MATH 365
Working with data files

1 Introduction

An important component of any scientific program is the ability to read and write data to files, and manipulate the data. Consider a generic file data.txt whose contents are:

data.txt

```
#pressure temperature average_energy
1.0 1.0 -50.0
1.0 1.5 -27.8
2.0 1.0 -14.5
2.0 1.5 -11.2
```

1.1 Reading and writing with Numpy

NumPy routines loadtxt() and savetxt() can be used useful for quickly loading and saving simple array data like that contained in data.txt. For example

mydata = np.loadtxt('data.txt')

creates an array with 4 rows and 3 columns. If a line starts with a number sign (#), it will be ignored as a comment. Blank lines are also ignored. You can see the dimension of the array with

mydata.shape

which in this case returns (4,3).

If we want to alter the data, say change the average_energy in the third row to -15.4 excecute

mydata[2,2]=-15.4

We could then save the new data into a new file

np.savetxt('data_alt.txt', mydata)

We can also load each variable into a separate 1-dimensional array. Setting the argument unpack=True and providing a variable for each column accomplishes this

p, T, AE= np.loadtxt('data.txt', unpack=True)

If you want to read in only some columns, you can use the usecols argument to specify which ones

p, AE= np.loadtxt('data.txt', unpack=True, usecols=[0,2])

Similarly, if you can save each variable as a separate 1-dimensional array. If we want to save the first and third columns

np.savetxt('data_alt.txt', mydata[:,range(0,3,2)])

Note the indexing of the array.

1.2 Formatting

By default, when using savetxt in Numpy the numbers will be written in scientific notation. The *fmt* argument can be used to specify the formatting. If one format is supplied, it will be used for all of the numbers.

The general form of the fmt argument is: *fmt = '%(width).(precision)(specifier)'* where width specifies the maximum number of digits, precision specifies the number of digits after the decimal point, and the possibilities for specifier are shown below. For integer formatting, the precision argument is ignored if you give it. For scientific notation and floating point formatting, the width argument is optional.

Specifier	Meaning	Example	Format Output for -34.5678
i	signed integer	%5i	-34
e	scientific notation	%5.4e	-3.4568e+001
f	floating point	%5.2f	-34.57

For example

np.savetxt('data_alt.txt', mydata, fmt=('%3.1f', '%3.1f', '%4.1f'))

saves the columns as floating point numbers, rather than scientific notation, in the file data_alt.txt.

1.3 Pandas

Pandas is a Python software library written for data manipulation and analysis. It can give more functionality with data than is directly available with Numpy arrays.

import pandas as pd

The name Pandas comes from an econometrics term for data sets: panel data. While there are many different important features of Pandas, here we will just talk about them in terms of CSV files.

CSV is an acronym for "comma separated values" but really most interpretations of it can be assumed to mean delimter-separated values files. This means data are separated in each row with delimiter characters. Most spreadsheet programs do this, as does the file data.txt. For example

csv_data = pd.read_csv('data.txt')
print(csv_data)

results in the following

```
   #pressure temperature average_energy
0                         1.0 1.0 -50.0
1                         1.0 1.5 -27.8
2                         2.0 1.0 -14.5
3                         2.0 1.5 -11.2
```

In order to access the data as arrays as we did with Numpy, we need to sort them in columns. This is done by adding the option *sep=' '* in pd.read_csv, leaving a blank space between the quotes so pandas can detect spaces between values. In addition, we can skip the first line that contains a comment by setting the parameter *header* to None and using *skiprows*

csv_data = pd.read_csv('data.txt',skiprows = 1,header=None, sep=' ')

which results in

```
     0    1      2
0  1.0  1.0 -50.0
1  1.0  1.5 -27.8
2  2.0  1.0 -14.5
3  2.0  1.5 -11.2
```

Before working with the data, it is a good idea to check the information that is stored in csv_data. First, check the dimension by

csv_data.shape

Second, check the data type (e.g. string, integer, floating point number) by

csv_data.dtypes

This gives output

```
0    float64
1    float64
2    float64
dtype: object
```

This means that columns 0-2 consist of floating point numbers, which is what we would expect.

In order to work with the floating point numbers in the data structure we need to assign a label to each column

csv_data.columns = ['Pressure','Temperature', 'Average_Energy']

Then we can access the first column with

csv_data.Pressure

and the second element in the first column with

csv_data.Pressure[1]

If we want to manipulate the data and do arithmetic operations, e.g. see if pressure and average energy are proportional we can find the constant of proportionality for the second row

const=csv_data.Pressure[1]/csv_data.Average_Energy[1]

We can also find an array of constants for each row

const_array=csv_data.Pressure/csv_data.Average_Energy

2 Individual Lab

You may use the file data.txt available in Blackboard to answer the following questions after reading the Working with data files lab.

1. As described in Section 1.1, what is the result in Python if you execute p.shape?
2. As described in Section 1.1, data is saved to the file data_alt.txt twice. Note that each time you use the command np.savetxt it overwrites the existing file (if there is one). What does the file data_alt.txt look like after the second execution, i.e. np.savetxt('data_alt.txt', mydata[:,range(0,3,2)])?
3. As described in Section 1.2, what does the file data_alt.txt look like after the execution np.savetxt('data_alt.txt', mydata, fmt=('%i3', '%3.1f', '%4.1f'))
4. As described in Section 1.3, what is the output when csv_data.shape is executed?
5. As described inSection 1.3 what is the output when const=csv_data.Pressure[1]/csv data.Average Energy[1] is executed?

3 Group work

Please work in pairs to load, plot and analyze the data in the file galton.dta available in Blackboard. This data file contains the heights of children and their parents. The file extension .dta comes form the statistical software package STATA which integrates well with Python. You will plot these data to see if the heights of parents and children look to be related linearly. Next week we will talk about fitting a least squares regression line to the data.

1. Let's start by loading the data and get comfortable with its contents by executing the following commands

   ```
   import pandas as pd
   df = pd.read_stata('galton.dta')
   df.shape
   df.dtypes
   ```

   Be prepared to describe the categories, their formats and values. I suggest printing a subset of the data values to help understand each category, e.g.

   ```
   print(df.family[1:10])
   ```

   Note that the category family is a family ID variable, as the data include multiple children per family.

2. Plot the heights of mother vs kid and father vs kid. Make sure to plot data points using either '.' or plt.scatter and not a continuous line. **Be prepared to discuss the potential linear relationship between the mothers' height and the kids' height. Also, the potential for the fathers' height to be linearly related to the kids' height.** It may help to make both axes equal to each other and try different scales to come to your conclusions.

3. Investigate the linear relationship between just the boys height and their father's height. In order to do this we must only use those rows of the data that refer to males. Start by creating an array with only the boys' height

   boy_height=df.height[df.male == 1]

   In order to compare it to the father's height we need to shorten the father array in the same manner so it only contains rows corresponding to their boy's height

   boy_father=df.father[df.male == 1]

   Plot the heights of the fathers vs their boys and **be prepared to discuss if you think there is a stronger linear relationship than those you saw in 2.**

4. Now investigate if there is looks like there is a linear relationship between just the girls's height and their mother's height. **Compare how their linear relationship looks to those you observed in 2 and 3.**