Hands-on Session Examples

ProDy Examples

In [39]:
```
from prody import parsePDB # we imported only a function that we need for
```

In [40]:
```
parsePDB ?
```

Let's parse some structures:

In [41]:
```
p38 = parsePDB('1p38')
```

```
@> PDB file is found in the local mirror (.\1p38.pdb).
DEBUG:.prody:PDB file is found in the local mirror (.\1p38.pdb).
@> 2962 atoms and 1 coordinate set(s) were parsed in 0.04s.
DEBUG:.prody:2962 atoms and 1 coordinate set(s) were parsed in 0.04s.
```

In [42]:
```
p38
```

Out[42]:
```
<AtomGroup: 1p38 (2962 atoms)>
```

`p38` is an AtomGroup object. You can think of it as a list of atoms, and indexing will work in the same way.

In [43]:
```
l = range(10)
```

In [44]:
```
l[0] # to get the first item in the list
```

Out[44]:
```
0
```

In [45]:
```
p38[0] # to get the first atom in the atom group
```

Out[45]:
```
<Atom: N from 1p38 (index 0)>
```

In [46]:
```
n = p38[0]
```

In [47]:
```
n
```

Out[47]:
```
<Atom: N from 1p38 (index 0)>
```

Dictionaries

There are also similarities between AtomGroup objects and dictionaries. Dictionaries map keys to values. In the following we will map some first names to last names:

In [48]:
```
d = {}
```

In [49]:
```
d ?
```

In [50]:
```
d['ahmet']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-50-94b25eed165b> in <module>()
----> 1 d['ahmet']

KeyError: 'ahmet'
```

In [51]:
```
d['ahmet'] = 'bakan'
```

In [52]:
```
d
```

Out[52]:
```
{'ahmet': 'bakan'}
```

In [53]:
```
d['tim'] = 'lezon'
```

In [54]:
```
d
```

Out[54]:
```
{'ahmet': 'bakan', 'tim': 'lezon'}
```

In [55]:
```
d['ahmet']
```

Out[55]:
```
'bakan'
```

AtomGroups - Chains

AtomGroup objects map one letter chain identifiers, e.g. 'A' to chain objects.

In [56]:
```
p38.numAtoms()
```

Out[56]: 2962

In [57]:
```
p38.numChains()
```

Out[57]: 1

In [58]:
```
p38['A']
```

Out[58]: <Chain: A from 1p38 (480 residues, 2962 atoms)>

In [59]:
```
chA = p38['A']
```

In [60]:
```
chA
```

Out[60]: <Chain: A from 1p38 (480 residues, 2962 atoms)>

In [61]:
```
len(p38) # lengh of AtomGroup is number atoms, since AtomGroup is like a l
```

Out[61]: 2962

In [62]:
```
len(chA) # whereas length of a chain object is number of residues, since c
```

Out[62]: 480

Some calculations

In [63]:
```
from prody import * # we now import everything
```

In [64]:
```
calcGyradius(p38) # we calculated radius of gyration for the AtomGroup
```

Out[64]: 22.057752024921747

In [65]:
```
calcGyradius(chA) # Chain object can also be an import to the same functic
```

Out[65]: 22.057752024921747

In [66]:
```
chA
```

Out[66]: <Chain: A from 1p38 (480 residues, 2962 atoms)>

In [67]:
```
p38
```

Out[67]:
```
<AtomGroup: 1p38 (2962 atoms)>
```

In [68]:
```
p38.numAtoms('water') # this gives us number of water atoms in the AtomGrc
```

Out[68]:
```
129
```

In [69]:
```
p38.numAtoms('protein') # this give number of protein atoms
```

Out[69]:
```
2833
```

In [70]:
```
p38.select('protein') # we select protein atoms
```

Out[70]:
```
<Selection: 'protein' from 1p38 (2833 atoms)>
```

In [71]:
```
p38.protein # same selection simplified like this
```

Out[71]:
```
<Selection: 'protein' from 1p38 (2833 atoms)>
```

In [72]:
```
p38.protein == p38.select('protein') # they are the same thing
```

Out[72]:
```
True
```

In [73]:
```
calcGyradius(p38.protein) # this is what we would do if we wanted to perfc
```

Out[73]:
```
21.960840914729118
```

Residues

In [74]:
```
chA[10] # get the residue number 10
```

Out[74]:
```
<Residue: ARG 10 from Chain A from 1p38 (11 atoms)>
```

In [75]:
```
calcPsi(chA[10]) # a residue can be an input to calcPsi function
```

Out[75]:
```
147.49025666398765
```

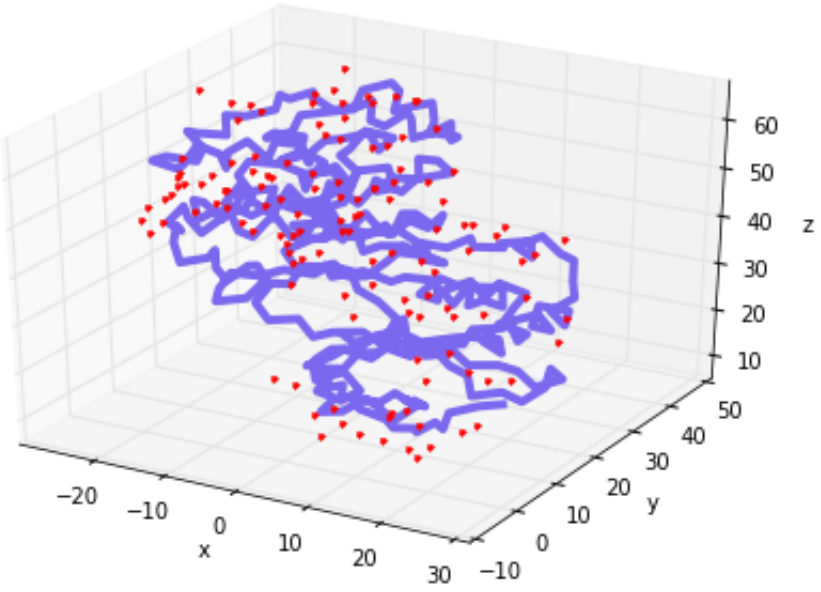
In [76]:
```
calcPsi ?
```

Show atoms

You can show atoms using the following function. For proteins only Calpha trace will be displayed. Heterogeneous atoms will be shown as points or dots

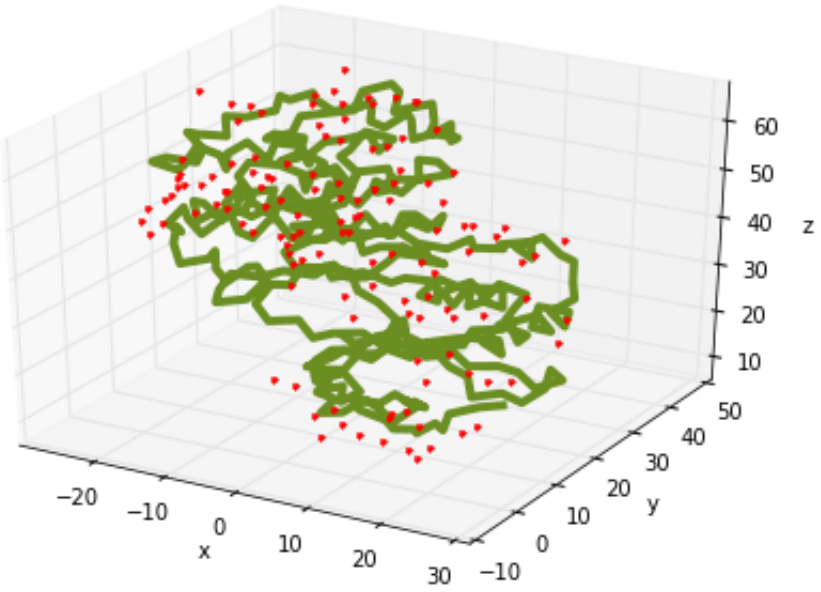
In [77]:
```
showProtein(p38)
```

Out[77]: <mpl_toolkits.mplot3d.axes3d.Axes3D at 0x56a9ed0>



In [78]:
```
showProtein(chA)
```

Out[78]: <mpl_toolkits.mplot3d.axes3d.Axes3D at 0x5b18170>



Arrays and Efficiency

In [79]:
```
l = range(10)
```

In [80]:
```
array(l) # this give us an array
```

Out[80]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [81]:
```
l = range(100000) # we will make a long list to show how efficient arrays
```

In [82]:
```
len(l)
```

Out[82]:
```
100000
```

In [83]:
```
a = array(l)
```

In [84]:
```
len(a)
```

Out[84]:
```
100000
```

We will time the operation of summing up items in a list and array. For lists, we will use built-in Python function `sum`. For arrays, we will use array method `.sum`. Array methods are C code working on defined types, and they can be much more faster than pure Python equivalents.

In [85]:
```
%timeit sum(l)
```

```
10 loops, best of 3: 23.4 ms per loop
```

This operation took 23400 microseconds on my laptop.

In [86]:
```
a.sum()
```

Out[86]:
```
704982704
```

In [87]:
```
a.max()
```

Out[87]:
```
99999
```

In [88]:
```
%timeit a.sum()
```

```
10000 loops, best of 3: 128 us per loop
```

This operation took 124 micro seconds

In [89]:
```
23400/125
```

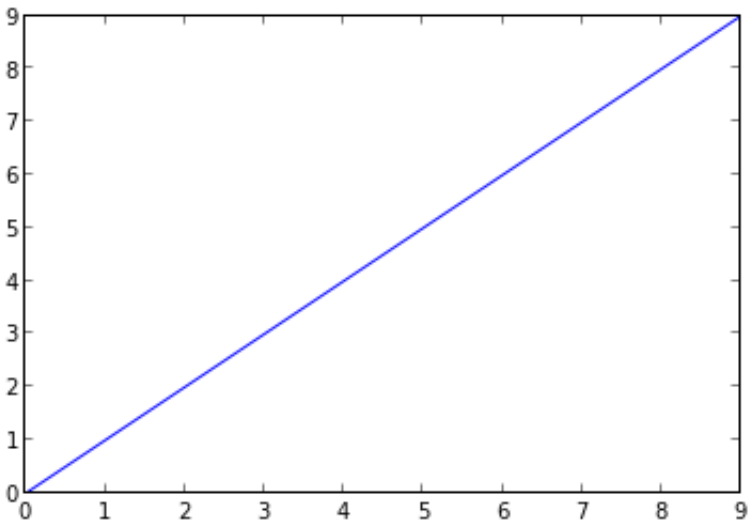
Out[89]: 187

Array summation is 187 times faster. ProDy uses NumPy arrays to store and modify data in structure file, so most operations will be as fast as it would be using C code developed for the sampe purpose.

Plotting example

In [90]:
```
plot( range(10) ) # simple plot
```

Out[90]: [<matplotlib.lines.Line2D at 0x5926ef0>]



In [91]:
```
random.random(10) # this returns random numbers from uniform distribution
```

Out[91]:
```
array([ 0.64976994,  0.36475811,  0.01070796,  0.21786143,  0.91924073,
        0.27599671,  0.6306186 ,  0.59448841,  0.90071934,  0.83750927])
```

In [92]:
```
random.random((3,3))
```

Out[92]:
```
array([[ 0.4052089 ,  0.16822682,  0.18026583],
       [ 0.60029805,  0.57140157,  0.37823275],
       [ 0.32850192,  0.62418365,  0.1196399 ]])
```

In [93]:
```
random_number = normal(10, 2, 1000) # will return 1000 random numbers from
```

In [1]:
```
hist(random number) # this show a distribution of numbers
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call
last)
<ipython-input-1-4fc3bba91e20> in <module>()
----> 1 hist(random_number) # this show a distribution of numbers

NameError: name 'random_number' is not defined
```

A crazy example

In this example, we will download an image from the internet and show it using Matplotlib

In [2]:
```
from urllib import urlopen # for reading online documents
from matplotlib import image # for reading image data
```

In [3]:
```
url_handle = urlopen('http://www.csb.pitt.edu/ProDy/_static/logo.png') # w
```

In [4]:
```
img_data = image.imread(url_handle) # this will read image data and conver
```

In [7]:
```
imshow(img_data) # You see the image below
```

Out[7]: <matplotlib.image.AxesImage at 0x555e670>

You can do much more with Matplotlib. Take a look at its gallery of plots: http://matplotlib.org/gallery.html
You can pick something similar to that you want to plot and then get the code example that was used to plot it.