CABS Documentation

Release 2012

Andrzej Koliński

July 25, 2012

CONTENTS

Contents:

CHAPTER

ONE

TUTORIAL

1.1 Calculating heat capacity, C_v

$$C_v(T) = \frac{\langle E^2 \rangle - \langle E \rangle^2}{T^2}$$

```
#!/usr/bin/env python
import multiprocessing as mp
import os
import numpy as np

import pycabs


def runCABS(temperature):
        # global for simplify arguments
        global name, sequence,secstr,template

        # function for running CABS with different temperatures
        # it will compute in directory name+_+temperature
        here = os.getcwd() # since pycabs changing directories...
        a = pycabs.CABS(sequence,secstr,template,name+"_"+str(temperature))
        a.createLatticeReplicas(replicas=1)
        a.modeling(Ltemp=temperature,Htemp=temperature, phot=85,cycles=2)
        #remember to come back to 'here' directory
        os.chdir(here)


#init these variables _before_ running cabs
name = "fnord"
# we have some template, it has to be as list
template=["/home/hydek/pycabs/playground/2pcy.pdb"]
# suppose we have porter prediction of sec. str.
sss =  pycabs.parsePorterOutput("/home/hydek/pycabs/proba/playground/porter.ss")
sequence = sss[0]
secstr = sss[1]
# now we have all data required to run CABS

temp_from = 2.0
```

```
temp_to  = 4.0
temp_interval = 0.07
temperatures=np.arange(temp_from,temp_to,temp_interval) # ranges of temperature

# create thread pool  with two parallel threads
pool = mp.Pool(processes=2)
pool.map(runCABS,temperatures) # run cabs threads

# HERE IS THE END OF PART WHERE WE RUN CABS in parallel fashion.

# Now you can do something with output data, we'll calculate heat capacity, Cv:
cv = np.empty(len(temperatures))
for i in range(len(temperatures)):
        t = temperatures[i]
        e_path = os.path.join(name+'_'+str(t),'ENERGY')
        energy = np.fromfile(e_path,sep='\n') # read ENERGY data into array 'energy'
        avg_energy2 = np.average(energy*energy) # <E^2>
        avg_energy = np.average(energy)                    # <E>^2
        cv[i] = (avg_energy2 - avg_energy*avg_energy)/(t*t) # (<E^2> - <E>^2) / T^2
# now we have heat capacity in cv array

# ... and display plot
from pylab import *
xlabel(r'temperature $T$')
ylabel(r'heat capacity $C_v = (\left<E^2\right> - \left<E\right>^2)/T^2$' )
xlim(temp_from,temp_to) # xrange
plot(temperatures,cv)
show()

#remember that you have name+_+temperature directories, delete it or sth
```

1.2 Monitoring of CABS energy during simulation

```
#!/usr/bin/env python
from pylab import *
from sys import argv
import os
import time
import numpy as np
import pycabs

class Energy(pycabs.Calculate):
    def calculate(self,data):
        for i in data:
            self.out.append(float(i)) # ENERGY file contains one value in a row

out = []
calc = Energy(out) # out is dynamically updated
m=pycabs.Monitor(os.path.join(argv[1],"ENERGY"),calc)
m.daemon = True
m.start()



ion()
y = zeros(1)
```

```
x = zeros(1)
line, = plot(x,y)
xlabel('CABS time step')
ylabel('CABS energy')

while 1:
    time.sleep(1)
    y = np.asarray(out)
    x = xrange(0,len(out))
    axis([0, amax(x)+1, amin(y)-5, amax(y)+5 ])
    line.set_ydata(y)  # update the data
    line.set_xdata(x)
    draw()
```

1.3 Monitoring of end-to-end distance of chain during simulation

```
#!/usr/bin/env python
from pylab import *
from sys import argv
import time
import os
import numpy as np
import pycabs

class E2E(pycabs.Calculate):
    def calculate(self,data):
        models = self.processTrajectory(data)
        for m in models:
            first = m[0:3]
            last = m[-3:]

            x = first[0]-last[0]
            y = first[1]-last[1]
            z = first[2]-last[2]
            self.out.append(x*x+y*y+z*z)

out = []
calc = E2E(out) # out is dynamically updated
m=pycabs.Monitor(os.path.join(argv[1],"TRAF"),calc)
m.daemon = True
m.start()


ion()
y = zeros(1)
x = zeros(1)
line, = plot(x,y)
xlabel('CABS time step')
ylabel('square of end to end distance')

while 1:
    time.sleep(1)
    y = np.asarray(out)
    x = xrange(0,len(out))
    axis([0, amax(x)+1, amin(y)-5, amax(y)+5 ])
    line.set_ydata(y)  # update the data
    line.set_xdata(x)
    draw()
```

CHAPTER

TWO

PYCABS API

class `pycabs.`**`CABS`**(*sequence*, *secondary_structure*, *templates_filenames*, *project_name*)

CABS main class.

Parameters

- **sequence** (*string*) – one line sequence of the target protein
- **secondary_structure** (*string*) – one line secondary structure for the target protein
- **templates_filenames** (*list*) – path to 3D protein model templates in pdb file format which you want to use for modeling. Cα numbering in templates must be aligned to target sequence
- **project_name** (*string*) – project_name and working directory name (uniq)

`calcConstraints`(*exclude_residues*=[ ], *other_constraints*=[ ])

Calculate distance constraints using templates 3D models.

Parameters

- **exclude_residues** (*list*) – indexes of residues without constrains
- **constrains** (*other*) – user-defined constrains as list of tuples: (residue_i_index,residue_j_index,constraint_strength)

`convertPdbToDcd`(*catdcd_path='/home/hydek/pycabs/FF/catdcd'*)

This is only simple wrapper to CatDCD software (http://www.ks.uiuc.edu/Development/MDTools/catdcd/), could be usable since *.dcd binary format is few times lighter than pdb, and many python libraries (ProDy, MDAnalysis) use *.dcd as trajectory input format. Before use, download CatDCD from http://www.ks.uiuc.edu/Development/MDTools/catdcd/ and modify catdcd_path.

`createLatticeReplicas`(*start_structures_fn*=[ ], *replicas=20*)

Create protein models projected onto CABS lattice, which will be used as replicas.

Parameters

- **start_structures_fn** (*list*) – list of paths to pdb files which should be used instead of templates models. This parameter is optional, and probably not often used. Without it script creates replicas from templates files.
- **replicas** (*integer*) – define number of replicas in CABS simulation. However 20 is optimal for most cases, and you don't need to change it in protein modeling case.

Note: If number of replicas is smaller than number of templates - program will create replicas using first *replicas* templates. If there is less templates than replicas, they are creating sequentially using template models.

---

getEnergy()
Read CABS energy values into list

Returns list of models energy

getTraCoordinates()
Read trajectory file into 2D list of coordinates

Returns 2D list of trajectory coordinates

rng_seed = None
seed for random generator

trafToPdb(*output_filename='TRAF.pdb'*)
Convert TRAF CABS pseudotrajectory file format into multimodel pdb

class pycabs.**Calculate**(*output*)
Inherit if you want to process data used with `Monitor` class.

Parameters **output** (*array/list*) – output array with calculated results

processTrajectory(*data*)
Use it in *calculate* method if you parsing TRAF file, and want to calculate something on structure

Returns array of model coordinates

exception pycabs.**Errors**(*value*)
Simple error messages

class pycabs.**Info**(*text*)
Simple message system

class pycabs.**Monitor**(*filename*, *calculate*)
Class for monitoring of CABS output data. You can run it and dynamically update output arrays with calculated results.

Parameters **calculate** (`Calculate`) – what to do with gathered data ?

daemon = None
if True, it will terminate when script terminates

run()
Run monitor in background

terminate()
Terminate monitor

class pycabs.**Template**(*filename*)
Class used for template storage of atom positions and distance calculation

Parameters **filename** – path to file with template (in PDB format)

distance (*idx_i*, *idx_j*)

Parameters

- **idx_i** (*integer*) – residue index (as in target sequence numbering)
- **idx_j** (*integer*) – residue index (as in target sequence numbering)

Returns euclidean distance between Cα(i) and Cα(j)

pycabs.**parsePorterOutput** (*porter_output_fn*)

Porter (protein secondary stucture prediction, http://distill.ucd.ie/porter/) output parser. Porter emailed output looks like:

```
IDVLLGADDGSLAFVPSEFSISPGEKIVFKNNAGFPHNIVFDEDSIPSGVDASKISMSEE
CEEEECCCCCCCCEECCEEEECCCCEEEEEECCCCCEEEEECCCCCCCCCCHHHHCCCCC



DLLNAKGETFEVALSNKGEYSFYCSPHQGAGMVGKVTVN
CCECCCCCEEEEECCCCEEEEEECCHHHHCCCEEEEEEC
```

Parameters **porter_output_fn** (*string*) – path to the porter output file

Returns tuple (sequence, secondary_structure)

pycabs.**parsePsipredOutput** (*psipred_output_fn*)

Psipred (protein secondary structure prediction, http://bioinf.cs.ucl.ac.uk/psipred/) output parser. Psipred output looks like:

```
> head psipred.ss
1 P C   1.000  0.000  0.000
2 K C   0.665  0.000  0.459
3 A E   0.018  0.000  0.991
4 L E   0.008  0.000  0.997
5 I E   0.002  0.000  0.998
6 V E   0.003  0.000  0.999
7 Y E   0.033  0.000  0.981
```

Parameters **psipred_output_fn** (*string*) – path to the psipred output file

Returns tuple (sequence, secondary_structure)

pycabs.**rmsd** (*reference*, *arr*)

Calculate coordinate Root Mean Square Deviation between two sets of coordinates.

$$cRMSD = \sqrt{\frac{\sum_{i=1}^{N} \|x_i - y_i\|^2}{N}}$$

Parameters

- **reference** (*list*) – 1D list of coordinates (length of 3N)
- **arr** (*list*) – 1D list of coordinates (length of 3N)

Returns RMSD value after optimal superimposition of two structures

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX