p38 Example

This examples shows how to analyze an ensemble of structures and extract dominant patterns in conformational variability, that is the dynamics of the protein that we can infer from experiments. Then, we compare this to theoretically predicted motions of the protein.

First, we import ProDy API:

In [7]:
```
from prody import *
```

List of Structures

We will use a predefined list of structures for this example:

In [8]:
```
p38_pdbs = ['1A9U', '1BL6', '1BL7', '1BMK', '1DI9', '1IAN', '1KV1', '1KV2'
            '1LEW', '1LEZ', '1M7Q', '1OUK', '1OUY', '1OVE', '1OZ1', '1P38'
            '1R39', '1R3C', '1W7H', '1W82', '1W83', '1W84', '1WBN', '1WBO'
            '1WBS', '1WBT', '1WBV', '1WBW', '1WFC', '1YQJ', '1YW2', '1YWR'
            '1ZYJ', '1ZZ2', '1ZZL', '2BAJ', '2BAK', '2BAL', '2BAQ', '2EWA'
            '2FSL', '2FSM', '2FSO', '2FST', '2GFS', '2GHL', '2GHM', '2GTM'
            '2GTN', '2I0H', '2NPQ', '2OKR', '2OZA', '3HVC', '3MH0', '3MH3'
            '3MH2', '2PUU', '3MGY', '3MH1', '2QD9', '2RG5', '2RG6', '2ZAZ'
            '2ZB0', '2ZB1', '3BV2', '3BV3', '3BX5', '3C5U', '3L8X', '3CTQ'
            '3D7Z', '3D83', '2ONL']
len(p38_pdbs)
```

Out[8]: 75

Reference Structure

We need to specify a reference structure and a reference chain. We are going to use `1p38`, which is an unbound and quite complete (smallers number of unresolved residues) structure of p38 MAPK.

It is always best to select the most complete structure as the reference structure.

In [9]:
```
p38 = parsePDB('1p38', subset='ca')
p38
```

```
@> PDB file is found in the local mirror (.\1p38.pdb).
DEBUG:.prody:PDB file is found in the local mirror (.\1p38.pdb).
```

```
@> 351 atoms and 1 coordinate set(s) were parsed in 0.02s.
DEBUG:.prody:351 atoms and 1 coordinate set(s) were parsed in 0.02s.
```

Out[9]:
```
<AtomGroup: 1p38_ca (351 atoms)>
```

In [10]:
```
p38_ref = p38.select('resnum 5 to 31 36 to 114 122 to 169 185 to 351').get
p38_ref
```

Out[10]:
```
<Chain: A from 1p38_ca (321 residues, 321 atoms)>
```

Comparing Chains

Let's read another p38 structure and compare it to the reference structure. The function that we will use for this comparison is the key to building the ensemble:

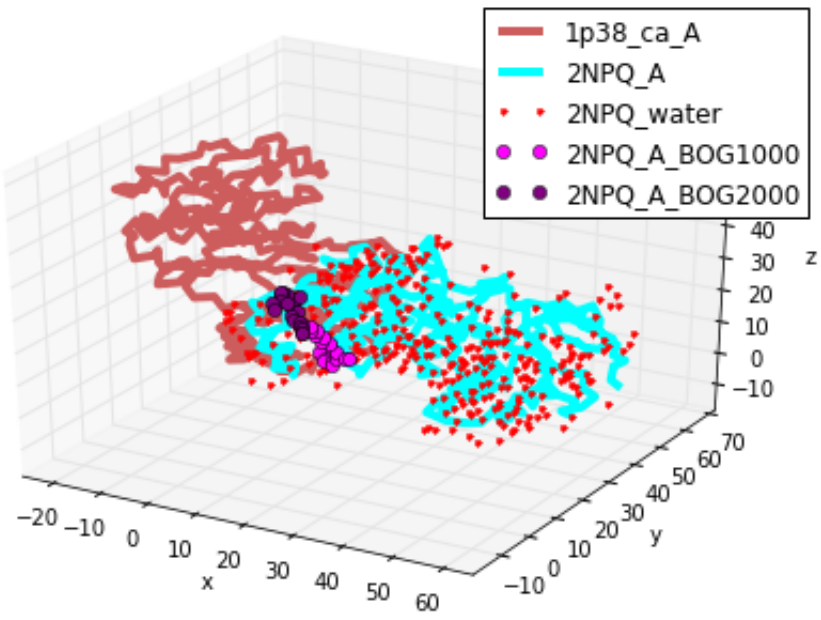
In [11]:
```
other = parsePDB(p38_pdbs[50])
other
```

```
@> PDB file is found in the local mirror (.\2npq.pdb).
DEBUG:.prody:PDB file is found in the local mirror (.\2npq.pdb).
@> 3014 atoms and 1 coordinate set(s) were parsed in 0.13s.
DEBUG:.prody:3014 atoms and 1 coordinate set(s) were parsed in 0.13s.
```

Out[11]:
```
<AtomGroup: 2NPQ (3014 atoms)>
```

In [12]:
```
showProtein(p38, other)
legend()
```

Out[12]:
```
<matplotlib.legend.Legend at 0x5633cd0>
```



In [13]:
```
mapping = mapOntoChain(other, p38_ref)
mapping
```

```
@> Evaluating AtomGroup 2NPQ: 1 chains are identified
DEBUG:.prody:Evaluating AtomGroup 2NPQ: 1 chains are identified
@> Trying to map atoms based on residue numbers and identities:
DEBUG:.prody:Trying to map atoms based on residue numbers and identities:
@>   Comparing Chain A from 2NPQ (len=326) with Chain A from 1p38_ca:
DEBUG:.prody:  Comparing Chain A from 2NPQ (len=326) with Chain A from
1p38_ca:
@>      Mapped: 321 residues match with 99% sequence identity and 98%
overlap.
DEBUG:.prody:   Mapped: 321 residues match with 99% sequence identity and
98% overlap.
```

Out[13]:
```
[(<AtomMap: Chain A from 2NPQ -> Chain A from 1p38_ca from 2NPQ (321
atoms)>,
  <AtomMap: Chain A from 1p38_ca -> Chain A from 2NPQ from 1p38_ca (321
atoms)>,
  99,
  98)]
```

`mapping` variable is a list of mapping(s). It may contain multiple mapping between chains in given structures. The first mapping is the best mapping, that is it has the higher sequence identitiy between the mapped protein chains.

In [14]:
```
best_mapping = mapping[0]
best_mapping[3]
```

Out[14]: 98

In [15]:
```
other_mapped_atoms = best_mapping[0]
ref_mapped_atoms = best_mapping[1]
calcRMSD(other_mapped_atoms, ref_mapped_atoms)
```

Out[15]: 51.190291246092897

We can find out which atoms are mapped in the `other` structure by getting atom flags `mapped`. This will return a boolean array, and for mapped atoms it will contain `True`, for those missing, it will contain `False`.

In [16]:
```
other_mapped_atoms.getFlags('mapped')
```

Out[16]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
```

```
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True,    True,    True,    True,
        True,    True,    True,    True,    True,    True], dtype=bool)
```

In [17]:

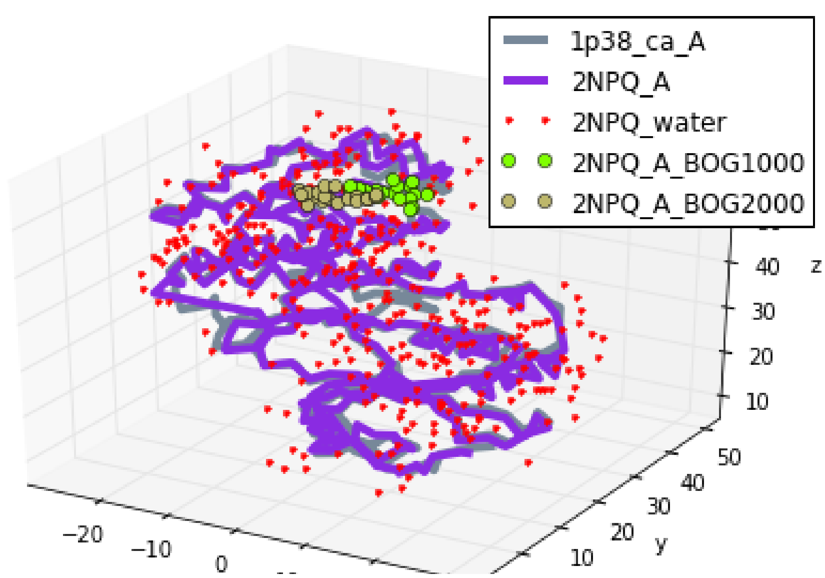
```
superpose(other_mapped_atoms, ref_mapped_atoms)
calcRMSD(other_mapped_atoms, ref_mapped_atoms)
```

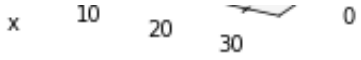
Out[17]: 1.0590477386767079

In [18]:

```
showProtein(p38, other)
legend()
```

Out[18]: <matplotlib.legend.Legend at 0x59c58b0>





Building PDB Ensemble

We will assemble conformations from structures in a `PDBEnsemble` object:

In [19]:
```
ens = PDBEnsemble('p38')
ens
```

Out[19]:
```
<PDBEnsemble: p38 (0 conformations; 0 atoms)>
```

In [20]:
```
ens.setAtoms(p38_ref)
ens.setCoords(p38_ref)
ens
```

Out[20]:
```
<PDBEnsemble: p38 (0 conformations; 321 atoms)>
```

In [21]:
```
l = range(10)
print l
for i in l:
    print i, i**2
for pdb in p38_pdbs:
    print pdb,
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
0 0
1 1
2 4
3 9
4 16
5 25
6 36
7 49
8 64
9 81
1A9U 1BL6 1BL7 1BMK 1DI9 1IAN 1KV1 1KV2 1LEW 1LEZ 1M7Q 1OUK 1OUY 1OVE
1OZ1 1P38 1R39 1R3C 1W7H 1W82 1W83 1W84 1WBN 1WBO 1WBS 1WBT 1WBV 1WBW
1WFC 1YQJ 1YW2 1YWR 1ZYJ 1ZZ2 1ZZL 2BAJ 2BAK 2BAL 2BAQ 2EWA 2FSL 2FSM
2FSO 2FST 2GFS 2GHL 2GHM 2GTM 2GTN 2I0H 2NPQ 2OKR 2OZA 3HVC 3MH0 3MH3
3MH2 2PUU 3MGY 3MH1 2QD9 2RG5 2RG6 2ZAZ 2ZB0 2ZB1 3BV2 3BV3 3BX5 3C5U
3L8X 3CTQ 3D7Z 3D83 2ONL
```

Now we will build the ensemble. We will use `mapOntoChain` function for each structure that we want to incorporate in the ensemble.

In [22]:
```
from time import time
start = time()
```

```
for pdb in p38_pdbs:
    pdb = parsePDB(pdb, subset='ca')
    mapping = mapOntoChain(pdb, p38_ref)[0]
    mapped = mapping[0]
    ens.addCoordset(mapped, weights=mapped.getFlags('mapped'), label=pdb.g
timing = time()-start
```

```
@> PDB file is found in the local mirror (.\1a9u.pdb).
DEBUG:.prody:PDB file is found in the local mirror (.\1a9u.pdb).
@> 351 atoms and 1 coordinate set(s) were parsed in 0.18s.
DEBUG:.prody:351 atoms and 1 coordinate set(s) were parsed in 0.18s.
@> Evaluating AtomGroup 1A9U_ca: 1 chains are identified
DEBUG:.prody:Evaluating AtomGroup 1A9U_ca: 1 chains are identified
@> Trying to map atoms based on residue numbers and identities:
DEBUG:.prody:Trying to map atoms based on residue numbers and
identities:
@>   Comparing Chain A from 1A9U_ca (len=351) with Chain A from
1p38_ca:
DEBUG:.prody:  Comparing Chain A from 1A9U_ca (len=351) with Chain A
from 1p38_ca:
@>      Mapped: 321 residues match with 99% sequence identity and 91%
overlap.
DEBUG:.prody:   Mapped: 321 residues match with 99% sequence identity
and 91% overlap.
@> PDB file is found in the local mirror (.\1bl6.pdb).
DEBUG:.prody:PDB file is found in the local mirror (.\1bl6.pdb).
@> 351 atoms and 1 coordinate set(s) were parsed in 0.04s.
```

In [23]:
```
ens
```

Out[23]:
```
<PDBEnsemble: p38 (75 conformations; 321 atoms)>
```

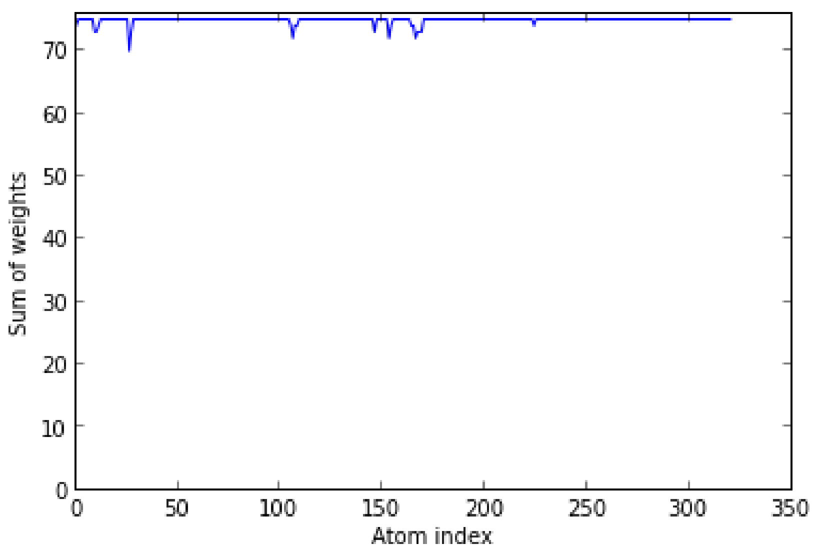
In [24]:
```
calcOccupancies(ens) # this will show us the number of structures that eac
```

Out[24]:
```
array([ 74.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  73.,  73.,
        74.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  70.,  73.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  74.,  72.,  74.,  74.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  73.,  75.,  75.,  75.,  75.,  75.,  75.,  72.,
        74.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  74.,
        74.,  72.,  73.,  73.,  73.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
        75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,  75.,
```

```
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   74.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,   75.,
        75.,   75.])
```

In [25]:
```
showOccupancies(ens)
```

Out[25]: [<matplotlib.lines.Line2D at 0x6442cf0>]



In [26]:
```
timing/len(ens)
```

Out[26]: 0.16486666679382325

In [27]:
```
trimPDBEnsemble ?
```

In [28]:
```
ens.iterpose()
```

```
@> Starting iterative superposition:
INFO:.prody:Starting iterative superposition:
@> Step #1: RMSD difference = 7.5550e-01
INFO:.prody:Step #1: RMSD difference = 7.5550e-01
@> Step #2: RMSD difference = 1.6867e-04
INFO:.prody:Step #2: RMSD difference = 1.6867e-04
@> Step #3: RMSD difference = 1.7398e-07
INFO:.prody:Step #3: RMSD difference = 1.7398e-07
@> Iterative superposition completed in 0.16s.
```

```
DEBUG:.prody:Iterative superposition completed in 0.16s.
@> Final superposition to calculate transformations.
INFO:.prody:Final superposition to calculate transformations.
@> Superposition completed in 0.07 seconds.
DEBUG:.prody:Superposition completed in 0.07 seconds.
```

PCA Calculations

In [29]:
```
pca = PCA('p38')
pca.buildCovariance(ens)
pca.calcModes()
pca
```

```
@> Covariance is calculated using 75 coordinate sets.
INFO:.prody:Covariance is calculated using 75 coordinate sets.
@> Covariance matrix calculated in 0.284000s.
DEBUG:.prody:Covariance matrix calculated in 0.284000s.
@> 20 modes were calculated in 1.28s.
DEBUG:.prody:20 modes were calculated in 1.28s.
```

Out[29]:
```
<PCA: p38 (20 modes; 321 atoms)>
```

In [30]:
```
pca[0]
```

Out[30]:
```
<Mode: 1 from PCA p38>
```

In [31]:
```
calcFractVariance(pca[0])
```

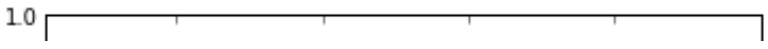
Out[31]:
```
0.2919181134661914
```

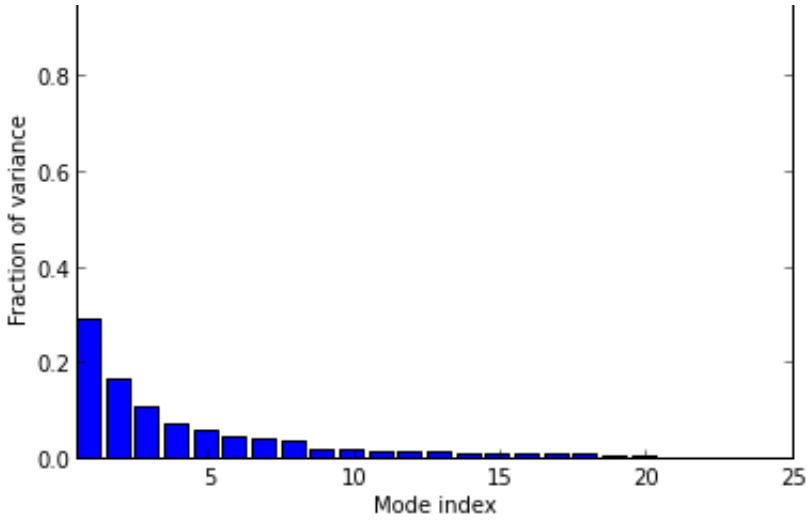
In [32]:
```
for i in range(10):
    mode = pca[i]
    print mode, calcFractVariance(mode)
```

```
Mode 1 from PCA p38 0.291918113466
Mode 2 from PCA p38 0.16509871699
Mode 3 from PCA p38 0.106324495574
Mode 4 from PCA p38 0.0723278872924
Mode 5 from PCA p38 0.0569148267664
Mode 6 from PCA p38 0.0438956532997
Mode 7 from PCA p38 0.0412148092555
Mode 8 from PCA p38 0.0340511072176
Mode 9 from PCA p38 0.0182693961103
Mode 10 from PCA p38 0.0170065627455
```

In [33]:
```
showFractVars(pca)
```

Out[33]:
```
<Container object of 20 artists>
```





ANM Calculations

In [34]:
```
anm = ANM('1p38')
anm.buildHessian(p38_ref)
anm.calcModes()
anm
```

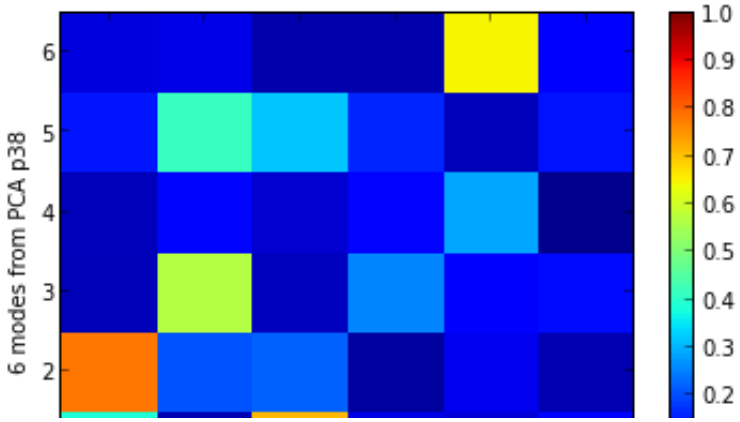
```
@> Hessian was built in 0.67s.
INFO:.prody:Hessian was built in 0.67s.
@> 20 modes were calculated in 1.21s.
DEBUG:.prody:20 modes were calculated in 1.21s.
```

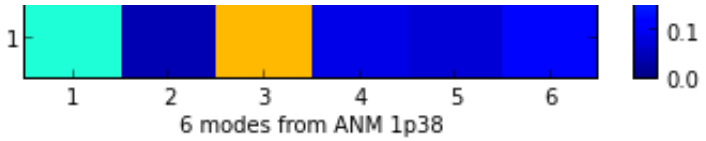
Out[34]:
```
<ANM: 1p38 (20 modes; 321 nodes)>
```

Comparison

In [35]:
```
showOverlapTable(anm[:6], pca[:6])
```

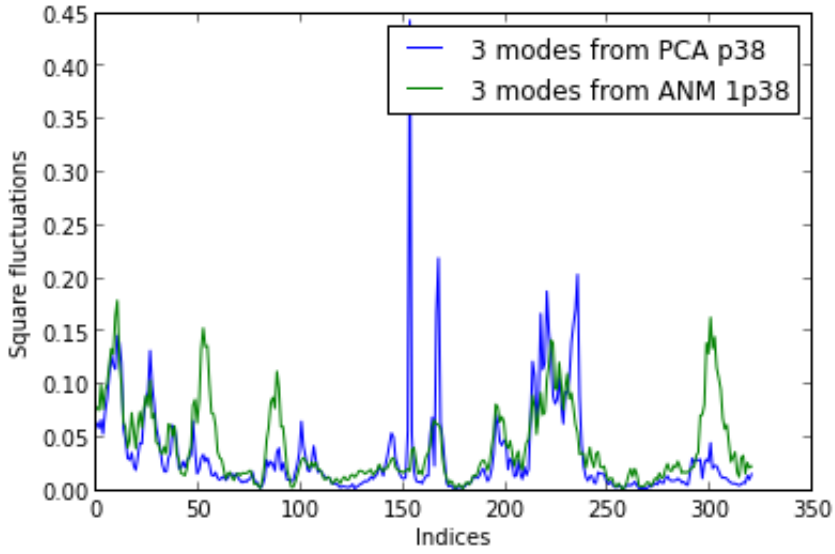
Out[35]:
```
(<matplotlib.collections.PolyCollection at 0x7757950>,
 <matplotlib.colorbar.Colorbar instance at 0x077968F0>)
```





In [36]:
```
showNormedSqFlucts(pca[:3])
showNormedSqFlucts(anm[:3])
legend()
```

Out[36]: <matplotlib.legend.Legend at 0x5b561d0>



In [37]:
```
writeNMD('p38_pca.nmd', pca[:6], p38_ref)
```

Out[37]: 'p38_pca.nmd'

Reading files

In [38]:
```
nmd = open('p38_pca.nmd')
nmd
```

Out[38]: <open file 'p38_pca.nmd', mode 'r' at 0x05AE89C0>

In [39]:
```
nmd = open('p38_pca.nmd')
for line in nmd:
    print line[:60]
```

```
nmwiz_load D:\workshop\p38_pca.nmd

name 6_modes_from_PCA_p38

atomnames CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA CA
resnames ARG PRO THR PHE TYR ARG GLN GLU LEU ASN LYS THR ILE
resids 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 2
```

```
chainids A A A A A A A A A A A A A A A A A A A A A A A A A A A A
bfactors 59.83 47.29 34.64 26.52 25.42 32.60 34.80 42.96 46.
coordinates 24.179 4.807 21.907 21.218 2.742 20.697 20.409 2
mode 1 7.35 0.025 0.004 0.086 0.031 -0.020 0.082 0.067 -0.03
mode 2 5.52 0.023 -0.046 0.038 0.021 -0.039 0.035 0.029 -0.0
mode 3 4.43 0.017 0.039 0.033 0.024 0.016 0.034 0.022 0.023
mode 4 3.66 -0.033 0.005 0.010 -0.034 -0.011 0.020 -0.030 -0
mode 5 3.24 0.017 0.069 0.055 -0.003 0.011 0.094 0.035 0.016
mode 6 2.85 -0.019 -0.022 -0.032 -0.030 -0.007 -0.010 -0.036
```

In [ ]: