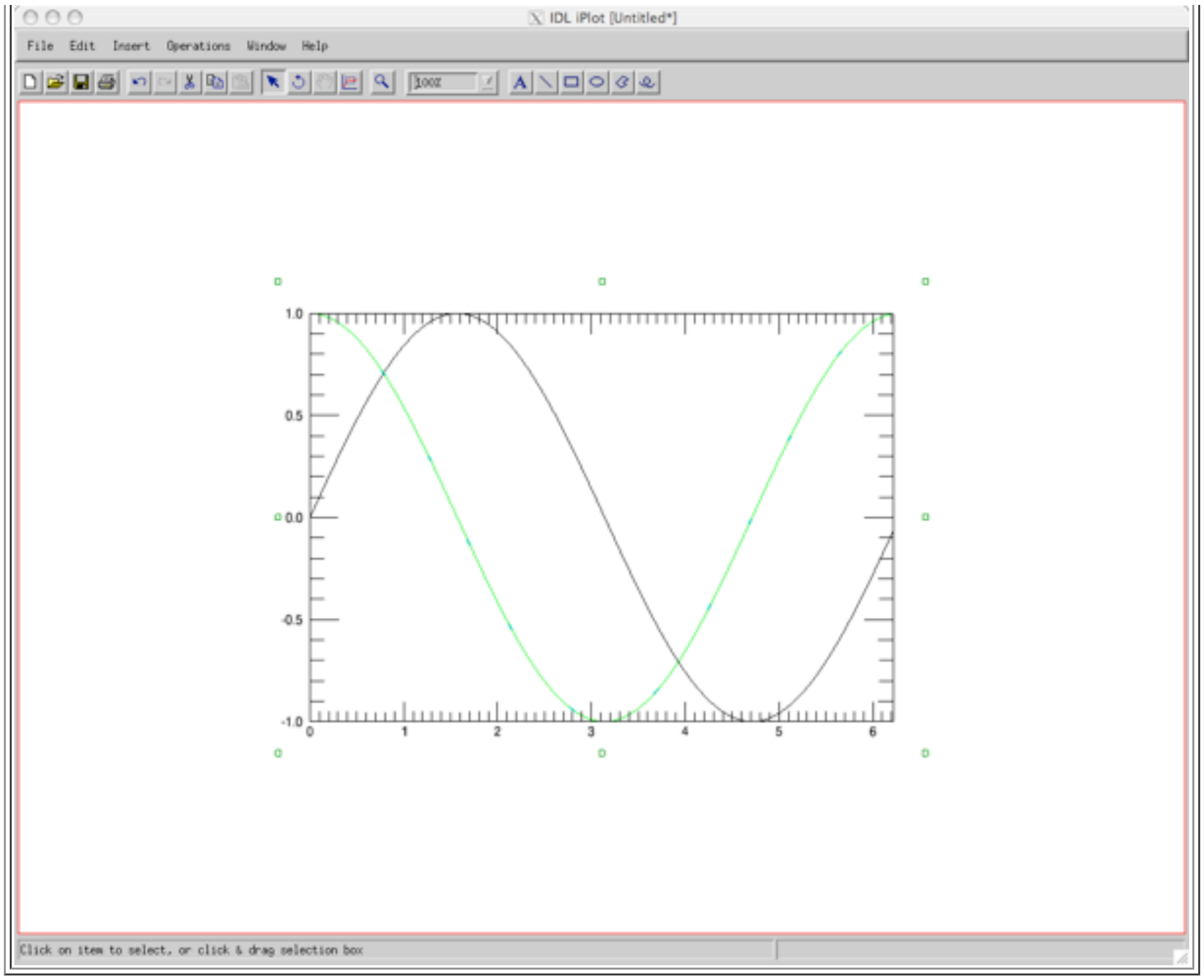
Which should give you two overlaid plots, like this:



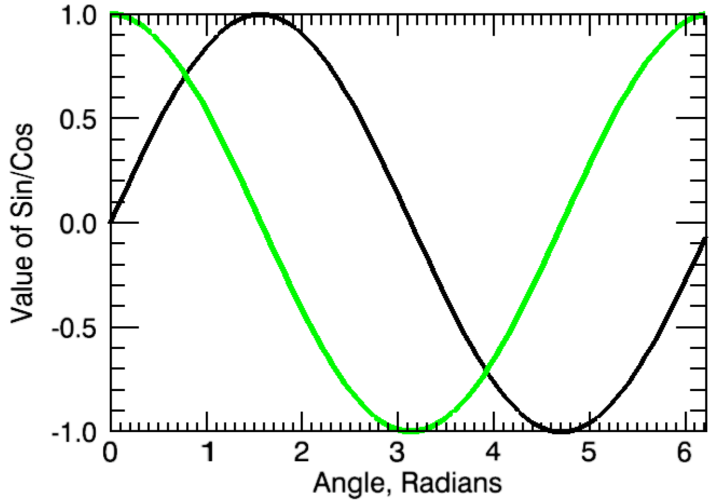
IDL has an interactive plotting tool called **IPLOT** that works in a similar way, using

```
IDL> x = 2*!PI/100 * FINDGEN(100)
IDL> iplot,x,sin(x)
IDL> iplot,x,cos(x),/overplot,color=[0,255,0]
```

Which should give you something that looks like the figure below.


IDL iPlot [Untitled*]
File Edit Insert Operations Window Help
100%
1.0
0.5
0.0
-0.5
-1.0
0
1
2
3
4
5
6
Click on item to select, or click & drag selection box

IPLOT allows you to modify components of the plot (through the Edit->Properties pulldown menus) and save the results. With a little manipulation, you can get a more attractive plot, like this:



Exercise 1

Plot the above functions using **IPLOT** and modify the plot settings to generate a labeled figure with larger fonts like you see above

Using procedures in *IDL*

Much like Unix itself, you can save *IDL* commands to a file and run the resulting procedure file. For example, to generate the same plots we made above, we can write those commands to a procedure file called **plotwaves.pro** and save it in your working directory (where you launched *IDL*). In order to use them as a procedure, the file needs to look like this:

```
pro plotwaves
  ; This is a procedure file to plot sin and cos in IPLOT
  x = 2*!PI/100 * FINDGEN(100)
  plott,x,sin(x)
  oplot,x,cos(x)
end pro
```

Here, the **;** symbol indicates a comment line, line **//** in java or C++. Once you have this file created, you can compile and execute it from the *IDL* prompt, like so:

```
IDL> .run plotwaves
IDL> plotwaves
```

As long as the file has the `.pro` extension, the file name is the same as the procedure name, and it is located in the working directory, you should see the same `IPLOT` window you saw before.

Configuring your *IDL* procedure library

Most *IDL* users create a directory to hold all their procedure files so they are easy to keep track of, and then train *IDL* to find them and use them. To do this, you need create an *IDL* startup file which loads the location of your personal library.

To do this, make a directory in your home directory to hold the library:

```
% cd ~/

% mkdir idllib

% cd idllib
```

Now copy the template startup file to this location

```
% cp /usr/local/lib/idllib/idl_startup.pro ./
```

Open this file in a text editor, and you should see something like this

```
print,"Loading Paths..."
!path='/usr/local/lib/idllib:'+!path
!path='/usr/local/lib/idllib/krctool:'+!path
```

These are paths to libraries located on your computer. Now, to add your library location, add the following line to the end of the file so it looks like this:

```
print,"Loading Paths..."
!path='/usr/local/lib/idllib:'+!path
!path='/usr/local/lib/idllib/krctool:'+!path
!path='~/idllib:'+!path
```

Save this file. To convince *IDL* to load your library path, we need to make one more change. Edit the file ~/.bashrc (you will need to use File->Open Hidden in *TextWrangler*). Make sure that the line

```
export IDL_STARTUP=/usr/local/lib/idllib/idl_startup.pro
```

is changed to

```
export IDL_STARTUP=~/idllib/idl_startup.pro
```

Move your procedure file **`plotwaves.pro`** to your **`idllib`** directory. Now, exit out of *IDL* (using the **`EXIT`** command). Then, refresh the **`.bashrc`** in memory by typing

```
% source ~/.bashrc
```

and start *IDL* again. You should be able to execute **`plotwaves`** straight away,

```
IDL> plotwaves
```

since *IDL* now knows where to look.

In general, when managing procedure files for *IDL*, you should

- Always store your procedure files in your **`idllib`** directory
- Name your files with the same name as your procedure.

Reading and plotting data with IDL

Reading data in *IDL* is similar to other programming languages, in the sense that you need to open a file, read it into an array or other variable, and then close it. However, this can be done with just a few lines of code. For example, the procedure **`readdata`**, below, will read in a file with 2 columns and 10 rows of data (specified as a *command line argument*) and print the results.

```
pro readata,file
; procedure for reading and printing a data file
  lun=1
  cols=2
  rows=10
  openr,lun,file
  data = fltarr(cols,rows)
  readf,lun,data
  close,lun
  print,data
end pro
```

Here, **`lun`** is the *logical unit number* which is assigned to the file using the **`openr`** command. We then define a floating point array with the proper number of **`rows`** and **`cols`** to **`data`** using **`fltarr`**. Then, we read the data in with **`readf`** and close the file with **`close`**.

You run this by storing it in your library folder, compiling, and executing the procedure in *IDL*:

```
IDL> .run readdata
IDL> readdata,"test.txt"
```

using a file called **`test.txt`** in your working directory. To plot the data, you can add an **IPLOT** command to the procedure above to plot the first and second columns of the data file:

```
pro readata,file
; procedure for reading and printing a data file
  lun=1
  cols=2
  rows=10
  openr,lun,file
  data = fltarr(cols,rows)
  readf,lun,data
  close,lun
  print,data
  iplot,data(0,*),data(1,*)
end pro
```

Exercise 2

Create a data file with 10 rows and 2 columns and read it into *IDL* and plot it using the example above.

Exercise 3

There is a pre-developed extension of this procedure for generic plotting called **iplot2d** located in the system library, **/usr/local/lib/idllib**. This procedure allows you to plot a file that may have a header with any number of columns and rows. For this exercise:

- Dowload this data file exoplanet.txt
- Plot the planet period (2 column, y axis) vs. the planet semi-major axis (3rd column x-axis) using the **iplot2d** procedure (see below).
- Modify the properties to plot points instead of lines, increase the font size, and label the plot
- Save and print the results

The **iplot2d** procedure can be executed by typing

```
IDL> iplot2d,"exoplanet.txt",cols=5,hskip=2,xx=2,yy=3
```

Where **xx** is the x column to plot, **yy** is the y column to plot, **cols** is the number of columns, and **hskip** is the number of header lines to skip.

home

M.Stonebank@surrey.ac.uk October 2001