

SUPPORTING MATERIAL 14

Barnaba: Software for Analysis of Nucleic Acids Structures and Trajectories

Contents

| | |
|---|-----------|
| 1 Calculate quantities for analysis | 1 |
| 2 Figure 1: RMSD, eRMSD, annotation | 3 |
| 3 Figure 2: Transition Paths. | 6 |
| 4 Figure 3: Torsion angles and scalar couplings. | 12 |
| 5 Figure 4: Cluster analysis. | 15 |
| 6 Figure 6: ENM. | 22 |
| 7 Figure 7: ENM performances. | 24 |
| 7.1 SBP model | 28 |
| 7.2 Save all results on disk | 31 |
| 7.3 Final plot | 31 |

1 Calculate quantities for analysis

These notebooks describe how to calculate the data and how to produce figures in the manuscript “Barnaba: Software for Analysis of Nucleic Acids Structures and Trajectories”. Here, we calculate different quantities over the entire trajectory: - eRMSD from reference structure - RMSD from reference structure - Base-pairing and base-stacking detection (annotation) - Dot-bracket annotation - Backbone torsion angles - ³J scalar couplings - Relative position and orientation between nucleobases as G-vectors, as defined in Bottaro, Di Palma, Bussi. NAR 2014.

All the data is saved to pickle files for later analysis. The MD trajectory is taken from the paper “RNA force field with accuracy comparable to state-of- the-art protein force fields”, PNAS, 2017.

First, we import the modules barnaba and pickles, and define the location of the topology/trajectory file, as well as the location of the native, reference structure.

```
In [1]: import barnaba as bb
import pickle

top = "topology.pdb"
traj = "trajectory.dcd"
native = "2K0C.pdb"
```

Now we calculate all the quantities described above and save the data to pickle file. Note that `heavy_atoms=True` in RMSD calculation. Default mode is backbone-only.

```
In [2]: # calculate eRMSD and store in a pickle file
fname = "ermsd.p"
```

```

ermsd = bb.ermsd(native,traj,topology=top)
pickle.dump(ermsd[1:],open(fname, "w"))

# calculate RMSD and store in a pickle file
fname = "rmsd.p"
print "# calculate %s" % fname
rmsd = bb.rmsd(native,traj,topology=top,heavy_atom=True)
pickle.dump(rmsd[1:],open(fname, "w"))

# calculate annotation and store in pickle file
fname = "pairs.p"
print "# calculate %s" % fname
stackings, pairings, res = bb.annotate(traj,topology=top)
pickle.dump([pairings[1:], res],open(fname, "w"))

# calculate dot-bracket annotation and store in pickle file
fname = "dotbracket.p"
dotbr,ss = bb.dot_bracket(pairings,res)
pickle.dump([dotbr[1:], res],open(fname, "w"))

# Calculate torsion angles
fname = "angles.p"
print "# calculate %s" % fname
angles,res = bb.backbone_angles(traj,topology=top)
pickle.dump([angles[1:],res],open(fname, "w"))

# calculate couplings and save to pickle
fname = "couplings.p"
print "# calculate %s" % fname
couplings,res = bb.jcouplings(traj,topology=top)
pickle.dump([couplings[1:],res],open(fname, "w"))

# calculate couplings and save to pickle
fname = "gvec.p"
print "# calculate %s" % fname
gvec,seq = bb.dump_gvec(traj,topology=top)
pickle.dump([gvec[1:],seq],open(fname, "w"))

# Loaded reference 2K0C.pdb
# Loaded target trajectory.dcd

# calculate rmsd.p
# found 294 atoms in common
# calculate pairs.p

# Loading trajectory.dcd

# calculate angles.p

# Loading trajectory.dcd

# calculate couplings.p

# Loading trajectory.dcd

```

```
# calculate gvec.p
# Loading trajectory.dcd
```

2 Figure 1: RMSD, eRMSD, annotation

Here, we plot the eRMSD and RMSD over time as shown in figure 1. We color the different data points according to the annotation: blue if the stem is formed and red if the stem is formed and the non-canonical interaction between U6-G9 is present. The first step is to read the pickles:

```
In [1]: import pickle

# read ermsd pickle
fname = "ermsd.p"
print "# reading pickle %s" % fname,
ermsd = pickle.load(open(fname, "r"))
print " - shape ", ermsd.shape

# Read RMSD pickle
fname = "rmsd.p"
print "# reading pickle %s" % fname,
rmsd = pickle.load(open(fname, "r"))
print " - shape ", rmsd.shape

# Read annotation pickle
fname = "pairs.p"
print "# reading pickle %s" % fname,
pairings,res = pickle.load(open(fname, "r"))
print " - shape ", len(pairings)

# Read dot-bracket pickle
fname = "dotbracket.p"
print "# reading pickle %s" % fname,
dotbr,res = pickle.load(open(fname, "r"))
print " - shape ", len(dotbr)

# reading pickle ermsd.p - shape (20000,)
# reading pickle rmsd.p - shape (20000,)
# reading pickle pairs.p - shape 20000
# reading pickle dotbracket.p - shape 20000
```

Now we create a list of length n=20000, where n is the length of the simulation. The entry is 2 if the stem is formed and SWt present, 1 if stem is formed and 0 otherwise.

```
In [2]: # bin structures according to annotation:
# bins_anno[j] = 2 if stem is formed and SWt present
# bins_anno[j] = 1 if stem is formed
# bins_anno[j] = 0 otherwise
bins_anno = [0]*len(pairings)
for j in range(0,len(pairings)):
    # if the stem is fully formed add 1
    if(dotbr[j] == "(((((...))))"):
        bins_anno[j] += 1
```

```

# search through the list if SWt between U6 and G9 is present.
for p in range(len(pairings[j][0])):
    res1 = res[pairings[j][0][p][0]]
    res2 = res[pairings[j][0][p][1]]
    interaction = pairings[j][1][p]
    if(res1=="U_6_0" and res2=="G_9_0" and interaction=="SWt"):
        bins_anno[j] += 1

```

We are now ready to plot the data: time series and histogram on the right.

```

In [4]: # Import numpy, matplotlib and seaborn.
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")

# define colors: gray, blue and red
flatui = [ "#95a5a6", "#3498db", "#e74c3c" ]
cmap = sns.color_palette(flatui)
col = [flatui[j] for j in bins_anno]

# define figure size
plt.figure(figsize=(7,6))
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.97, top=0.97,
                    wspace=None, hspace=None)
ax1 = plt.subplot2grid((2, 3), (1, 0), colspan=2)

# plot eRMSD over time
xx = np.linspace(0,180,ermsd.shape[0])
ax1.scatter(xx,ermsd,c=col,s=1)

# set limits and labels
ax1.set_xlim(-1,181)
ax1.set_ylabel("eRMSD from native")
ax1.set_ylim(0,2.5)
ax1.set_xlabel(r"Time ( $\mu$ s)")

# Define new axis and do histograms for the three categories
ax2 = plt.subplot2grid((2, 3), (1, 2))
bins = np.linspace(0,2.5,100)
hh1,ee1 = np.histogram([ermsd[e] for e in range(ermsd.shape[0]) if bins_anno[e]==0],bins=bins)
hh2,ee2 = np.histogram([ermsd[e] for e in range(ermsd.shape[0]) if bins_anno[e]==1],bins=bins)
hh3,ee3 = np.histogram([ermsd[e] for e in range(ermsd.shape[0]) if bins_anno[e]==2],bins=bins)
# do horizontal barplot with left padding
ax2.barh(0.5*(ee1[1:]+ee1[:-1]),hh1,color=flatui[0],height=0.026,linewidth=0)
ax2.barh(0.5*(ee1[1:]+ee1[:-1]),hh2,color=flatui[1],height=0.026,left=hh1,linewidth=0)
ax2.barh(0.5*(ee1[1:]+ee1[:-1]),hh3,color=flatui[2],height=0.026,left=hh1+hh2,linewidth=0)
# set labels, limits
ax2.set_xlabel("Count")
ax2.set_ylim(0,2.5)
# draw line at eRMSD = 0.7
ax2.axhline(0.7,ls='--',c='k',lw=1)

# now do the same as above, but for RMSD
ax3 = plt.subplot2grid((2, 3), (0, 0), colspan=2)

```

```

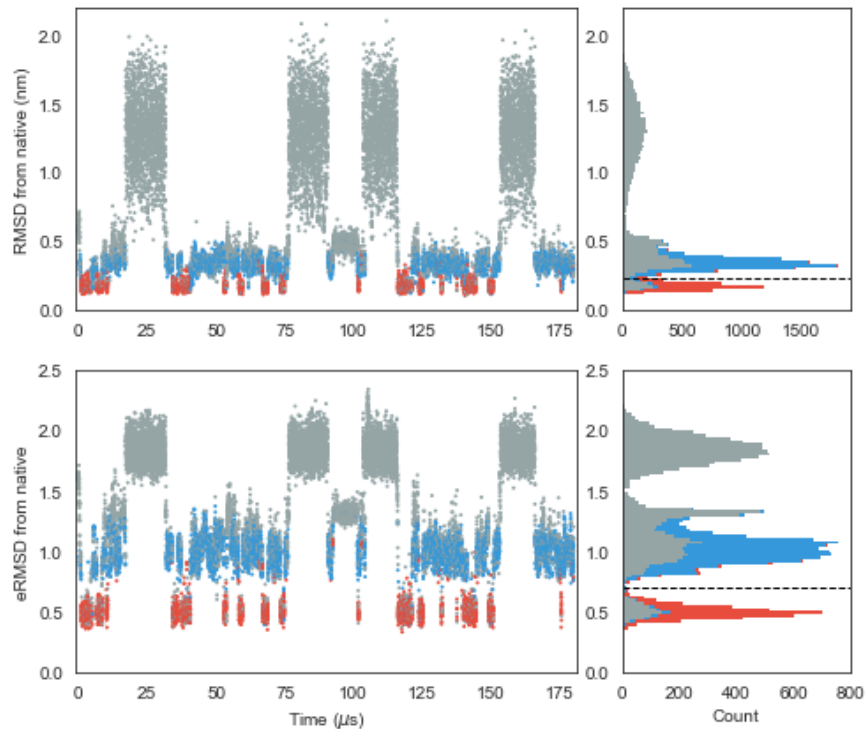
# plot time series
ax3.scatter(xx,rmsd,c=col,s=1)
ax3.set_xlim(-1,181)
ax3.set_ylim(0,2.2)
ax3.set_ylabel("RMSD from native (nm)")

# histogram
ax4 = plt.subplot2grid((2, 3), (0, 2))
bins = np.linspace(0,2.2,100)
hh1,ee1 = np.histogram([rmsd[e] for e in range(ersmd.shape[0]) if bins_anno[e]==0],bins=bins)
hh2,ee2 = np.histogram([rmsd[e] for e in range(ersmd.shape[0]) if bins_anno[e]==1],bins=bins)
hh3,ee3 = np.histogram([rmsd[e] for e in range(ersmd.shape[0]) if bins_anno[e]==2],bins=bins)
ax4.barh(0.5*(ee1[1:]+ee1[:-1]),hh1,color=flatui[0],height=0.025,linewidth=0)
ax4.barh(0.5*(ee1[1:]+ee1[:-1]),hh2,color=flatui[1],height=0.025,left=hh1,linewidth=0)
ax4.barh(0.5*(ee1[1:]+ee1[:-1]),hh3,color=flatui[2],height=0.025,left=hh1+hh2,linewidth=0)

# set limits and draw line at 0.23 nm
ax4.axhline(0.23,ls='--',c='k',lw=1)
ax4.set_ylim(0,2.2)

plt.show()
#plt.savefig("figure1.png",dpi=600)

```



3 Figure 2: Transition Paths.

This notebook shows how to extract the transition pathways from the MD trajectory, thereby allowing to understand what is the nature and order of events leading to folding. In particular, we here study the formation of the Watson-Crick base-pairs in the stem and of the Sugar-Watson interaction, characterized by the $G9\chi$ angle in the unusual syn conformation.

First, we identify transition paths (TP) from folded to unfolded (and vice-versa) as described in “How Fast-Folding Proteins Fold, Science, 2011”. In brief, we consider folding/unfolding transitions only when the system transitions fully between unfolded (eRMSD > 1.9) and folded (eRMSD < 0.6) substates. In order to define transition paths, we analyze the trajectory forward in time, and assign folded/unfolded states depending on the last state visited. If, as an example, the trajectory starts from folded (eRMSD < 0.6), all the frames will be considered as folded until the unfolded state (eRMSD > 1.9) is visited. We then analyze the trajectory backwards in time. Frames that are assigned to one state (e.g., unfolded) in the forward direction and a different state (e.g., folded) in the reverse direction belong to the transition path connecting the two states.

```
In [61]: import pickle
import numpy as np
# read ermsd from pickle
fname = "ermsd.p"
print "# reading pickle %s" % fname,
ermsd = pickle.load(open(fname, "r"))
print " shape ", ermsd.shape
print "####"
# define thresholds for folded and unfolded states, in eRMSD values.
thr_u = 1.9
thr_f = 0.6

ll = ermsd.shape[0]

# make assignments of folded/unfolded.
# Initial values are set as -1 (unininitialized).
# If the first/last eRMSD is between the two cutoffs,
# the state cannot be assigned
forward = -np.ones(ll)
backward = -np.ones(ll)
folded_idx = []
unfolded_idx = []
# loop over the trajectory
for j in range(ll):
    # fill forward. fw is < 0 if eRMSD < thr_f and fw > 1 if eRMSD > thr_u
    fw = (ermsd[j]-thr_f)/(thr_u-thr_f)
    if(fw>=1):
        forward[j] = 1
        unfolded_idx.append(j)
    elif(fw<=0):
        forward[j] = 0
        folded_idx.append(j)
    else:
        forward[j] = forward[j-1]

# fill backwards
k = ll-j-1
bw = (ermsd[k]-thr_f)/(thr_u-thr_f)
if(bw>=1):
```

```

        backward[j] = 1
    elif(bw<=0):
        backward[j] = 0
    else:
        backward[j] = backward[j-1]

# reverse list
backward = backward[::-1]

# create list of lists with indexes identifying
# transition paths.
tps = []
tmp = []
for j in range(11):
    if(forward[j]+backward[j]==1):
        tmp.append(j)
    else:
        if(len(tmp)!=0):
            tps.append(tmp)
            tmp = []
if(len(tmp)!=0): tps.append(tmp)

# establish whether is folding or unfolding.
# print info and check that the state before/after the TP
# is unfolded or folded.

for el in tps:
    if(el[0] != 0):
        if(ermsd[el[0]-1] >= thr_u):
            assert(ermsd[el[-1]+1]<= thr_f)
            print "# Folding trajectory: number of frames %d " % len(el)
        if(ermsd[el[0]-1] <= thr_f):
            assert(ermsd[el[-1]+1]>= thr_u)
            print "# Unfolding trajectory: number of frames %d " % len(el)
print "#####"
print "# I have found %d folding/unfolding transitions " % len(tps)

# reading pickle ermsd.p shape (20000,)
####
# Unfolding trajectory: number of frames 719
# Folding trajectory: number of frames 279
# Unfolding trajectory: number of frames 169
# Folding trajectory: number of frames 1225
# Unfolding trajectory: number of frames 164
# Folding trajectory: number of frames 34
# Unfolding trajectory: number of frames 294
# Folding trajectory: number of frames 1069
#####
# I have found 8 folding/unfolding transitions

```

Now we monitor base-pair formation and the χ angle in G9 during folding and unfolding pathways. We read these information from pickle files. The array *angles* has dimensions $(n,m,7)$, where $n=20000$ is the number of frames, $m=14$ is the number of nucleotides and the third index runs over the seven torsion angles $(\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \chi)$

```

In [25]: # read angles pickle
fname = "angles.p"
print "# reading pickle %s" % fname,
angles,res = pickle.load(open(fname, "r"))
print " - shape", angles.shape

# Read annotation pickle
fname = "pairs.p"
print "# reading pickle %s" % fname,
pairings,res = pickle.load(open(fname, "r"))
print " - shape ", len(pairings)

# reading pickle angles.p - shape (20000, 14, 7)
# reading pickle pairs.p - shape 20000

```

We now define a function that reads the list of base-pairs. This function will be used later for plotting, and can be used to count the number of native base-pair interaction, as shown below.

```

In [62]: # this function loops over the indexes and read the annotation relative
# to base-base interactions in the stem. It returns an array (q,6),
# where q is the length of the list idxs.
# The value of the entry is defined as follows:
# 1.0 if the WC base-pair (or SWt) is formed in the stem
# 0.0 if there are no interactions
# -1.0 if there is a non-native base pair
def count_stem(pairs,idxs):
    pairs = {0:13,1:12,2:11,3:10,4:9,5:8}
    wcpairs = np.zeros((len(idxs),6))
    for j,i in enumerate(idxs):
        for p in range(len(pairings[i][0])):
            res1 = pairings[i][0][p][0]
            res2 = pairings[i][0][p][1]
            interaction = pairings[i][1][p]
            if(res1 not in pairs): continue
            if(res2 != pairs[res1]): continue

            if(res1==5):
                # if is
                if(interaction == "SWt"):
                    wcpairs[j,res1] = 1.0
                else:
                    wcpairs[j,res1] = -1.0
            else:
                if(interaction=="WCc" or interaction == "WWc"):
                    wcpairs[j,res1] = 1.0
                else:
                    wcpairs[j,res1] = -1.0
    return wcpairs

stem_fold = count_stem(pairings,folded_idx)
stem_unfold = count_stem(pairings,unfolded_idx)

# sum elements where stem_fold is 1 to count interactions.
# The last column refers to the SWt interaction, and it is

```



```

# not considered when counting Watson-Crick interactions
number_wc_fold = np.sum(stem_fold[:, :-1]>0,axis=1)
number_wc_unfold = np.sum(stem_unfold[:, :-1]>0,axis=1)

print "Average number of formed WC pairs in folded %4.2f" % np.average(number_wc_fold)
print "Average number of formed WC pairs in unfolded %4.2f" % np.average(number_wc_unfold)

```

Average number of formed WC pairs in folded 5.00
Average number of formed WC pairs in unfolded 0.00

For each frame in the TP, we calculate i) which native base-pairs are formed and ii) the value of the chi angle in G9. The time evolution of these quantities over one TP is plotted below. Formed base pairs are shown in green, absent in gray and non-native base pairs in red.

```

In [198]: # loop over TPs and plot
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

colors = ["#980002", "#7d7f7c", "#0b4008"]
mycm = ListedColormap(sns.color_palette(colors))
lst = [1]
for i in lst:

    oo = 75
    nidxs = range(tps[i][0]-oo, tps[i][-1]+oo)
    stem_tp = count_stem(pairings, nidxs)
    #fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(3.5, 3.4), sharex=True)
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(3.5, 3.4), sharex=True)
    # plot heatmap

    g = sns.heatmap(stem_tp.T, cbar=False, ax=ax1, cmap=mycm, \
                    yticklabels=["G1-C14", "G2-C13", "C3-G12", "A4-U11", "U5-G10", "U6-G9"], \
                    lw=0.0, alpha=0.9, rasterized=True, edgecolor=None)

    #set nice tick labels
    #g.set_yticklabels(g.get_yticklabels(), rotation=0)
    ax1.set_ylim(0, 6.0)
    ax1.set_xticks([])

    # overlay value of chi in G9
    aa = np.copy(angles[nidxs, 8, 6])
    # plot in 0-2pi range
    aa[np.where(aa<0.0)] += 2.*np.pi
    # convert from rad to deg
    aa *= (180./np.pi)
    # plot
    ax2.plot(np.arange(len(nidxs)), aa, c='k', lw=1.2)

    # set limits
    ax2.set_ylim(20, 320)
    ax2.set_ylabel(r"\chi$ G9 (deg)")
    ax2.set_yticks([60, 120, 180, 240, 300])

```

```

l1 = np.linspace(0,2500,6)
l11 = (l1)/9. + oo
ax2.set_xticks(l11)
l12 = np.linspace(tps[i][0],tps[i][-1],6)
print l12
ax2.set_xticklabels(["%3.1f" % (x*0.009) for x in l12],rotation=0)
ax2.set_xlabel("Time ( $\mu$ s)")

# draw lines at the beginning and end of TP
#ax1.axvline(oo,color='y',lw=2.5,zorder=10,ls="--")
ax1.fill([0,0,oo,oo],[0,6,6,0],zorder=10,hatch="//",edgecolor='w',fc='none',alpha=0.5)
ax2.fill([0,0,oo,oo],[0,360,360,0],zorder=10,hatch="//",edgecolor='0.5',fc='none',alpha=0)

ax1.fill([oo+len(tps[i]),oo+len(tps[i]),2*oo+len(tps[i]),2*oo+len(tps[i])),\
        [0,6,6,0],zorder=10,hatch="//",edgecolor='w',fc='none',alpha=0.5)

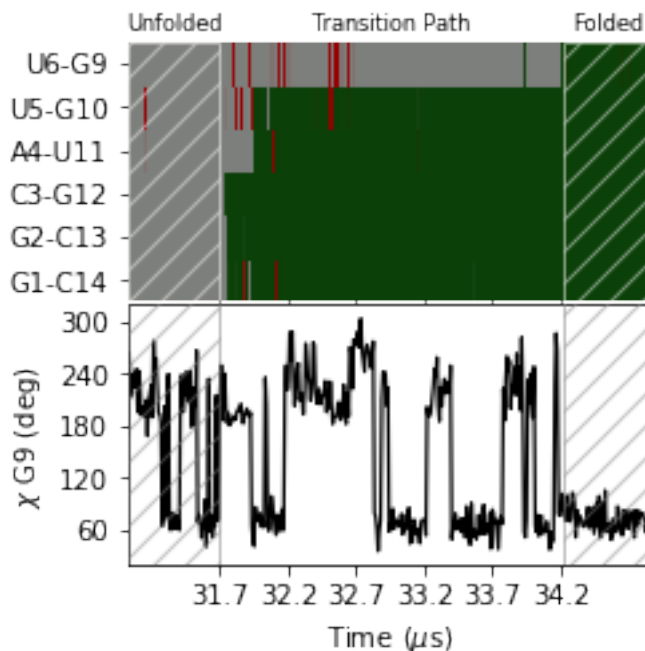
ax2.fill([oo+len(tps[i]),oo+len(tps[i]),2*oo+len(tps[i]),2*oo+len(tps[i])),\
        [0,360,360,0],zorder=10,hatch="//",edgecolor='0.5',fc='none',alpha=0.5)

#ax1.axvline(len(tps[i]) + oo,color='y',ls="--",lw=2.5)
#ax2.axvline(oo,color='y',lw=2.5,zorder=10,ls="--")
#ax2.axvline(len(tps[i]) + oo,color='y',lw=2.5,ls="--")
ax1.text(oo/2,6.25,"Unfolded",ha='center',fontsize=8)
ax1.text(1.5*oo+len(tps[i]),6.25,"Folded",ha='center',fontsize=8)
ax1.text(oo+0.5*(len(tps[i])),6.25,"Transition Path",ha='center',fontsize=8)
fig.subplots_adjust(left=0.2, bottom=0.15, right=0.98, top=0.95, wspace=0.0, hspace=0.01)
plt.show()

#plt.savefig("figure_02a.png",dpi=600)
#plt.close()

```

[3523. 3578.6 3634.2 3689.8 3745.4 3801.]



Finally, we quantify the order of the events in folding by calculating the two following quantities: 1. the fraction of formed WC-pairs ($q1$) 2. the normalized quantity $q2 = 0.5 * (1 + \cos(\chi - 1.104))$, that has a maximum $q2 = 1$ when $\chi = 1.014rad$.

We then average $q1, q2$ over the TP, so that quantities that reach a native-like value (i.e. 1) early during folding have a high value and those that form late get a low value. Here, we can see that the Watson-Crick base pairs form early in folding, while the transition of the χ angle to syn with the formation of the SWt base-pairs occurs later.

```
In [187]: averages = []
averages2 = []
for i in range(len(tps)):

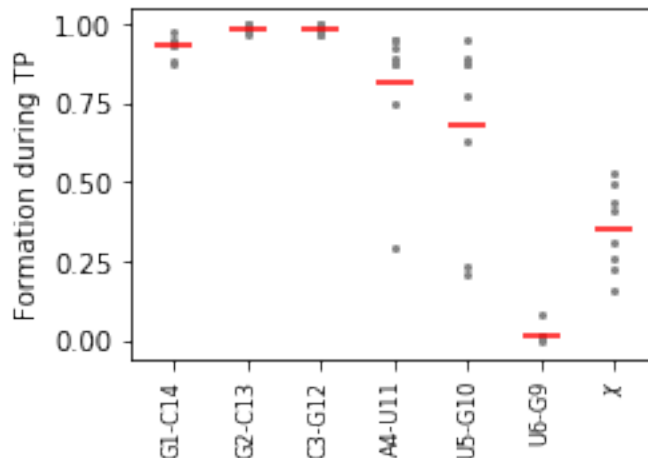
    # count the base-pair interaction in stem
    stem_tp = count_stem(pairings, tps[i])

    # calculate the fraction of formed WC pairs in stem
    q1 = np.sum(stem_tp[:, :-1], axis=1)/5.
    # calculate the cosine distance from the syn conformation
    q2 = 0.5*(1.+ np.cos(angles[tps[i], 8, 6]-1.104) )
    averages.append([np.average(q1), np.average(q2)])

    q3 = np.average(stem_tp[:, 0]>0)
    q4 = np.average(stem_tp[:, 1]>0)
    q5 = np.average(stem_tp[:, 2]>0)
    q6 = np.average(stem_tp[:, 3]>0)
    q7 = np.average(stem_tp[:, 4]>0)
    q8 = np.average(stem_tp[:, 5]>0)
    averages2.append([q3, q4, q5, q6, q7, q8, np.average(q2)])

averages = np.array(averages)
averages2 = np.array(averages2)
averages3 = np.average(np.array(averages2), axis=0)

fig, ax1 = plt.subplots(1, 1, figsize=(3.5, 2.5))
for j in range(averages2.shape[1]):
    ax1.scatter([j]*averages2.shape[0], averages2[:, j], c='0.5', s=4, marker="o")
ax1.scatter(range(averages2.shape[1]), averages3, c='r', marker="_", s=200)
#ax1.plot([0, 1], [0, 1], c='k')
#ax1.set_aspect(1.)
plt.xticks([0, 1, 2, 3, 4, 5, 6], ["G1-C14", "G2-C13", "C3-G12", "A4-U11", "U5-G10", "U6-G9", r"$\chi$"], \
           rotation=90, fontsize=8)
ax1.set_yticks([0, 0.25, 0.5, 0.75, 1.0])
plt.ylabel("Formation during TP")
#ax1.set_xlabel("q1 - Watson-Crick pairs", fontsize=10)
#ax1.set_ylabel(r"q2 - $\chi$ angle, G9", fontsize=10)
#plt.show()
fig.subplots_adjust(left=0.2, bottom=0.25, right=0.98, top=0.98, wspace=0.1, hspace=0.1)
plt.savefig("figure_02b.png", dpi=600)
plt.close()
```



4 Figure 3: Torsion angles and scalar couplings.

In this notebook we show how to calculate torsion angles and plot their distribution. In the last part, we calculate 3J scalar couplings and compare with experimental data.

```
In [2]: import numpy as np
import pickle
```

```
# read eRMSD from pickle
fname = "ermsd.p"
print "# reading pickle %s" % fname,
ermsd = pickle.load(open(fname, "r"))
print " shape ", ermsd.shape

# consider structures where eRMSD is smaller than 1.5
qq = np.where(ermsd<1.5)[0]

# read torsion angles from pickle
fname = "angles.p"
print "# reading pickle %s" % fname,
angles, res = pickle.load(open(fname, "r"))
print " shape ", angles.shape

# calculate couplings and save to pickle
fname = "couplings.p"
print "# reading pickle %s" % fname,
couplings, res = pickle.load(open(fname, "r"))
print " - shape ", couplings.shape

# read file with experimental J-couplings
from barnaba.definitions import couplings_idx
```

```

fh = open("jcouplings.dat")
key = couplings_idx.keys()
exp = [[] for z in range(len(res))]
for line in fh:
    if("#" in line): continue
    ii = int(line.split()[0])-1
    exp[ii].append([line.split()[1],float(line.split()[2])])
fh.close()

# reading pickle ermsd.p shape (20000,)
# reading pickle angles.p shape (20000, 14, 7)
# reading pickle couplings.p - shape (20000, 14, 12)

In [4]: import matplotlib.pyplot as plt
import seaborn as sns
import barnaba as bb

# now do the plot. set ticks and colors
sns.set_style("ticks")
sns.set_context("paper")
colors = [ "#FFD300", "#009F6B", "#000000" ]
cols = sns.color_palette("colorblind", 3)

# define bins for histogram
bins = np.linspace(0,360,100)
# plot angles in nucleotides 6,7 and 9
labs = ["U6", "U7", "G9"]
rr = [5,6,8]

# plot beta, gamma, delta and epsilon angles
angs = [1,2,3,4]
nams = [r"$\beta$", r"$\gamma$", r"$\delta$", r"$\epsilon$"]

# these two lists are used to plot specific scalar couplings of interest
myk = [ ["1H5P", "2H5P", "C4Pb"], \
        ["2H5H4", "1H5H4"], \
        ["H1H2", "H2H3", "H3H4"], \
        ["H3P", "C4Pe"] ]
mykn = [ ["1H5-P", "2H5-P", "C4-P"], \
         ["2H5-H4", "1H5-H4"], \
         ["H1'-H2'", "H2'-H3'", "H3'-H4'"], \
         ["H3-P", "C4-P+1"] ]

# define figure
plt.figure(figsize=(9,12))
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.87, top=0.97,
                    wspace=0.1, hspace=0.23)

# plot
for k,a in enumerate(angs):
    ax1 = plt.subplot2grid((4, 2), (k, 0), colspan=1)
    for j,r in enumerate(rr):
        # convert to deg and move to 0:360 range

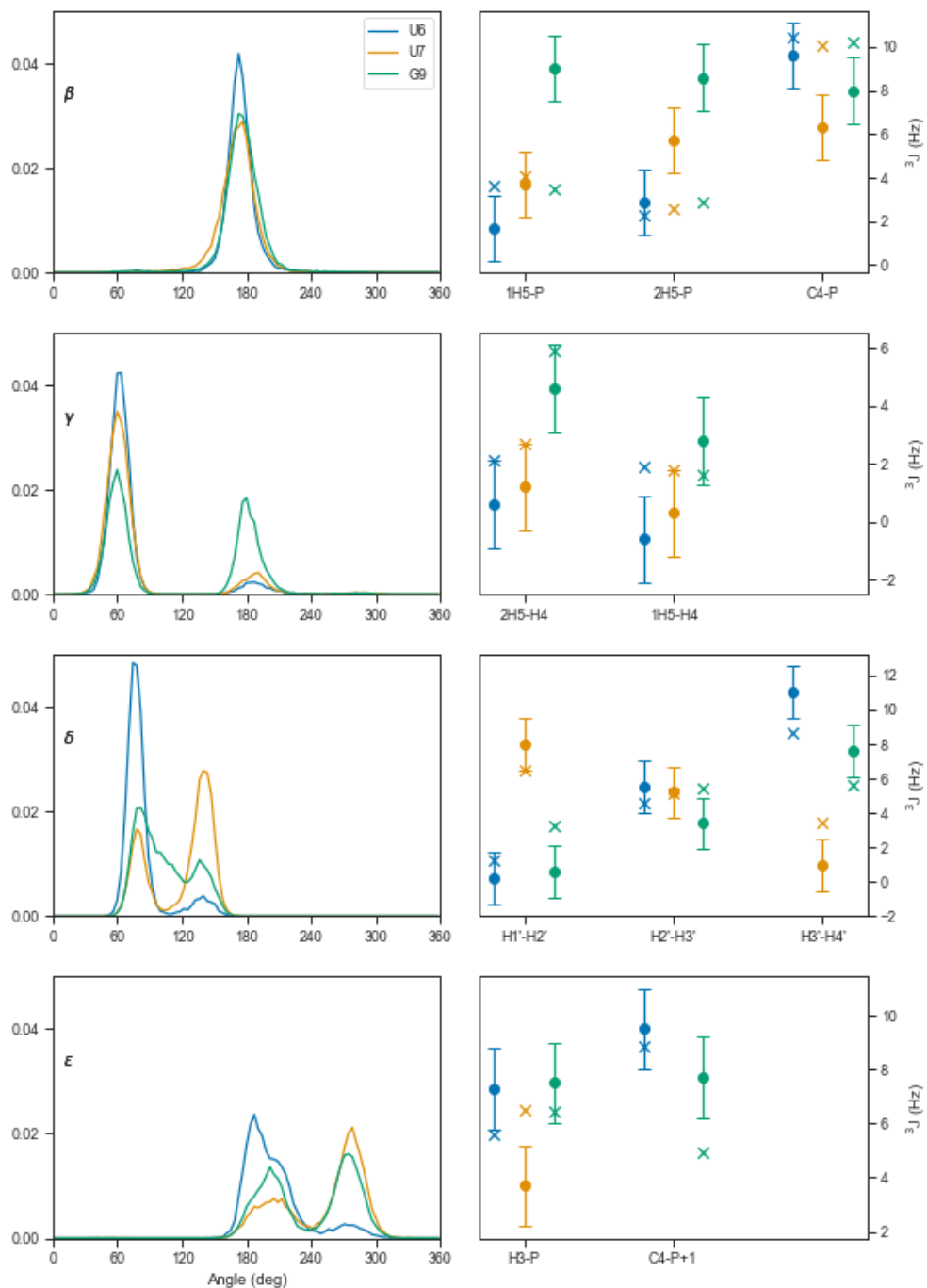
```

```

dd = angles[qq,r,a]*(180./np.pi)
dd[np.where(dd<0.0)] += 360
# make histogram
hh1,ee1 = np.histogram(dd,bins=bins,density=True)
xx = 0.5*(ee1[1:]+ee1[:-1])
# set nice limits and ticks
plt.xlim(0,360)
plt.ylim(0,0.05)
plt.xticks([0,60,120,180,240,300,360])
plt.yticks([0,0.02,0.04])
plt.text(10,0.033,nams[k])
# plot histogram. Make labels for legend
if(k==0):
    plt.plot(xx,hh1,c=cols[j],label=labs[j])
else:
    plt.plot(xx,hh1,c=cols[j])
if(k==0):
    plt.legend()
if(k==3):
    plt.xlabel("Angle (deg)")
ax2 = plt.subplot2grid((4, 2), (k, 1), colspan=1)
# now plot experimental data. This is done manually,
# because we hand-picked some of the couplings
xpos = 0
plt.xlim(-0.5,12.5)
xt = []
xt1 = []
ax2.yaxis.tick_right()
ax2.yaxis.set_label_position("right")
ax2.set_ylabel("$\sim 3J$ (Hz)")
for z,c in enumerate(myk[k]):
    xt.append(xpos+1)
    xt1.append(mykn[k][z])
    for j,r in enumerate(rr):
        for exps in exp[r]:
            if(exps[0] == c):
                # scatter experimental data (cross)
                (_, caps, _) = plt.errorbar(xpos,exps[1], yerr=1.5,c=cols[j],fmt='o'
                    ,capsize=4,markersize=6)
                for cap in caps:
                    cap.set_markeredgewidth(1.0)
                # scatter average from simulation
                yyd = couplings[qq,r,key.index(c)]
                plt.scatter(xpos,np.average(yyd),c=cols[j],marker="x",s=45)
            xpos += 1
        xpos += 2
plt.xticks(xt,xt1)

plt.show()

```



In []:

5 Figure 4: Cluster analysis.

In this notebook we show how to perform the cluster analysis shown in figure 4. We first read pickle files generated in 00_calculate

```

In [2]: import pickle
import numpy as np

# calculate eRMSD from native
fname = "ermsd.p"
print "# reading pickle %s" % fname,
ermsd = pickle.load(open(fname, "r"))
print " - ", ermsd.shape

# calculate RMSD from native
fname = "rmsd.p"
print "# reading pickle %s" % fname,
rmsd = pickle.load(open(fname, "r"))
print " - ", rmsd.shape

# Calculate G-vectors
fname = "gvec.p"
print "# reading pickle %s" % fname,
gvec,seq = pickle.load(open(fname, "r"))
print " - shape", gvec.shape

# remove samples with ermsd larger than 1.5 and reshape.
# the flattened array is later fed to the clustering routine
n = gvec.shape[0]
qq = np.where(ermsd<1.5)[0]
gvecs = gvec.reshape(n,-1)[qq]
print "# Gvec, new shape ", gvecs.shape

# reading pickle ermsd.p - (20000,)
# reading pickle rmsd.p - (20000,)
# reading pickle gvec.p - shape (20000, 14, 14, 4)
20000
# Gvec, new shape (13958, 784)

```

Now we call DBSCAN function from sklearn, giving as input the list of g-vectors. The parameters eps and min_samples are be system/simulation dependent. As a rough rule-of-thumb, reasonable results are obtained by setting eps in the range 0.2-0.6. A pragmatic choice is to tune eps/min sample so as to obtain ~ 10 clusters with a intra-centroid (IC) eRMSD distance below 0.9.

```

In [7]: import barnaba.cluster as cc

# calculate clusters. Call the function dbscan and return list of labels and center indeces
new_labels, center_idx = cc.dbscan(gvecs,list(qq),eps=0.45,min_samples=70)

# write to pickle for later
pickle.dump([new_labels,center_idx],open("cluster.p", "w"))

# eps:1.684 min_samples:70 nclusters: 10
# silhouette score: 0.1315
# Avg silhouette: 0.3627
# assigned samples :8444 total samples:13958
# N size          max eRMSD (IC)          med eRMSD (IC) max eRMSD (centroid) med eRMSD (centroid) center

```



```

# 00 3326          0.864          0.468          0.615          0.384 01 16744
# 01 1056          0.997          0.654          0.764          0.583 02 6453
# 02 0994          0.884          0.467          0.617          0.383 05 10605
# 03 0693          0.898          0.522          0.666          0.434 08 5335
# 04 0614          0.813          0.477          0.572          0.387 07 14232
# 05 0597          0.945          0.550          0.733          0.460 06 6223
# 06 0535          0.963          0.604          0.711          0.519 00 8389
# 07 0334          0.828          0.510          0.597          0.427 04 7986
# 08 0295          0.836          0.550          0.648          0.471 03 988
# Write centroids to PDB files and cluster members to .xtc

```

One possible way to visualize the cluster is to perform a principal component analysis and make a scatter plot.

```

In [12]: # Do plots. Import matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
import matplotlib as mpl
from matplotlib.collections import PatchCollection
import matplotlib.patches as mpatches

# calculate PCA
v,w = cc.pca(gvecs,nevecs=3)
# Define figure and set aspect
fig, ax = plt.subplots(figsize=(6.3,4.8))
ax.set_aspect(1)

# calculate explained variance for the first two components
plt.xlabel(r'PC1 (%2.0f%s)' % (v[0]*100,"%"),fontsize=12)
plt.ylabel(r'PC2 (%2.0f%s)' % (v[1]*100-v[0]*100,"%"),fontsize=12)
print("# Explained variance PC1=%5.1f 2:=%5.1f 3=%5.1f" % (v[0]*100,v[1]*100,v[2]*100))

# define colors for clusters. Noise is gray point
cp = sns.color_palette("hls",n_colors=len(center_idx),desat=0.8)
colors = [cp[j-1] if(j!=0) else (0.77,0.77,0.77) for j in new_labels]
size = [0.12 if(j==0) else 0.0 for j in new_labels]
# scatterplot the noise
plt.scatter(w[:,0],w[:,1],s=size,c=colors,zorder=0)

# make circles in the center of the cluster
patches = []
new_labels = np.array(new_labels)
for i,k in enumerate(center_idx):
    plt.text(w[k,0],w[k,1],str(i+1),ha='center',va='center',fontsize=12)
    rr = np.sqrt(1.*len(np.where(new_labels==i+1)[0])/len(new_labels))
    circle = mpatches.Circle((w[k,0],w[k,1]), rr, ec='k')
    patches.append(circle)
p = PatchCollection(patches, cmap=mpl.cm.tab10, alpha=0.6)
p.set_array(np.arange(len(center_idx))+1)
ax.add_collection(p)

# set nice limits
ax.xaxis.label.set_size(6)

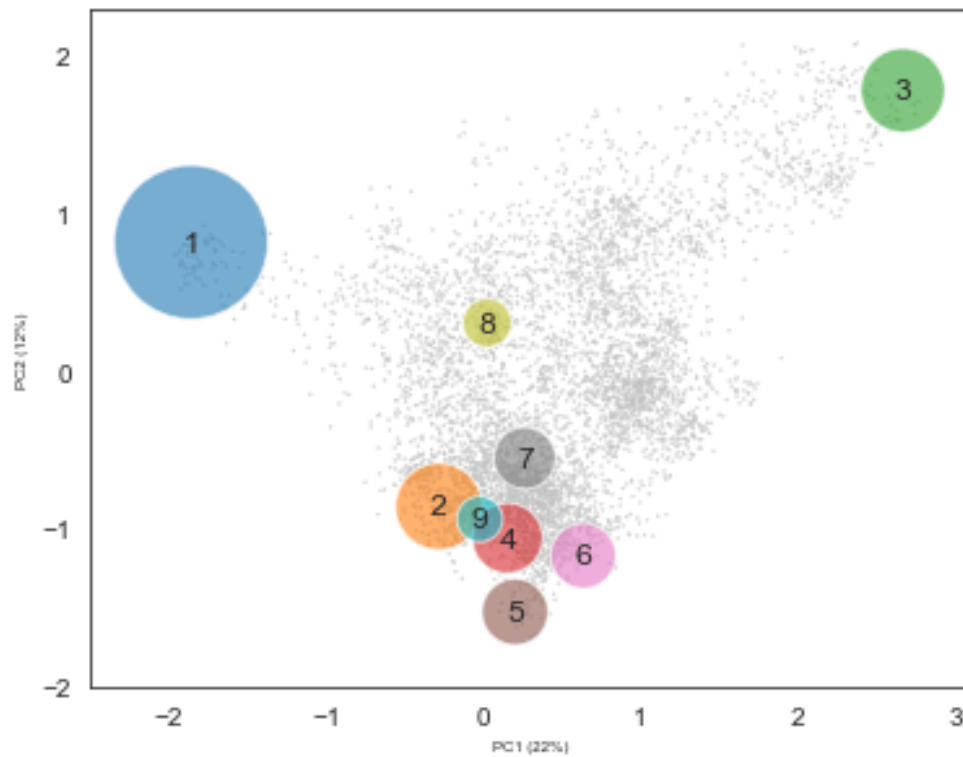
```

```

ax.yaxis.label.set_size(6)
ax.set_xlim([-2.5,3.1])
ax.set_ylim([-2.,2.3])
ax.set_xticks([-2,-1,0,1,2,3])
ax.set_yticks([-2,-1,0,1,2])
plt.savefig("clusters.png",dpi=600)
plt.close()
plt.show()

```

Cumulative explained variance of component: 1= 21.6 2:= 33.3 3= 40.0



As a last step, we analyze all the cluster individually. We save centroids to a PDB and cluster members to an xtc trajectory file. For each cluster, we calculate the ermsd and rmsd between centroid and cluster members.

```

In [14]: import mdtraj as md
import barnaba as bb
print("# Write centroids to PDB files and cluster members to .xtc")
top = "topology.pdb"
traj = "trajectory.dcd"
t = md.load(traj, top=top)
dd1 = []
dd2 = []
ll = []
for i,k in enumerate(center_idx):
    t[qq[k]].save_pdb("cluster_%03d.test.pdb" % (i))

```

```

idxs = [ii for ii, kk in enumerate(new_labels) if(kk==i+1)]
t[qq[idxs]].save_xtc("cluster_%03d.traj.xtc" % (i))
ll.append(100.*len(idxs)/(n))
ermsd_t = bb.ermsd("cluster_%03d.test.pdb" % (i), "cluster_%03d.traj.xtc" % (i),
topology=top)
rmsd_t = bb.rmsd("cluster_%03d.test.pdb" % (i), "cluster_%03d.traj.xtc" % (i),
topology=top)
dd1.append(ermsd_t)
dd2.append(rmsd_t)

# Write centroids to PDB files and cluster members to .xtc

# Loaded reference cluster_000.test.pdb
# Loaded target cluster_000.traj.xtc

# found 165 atoms in common

# Loaded reference cluster_001.test.pdb
# Loaded target cluster_001.traj.xtc

# found 165 atoms in common

# Loaded reference cluster_002.test.pdb
# Loaded target cluster_002.traj.xtc

# found 165 atoms in common

# Loaded reference cluster_003.test.pdb
# Loaded target cluster_003.traj.xtc

# found 165 atoms in common

# Loaded reference cluster_004.test.pdb
# Loaded target cluster_004.traj.xtc

# found 165 atoms in common

# Loaded reference cluster_005.test.pdb
# Loaded target cluster_005.traj.xtc

# found 165 atoms in common

# Loaded reference cluster_006.test.pdb
# Loaded target cluster_006.traj.xtc

```

```

# found 165 atoms in common

# Loaded reference cluster_007.test.pdb
# Loaded target cluster_007.traj.xtc

# found 165 atoms in common
# found 165 atoms in common

# Loaded reference cluster_008.test.pdb
# Loaded target cluster_008.traj.xtc

```

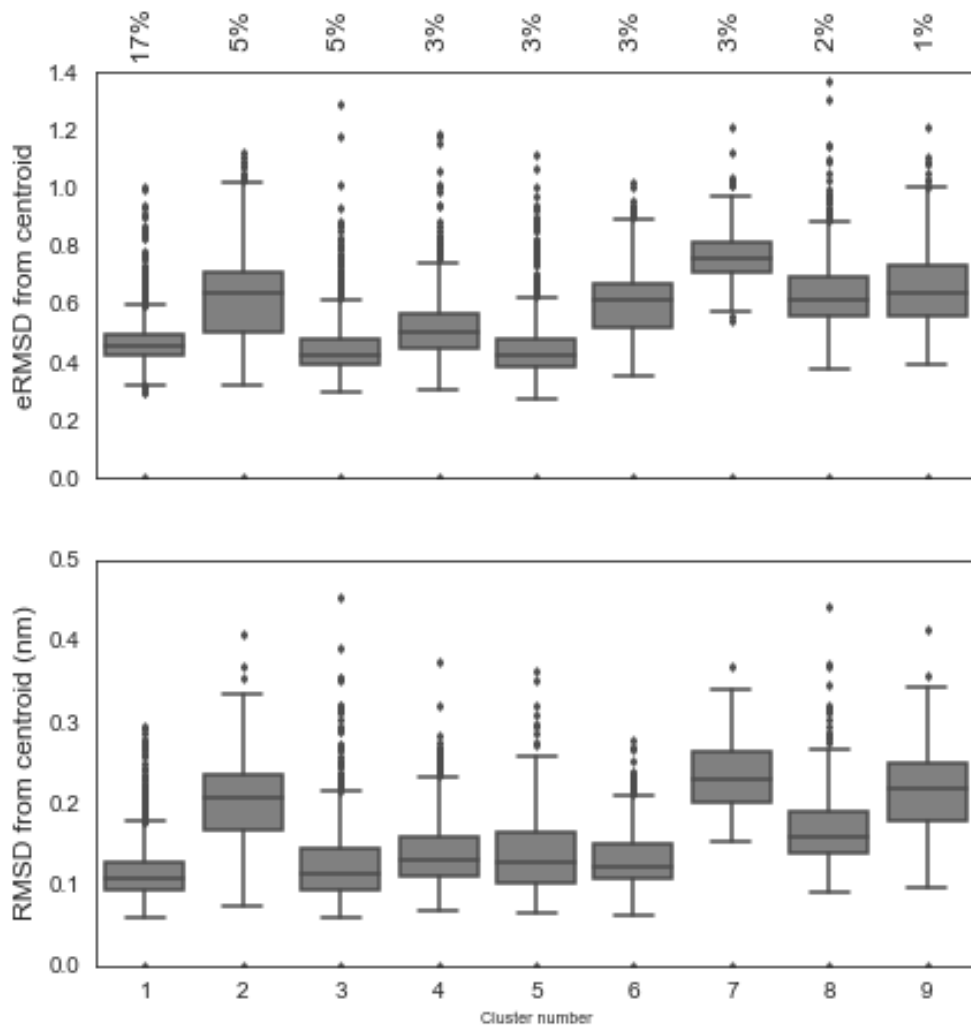
And now we box-plot the data.

```

In [25]: # define figure
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True, figsize=(6.2, 6.3))
# do the boxplot
ax1 = sns.boxplot(data=dd1, color='0.5', ax=ax1, fliersize=2.5)
ax2 = sns.boxplot(data=dd2, color='0.5', ax=ax2, fliersize=2.5)

# write percentages
for j in range(9):
    ax1.text(j, 1.5, "%4.0f%s" % (ll[j], "%"), ha="center", va="center", rotation=90, fontsize=12)
# set limits and labels
ax1.set_ylim(0, 1.4)
ax1.set_ylabel("eRMSD from centroid", fontsize="12")
ax2.set_ylim(0, 0.5)
ax2.set_ylabel("RMSD from centroid (nm)", fontsize="12")
ax2.set_xlabel("Cluster number", fontsize="7")
ax2.set_xticklabels(["1", "2", "3", "4", "5", "6", "7", "8", "9"])
plt.subplots_adjust(right=0.98, top=0.98)
#plt.savefig("cluster_stats.png", dpi=600)
#plt.close()
plt.show()

```



Finally, we produce the dynamic secondary structures. We start from the native

```
In [5]: import os
native = "2KOC"
cmd1 = "barnaba ANNOTATE --trj %s.pdb --top %s.pdb -o %s" % (native,native,native)
cmd2 = "barnaba SEC_STRUCTURE --ann %s.ANNOTATE.stacking.out %s.ANNOTATE.pairing.out -o %s" % (native,native,native)
os.system(cmd1)
os.system(cmd2)
```

Out[5]: 0

And we do the same for the first three clusters:

```
In [14]: top = "topology.pdb"
for i in range(3):
cmd1 = "barnaba ANNOTATE --trj cluster_00%d.traj.xtc --top %s -o c%d" % (i,top,i)
cmd2 = "barnaba SEC_STRUCTURE \
--ann c%d.ANNOTATE.stacking.out c%d.ANNOTATE.pairing.out -o c%d" % (i,i,i)
```

```

os.system(cmd1)
os.system(cmd2)
print cmd1

```

```

barnaba ANNOTATE --trj cluster_000.traj.xtc --top topology.pdb -o c0
barnaba ANNOTATE --trj cluster_001.traj.xtc --top topology.pdb -o c1
barnaba ANNOTATE --trj cluster_002.traj.xtc --top topology.pdb -o c2

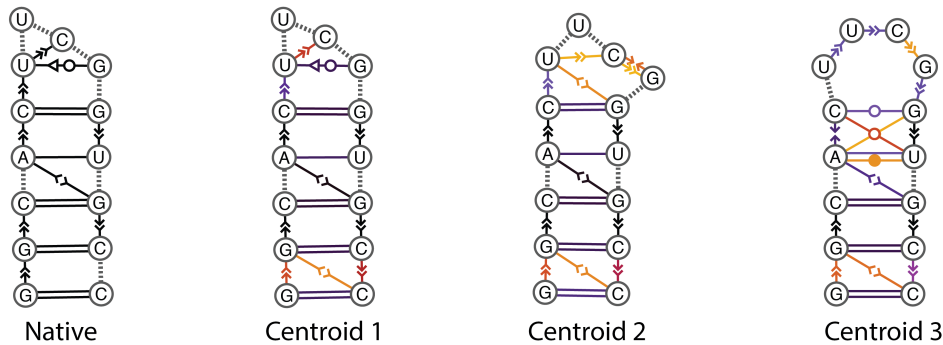
```

```
In [16]: %ls *.svg
```

```

2K0C.SEC_STRUCTURE_175steps.svg  c1.SEC_STRUCTURE_234steps.svg
c0.SEC_STRUCTURE_166steps.svg   c2.SEC_STRUCTURE_232steps.svg

```



6 Figure 6: ENM.

We here construct the elastic network model, calculate C2-C2 fluctuations and compare with experimental SHAPE data.

```

In [2]: %matplotlib inline
# import barnaba
import barnaba.enm as enm
import numpy as np

# fetch PDBs from rcsb.org. Many of them will be used for figure 7.
flist=["1C2X.pdb", "2GDI.pdb", "3DIG.pdb", "1Y26.pdb", "2A64.pdb", "1NBS.pdb", \
       "1YOQ.pdb", "1CSL.pdb", "1JZV.pdb"]

import os
for pdb in flist:
    cmd = "curl https://files.rcsb.org/download/%s > %s" % (pdb,pdb)
    os.system(cmd)

# Calculate ENM
enm_1C2X=enm.Enm("1C2X.pdb",cutoff=0.7,sele_atoms="AA",sparse=True)
enm_3DIG=enm.Enm("3DIG.pdb",cutoff=0.7,sele_atoms="AA",sparse=True)

# calculate C2-C2 fluctuations
fluc_C2_1C2X,res_list_1C2X=enm_1C2X.c2_fluctuations()
fluc_C2_3DIG,res_list_3DIG=enm_3DIG.c2_fluctuations()

```

```

# read experimental data
shape_1C2X=np.loadtxt('1C2X.shape',usecols=[2]).clip(0)
shape_3DIG=np.loadtxt('3DIG.shape',usecols=[2]).clip(0)

# Read (2570, 3) coordinates
# Using sparse matrix diagonalization
# Read (3809, 3) coordinates
# Using sparse matrix diagonalization

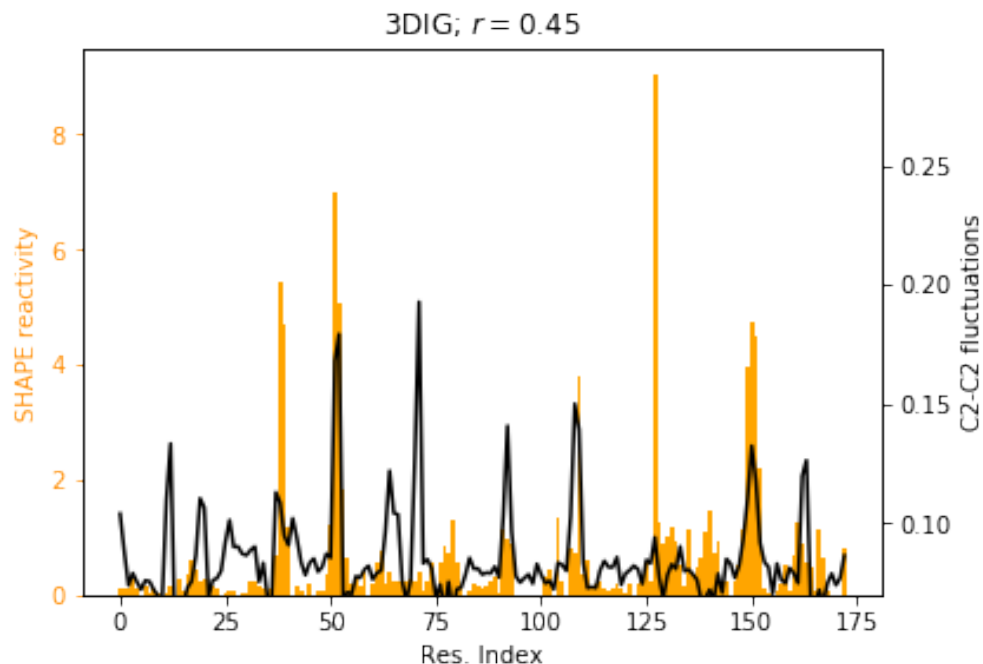
In [14]: from scipy.stats.stats import pearsonr
import matplotlib.pyplot as plt
import seaborn as sns

fig, ax1 = plt.subplots()
ax1.bar(np.arange(shape_3DIG.shape[0]-1),shape_3DIG[:-1],1,label='SHAPE',color='orange')
ax1.set_xlabel('Res. Index')
# Make the y-axis label, ticks and tick labels match the line color.
ax1.set_ylabel('SHAPE reactivity', color='darkorange')
ax1.tick_params('y', colors='darkorange')

ax2 = ax1.twinx()

ax2.plot(fluc_C2_3DIG[:-1],label='ENM',c='k')
ax2.set_ylabel('C2-C2 fluctuations', color='k')
ax2.tick_params('y', colors='k')
ax2.set_ylim(0.07,0.299)
fig.tight_layout()
plt.title('3DIG; $r$=%.2f$'%pearsonr(shape_3DIG[:-1],fluc_C2_3DIG[:-1])[0])
plt.savefig('ENM-SHAPE_3DIG.pdf',bbox_inches="tight")

```



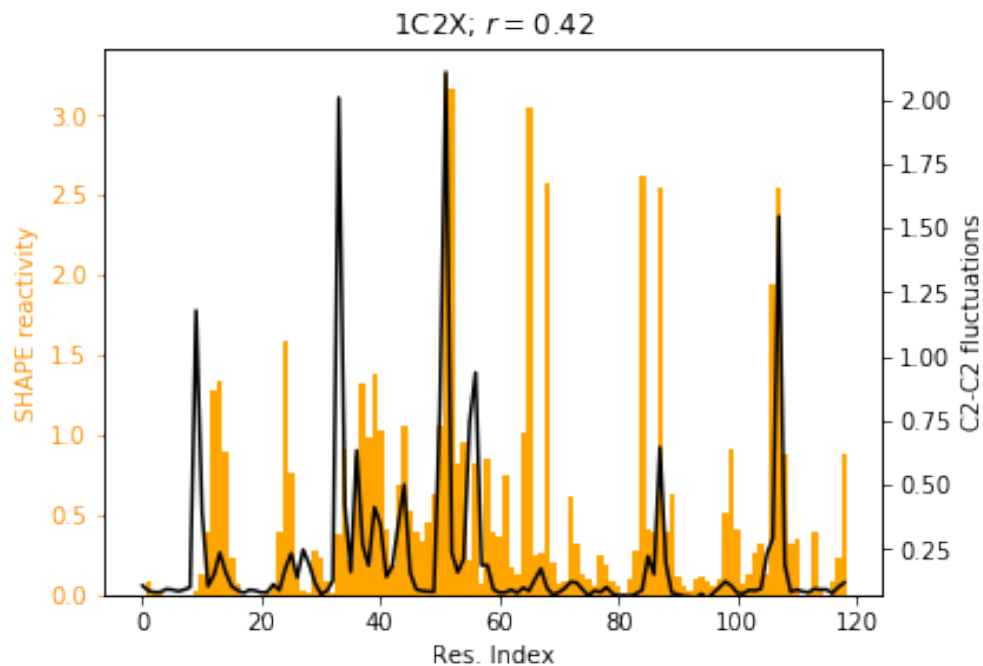
```

In [15]: fig, ax1 = plt.subplots()
ax1.bar(np.arange(shape_1C2X.shape[0]-1),shape_1C2X[:-1],1,label='SHAPE',color='orange')
ax1.set_xlabel('Res. Index')
# Make the y-axis label, ticks and tick labels match the line color.
ax1.set_ylabel('SHAPE reactivity', color='darkorange')
ax1.tick_params('y', colors='darkorange')

ax2 = ax1.twinx()

ax2.plot(fluc_C2_1C2X[:,],label='ENM',c='k')
ax2.set_ylabel('C2-C2 fluctuations', color='k')
ax2.tick_params('y', colors='k')
ax2.set_ylim(0.07,2.2)
fig.tight_layout()
plt.title('1C2X; $r=%.2f$'%pearsonr(shape_1C2X[:-1],fluc_C2_1C2X[:,])[0])
plt.savefig('ENM-SHAPE_1C2X.pdf',bbox_inches="tight")

```



In []:

7 Figure 7: ENM performances.

We fetch nine PDB from the protein data bank and calculate the time it takes to calculate the ENM and the C2-C2 fluctuations

```

In [24]: # import barnaba
import barnaba.enm as enm

```



```

import time
import numpy as np

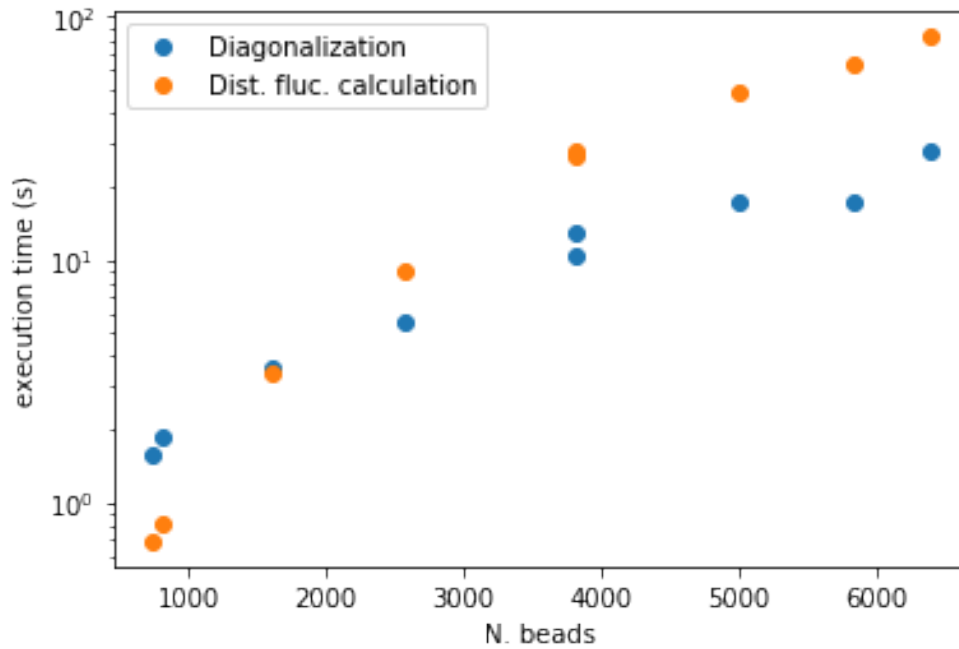
times=[]
n_beads_AA=[]
n_res=[]
for fname in flist:
    t0=time.time()
    enm_tmp=enm.Enm(fname,cutoff=0.7,sele_atoms="AA",sparse=True)
    t1=time.time()-t0
    fluc,r1=enm_tmp.c2_fluctuations()
    t2=time.time()-t0-t1
    n_beads_AA.append(enm_tmp.n_beads)
    n_res.append(len(r1))
    times.append([t1,t2])
t_AA_sparse=np.array(times)

# Read (2570, 3) coordinates
# Using sparse matrix diagonalization
# Read (3817, 3) coordinates
# Using sparse matrix diagonalization
# Read (3809, 3) coordinates
# Using sparse matrix diagonalization
# Read (1608, 3) coordinates
# Using sparse matrix diagonalization
# Read (6383, 3) coordinates
# Using sparse matrix diagonalization
# Read (5837, 3) coordinates
# Using sparse matrix diagonalization
# Read (4996, 3) coordinates
# Using sparse matrix diagonalization
# Read (739, 3) coordinates
# Using sparse matrix diagonalization
# Read (800, 3) coordinates
# Using sparse matrix diagonalization

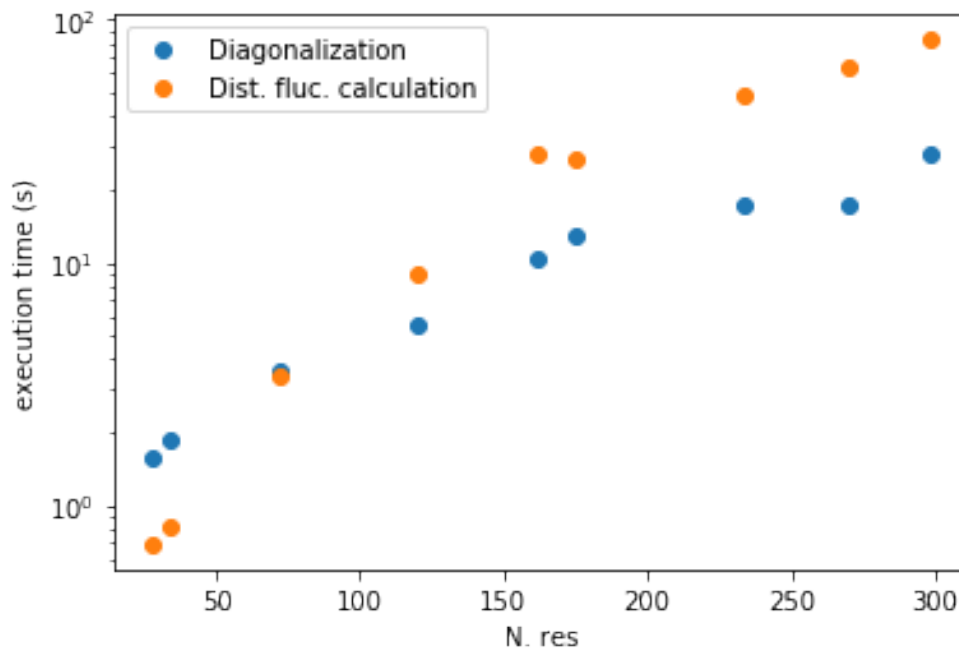
In [9]: import matplotlib.pyplot as plt
import seaborn as sns

plt.plot(n_beads_AA,t_AA_sparse[:,0],marker='o',ls='',label='Diagonalization')
plt.plot(n_beads_AA,t_AA_sparse[:,1],marker='o',ls='',label='Dist. fluc. calculation')
plt.ylabel('execution time (s)')
plt.xlabel('N. beads')
plt.legend()
plt.yscale('log')

```



```
In [10]: plt.plot(n_res,t_AA_sparse[:,0],marker='o',ls='',label='Diagonalization')
plt.plot(n_res,t_AA_sparse[:,1],marker='o',ls='',label='Dist. fluc. calculation')
plt.ylabel('execution time (s)')
plt.xlabel('N. res')
plt.legend()
plt.yscale('log')
```



```

In [12]: times=[]
         n_beads_AA=[]
         for fname in flist:
             t0=time.time()
             enm_tmp=enm.Enm(fname,cutoff=0.7,sele_atoms="AA",sparse=False)
             t1=time.time()-t0
             fluc,r1=enm_tmp.c2_fluctuations()
             t2=time.time()-t0-t1
             n_beads_AA.append(enm_tmp.n_beads)
             times.append([t1,t2])
         t_AA_vanilla=np.array(times)

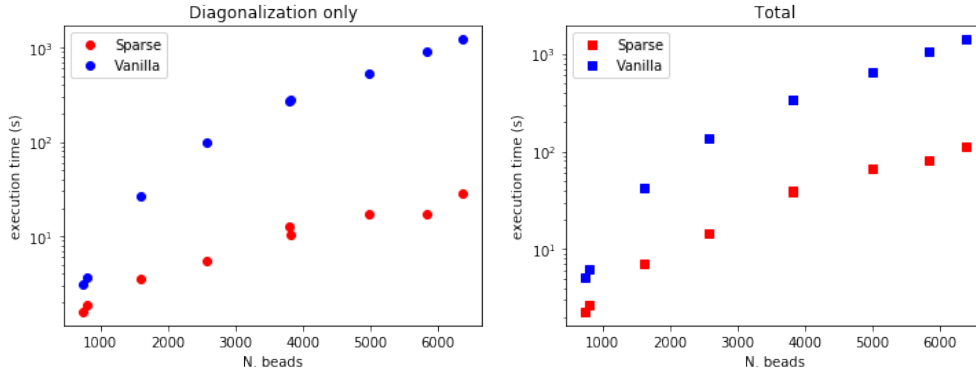
# Read (2570, 3) coordinates
# Read (3817, 3) coordinates
# Read (3809, 3) coordinates
# Read (1608, 3) coordinates
# Read (6383, 3) coordinates
# Read (5837, 3) coordinates
# Read (4996, 3) coordinates
# Read (739, 3) coordinates
# Read (800, 3) coordinates

In [13]: plt.figure(figsize=(12,4))
         plt.subplot2grid((1,2),(0,0))
         plt.title('Diagonalization only')
         plt.plot(n_beads_AA,t_AA_sparse[:,0],marker='o',ls='',label='Sparse',c='r')
         plt.plot(n_beads_AA,t_AA_vanilla[:,0],marker='o',ls='',label='Vanilla',c='b')
         plt.ylabel('execution time (s)')
         plt.xlabel('N. beads')
         plt.legend()
         plt.yscale('log')

         plt.subplot2grid((1,2),(0,1))
         plt.title('Total')
         plt.plot(n_beads_AA,np.sum(t_AA_sparse,axis=1),marker='s',ls='',label='Sparse',c='r')
         plt.plot(n_beads_AA,np.sum(t_AA_vanilla,axis=1),marker='s',ls='',label='Vanilla',c='b')
         plt.ylabel('execution time (s)')
         plt.xlabel('N. beads')
         plt.legend()

         plt.yscale('log')

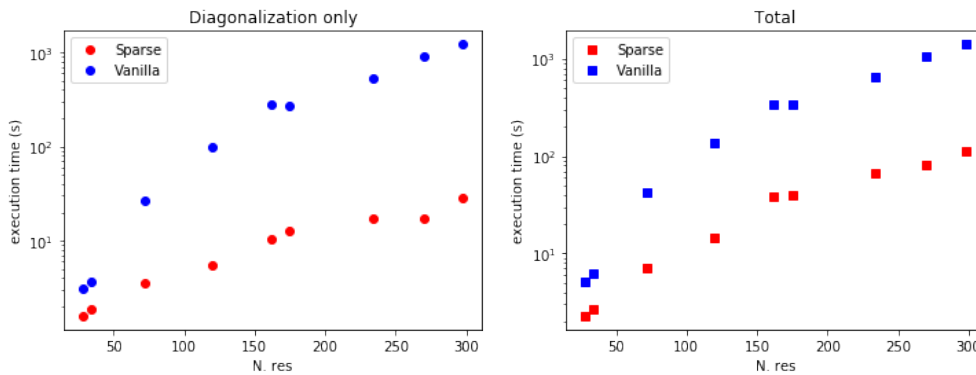
```



```
In [14]: plt.figure(figsize=(12,4))
plt.subplot2grid((1,2),(0,0))
plt.title('Diagonalization only')
plt.plot(n_res,t_AA_sparse[:,0],marker='o',ls='',label='Sparse',c='r')
plt.plot(n_res,t_AA_vanilla[:,0],marker='o',ls='',label='Vanilla',c='b')
plt.ylabel('execution time (s)')
plt.xlabel('N. res')
plt.legend()
plt.yscale('log')

plt.subplot2grid((1,2),(0,1))
plt.title('Total')
plt.plot(n_res,np.sum(t_AA_sparse,axis=1),marker='s',ls='',label='Sparse',c='r')
plt.plot(n_res,np.sum(t_AA_vanilla,axis=1),marker='s',ls='',label='Vanilla',c='b')
plt.ylabel('execution time (s)')
plt.xlabel('N. res')
plt.legend()

plt.yscale('log')
```



7.1 SBP model

```
In [16]: times=[]
n_beads_SBP=[]
```

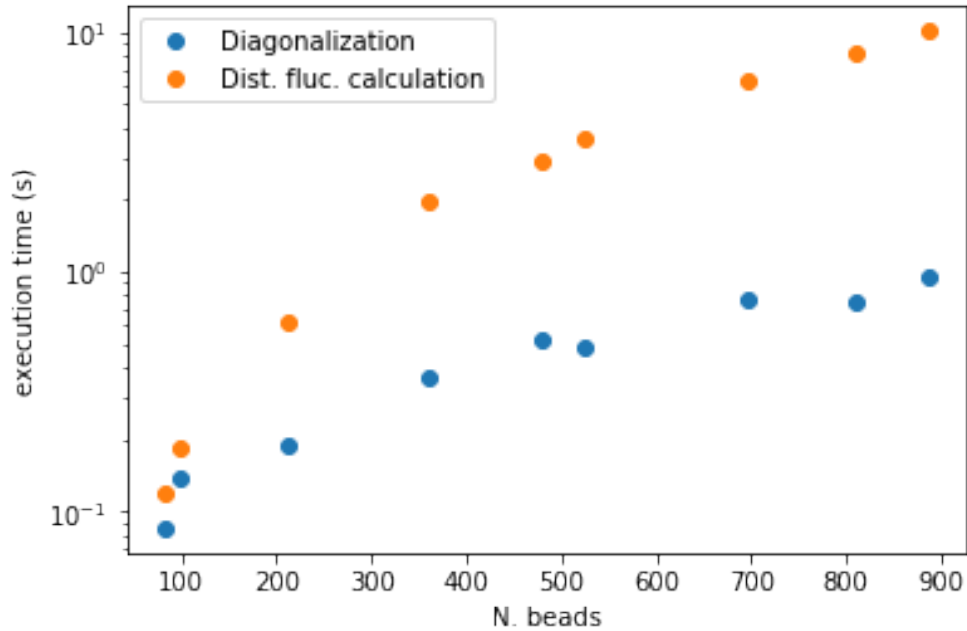
```

for fname in flist:
    t0=time.time()
    enm_tmp=enm.Enm(fname,sparse=True,cutoff=1.0)
    t1=time.time()-t0
    fluc,r1=enm_tmp.c2_fluctuations()
    t2=time.time()-t0-t1
    n_beads_SBP.append(enm_tmp.n_beads)
    times.append([t1,t2])
t_SBP_sparse=np.array(times)

# Read (360, 3) coordinates
# Using sparse matrix diagonalization
# Read (480, 3) coordinates
# Using sparse matrix diagonalization
# Read (525, 3) coordinates
# Using sparse matrix diagonalization
# Read (213, 3) coordinates
# Using sparse matrix diagonalization
# Read (888, 3) coordinates
# Using sparse matrix diagonalization
# Read (810, 3) coordinates
# Using sparse matrix diagonalization
# Read (697, 3) coordinates
# Using sparse matrix diagonalization
# Read (82, 3) coordinates
# Using sparse matrix diagonalization
# Read (98, 3) coordinates
# Using sparse matrix diagonalization

In [17]: plt.plot(n_beads_SBP,t_SBP_sparse[:,0],marker='o',ls='',label='Diagonalization')
plt.plot(n_beads_SBP,t_SBP_sparse[:,1],marker='o',ls='',label='Dist. fluc. calculation')
plt.ylabel('execution time (s)')
plt.xlabel('N. beads')
plt.legend()
plt.yscale('log')

```



```
In [18]: times=[]
n_beads_SBP=[]
for fname in flist:
    t0=time.time()
    enm_tmp=enm.Enm(fname,sparse=False,cutoff=1.0)
    t1=time.time()-t0
    fluc,rl=enm_tmp.c2_fluctuations()
    t2=time.time()-t0-t1
    n_beads_SBP.append(enm_tmp.n_beads)
    times.append([t1,t2])
t_SBP_vanilla=np.array(times)
```

```
# Read (480, 3) coordinates
# Read (525, 3) coordinates
# Read (213, 3) coordinates
# Read (888, 3) coordinates
# Read (810, 3) coordinates
# Read (697, 3) coordinates
# Read (82, 3) coordinates
# Read (98, 3) coordinates
```

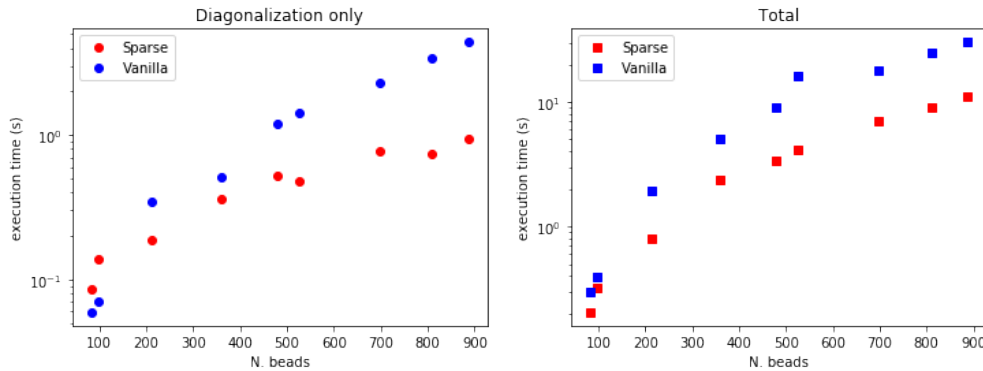
```
In [19]: plt.figure(figsize=(12,4))
plt.subplot2grid((1,2),(0,0))
plt.title('Diagonalization only')
plt.plot(n_beads_SBP,t_SBP_sparse[:,0],marker='o',ls='',label='Sparse',c='r')
plt.plot(n_beads_SBP,t_SBP_vanilla[:,0],marker='o',ls='',label='Vanilla',c='b')
plt.ylabel('execution time (s)')
plt.xlabel('N. beads')
plt.legend()
```

```

plt.yscale('log')

plt.subplot2grid((1,2),(0,1))
plt.title('Total')
plt.plot(n_beads_SBP,np.sum(t_SBP_sparse,axis=1),marker='s',ls='',label='Sparse',c='r')
plt.plot(n_beads_SBP,np.sum(t_SBP_vanilla,axis=1),marker='s',ls='',label='Vanilla',c='b')
plt.ylabel('execution time (s)')
plt.xlabel('N. beads')
plt.legend()
plt.yscale('log')

```



7.2 Save all results on disk

```

In [20]: fout=open('performances.dat','w')
fout.write('# PDB | n_beads_AA | n_beads_SBP | n_res | \
t_AA_sprs diag | t_AA_sprs fluc | t_AA_vnll diag | t_AA_vnll fluc | \
t_SBP_sprs diag | t_SBP_sprs fluc | t_SBP_vnll diag | t_SBP_vnll fluc \n')
for i,finname in enumerate(flist):
    pdb=finname.split(".")[0]
    fout.write("%s %d %d %d %e %e %e %e %e %e %e \n" % \
(pdb,n_beads_AA[i],n_beads_SBP[i],n_res[i],\
t_AA_sparse[i,0],t_AA_sparse[i,1],t_AA_vanilla[i,0],t_AA_vanilla[i,1],\
t_SBP_sparse[i,0],t_SBP_sparse[i,1],t_SBP_vanilla[i,0],t_SBP_vanilla[i,1]))
fout.close()

```

7.3 Final plot

```

In [21]: plt.figure(figsize=(12,4))
plt.subplot2grid((1,2),(0,0))
plt.subplots_adjust(wspace=0)
plt.title('Diagonalization only')
plt.plot(n_res,t_SBP_sparse[:,0],marker='^',ls='',label='Sparse',c='r',ms=8,
markeredgecolor='k')
plt.plot(n_res,t_SBP_vanilla[:,0],marker='^',ls='',label='Vanilla',c='gold',ms=8,
markeredgecolor='k')
#plt.legend()
plt.plot(n_res,t_AA_sparse[:,0],marker='o',ls='',label='Sparse',c='r',ms=8,
markeredgecolor='k')

```

```

plt.plot(n_res,t_AA_vanilla[:,0],marker='o',ls='',label='Vanilla',c='gold',ms=8,
markeredgecolor='k')
plt.ylabel('execution time (s)')
plt.xlabel('N. res')
#plt.legend()
plt.yscale('log')
plt.xscale('log')
plt.ylim(0.01,2000)

ax=plt.subplot2grid((1,2),(0,1))
plt.title('Total')
ax.plot(n_res,np.sum(t_AA_sparse,axis=1),marker='o',ls='',label='Sparse',c='r',ms=8,
markeredgecolor='k')
ax.plot(n_res,np.sum(t_AA_vanilla,axis=1),marker='o',ls='',label='Dense',c='gold',ms=8,
markeredgecolor='k')
plt.legend()
ax.plot(n_res,np.sum(t_SBP_sparse,axis=1),marker='^',ls='',label='Sparse',c='r',ms=8,
markeredgecolor='k')
ax.plot(n_res,np.sum(t_SBP_vanilla,axis=1),marker='^',ls='',label='Dense',c='gold',ms=8,
markeredgecolor='k')
ax.yaxis.tick_right()
plt.xlabel('N. res')
#plt.legend()
plt.ylim(0.01,2000)

plt.yscale('log')
plt.xscale('log')
plt.savefig('ENM-performance.pdf',bbox_inches="tight")

```

