Exercise 5

Molecular Statistics, Week 5

2014

1 Introduction

Often it will be necessary to data-mine, manipulate and visualize data obtained manually from experiments or from other software. Python is great for this and the goals of this exercise is:

1. Use Python to load/read data
2. Use Numpy to manipulate data
3. Use matplotlib to illustrate
4. Save Numpy data

1.1 Changing the look of matplotlib

2 Exercises

Todays exercises will each be based on different sets of data which requires different representation.

2.1 Dissociation energy of water dimer

distance of the hydrogen bond is defined as the distance between the oxygen and the hydrogen

1. Convert the distance from A.U. to Ångstroem.
2. Convert the energy to kJoule/mol. Plot the result
3. Convert the energy to kcal/mol. Plot the result.

2.2 Proton transfer / reaction path

2.3 Random precision errors in assigned chemical shifts

When assigning measured chemical shifts of a protein to their respective amino-acids you will have to match the chemical shift measured by one experiment with another. Due to experimental error these values are not exactly the same, even though they should be in theory. Because of this it can be difficult to be sure that the two matched chemical shifts actually origin from the same amino-acid. To avoid making assignment errors it is thus informative to know how well the measured chemical shifts 'should' match.
The file `chemical_shift_errors.txt` obtained from the course website contains all differences (or errors) in assigned chemical shifts of all amide protons for a single protein in a single column format.

4. Load the file containing the data and store it a variable.
5. Plot a histogram of the data using Matplotlib.

When plotting a histogram you can select the number of bins to present the data with by giving the argument `bins=10`. (10 is the default value in Matplotlib). If you try changing the number of bins you can severely affect how the data 'looks', especially if your number of datapoints are relatively low. The Freedman-Diaconis Rule can be used to select the number of bins automatically. The following code takes as argument the data and returns the optimal number of bins according to the Freedman-Diaconis Rule.

```
def bins(data):
    data.sort()
    n=len(data)
    width = 2*(data[3*n/4]-data[n/4])*n**(-1./3)
    return int((data[-1]-data[0])/width)
```

6. Plot the histogram again using the Freedman-Diaconis Rule to select number of bins.

If these errors are completely random, they should approximately follow a normal distribution (also known as a Gaussian distribution). We can use the module `scipy.stats` to fit a distribution to a dataset. Import this in your program as follows:

```
import scipy.stats as ss
```

We will begin by fitting a normal distribution to our data. The command `ss.norm.fit(data)` returns the mean and standard deviation that best describes the data. To draw this curve we will need a set of x and y-values that cover our data range.

7. Use `np.arange()` together with the `max()` and `min()` functions to generate x-values that range from the lowest data point to the highest in steps of `1e3`. Store these in a variable called `x`.

The function `ss.norm.pdf(x, param_1, param_2)` returns the probability densities for a normal distribution for all values of `x`. `param_1` and `param_2` are the mean and standard deviation you obtained from the fit. Fitting more complicated distributions will return more than two parameters. To avoid having to adjust the number of arguments for each distribution you look at, the following works for every distribution in the `scipy.stats` package:

```
parameters = norm.fit(data)
y = norm.pdf(x, *parameters)
```

8. Fit a normal distribution to your data.
9. Try to plot the fitted distribution together with the histogram. *Hint!* Use `normed=1` as argument to your histogram.
10. Try fitting other popular distributions such as the Cauchy/Lorenz distribution `ss.cauchy` or the Student's t-distribution `ss.t`.

2.4 some kind of 3d plot

2.5 Protein structure determination

Not all protein structures can easily be determined experimentally. These kinds of proteins will often have their structure determined by simulation where a force-field is used to describe how the atoms interact with each other. If a good force-field is used, the correct (also called native) structure should correspond to the lowest energy of the force-field. When developing new force-fields you want to see how well the energy correlate with the deviation from the native structure.
The file `rmsd_energy_unfolded.txt` obtained from the course website, contains the energies, in units of $kcal/(mol \cdot RT)$ at $300K$, of several proposed structures as well as the atomic root mean square deviation (rmsd) in Å from the native structure. You will never get an rmsd of zero for structures proposed by simulation, since the force-field will never fully describe all interactions correctly, however rmsds under 3-5 Å is usually considered accurate.
Note: The first column contain the rmsds and the second one the energies.

1. Load the datafile in Python and put the rmsds in one list, and the energies in another.
2. Plot the rmsds vs. energies using small black dots. Does the force-field used seem to be good in this case?

Since the local energi minimum is not exactly at 0 Å it can be hard to know if you have achieved the correct structure and the non-zero rmsd stems from the force-field, or if an incorrect structure is found. To test this a second simulation is usually run starting from the native structure to what the structure relaxes to with the given force-field.

3. Download and load the file `rmsd_energy_native.txt`.
4. Plot this together with the data from before, using a red colour. *Note!* Change the border colour of the dots to red as well.
5. Based on this second simulation, would you say that the force-field performs well?