February 8 2012

Retrieved structures, centers and selections from the "paper corrections/hydropobicity moment" subfolder of my earlier work

Looked up in "hhomp location of dataset/search results" the clusters including 1A0S (ScrY), 1QD6 (OMPLA), and 1QJP (OmpA)

ScrY: 18.1.1

OMPLA: 12.6.1

OmpA: 8.1.1

Looked up their sequences in "hhomp location of dataset/sequences.csv"

ScrY:

SGFEFHGYARSGVIMNDSGASTKSGAYITPAGETGGAIGRLGNQADTYVEMNLEHKQTLDNGATTRFKVMVADGQTSYNDWTASTSDLNVRQAFVELGNLPTFAGPFKGSTLWAGKRFDRDNFDIHWIDSDVVFLAGTGGGIYDVKWNDGLRSNFSLYGRNFGDIDDSSNSVQNYILTMNHFAGPLQMMVSGLRAKDNDERKDSNGNLAKGDAANTGVHALLGLHNDSFYGLRDGSSKTALLYGHGLGAEVKGIGSDGALRPGADTWRIASYGTTPLSENWSVAPAMLAQRSKDRYADGDSYQWATFNLRLIQAINQNFALAYEGSYQYMDLKPEGYNDRQAVNGSFYKLTFAPTFKVGSIGDFFSRPEIRFYTSWMDWSKKLNNYASDDALGSDGFNSGGEWSFGVQMETWF

OMPLA:  
FTLYPYDTNYLIYTQTSDLNKEAIASYDWAENARKDEVKFQLSLAFPLWRGILGPNSVLGASYTQKSWWQLSNSEESSPFRETNYEPQLFLGFATDYRFAGWTLRDVEMGYNHDSNGRSDPTSRSWNRLYTRLMAENGNWLVEVKPWYVVGNTDDNPDITKYMGYYQLKIGYHLGDAVLSAKGQYNWNTGYGGAELGLSYPITKHVRLYTQVYSGYGESLIDYNFNQTRVGVGVMLNDLF

OmpA:

APKDNTWYTGAKLGWSQHENKLGAGAFGGYQVNPYVGFEMGYDWLGRMPYAYKAQGVQLTAKLGYPITDDLDIYTRLGGMVWRADTYSNVYGKNHDTGVSPVFAGGVEYAITPEIATRLEYQWTNGMLSLGVSYRFG

Downloaded the alignments for those clusters as "OMPLA without pdb.clu" etc.

Added the sequences above to these alignments with the identifiers "chaina\_ompla" etc. (since these were taken from chain A of the pdb file, according to what I remember reading of how Daniel made the aligned pdb files), removed anything that wasn't actually a protein sequence (ss pred, ss conf, bb pred, bb conf, cluster name), and aligned, outputting as "OMPLA with pdb.aln" etc.

Copied "published params.csv", containing the ez-beta parameters from the first ez-beta paper, from "paper corrections/ompla moments"

February 15, 2012

Took "identity.csv" from folder "ompla gaps": see that folders log for details on its creation

These selections and centers are older than theones used in the final paper I sent Vik. They may be outdated, though I don't remember what I would have changed; all info of that type should be in my "paper corrections" and "final paper" logs.

Generated family moments for OmpA, ScrY and OMPLA using the following code:

import numpy as np

import zenergy

import csv

import selections

from functools import partial

import Bio.AlignIO

from sundries import CIDict

from sundries import file\_dict

from Bio.PDB import PDBParser

import warnings

import os

import matrices

import math

class MissingPDBSequence(Exception):

pass

class MultiplePDBSequences(Exception):

pass

class AlignmentOracle(object):

'''

Initialized with a multiple sequence alignment and a name to find a

particular sequence of interest. Can give amino acids from other

sequences in the alignment that correspond to amino acids in the

sequence

of interest, ignoring amino acids that are lined up with gaps in the

sequence of interest.

'''.replace(' ', '')

def \_\_init\_\_(self, alignment, pdb\_name = 'pdb'):

self.data = alignment

self.pdb\_index = None

for index, record in enumerate(self.data):

if pdb\_name in record.id:

if self.pdb\_index is None:

self.pdb\_index = index

else:

raise MultiplePDBSequences('More than one sequence had '+\

pdb\_name + ' in its title')

if self.pdb\_index == None:

raise MissingPDBSequence('None of the sequences had '+\

pdb\_name + ' in their title')

self.\_pdb\_sequence = self.data[self.pdb\_index].seq

def pdb\_sequence(self, selection = None):

return self.sequence(self.pdb\_index, selection)

def sequence(self, index, selection = None):

pos = 0

for j, letter in enumerate(self.data[index].seq):

if self.\_pdb\_sequence[j] != '-':

if selection is None or pos in selection:

yield letter.upper()

pos += 1

def get\_alignment(self):

return self.data

def get\_pdb\_index(self):

return self.pdb\_index

def get\_pdb\_seq\_record(self):

return self.data[self.pdb\_index]

def load\_centers(iterable):

dict\_ = CIDict()

for row in iterable:

if row[0] != '':

dict\_.update({row[0]:row[1]})

for key, value in dict\_.items():

# Turns '(1,2,3)' etc, that is, textual representations of vectors,

# into Vector objects. Will cut off the last digit of the third

# componant, but I don't care because the third componant will

# always be 0.0

dict\_[key] = np.array([float(y[:-1]) for y in value[1:].split()])

return dict\_

def moment(structure, selection, center, mag\_function, res\_retrieve):

sum\_ = np.zeros(3)

for residue in structure.get\_residues():

# My selection files just give residue numbers. If two residues

# have the same number, if there're insertions, then the selection

# files are ambiguous and I need to change how I do them.

# For now, anything identified by more than just a residue number

# is ignored. The reason this is okay is that, as far as I can tell,

# in Dan's structures that just means it ignores water.

# If the structures have complex residue ids, then I need different

# selection files.

if residue.get\_id()[0] != ' ':

if residue.get\_id()[0] != 'W' and \

residue.get\_id()[1] in selection:

print('Residue with id {0} in structure {1}'\

.format(residue.get\_id(), structure) + \

'was in selection but was ignored')

continue

try:

resn = res\_retrieve.next()

except StopIteration:

raise Exception('Oracle ran out of letters on residue ' +\

repr(residue.get\_id()))

#resn = one\_letter[residue.get\_resname()]

if residue.get\_id()[1] not in selection:

continue

# Vector points from center to Ca

coordinates = residue.child\_dict['CA'].get\_coord()

vector = coordinates - center

# Take the projection on the xy plane

vector[2] = 0

# Normalize the vector

normalized = vector / np.linalg.norm(vector)

# Give it a magnitude determined by the 'function' argument

try:

complete = normalized \* mag\_function(residue, resn)

except zenergy.NoParameters:

# For now, for the purposes of replicating old data:

complete = normalized \* .5

sum\_ += complete

return sum\_

def calculator\_adapter(calc, residue, resn):

# The moment function uses a function of a residue

# The ez\_beta calculator is a function of a residue type and depth

return calc.calculate(resn,

residue.child\_dict['CA'].get\_coord()[2])

# Load selections:

with open('cored 1 selections with 1qd5.csv', 'rb') as f:

reader = csv.reader(f)

resi\_lists = selections.selections\_by\_resi(reader)

print('Selections retrieved... ' + repr(type(resi\_lists)))

# Load centers:

with open('cored 1 centers with 1qd5.csv', 'rb') as f:

reader = csv.reader(f)

centers = load\_centers(reader)

print('Centers retrieved... ' + repr(type(centers)))

# Initialize a calculator:

with open('published params.csv', 'rb') as f:

reader = csv.reader(f)

calc = zenergy.Calculator(reader, normalize = True)

print('Calculator created... ' + repr(type(calc)))

with open('identity.csv', 'rb') as f:

reader = csv.reader(f)

identity = matrices.retrieve\_matrix(reader)

print('Identity matrix loaded... ' + repr(type(identity)))

# The slow part, open the structures:

structure\_files = file\_dict('structures with 1qd5',

['aligned\_(1QD6).pdb',

'aligned\_(1QJP).pdb',

'aligned\_(1A0S).pdb'])

parser = PDBParser()

with warnings.catch\_warnings():

warnings.simplefilter('ignore')

structures = [parser.get\_structure(id\_, path) \

for id\_, path in structure\_files.items()]

structure\_dict = CIDict([(structure.get\_id(), structure)

for structure in structures])

# Yes this is stupid, next time I'll just use a dictionary, no list

print('Structured parsed...' + repr(structures))

# Open the multiple sequence alignments:

alignments = dict([(name, Bio.AlignIO.read(filename, 'clustal')) \

for name, filename in \

file\_dict(os.getcwd(), ['(.\*) with pdb.aln']).items()])

# Remake the alignment dictionary with the pdbids that are being used

# everywhere else in this script, instead of protein names:

alignments ={'1A0S': alignments['ScrY'],

'1QD6': alignments['OMPLA'],

'1QJP': alignments['OmpA']}

# Make an oracle for each alignment:

oracles = CIDict([(pdbid, AlignmentOracle(alignment, pdb\_name = 'chaina'))\

for pdbid, alignment in alignments.items()])

print('Alignments loaded... ' + repr(oracles))

pdb\_moments = CIDict([(structure.get\_id(),

moment(structure, resi\_lists[structure.get\_id()],

centers[structure.get\_id()],

partial(calculator\_adapter, calc),

oracles[structure.get\_id()].pdb\_sequence()))

for structure in structures])

print('pdb moments calculated! ' + repr(pdb\_moments))

family\_moments = CIDict((pdbid, list()) for pdbid in alignments.keys())

for pdbid in family\_moments.keys():

for seq\_index in range(len(oracles[pdbid].get\_alignment())):

family\_moment = moment(structure\_dict[pdbid], resi\_lists[pdbid],

centers[pdbid],

partial(calculator\_adapter, calc),

oracles[pdbid].sequence(seq\_index))

pdb\_sequence = oracles[pdbid].get\_pdb\_seq\_record().seq

sequence = oracles[pdbid].get\_alignment()[seq\_index].seq

normalized\_distance = matrices.compare(pdb\_sequence, sequence,

identity)

seq\_id = oracles[pdbid].get\_alignment()[seq\_index].id

family\_moments[pdbid].append((seq\_id, normalized\_distance,

family\_moment))

print('family moments calculated! ' + str(type(family\_moments)))

for pdbid in family\_moments.keys():

with open('prelim test {}.csv'.format(pdbid), 'wb') as f:

writer = csv.writer(f)

# Sort moments by distance from pdb sequence, descending:

sorted\_moments = sorted(family\_moments[pdbid],

key = lambda x: x[1])[::-1]

for seq\_id, normalized\_distance, family\_moment in sorted\_moments:

writer.writerow([seq\_id, normalized\_distance,

np.linalg.norm(family\_moment),

math.atan2(family\_moment[1],family\_moment[0])])

These were the moments it produced for pdb files:

pdb moments calculated! {'1qjp': array([-1.79500426, -2.95080628, 0. ]), '1qd6': array([-1.05537928, 1.69320925, 0. ]), '1a0s': array([-2.30540811, -6.5496449 , 0. ])}

These are close to, but not exactly the same as, various moments I have saved in the "pymol/moments" folder.

The script also produced 3 csv files, containing the norms and directions of the family moments for ScrY, OMPLA, and OmpA, sorted by descending sequence similarity to the sequence of the pdb structure. I checked the first entry of the ScrY file (the one containing the sequence of the pdb structure) to make sure its norm and direction match those of the moment printed above, and it does.

There are no sequences in ScrY's alignment with more than 20% sequence identity to ScrY. OMPLA and OmpA have small, sensitive moments. The results, because of this, are not especially meaningful.

March 7 2012

Added to folder "non normalized moments/exc centers/exc centers.csv" and "non normalized moments/exc centers/ exc selections.csv". As documented in the "non normalized moments" folder, using these centers and selections, with normalization, reproduces the exclusive moment I gave to Vik in my final paper.

I modified theabove code, to produce new family moments, with those selections. I also made small changes to the code, to make the output more clear. Also, this is very important, I disabled the normalization that happens as part of calculating the moment.

A summary of the changes:

diff --git a/test moments/generate moments.py b/test moments/generate moments.py

index af4d04b..f4c9db5 100755

--- a/test moments/generate moments.py

+++ b/test moments/generate moments.py

@@ -130,27 +130,38 @@ def calculator\_adapter(calc, residue, resn):

residue.child\_dict['CA'].get\_coord()[2])

# Load selections:

-with open('cored 1 selections with 1qd5.csv', 'rb') as f:

+selections\_file = 'exc selections.csv'

+with open(selections\_file, 'rb') as f:

reader = csv.reader(f)

resi\_lists = selections.selections\_by\_resi(reader)

-print('Selections retrieved... ' + repr(type(resi\_lists)))

+print('Retrieved selections from {}... '.format(selections\_file) \

+ + repr(type(resi\_lists)))

# Load centers:

-with open('cored 1 centers with 1qd5.csv', 'rb') as f:

+centers\_file = 'exc centers.csv'

+with open(centers\_file, 'rb') as f:

reader = csv.reader(f)

centers = load\_centers(reader)

-print('Centers retrieved... ' + repr(type(centers)))

+print('Retrieved centers from {}... '.format(centers\_file) \

+ + repr(type(centers)))

# Initialize a calculator:

-with open('published params.csv', 'rb') as f:

+normalize = False

+params\_file = 'published params.csv'

+with open(params\_file, 'rb') as f:

reader = csv.reader(f)

- calc = zenergy.Calculator(reader, normalize = True)

-print('Calculator created... ' + repr(type(calc)))

-

-with open('identity.csv', 'rb') as f:

+ calc = zenergy.Calculator(reader, normalize = normalize)

+print('Calculator created with normalize set to {}, '

+ + 'and parameters from {}... '.format(normalize, params\_file) \

+ + repr(type(calc)))

+

+# Load matrix:

+matrix\_file = 'identity.csv'

+with open(matrix\_file, 'rb') as f:

reader = csv.reader(f)

identity = matrices.retrieve\_matrix(reader)

-print('Identity matrix loaded... ' + repr(type(identity)))

+print('Retrieved matrix from {}... '.format(matrix\_file) \

+ + repr(type(identity)))

# The slow part, open the structures:

structure\_files = file\_dict('structures with 1qd5',

@@ -167,7 +178,7 @@ with warnings.catch\_warnings():

structure\_dict = CIDict([(structure.get\_id(), structure)

for structure in structures])

# Yes this is stupid, next time I'll just use a dictionary, no list

-print('Structured parsed...' + repr(structures))

+print('Structured parsed... ' + repr(structures))

# Open the multiple sequence alignments:

alignments = dict([(name, Bio.AlignIO.read(filename, 'clustal')) \

@@ -212,12 +223,17 @@ for pdbid in family\_moments.keys():

print('family moments calculated! ' + str(type(family\_moments)))

for pdbid in family\_moments.keys():

- with open('prelim test {}.csv'.format(pdbid), 'wb') as f:

+ with open('exc test {}.csv'.format(pdbid), 'wb') as f:

writer = csv.writer(f)

+ writer.writerow(['pdbid', '%id w/ pdb seq',

+ 'mag', 'dir', 'x','y'])

# Sort moments by distance from pdb sequence, descending:

sorted\_moments = sorted(family\_moments[pdbid],

key = lambda x: x[1])[::-1]

for seq\_id, normalized\_distance, family\_moment in sorted\_moments:

writer.writerow([seq\_id, normalized\_distance,

np.linalg.norm(family\_moment),

- math.atan2(family\_moment[1],family\_moment[0])])

+ (180/math.pi) \* math.atan2(family\_moment[1],

+ family\_moment[0]),

+ family\_moment[0],

+ family\_moment[1]])

I omit the code because it does not function properly. The output includes:

pdb moments calculated! {'1qjp': array([ 11.14038282, 7.92281364, 0. ]), '1qd6': array([-0.34006503, 1.61756824, 0. ]), '1a0s': array([ 2.19463583, 4.11484775, 0. ])}

I checked the 1a0s moment against that contained in "non normalized moments/non normalized exclusive\_moment.csv" and it's not the same.

To produce "non normalized moments/non normalized exclusive\_moment.csv", I had it use 0 as an energy for a residue where there's no parameters. Here I'm still using .5.