CABS Documentation

Release 2012

Andrzej Koliński

March 02, 2013

CONTENTS

Download CABS:

`pycabs.tar.gz` (or github repository: pycabs)

Contents:

CHAPTER

ONE

TUTORIAL

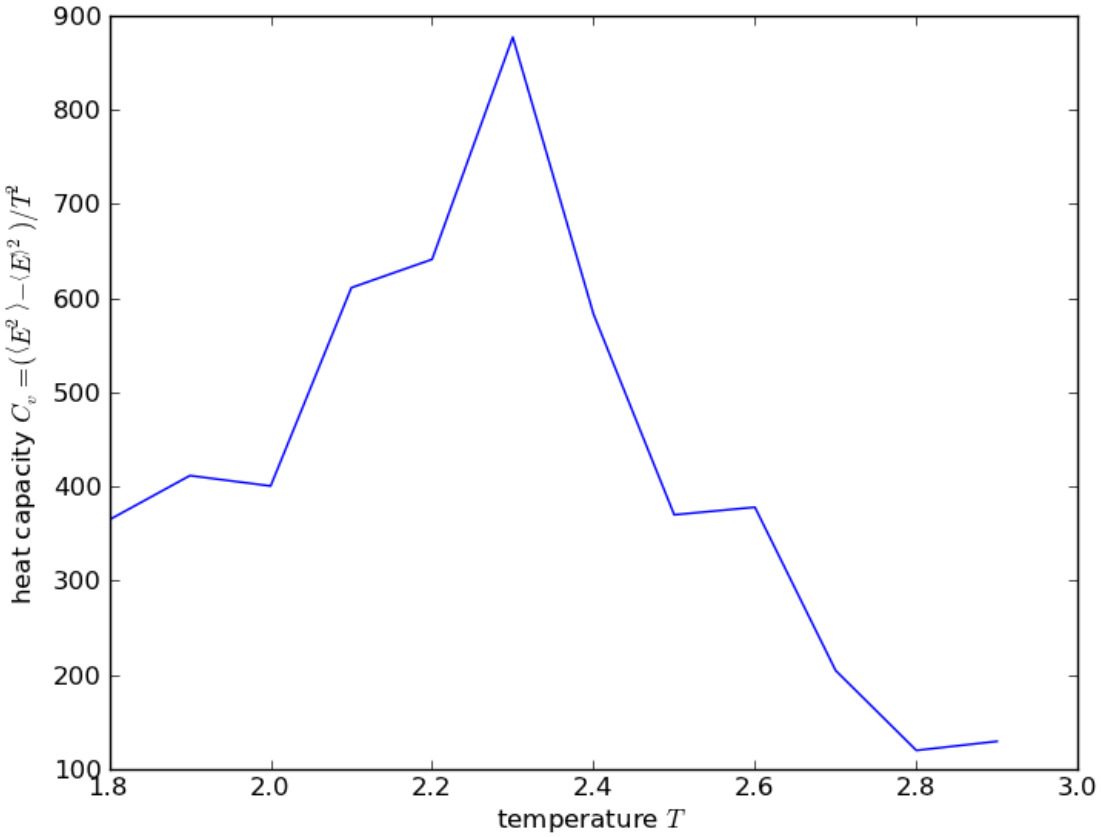
1.1 Calculating heat capacity, C_v

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

```
#!/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=300,cycles=100,dynamics=True)
        #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/user/pycabs/playground/2pcy.pdb"]
# suppose we have porter prediction of sec. str.
sss =  pycabs.parsePorterOutput("/home/user/pycabs/proba/playground/porter.ss")
sequence = sss[0]
secstr = sss[1]
# now we have all data required to run CABS

temp_from = 1.5
temp_to  = 3.0
```

```
temp_interval = 0.1
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] = np.std(energy)*np.std(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
```

Download script: `heat_capacity.py`.



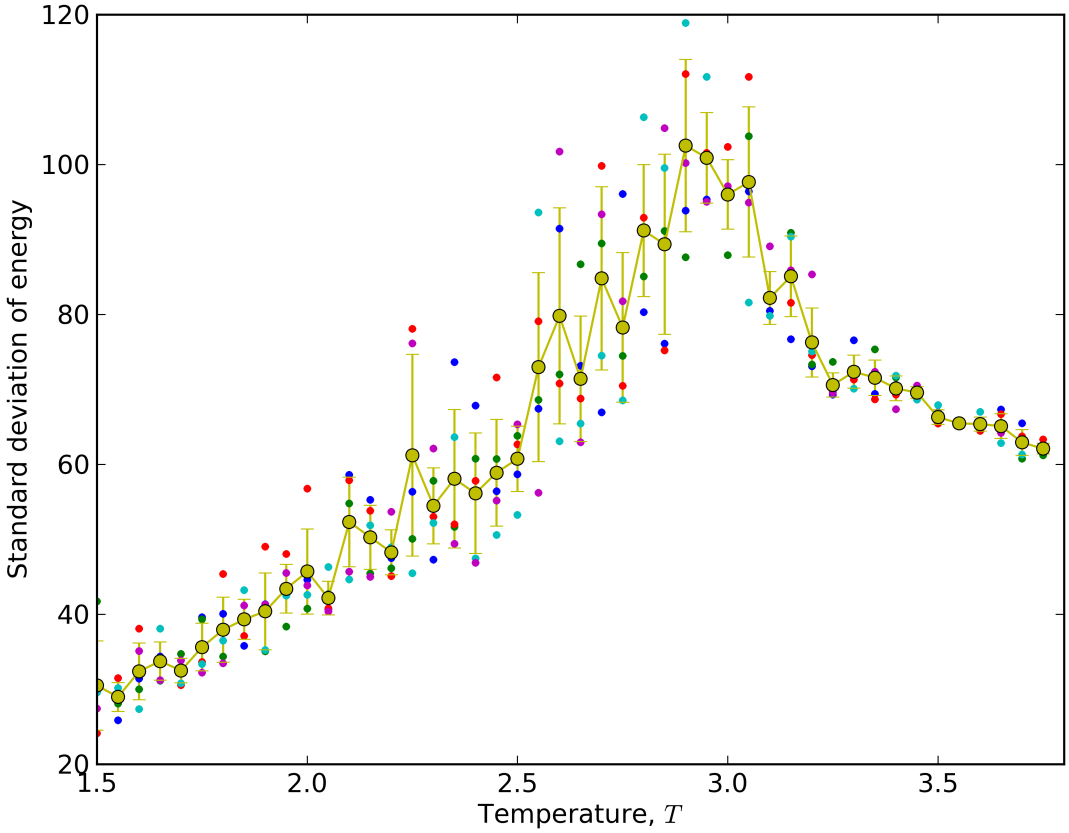
1.2 Study folding pathway: 1) create standard deviation and mean energy plots for Barnase

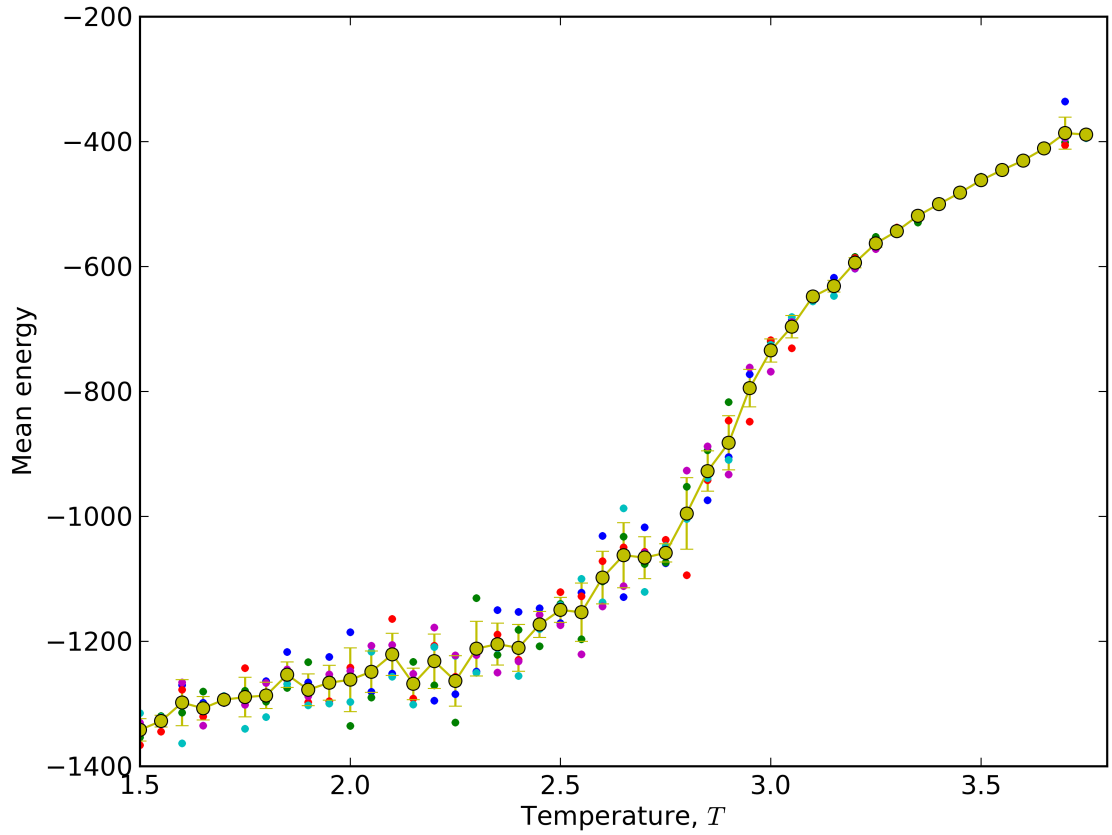
```
#!/usr/bin/env python
# 2013, Michal Jamroz, public domain. http://biocomp.chem.uw.edu.pl

import os, random, pylab, glob, pycabs, numpy as np, multiprocessing as mp
# first of all, download pyCABS and set self.FF = "" to the FF directory with cabs files
# to compile CABS, use: g77 -O2 -ffloat-store -static -o cabs CABS.f
# to compile CABS_dynamics, use: g77 -O2 -ffloat-store -static -o cabs_dynamics CABS_dynamics.f
# to compile lattice model builder, use g77 -O2 -ffloat-store -static build_cabs61.f
# in FF directory.

sequence, secstr = pycabs.parseDSSPOutput("/where/is/my/barnase/1bnr.dssp") # define file with secon
name = "barnase" # name for the project. Script will create subdirectories with this name as suffix
template = ["/where/is/my/barnase/1bnr.pdb"] # set path to the start structure (here - native structu
independent_runs=5  # set number of independent simulations for each temperature
temp_from = 1.5     # define range of simulation temperatures, here is 1.0 - 2.8 with interval of 0.
temp_to  = 3.8
temp_interval = 0.05
temperatures=np.arange(temp_from,temp_to,temp_interval)

def runCABS(temperature):
```

```
    global name, sequence,secstr,template,independent_runs
    here = os.getcwd()
    for i in range(independent_runs):
        temp = "%06.3f" %(temperature)
        dir_name= name+"_"+str(i)+"_T"+temp  # create unique name for simulation dir
        a = pycabs.CABS(sequence,secstr,template,dir_name)
        a.rng_seed = random.randint(1,10000) # set random generator seed for each independent simulatio
        a.createLatticeReplicas(replicas=1)  # create lattice model for CABS
        a.modeling(Ltemp=temperature,Htemp=temperature, phot=300,cycles=100,dynamics=True) # start mode
        os.chdir(here)

pool = mp.Pool()  # it use all available CPUs on workstation. If user want to use only - for example
pool.map(runCABS,temperatures) # run simulations in parallel way, each simulation on each available C

# postprocessing (comment out two lines above to avoid starting over simulations. If you want to only

cv = np.empty([independent_runs,len(temperatures)])
avgene = np.empty([independent_runs,len(temperatures)])
for j in range(independent_runs):
    for i in range(len(temperatures)):
        t = temperatures[i]
        temp = "%06.3f" %(t)
        e_path = os.path.join(name+'_'+str(j)+'_T'+temp,'ENERGY') # path constructed in the same way li
        energy = np.fromfile(e_path,sep='\n')[1000:] # read CABS energies for each trajectory model (se
        cv[j][i] = np.std(energy)  # calculate standard deviation (numpy std function) of energy, for e
        avgene[j][i] = np.mean(energy) # calculate mean (numpy mean function) of energy


mean_sigma = np.mean(cv,axis=0)     # average over independent simulations
stddev_sigma = np.std(cv,axis=0)
mean_ene = np.mean(avgene,axis=0)  # calculate of standard deviations and mean values over independen
stddev_ene = np.std(avgene,axis=0)

# plotting data with pylab python module. Read matplotlib manual (http://matplotlib.org/) for explana
pylab.ylabel(r'Standard deviation of energy' )
pylab.xlabel(r'Temperature, $T$')
pylab.xlim(temp_from,temp_to)
for i in range(independent_runs):
    pylab.plot(temperatures, cv[i], '.')
pylab.errorbar(temperatures,mean_sigma,yerr=stddev_sigma,fmt='o-')
pylab.savefig("stdE_barnase.png",dpi=600)
pylab.close()

pylab.ylabel(r'Mean energy' )
pylab.xlabel(r'Temperature, $T$')
pylab.xlim(temp_from,temp_to)
for i in range(independent_runs):
    pylab.plot(temperatures, avgene[i], '.')
pylab.errorbar(temperatures,mean_ene,yerr=stddev_ene,fmt='o-')
pylab.savefig("meanE_barnase.png",dpi=600)
pylab.close()


# as a results user will get a lot of barnase* subdirectories, stdE_barnase.png and meanE_barnase.png
# EOF
```

Download script: `folding_pathway.py`. Download necessary files: `1bnr.pdb`, `1bnr.dssp`

Results:


120
100
80
60
40
20
Standard deviation of energy
1.5
2.0
2.5
3.0
3.5
Temperature, T



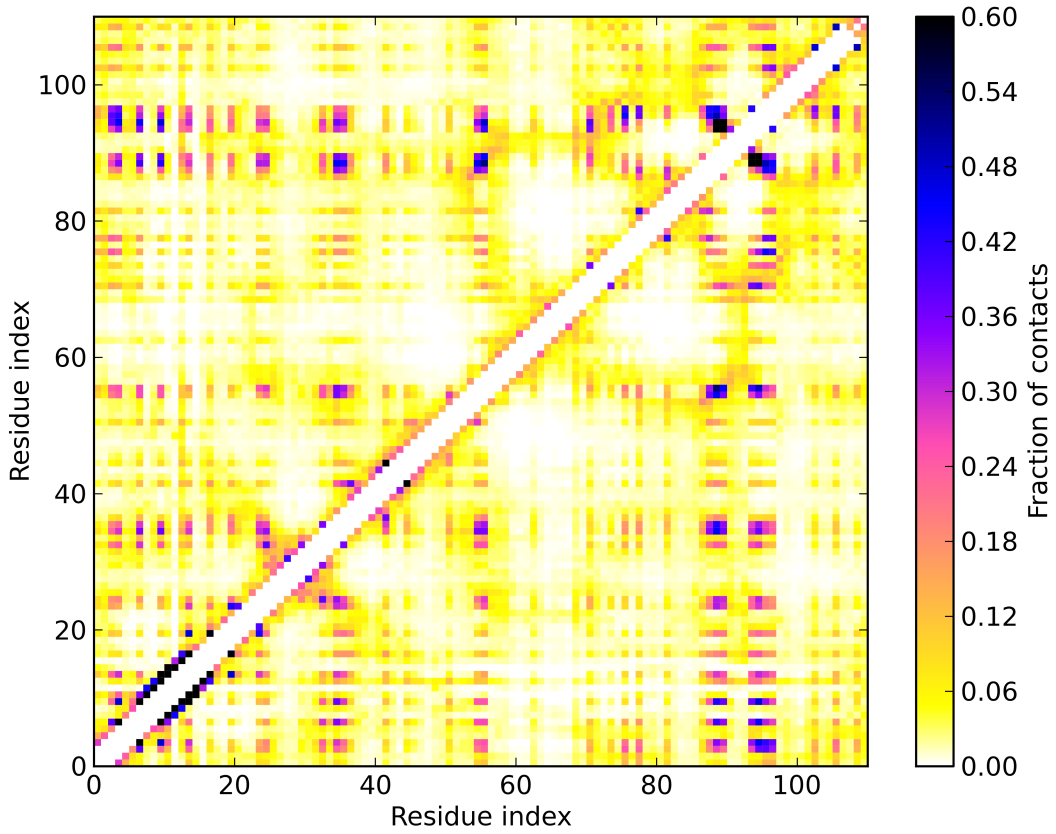
1.3 Study folding pathway: 2) calculate average contact map over trajectory of sidegroups in temperature 2.1

```
#!/usr/bin/env python
# 2013, Michal Jamroz, public domain. http://biocomp.chem.uw.edu.pl
import pycabs,os,numpy as np

name = "barnase" # set project name same like in folding_pathway.py script
max_sd_temperature=2.9 # read from the plot temperature for which maximum deviation of energy is obse
independent_runs=5      # set the same value like in folding_pathway.py script

# read trajectory of sidegroups (TRASG) from all independent simulations to the trajectory variable
trajectory = []
for j in range(independent_runs):
        temp = "%06.3f" %(max_sd_temperature)
        e_path = os.path.join(name+'_'+str(j)+'_T'+temp,'TRASG')
        trajectory += pycabs.loadSGCoordinates(e_path)[1000:] # second half of sidechains trajectory

contact = pycabs.contact_map(trajectory,7.0) # calculate averaged contact map over trajectories. Cut

# plot contact map with pylab. Read matplotlib manual (http://matplotlib.org/) for explanation of the
from pylab import xlabel,ylabel,pcolor,colorbar,savefig,xlim,ylim,cm
from numpy import indices
```

```
l=len(trajectory[0])
rows, cols = indices((l,l))
xlabel("Residue index")
xlim(0, len(contact))
ylim(0, len(contact))
ylabel("Residue index")
pcolor(contact, cmap=cm.gnuplot2_r,vmax=0.6) # vmax is range of colorbar. Here is set to 0.8, which g
cb = colorbar()
cb.set_label("Fraction of contacts")
savefig("heatmap"+str(max_sd_temperature)+".png",dpi=600)

# optionally, write contact map values into the text file formatted for GNUplot
fw = open("contact_map.dat","w")
for i in range(len(contact)):
   for j in range(len(contact)):
      fw.write("%5d %5d %7.5f\n" %(i+1,j+1,contact[i][j]))
   fw.write("\n")
fw.close()

# example GNUplot script for plotting contact map of contact_map.dat file:
'''
set terminal unknown
plot 'contact_map.dat' using 1:2:3
set xrange[GPVAL_DATA_X_MIN:GPVAL_DATA_X_MAX]
set yrange[GPVAL_DATA_Y_MIN:GPVAL_DATA_Y_MAX]


set terminal postscript eps enhanced color "Helvetica" 14
set output 'contact_map.eps'
set size ratio 1
unset key
set xlabel 'Residue index'
set ylabel 'Residue index'
set cbrange[:0.8]
set palette negative
plot 'contact_map.dat' with image
'''

# write it to the file.gp and run: gnuplot file.gp to get postscript file with heat map plot
```

Download script: `contact_map.py`.

Results:



Optionally, GNUplot script for plotting contact_map.dat file generated by contact_map.py script:

```
set terminal unknown
plot 'contact_map.dat' using 1:2:3
set xrange[GPVAL_DATA_X_MIN:GPVAL_DATA_X_MAX]
set yrange[GPVAL_DATA_Y_MIN:GPVAL_DATA_Y_MAX]


set terminal postscript eps enhanced color "Helvetica" 14
set output 'contact_map.eps'
set size ratio 1
unset key
set xlabel 'Residue index'
set ylabel 'Residue index'
set cbrange[:0.8]
set palette negative
plot 'contact_map.dat' with image
```

Download script: `gnuplot_script.gp`.

1.4 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()
```

Download script: `monitoring_energy.py`.

1.5 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()
```

Download script: `monitoring_e2e_distance.py`.

1.6 De-novo modeling of 2PCY structure

```
#!/usr/bin/env python
import pycabs
from Pycluster import *
from numpy import array

data =  parsePorterOutput("/home/user/pycabs/proba/playground/porter.ss") # read PORTER (or PsiPred)
working_dir = "de_novo" # name of project
templates = [] # deNOVO
a = pycabs.CABS(data[0],data[1],templates,working_dir) # initialize CABS, create required files
# DENOVO a.generateConstraints()
a.createLatticeReplicas() # create start models from templates
a.modeling(Htemp=3.0,cycles=20,phot=100) # start modeling with default INP values and create TRAF.pdb
tr = a.getTraCoordinates() # load TRAF into memory and calculate RMSD all-vs-all :


#calculating RMSD 2D array for clustering
distances = zeros((len(tr),len(tr)))
for i in range(len(tr)):
        for j in range(i,len(tr)):
                rms = pycabs.rmsd(tr[i],tr[j])
                distances[i][j] = distances[j][i] = rms
```

```
#save RMSD array as heat map
heat_map(distances,"Protein model","Protein model","RMSD")


# clustering by K-medoids method (with 5 clusters)
clusterid,error,nfound = kmedoids(distances,nclusters=5,npass=15,initialid=None)
print clusterid,error
clusterid,error,nfound = kcluster(distances,nclusters=5,npass=15)

# save cluster medoids to file
pycabs.saveMedoids(clusterid,a)
print clusterid,error
```

Download script: `de_novo.py`.

CHAPTER

TWO

PYCABS API

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

CABS main class. .. warning:: Manually update self.FF variable here (path to the FF directory with CABS files)

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)

convertPdbToDcd(*catdcd_path='/home/user/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.

---

generateConstraints(*exclude_residues*=[ ], *other_constraints*=[ ])
Calculate distance constraints using templates 3D models. Constraint will be a square well of size d-std_dev-1.0,d+std_dev+1.0, where d is mean distance among templates between Cα atoms (if constraint will be exceeded, there is penalty, scaled by weight.

Weight is defined as a fraction of particular average distance among templates i.e. if pair of residues exist in 2 of 3 templates, weight will be 0.66. Using multiple sequence alignments it should provide stronger constraints on consistently aligned parts.

Parameters

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

generateConstraintsOld(*exclude_residues*=[ ], *other_constraints*=[ ])
Calculate distance constraints using templates 3D models. Constraint will be a square well of size min(d), max(d) where d is mean distance among templates between Cα atoms (if constraint will be exceeded, there is penalty, scaled by weight.

Weight is defined as a fraction of particular average distance among templates i.e. if pair of residues exist in 2 of 3 templates, weight will be 0.66. Using multiple sequence alignments it should provide stronger constraints on consistently aligned parts.

Parameters

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

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 (list[1][6] is sixth coordinate of second trajectory model = z coordinate of second atom of second model)

loadSGCoordinates()
Read center of mass of sidegroups from TRASG file

Returns 2D list of sidechains coordinates

modeling(*Ltemp=1.0*, *Htemp=2.0*, *cycles=100*, *phot=300*, *constraints_force=1.0*, *dynamics=False*)
Start CABS modeling

param Ltemp Low temperature for Replica Exchange Monte Carlo

type Ltemp float

param Htemp High temperature for Replica Exchange Monte Carlo

type Htemp float

param cycles number of Replica Exchange cycles

type cycles integer

param iphot number of microcycles (inside REMC loop)

type iphot integer

param constraints_force Slope of constraints force potential

type constraints_force float

Parameters **dynamics** – Use of special CABS version for dynamics pathway studies :type dynamics: boolean

rng_seed = None
seed for random generator

savePdbModel (*model_idx*, *filename=''*)
Save trajectory model into pdb file

Parameters

- **model_idx** – index of model in the CABS trajectory
- **filename** – name of the output file. If empty, it saves to model_index.pdb

sgToPdb (*output_filename='TRASG.pdb'*)
Convert TRASG (sidechains pseudoatoms) into multimodel pdb. Default filename TRASG.pdb

trafToPdb (*output_filename='TRAF.pdb'*)
Convert TRAF CABS pseudotrajectory file format into multimodel pdb (default filename TRAF.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 storage of templates atom positions and distance calculation

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

Returns Nx3 list of coordinates

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.**contact_map**(*trajectory*, *contact_cutoff*)

Compute fraction of contacts in a trajectory, where trajectory is 2D list of coordinates (trajectory[2][5] is the z-th coordinate of second atom of third model)

Parameters

- **trajectory** – 2D trajectory of atoms (Cα, sidegroups center of mass, etc.)
- **contact_cutoff** – cutoff defining contact

Returns 2D array of fraction of contacts (number of contacts divided by trajectory length) for each pair of residue.

pycabs.**heat_map**(*data*, *x_label*, *y_label*, *colormap_label*, *output_file='heatmap.png'*, *cmap='Greys'*)

Save heat map using pylab

Parameters **data** (*float*) – 2D list of values

pycabs.**loadSGCoordinates**(*filename*)

Read center of mass of sidegroups from TRASG file

Parameters **filename** – path to the TRASG file

Returns 2D list of sidechains coordinates

pycabs.**loadTRAFCoordinates**(*filename*)

Read trajectory file into 2D list of coordinates

Returns 2D list of trajectory coordinates (list[1][6] is sixth coordinate of second trajectory model = z coordinate of second atom of second model)

pycabs.**parseDSSPOutput**(*filename*)

Helper function for extracting sequence and secondary structure assignments from the DSSP output. Useful for dynamics studies or other where we know protein structure.

You can download DSSP files directly from PDB server: http://www.pdb.org/pdb/files/PDBID.dssp

pycabs.**parsePDBfile**(*pdb_filename*)

Function for parsing of Cα coordinates from PDB file.

Parameters **pdb_filename** (*string*) – path to PDB file

Returns 1D list of Cα coordinates (for example: list[4] is y-th coordinate of second atom)

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

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

```
IDVLLGADDGSLAFVPSEFSISPGEKIVFKNNAGFPHNIVFDEDSIPSGVDASKISMSEE
CEEEECCCCCCCCEECCEEEECCCCEEEEEECCCCCEEEEECCCCCCCCCCCHHHHCCCCC
```

```
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

pycabs.**saveMedoids**(*clusters*, *cabs*)

Save cluster medoids in PDB file format.

Parameters

- **clusters** – cluster indices as a output of C Clustering Library
- **clusters** – list

CHAPTER

THREE

INDICES AND TABLES

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

PYTHON MODULE INDEX

p

INDEX