Computational Chemistry Laboratory I (CBE 60547)

Prateek Mehta, William F. Schneider

2015-01-22 Thu

1 Introduction to Linux

All computers are run by an `operating system`. What you see, on your phone, tablet, or laptop, is a `graphical user interface` that runs on top of that operating system. Far and away the most common operating systems are based on Unix. MacOS is a variant of Unix, for instance. Web serves, computational servers, . . . , run another variant named Linux. Serious computational work, such as computational chemistry, is done essentially exclusively on Linux computers.

The `command line` is the lowest level that one commonly interacts with a computer at. Phones and laptops hide the command line from us. On your Mac laptop, you can get to it through the `terminal`.

1.1 Logging onto the CRC

For consistency, we will use the Linux computers hosted by the Notre Dame Center for Research Computing (CRC) for our computational work. The CRC hosts a number of "front-end" computers that we can access and interact with directly, through the internet. Hinding behind those front-ends are racks and racks of computers ("nodes"), lashed together with high-speed connections, that are access through a queue and are used for really serious calculations. The front-ends and the nodes all share the same, network-based file system ("afs", `https://www.openafs.org/`), so your files are the same every. Every user has a home directory. For instance, mine is /afs/crc.nd.edu/user/w/wschnei1.

One way to access the CRC is through a terminal window on your computer (`putty` on a Windows computer, `terminal` on Mac/Linux User) and login using your regular ND credentials.

```
1 ssh -Y wschnei1@crcfe01.crc.nd.edu
```

This method works but gives relatively limited functionality. An alternative approach provided by the CRC and that is a bit more feature rich is to use fastX through a browser window. Go to `https://crcfe01.crc.nd.edu` and log in using your ND credentials. Then choose a single xterm session to get started.

1.2 Common Linux Commands

At the Linux command line you will be working interactively with a `=shell=`. The shell provides a lot of built-in features, is programmable, and gives access to There are many different shells, but

the most popular today is bash. Here is a list of common linux commands, along with an example of their output. Try typing these in your xterm window.

- `ls`: List the contents of a directory

```
ls
```

- `pwd`: Get the path of the current directory

```
pwd
```

- `mkdir`: Create a directory

```
mkdir example-directory
ls
```

- `cd`: Change directory

```
cd example-directory pwd cd .. # move 1 directory up pwd
```

- `cp` Copy files

```
cp file1 file2
```

- `mv`: Move files

```
mv file1 file2
```

- `rm`: Remove files

```
rm filename
```

- `quota`: Check the amount of space allocated to you by the CRC.

```
quota
```

- `printenv`: Show the ENVIRONMENT VARIABLES that control how t

1.3 Obtaining lab notes

The lab notes and homeworks can be obtained from the course website. They are also conveniently stored in a git repository. To obtain the files just type in

```
git clone https://github.com/wfschneidergroup/computational-chemistry
```

You can periodically update your copy of the notes by changing into the course directory and running the command `git pull`, i.e.,

```
cd computational-chemistry
git pull
```

2 The very least you have to know about GNU Emacs

For the purposes of this course, we will use GNU Emacs as our text editor, and in particular Emacs org-mode. Getting started with it can be a little challenging, but learning how to use it offers many benefits, which we shall soon see. If you really, really want to use another editor, you are free to do so, but it is likely that you will lose some customized emacs features which will help you with this course.

Here are a few things to get started.

- Starting Emacs. To start emacs type `emacs &` on the command line. This will launch emacs with the jmax starter kit (https://github.com/jkitchin/jmax). You can add your own customizations when you start learning more.
- The first screen you will see is called the scratch buffer. It is mainly for doing temporary things that are not generally required to be saved.
- Emacs is built around modifier keys, e.g., <Control>, <Alt>, <Shift>, <Esc>. These keys allow you to give instructions to emacs, e.g. keyboard shortcuts to open a file, close a file, run a command, etc. The commands are usually are in the form of <modifier key><some letter>.
- To create a new file, click on the file menu and select `visit new file`. This will prompt you to enter the filename at the small window at bottom of the screen. This small window is called the mini-buffer. Create a file `test.org`.
- An alternate way to open a file (or create a new file if it does not exist) is to type `Control-x Control-f` (or `C-x C-f` in Emacs notation). It should bring up the same prompt to enter the filename in your mini-buffer.
- Once your file is open, you can try typing something in it. Now save your file. You can do this from the file menu or using the command, `C-x C-s`.
- Cut/Copy/Paste. The biggest difference between Emacs and other text editors is how to cut, copy and paste text. We can use the edit menu to do this, or use the short cuts,
  - `C-w`. Cut selected text.
  - `M-x`. Copy selected text. (Here M is the `Alt` key for windows/linux users, the `Command` key for Mac users)
  - `C-y`. Paste cut/copied text.
- To close emacs, you can click on the 'X' sign on the corner of your screen or type, `C-x C-c`.
- Now that you are back at your terminal, use emacs to open the notes for today's lab. Change into the the directory you just cloned, and type `emacs lab1.org &`. This is the org-mode file that was used to create the pdf. All the commands that you just ran in the terminal were run inside this document!
- Press <TAB> to expand the headings and see what they contain. Navigate to this section of the document.
- Finally, we will consider how to run an emacs command, namely one that will make the equations in the document readable. Type `M-x`. Now type `org-toggle-latex-overlays`.

Alternately, you can click on this link, `org-toggle-latex-overlays`. You should be able to see the Schrodinger equation below.

- $H\psi = E\psi$
- **Tip:** If you find yourself stuck somewhere, type <ESC> four times or type `C-g`.

3 Introduction to Python

Python is a programming language which we will use to solve many of the homework problems, especially density functional theory calculations using VASP (homeworks 5, homework 6, and probably your class project). It is therefore important that you familiarize yourself with using it. The numerical and plotting features in python are mostly similar to `MATLAB`, with a few subtle differences. I would recommend that you try to solve the first problem in homework 2 using python. A few examples are below, adapted (a fancy way of saying pretty much copied) from John Kitchin's example files. We won't cover all of these, but you can use them for reference. To execute a code block type `C-c C-c`.

3.1 Simple calculations

A good overview of basic python operators can be found at http://www.tutorialspoint.com/python/python_basic_operators.htm

Here are some simple examples

```
print 2+3
print 4-6
print 2*7
print 4.0 / 6.0
```

```
5
-2
14
0.666666666667
```

Division is a little tricky. Python distinguishes between integer division and float division. In the first line we have integer division, where the remainder is discarded and an integer is returned. If any number is a float (indicated by a decimal or because it is converted to a float) then a float is returned.

```
print 2/3
print 2./3.
print 2/3.
print 2/float(3)  # the float function casts the integer to a float
```

```
0
0.666666666667
0.666666666667
0.666666666667
```

```
print 2*3
print 2*3.0
```

```
6
6.0
```

We can also do powers with **

```
print 2**3
print 2**0.5
print 2^4    # Binary XOR operator!
```

```
8
1.41421356237
6
```

The modulus operator (%) divides the left hand operand by the right hand operand and returns the remainder.

```
print 5 % 4
print 5. % 4.
```

```
1
1.0
```

3.2 Formatted printing

http://docs.python.org/library/stdtypes.html#string-formatting-operations

We will usually want to print more than a number, e.g. some descriptive text and the number. We also will want to format numbers so we do not see 9 decimal places all the time. We use string formatting for that. Here are some typical examples.

In a string we can specify where to put numbers with positional arguments like {0}. That says take the first argument (python starts counting at zero) and put it in place of {0}.

```
a = 4.5 + 2
print 'The answer is {0}'.format(a)
```

```
The answer is 6.5
```

We can have more than one number to format like this.

```
a = 5**3
b = 23
print 'a = {1} and b = {0}'.format(b,a)
```

```
a = 125 and b = 23
```

Alternatively, we can use named arguments to specify the values. It is your choice which one to do. Named arguments require more typing, but are easier to understand.

```
a = 5**3
b = 23
print 'a = {ans0} and b = {ans1}'.format(ans0=a,
                                         ans1=b)
```

```
a = 125 and b = 23
```

To do formatting, we need additional syntax. We use {i:format} to specify how the value should be formatted. Here we show how to specify only three decimal places on a results. See this link for a lot more details of formatting strings.

```
a = 2./3.
print 'a = {0}'.format(a)
print 'a = {0:1.3f}'.format(a)
```

```
a = 0.666666666667
a = 0.667
```

3.3 Data types

Numeric types http://docs.python.org/library/stdtypes.html#numeric-types-int-float-long-complex

Strings http://docs.python.org/library/stdtypes.html#string-methods

3.3.1 lists/tuples

Lists and tuples are similar in that they are both sets of data. A list is delimited by [] (square brackets) and a tuple is delimited by () (parentheses). The difference between them is a list can be changed after it is created (it is mutable), but a tuple cannot (it is immutable).

```
# short list example
a = [1, 2, 3, 4] # a list
print a
print len(a)
print a[0] # first element
print a[-1] # last element
print a[3]  # also last element
print 2*a   # surprise!!!
```

```
[1, 2, 3, 4]
4
1
4
4
[1, 2, 3, 4, 1, 2, 3, 4]
```

We can create a list with the `range` command:

```
a = range(4)
print a

b = range(4,10)
print b

print a + b  # surprise again!!!
```

```
[0, 1, 2, 3]
[4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Note that algebraic/math operations are not defined for lists the way they are for Matlab. We have to use `numpy.array` for that, which we will see later.

```
# short list example
a = [1, 2, 3, 4] # a list
print a
a[1] = 56 # change the value of 2nd element
print a
```

```
[1, 2, 3, 4]
[1, 56, 3, 4]
```

Tuples are like lists except they cannot be modified after creation.

```
a = (1,2,3,4)
print len(a)
print a[0]
print a[-1]
a[1] = 56 #this is not allowed!
```

3.4 Conditional statements

conditional operators http://docs.python.org/library/stdtypes.html#comparisons

Python has the standard conditional operators for testing if a quantity is equal to (`=`), less than (`<`), greater than (`>`), lessthan or equal to (`<`) greater than or equal to (>=) and not equal (!=). These generally work on numbers and strings.

```
print 4 == 2.
print 'a' != 'A'
print 4 > 3
print 4 <= 3
print 'a' < 'b' # hmmm....
```

```
False
True
True
False
True
```

We use these conditional operators to determine whether conditional statements should be run or not.

```
a = 4
b = 5

if a < b:
    print 'a is less than b'
```

```
a is less than b
```

In this next example we use an `else` statement. Note the logic is not complete, if a=b in this case, we would get the statement "a is less than b" printed.

```
a = 14
b = 5

if a > b:
    print 'a is greater than b'
```

```
else:
    print 'a is less than b'
```

```
a is greater than b
```

Here is a more complete logic that uses `elif` to add an additional logic clause.

```
a = 4
b = 4
if a > b:
    print 'a is greater than b'
elif a == b:
    print 'a is equal to b'
else:
    print 'a is less than b'
```

```
a is equal to b
```

Finally, to illustrate that the first conditional statement that evaluates to True is evaluated, consider this example:

```
a = 4
b = 4
if a > b:
    print 'a is greater than b'
elif a >= b:
    print 'a is greater than or equal to b'
elif a == b:
    print 'a is equal to b'
elif a <= b:
    print 'a is less than or equal to b'
else:
    print 'a is less than b'
```

```
a is greater than or equal to b
```

3.5 Loops

http://docs.python.org/tutorial/datastructures.html#looping-techniques for while/break/continue enumerate, zip

```
for i in [0,1,2,3]:
    print i


for i in range(4):
    print i
```

```
0
1
2
3
0
1
2
3
```

3.6 functions

http://docs.python.org/tutorial/controlflow.html#defining-functions

We can define functions with the `def` statement, and specify what they `return`

```
def myfunc(x):
    return x*x

print myfunc(3)
print myfunc(x=3)
```

```
9
9
```

3.7 Classes and objects

http://docs.python.org/tutorial/classes.html

3.8 Modules

http://docs.python.org/tutorial/modules.html

The default Python environment has minimal functionality. We can `import` additional functionality from modules. The full standard library is documented at http://docs.python.org/library/. It is not likely you will use everything there, but it is helpful to be familiar with what is available so you do not reinvent solutions.

We import modules, and then we can access functions inthe module with the `.` operator.

```
# list contents of current directory
import os
for item in os.listdir('.'):
    print item
```

```
lab1.org
lab1.pdf
lab1.tex
```

You can import exactly what you need also with the `from/import` syntax

```
# list contents of current directory
from os import listdir
for item in listdir('.'):
    print item
```

```
lab1.org
lab1.pdf
lab1.tex
```

Finally, you can change the name of a module. This may be done for readability, or to shorten the amount of typing.

```
# list contents of current directory
import os as operating_system
```

```
for item in operating_system.listdir('.'):
    print item
```

```
lab1.org
lab1.pdf
lab1.tex
```

3.8.1 Some common standard modules

http://docs.python.org/tutorial/stdlib.html os, sys, glob, re

3.9 file I/O

reading, writing files http://docs.python.org/library/stdtypes.html#file-objects

3.10 Error handling

http://docs.python.org/tutorial/errors.html

Errors happen, and when they do they usually kill your script. Sometimes that is not desirable, and it is nice to catch errors, handle them, and keep on going. When errors occur in python, an Exception is raised. We can use `try/except` code blocks to try some code, and then respond to any exceptions that occur.

```
try:
    1/0
except ZeroDivisionError, e:
    print e
    print 'an error was found'
```

```
integer division or modulo by zero
an error was found
```

3.11 Numerical Python (Numpy)

3.11.1 The basics

http://docs.scipy.org/doc/numpy/reference/

```
import numpy as np
a = np.array([1,2,3,8])

print a*a          # element-wise operation
print np.dot(a,a)  # linear-algebra dot product
```

```
[ 1  4  9 64]
78
```

Numpy defines lots of functions that operate element-wise on arrays.

```
import numpy as np
a = np.array([1, 2, 3, 4])
print a**2
print np.sin(a)
print np.exp(a)
print np.sqrt(a)
```

```
[ 1  4  9 16]
[ 0.84147098  0.90929743  0.14112001 -0.7568025 ]
[  2.71828183   7.3890561   20.08553692  54.59815003]
[ 1.          1.41421356  1.73205081  2.        ]
```

```
import numpy as np
a = np.array([1, 2, 3, 4])
print a.min(), a.max()
print a.sum()  # sum of elements
print a.mean() # average
print a.std()  # standard deviation
```

```
1 4
10
2.5
1.11803398875
```

Below are some recipes for doing linear algebra and polynomials.

3.11.2 Linear algebra

`numpy.linalg` provides a lot of the linear algebra functionality we need. See http://docs.scipy.org/doc/numpy/reference/routines.linalg.html for details of all the things that are possible. For example, given these linear equations:

$x + y = 3$, $x - y = 1$

we can represent these equations in matrix form $Ax = b$ and solve them.

```
import numpy as npx
import numpy.linalg as la

A = np.array([[1, 1],
              [1, -1]])

b = np.array([3, 1])

print la.solve(A, b)
```

```
[ 2.  1.]
```

You might be familiar with the following solution:

$x = A^{-1}b$

We can also compute that:

```
import numpy as np
import numpy.linalg as la

A = np.array([[1, 1],[1, -1]])
b = np.array([3, 1])

print np.dot(la.inv(A), b)
```

```
[ 2.  1.]
```

Finally, we can do linear least squares easily. Suppose we have these three equations, and two unknowns:

$x + y = 3$,

$x - y = 1$,

$x - y = 0.9$

```
import numpy as np
import numpy.linalg as la

A = np.array([[1, 1],
              [1, -1],
              [1, -1]])
b = np.array([3, 1, 0.9])

[x, residuals, rank, s] = la.lstsq(A,b)
print x
```

```
[ 1.975   1.025]
```

3.11.3 Polynomials

numpy can do polynomials too. We express polynomials by the coefficients in front of the powers of x, e.g. $4x^2 + 2x - 1 = 0$ is represented by [4, 2, -1].

```
import numpy as np
p = [4, 2, -1]
print np.roots(p)
```

```
[-0.80901699   0.30901699]
```

```
import numpy as np
p = [4, 2, -1]
print np.polyder(p) # coefficients of the derivative
print np.polyint(p)
```

```
[8 2]
[ 1.33333333  1.          -1.          0.         ]
```

We can also readily evaluate polynomials at specific points:

```
import numpy as np
p = [4,2,-1]
print np.polyval(p,[0, 1, 2])
```

```
[-1  5 19]
```

Polynomials are very convenient functions to fit to data. the `numpy.polyfit` command does this, and returns the coefficients.

```
import numpy as np
x = [0, 2, 3, 4]
y = [1, 5, 7, 9]
p = np.polyfit(x, y, 1)
print 'slope = {0}\nintercept = {1}'.format(*p)
print p
```

```
slope = 2.0
intercept = 1.0
[ 2.  1.]
```

3.12 Scientific Python (Scipy)

http://docs.scipy.org/doc/scipy/reference/

`scipy` provides all the functionality we need for integration, optimization, interpolation, statistics, and File I/O. You should look up the link on integration. It might be useful for homework 2.

3.12.1 Solving Equations

Here is a typical usage for solving the equation $x^2 = 2$ for x. We have to define a function that is $f(x) = 0$, and then use the `scipy.optimize.fsolve` function to solve it with an initial guess.

```
from scipy.optimize import fsolve

def f(x):
    y = 2 - x**2
    return y

x0 = 1.4 # Intitial Guess
x = fsolve(f, x0)
print x
print type(x)
```

```
[ 1.41421356]
<type 'numpy.ndarray'>
```

3.12.2 Integration Example

1. Using Quad

```
from scipy.integrate import quad
import numpy as np

def integrand(x):
    return (1 - np.cos(x))**2

ans, err = quad(integrand, 0, 2)
print ans
```

```
0.992204522522
```

2. Using trapz

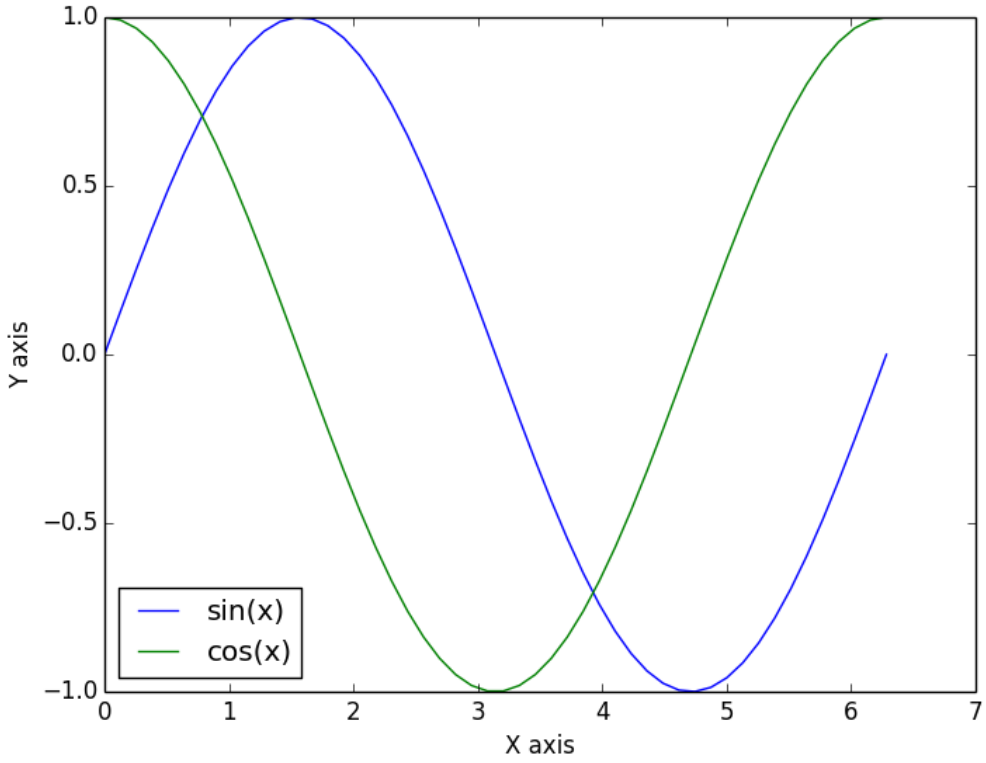
```
import numpy as np

# Array of 100 points between 0 and 2
x2 = np.linspace(0, 2, 100)
y2 = x2**3

print np.trapz(y2, x2)
```

```
4.00040812162
```

3.13 Plotting with Python (Matplotlib)

http://matplotlib.sourceforge.net/ `matplotlib` is the prime plotting module for python. The syntax is similar to Matlab. The best way to learn matplotlib is to visit the gallery (http://matplotlib.sourceforge.net/gallery.html) and look for examples that do what you want. Here is a simple example.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,2*np.pi)
y = np.sin(x)

plt.plot(x,y)
plt.plot(x,np.cos(x))
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend(['sin(x)', 'cos(x)'], loc='best')
plt.savefig('images/Lab1.png')
plt.show()
```



4 FDA

Now let us run our first computational chemistry code. Change into the `computational-chemistry/Lab1/FDA/fda` directory and run the command,

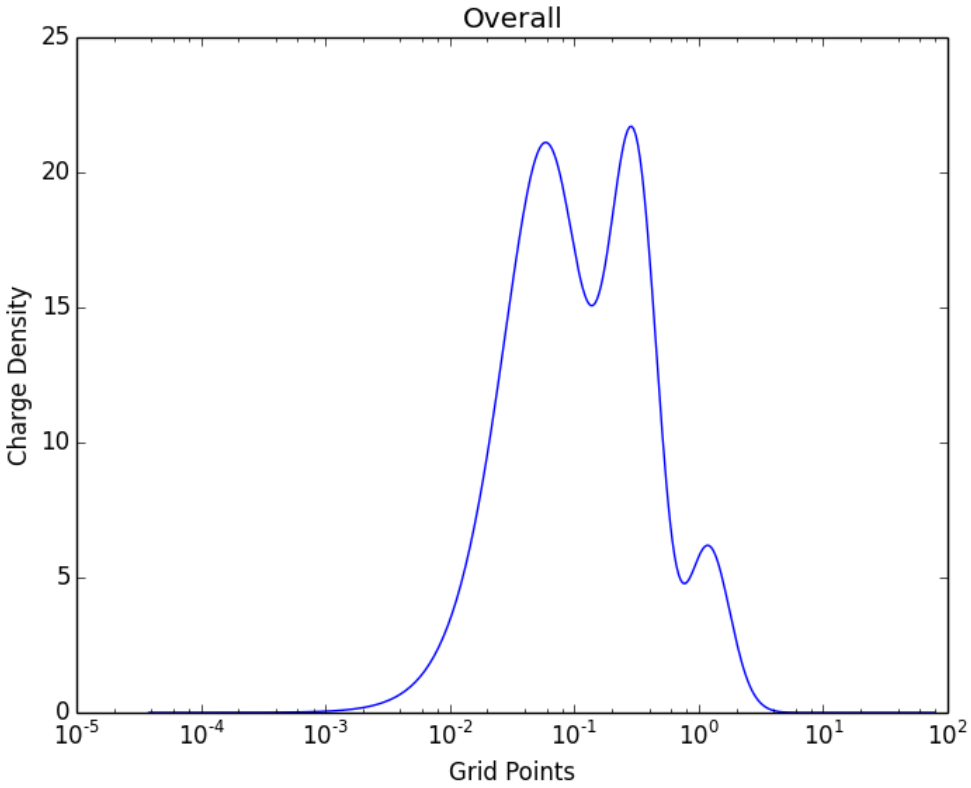
```
./fda Ar
```

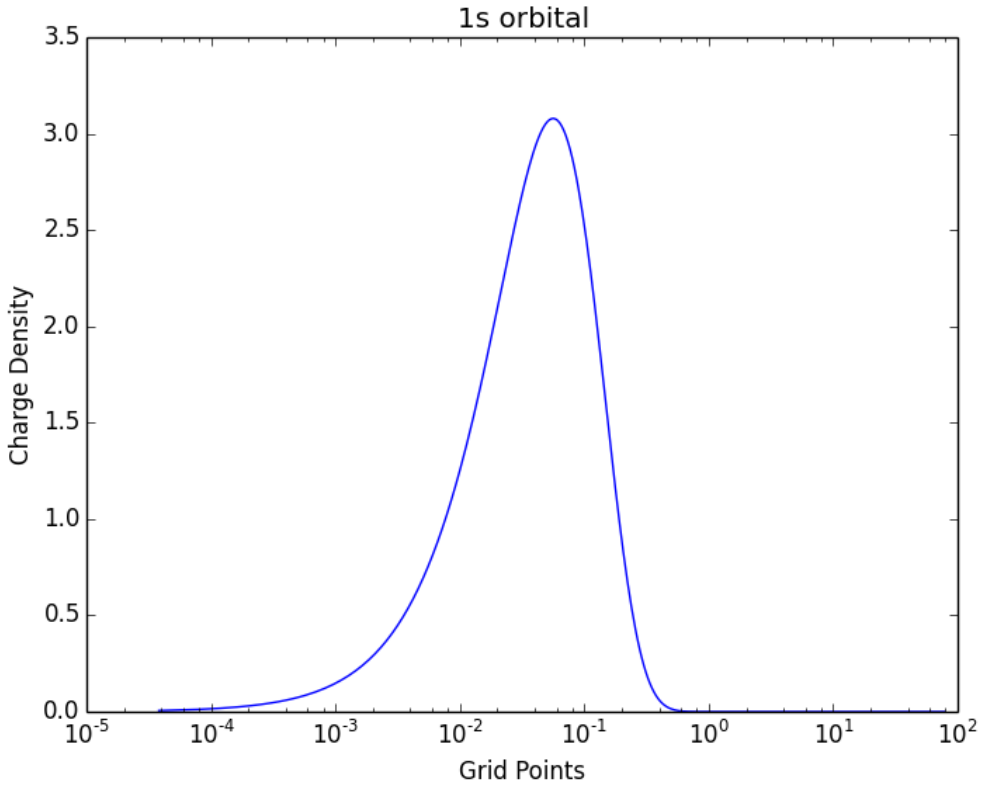
This will use the `Ar.inp` file as input and produce two output files, `Ar.dmp` and `Ar.out`. The `OOREAD.ME` file contains some information about the format of the input file. You can open these files with Emacs to read what they contain.

4.1 Parsing the dmp file

In homework 2, you will need to parse the `.dmp` file to plot some data. It contains the radial grid values and total charge density in two columns, followed by the charge density of each oribital on the same grid. Here we consider an example of how to do this with python.

```
import matplotlib.pyplot as plt
import numpy as np

# Lets open the file in read mode
with open('FDA/fda/Ar.dmp', 'r') as f:

    # Reading all the lines in the file
    # Each line is stored as an element of a list
    lines = f.readlines()

    # First we read the grid points and the total charge densities
    grid_points = []
    total_charge_densities = []

    for line in lines[3:303]:

        # Each is a string with two columns
        grid_point, tot_charge_density = line.split()

        # We need to convert each line to a float add it to our lists
        grid_points.append(float(grid_point))
        total_charge_densities.append(float(tot_charge_density))

    # Now for the 1s orbital
    one_s_charge_density = []

    for x in lines[304:604]:
        one_s_charge_density.append(float(x))

    # Alternately,
    one_s_charge_density_alt = [float(x) for x in lines[304:604]]

plt.figure()
plt.semilogx(grid_points, total_charge_densities)
plt.xlabel('Grid Points')
plt.ylabel('Charge Density')
plt.title('Overall')
plt.savefig('images/Ar-overall-charge-density.png')

plt.figure()
plt.semilogx(grid_points, one_s_charge_density)
plt.xlabel('Grid Points')
plt.ylabel('Charge Density')
plt.title('1s orbital')
plt.savefig('images/Ar-1s-charge-density.png')
plt.show()
```


Overall
Charge Density
Grid Points
25
20
15
10
5
0
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}


1s orbital
Charge Density
Grid Points
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}