# August 22, 2012

The following is code to calculate zdiff's. It's been tested; the details are in "june 2012 lab meeting log".

**from** **sundries** **import** CIDict

**from** **Bio.PDB** **import** PDBParser

**import** **warnings**

**from** **Bio** **import** AlignIO

**def** z(residue):

*'''Returns the z coordinate of a residue object's Calpha.'''*

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

**class** **Gap**(object):

*'''Represents a gap in a Position object.'''*

**pass**

**class** **Position**(object):

**def** \_\_init\_\_(self, pairs):

self.residues = CIDict(pairs)

**def** zdiff(self, template\_id, unknown\_id):

template\_res = self.residues[template\_id]

unknown\_res = self.residues[unknown\_id]

**if** type(template\_res) **is** Gap **or** type(unknown\_res) **is** Gap:

**return** None

**else**:

**return** z(template\_res) - z(unknown\_res)

**def** resi(self, stru\_id):

**return** self.residues[stru\_id].get\_id()[1]

**class** **NotFoundError**(Exception):

**pass**

**class** **Zdiff**(object):

**def** \_\_init\_\_(self, \*stru\_seq\_pairlist):

pos\_inputs = list()

**for** structure, sequence **in** stru\_seq\_pairlist:

residues = structure.get\_residues()

pairs\_for\_pos = list()

**for** letter **in** sequence:

**if** letter == '-':

seq\_unit = Gap()

**else**:

seq\_unit = residues.next()

pairs\_for\_pos.append((structure.get\_id(), seq\_unit))

pos\_inputs.append(pairs\_for\_pos)

self.positions = [Position(id\_res\_pairlist) \

**for** id\_res\_pairlist **in** zip(\*pos\_inputs)]

**def** get(self, template\_id, unknown\_id, resi, start=0):

**for** pos **in** self.positions[start:]:

**if** pos.resi(unknown\_id) == resi:

**return** pos.zdiff(template\_id, unknown\_id)

**raise** NotFoundError('resi {} of {} not found' \

.format(resi, unknown\_id))

**def** report(self, template\_id, unknown\_id):

output = (pos.zdiff(template\_id, unknown\_id) \

**for** pos **in** self.positions)

**return** filter(**lambda** x: x **is** **not** None, output)

**with** warnings.catch\_warnings():

warnings.simplefilter('ignore')

scry\_stru = PDBParser().get\_structure('1A0S', 'aligned\_1A0S.pdb')

maltoporin\_stru = PDBParser().get\_structure('1AF6', 'aligned\_1AF6.pdb')

alignment = AlignIO.read('Swiss-PDB structural alignment.aln', 'clustal')

**for** seq\_record **in** alignment:

**if** seq\_record.id == 'aligned\_1A0S':

scry\_seq = seq\_record

**if** seq\_record.id == 'aligned\_1AF6':

maltoporin\_seq = seq\_record

output = Zdiff((scry\_stru, scry\_seq), (maltoporin\_stru, maltoporin\_seq))

**with** open('NEW 1a0s as template.txt', 'w') **as** f:

f.writelines(str(zdiff) + '**\n**' \

**for** zdiff **in** output.report('1A0S', '1AF6'))

**with** open('NEW 1af6 as template.txt', 'w') **as** f:

f.writelines(str(zdiff) + '**\n**' \

**for** zdiff **in** output.report('1af6', '1a0s'))

My current goal is to turn this code into a module that can be accessed with easy-to-remember commands, and check that this module can reproduce the old results.

Copying some of the data from "june 2012 lab meeting" so I can try to replcate it. From "june 2012 lab meeting/zdiff calculator validation", I copied two structures, an alignment, and a list of zdiffs. The list of zdiffs does not have resi's, so I am pretending that I know that it is in order by resi. These files are:  
aligned\_1A0S.pdb  
aligned\_1AF6.pdb  
OLD 1a0s as template.txt  
Swiss-PDB structural alignment.aln

They all have their original names, except for "OLD 1a0s as template.txt", which used to be "NEW 1a0s as template.txt".

Remade the zdiff's with new code. The zdiff's at the top and bottom came out correct, and there was the right number of them, so I figure it probably works.

The code I used was this:

**from** sundries **import** CIDict

**from** Bio.PDB **import** PDBParser

**import** warnings

**from** Bio **import** AlignIO

# The guts that it runs on

**def** z(residue):

'''Returns the z coordinate of a residue object's Calpha.'''

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

**class** Gap(object):

'''Represents a gap in a Position object.'''

**pass**

**class** Position(object):

**def** \_\_init\_\_(self, pairs):

self.residues = CIDict(pairs)

**def** zdiff(self, template\_id, unknown\_id):

template\_res = self.residues[template\_id]

unknown\_res = self.residues[unknown\_id]

**if** type(template\_res) **is** Gap **or** type(unknown\_res) **is** Gap:

**return** None

**else**:

**return** z(template\_res) - z(unknown\_res)

**def** resi(self, stru\_id):

'''Return resi of position in specified structure'''

# Get the residue of the specified structure that is in this

# position in the alignment.

target = self.residues[stru\_id]

# It might actually be a gap; return None of it is

**if** type(target) **is** Gap:

**return** None

# Otherwise, return the id of the residue. It's a Biopython

# Residue object, so this is done with its get\_id() method.

**else**:

**return** self.residues[stru\_id].get\_id()[1]

**class** NotFoundError(Exception):

**pass**

**class** Zdiff(object):

**def** \_\_init\_\_(self, \*stru\_seq\_pairlist):

pos\_inputs = list()

**for** structure, sequence **in** stru\_seq\_pairlist:

residues = structure.get\_residues()

pairs\_for\_pos = list()

**for** letter **in** sequence:

**if** letter == '-':

seq\_unit = Gap()

**else**:

seq\_unit = residues.next()

pairs\_for\_pos.append((structure.get\_id(), seq\_unit))

pos\_inputs.append(pairs\_for\_pos)

self.positions = [Position(id\_res\_pairlist) \

**for** id\_res\_pairlist **in** zip(\*pos\_inputs)]

**def** get(self, template\_id, unknown\_id, resi, start=0):

**for** pos **in** self.positions[start:]:

**if** pos.resi(unknown\_id) == resi:

**return** pos.zdiff(template\_id, unknown\_id)

**raise** NotFoundError('resi {} of {} not found' \

.format(resi, unknown\_id))

**def** report(self, template\_id, unknown\_id):

output = (pos.zdiff(template\_id, unknown\_id) \

**for** pos **in** self.positions)

**return** filter(lambda x: x **is** **not** None, output)

**def** resi\_report(self, template\_id, unknown\_id):

output = ((pos.resi(template\_id),

pos.zdiff(template\_id, unknown\_id)) \

**for** pos **in** self.positions)

**return** filter(lambda x: x[1] **is** **not** None, output)

# The API for making zdiff files

**def** calc(template\_name, target\_name, template\_structure\_filename,

target\_structure\_filename, alignment\_filename,

write\_to, comment, format\_='clustal'):

# Open relevant files

with open(template\_structure\_filename, 'r') as template\_structure\_file,\

open(target\_structure\_filename, 'r') as target\_structure\_file,\

open(alignment\_filename, 'r') as alignment\_file:

# Load structures with Biopython's PDB file parser

# Daniel's aligned structures are missing some inessential

# information, and as a consequence the parser gives thousands

# of warnings. Gotta ignore these.

with warnings.catch\_warnings():

warnings.simplefilter('ignore')

template\_structure = PDBParser().\

get\_structure(template\_name,

template\_structure\_file)

target\_structure = PDBParser().\

get\_structure(target\_name,

target\_structure\_file)

# Open alignment using Biopython's parser

alignment = AlignIO.read(alignment\_filename, format\_)

# Find the template and target sequences in the alignment

**for** seq\_record **in** alignment:

**if** seq\_record.id == template\_name:

templ\_seq = seq\_record

**if** seq\_record.id == target\_name:

targ\_seq = seq\_record

# Calculate zdiff

results = Zdiff((template\_structure, templ\_seq),

(target\_structure, targ\_seq))

# Write results to a file

with open(write\_to, 'w') as o:

# Write some coments so I know which zdiff file this is

o.write('# Template: ')

o.write(template\_name + ' (' + template\_structure\_filename + ')\n')

o.write('# Target: ')

o.write(target\_name + ' (' + target\_structure\_filename + ')\n')

o.write('# Alignment: ' + alignment\_filename + '\n')

o.write('# ' + comment + '\n')

# Write the actual data

# Weird quirk of the zdiff objects - before giving a report, it

# requires the id's of the structures. I don't know if I wrote

# it to support more than two, or if I was planning to, or what.

**for** resi, zdiff **in** results.resi\_report(template\_name, target\_name):

o.write(str(resi) + ', ' + str(zdiff) + '\n')

I ran the "calc" function as so:

calc('aligned\_1A0S', 'aligned\_1AF6', 'aligned\_1A0S.pdb', 'aligned\_1AF6.pdb', 'Swiss-PDB structural alignment.aln', 'new.zdiff', 'reproduction of structural alignment to test code')

I copied the folder "june 2012 lab metting/pdbs from hhomp" to "zdiff module/pdbs from hhomp". This has the pdb structures of the proteins of known structure that are nearby other proteins of known structure in the HHOMP clustermap.

# September 4, 2012

Daniel's derivation of Ezβ involves aligning the sequences of PDB structures to whatever HHOMP clusters they had the closest match with in HHOMP's search.

Some of these HHOMP clusters have proteins of known structure in them. The proteins of known structure are not in the alignments, but they are mentioned in the cluster description. The alignment with OMPLA in it has the sequence from 1QD6 in the alignment, though, with "pdb" in the title. But the others don't have sequences with "pdb" in their titles, and in one case I searched the sequences for one matching a pdb file but couldn't find it.

Sometimes these structures are the very same that are in Daniel's dataset. Sometimes they are not, though. I will find the zdiff between the predicted structures of these proteins using proteins from Daniel's dataset, and the real structures.

I found five clusters that I can use for this purpose.

cluster73: 18.1.1 and 18.1.2  
Daniel's dataset: Sucrose porin (1A0S), mapped to cluster73  
HHOMP's dataset: Maltoporin (2MPR) in 18.1.1

cluster99: 16.1.1 and 16.1.2  
Daniel's: OmpC (2J1N), mapped to cluster99  
HHOMP's: PhoE (1PHO) and OmpF (2OMF), both in 16.1.1  
I've heard OmpC is pretty similar to these two, the seq identity might be really high

cluster71: 14.1.1, 14.1.5, and 14.1.7  
Daniel's: TodX (3BS0), mapped to cluster71  
HHOMP: FadL (1T16) in 14.1.5

cluster?:  
Daniel's: HasR (3CSL) mapped to cluster 22.4.6  
HHOMP's: BBtuB (1NQE) in 22.4.5  
These are basically on opposite sides of the 22-stranded cluster, but the HHOMP guys put them both in 22.4. I hope that means they're similar.  
However, no small contains both of these. Only cluster124, 139, 149, 152, and 153, and all those clusters are huge, much bigger than the clusters the sequences in Daniel's datgaset were mapped to.  
I'm leaving these ones out.

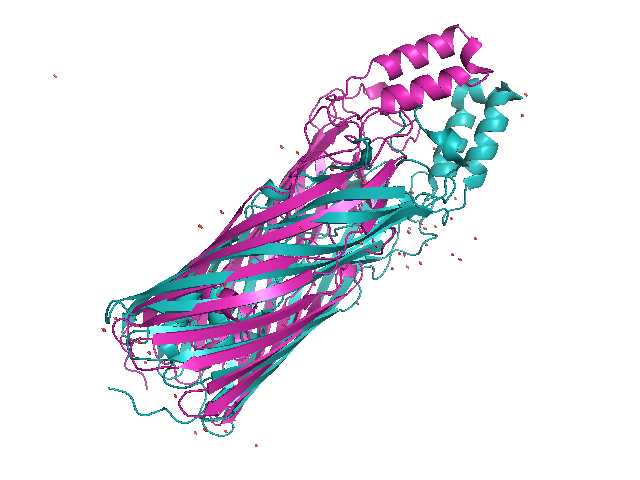
cluster18: 22.1.3, 22.1.4, 22.1.5, 22.1.6, 22.1.7  
Daniel's: FauA (3EFM), mapped to cluster18  
HHOMP'S: FhuA (1BY5) in 22.1.4  
It looks like a lot, but this all looks like one big blob on the clustermap. Still, I guess if they were more closely related, it would be concentrated to a point, and the sequences find their closest relatives.

First, I need to make aligned structures (of the structures in HHOMP, so that the z coordinate of each atom represents its distance from the plane running through the center of the membrane. I will do this by structural alignment to the homolog with which I will calculate zdiff.

I created a folder called "zdiff module/comparison structures". In this folder I made subfolders "cluster73", "cluster99", "cluster71" and "cluster18". IN each subfolder I put one of Daniel's aligned structures that was mapped to this cluster (file taken from "pymol/structures", which in turn was sent to me by Daniel a long long time ago, before I kept good records), as well as the structure (or structure**s**, in the case of cluster99) that HHOMP lists as belonging to the cluster (downloaded from www.pdb.org).

I opened Daniel's aligned structure of 3EFM and 1BY5 in PyMOL, and aligned 1BY5 to 3EFM using the "cealign" command. According to the PyMOL wiki, CEAlign is a structure-based alignment that does not use sequence information. I saved the result as "zdiff module/comparison structures/cluster18/1BY5 aligned to daniel's 3EFM.pdb".

I opened Daniel's aligned structure of 3BS0 and the PDB's 1T16. This has two chains, chain A and B, each of which is a β barrel. I deleted chain B using PyMOL's "remove" command. I aligned 1T16 to aligned\_3BS0 using cealign. I saved it as "1T16 aligned to daniel's 3BS0", following the same naming scheme. I also loaded Daniel's aligned structure of 1T16 and compared them. One of them is magenta and one of them is cyan in the picture below. It is oriented along the right axis. However, the difference in rotation is important. There has to be a strand-by-strand matchup with 3BS0; it has to have the same rotation about that axis as 3BS0.



Same procedure with 2MPR onto Daniel's 1A0S, had to delete all but chain A in 2MPR as with 1T16.

Did the same with 1POR and 2OMF, onto Daniel's 2J1N.

# September 9, 2012

I created multiple sequence aglinments with ClustalW, each containing the sequence of a PDB structure that is in an HHOMP cluster, the HHOMP cluster that contains it (according to the annotations of the clusters, since these sequences do not always appear in the MSA of the clusters given by HHOMP), and a structure from Daniel's dataset that was mapped to that cluster by HHOMP's search function. The alignments are in “zdiff module/gonnet aligned”. They were created using the following Python function:

**def** zdiff\_align(matrix, output\_dir):

walk = list(os.walk('comparison structures'))

# Get the names of the clusters:

# [0][1] is the list of foldernames in the rootmost directory

clusters = walk[0][1]

**for** path, folder\_list, file\_list **in** walk[1:]:

# Establish what cluster we're talking about

**for** name **in** clusters:

**if** name **in** path:

cluster = name

**for** filename **in** file\_list:

# Align each pair of an HHOMP structure, and the structure in

# our dataset that was matched to the same cluster, with

# all the sequences of the cluster

match = re.match("(....) aligned to daniel's (....)\.pdb",

filename)

**if** match **is** None:

**continue**

# Retrieve from the regex match the pdbid of the protein in

# HHOMP:

hhomp\_pdbid = match.group(1)

# Retrieve from the regex match the pdbid of the protein in

# our dataset that was mapped to this cluster:

our\_pdbid = match.group(2)

# Retrieve the path names of the PDB files representing both

hhomp\_path = path + '/' + match.group(0)

our\_path = path + '/aligned\_{0}.pdb'.format(our\_pdbid)

# Make the alignment

# Create the output directory if it does not already exist

**try**:

# This will not successfully make the directory if

# nested directories would have to be created

os.mkdir(output\_dir)

**except** OSError:

# This is the error you'd get if the directory already

# exists

# So, our work is done, don't worry about it:

**pass**

output\_path = output\_dir + '/{}, {} as target, {} as template'\

.format(cluster, hhomp\_pdbid,

our\_pdbid)

alignments.align(output\_path, cluster, matrix,

hhomp\_path, our\_path)

I ran this function with these arguments:

zdiff\_align('gonnet', 'gonnet\_aligned')

This function uses the “alignments” module that I wrote. The current code of this module can be found by looking at today's commit on my github repository, at “github.com/sinisterdexter/beta-barrel-oligomerization”.

## September 28, 2012

Renamed this folder from "zdiff module" to "zdiffs". I am not primarily developing a module, but doing a scientific experiment. I do want a module, since I want the experiment to be repeatable, if I start to use different alignments.

Today I am concerned with calculating the typical similarity of the sequences in the HHOMP database with the sequences of the proteins I am using as templates. I would also like to find out the sequence distances between the pairs of structures that I calculate zdiff for. If they are far lower than the typpical distances of a sequence-template pair, then they don't say very much.

Found a bug in the alignment-maker: tries to put too long a path into the name of the pdb sequence in the alignment. Ends up not showing any information: "comparison\_structures/cluster1" It should only put in the filename, which has the PDB ID and the letters "pdb" for easy searching.

## October 1, 2012

I'm rerunning the command

zdiff\_align('gonnet', 'gonnet\_aligned')

after modifying the alignments module to give the PDB sequences in the alignments shorter names, including only the base name of the path in which they are found.

### Protocol for checking the effectiiveness of a particular method of alignment, using zdiff:

**Making alignments**

The first step is to create alignments that contain the sequences of known structure. Create alignment files with the following names:

* cluster18, 1BY5 as target, 3EFM as template
* cluster71, 1T16 as target, 3BS0 as template
* cluster73, 2MPR as target, 1A0S as template
* cluster99, 1PHO as target, 2J1N as template
* cluster99, 2OMF as target, 2J1N as template

Each of these templates are in the Ez-β dataset, and they are associated with the cluster for which the template is used for modeling. The "targets" are structures of proteins that, according to HHOMP's 8annotation, are contained within the cluster (though they are not in the cluster's multiple sequence alignment). The details of how these targets were chosen are in the September 4 entry of this log.

These alignments should contain the sequences suggested by their names. The names or id's of the target and template should be "template\_{PDBID}" and "target\_{PDBID}" respectively.

If the method is the use of ClustalW with a particular matrix, the function *zdiff\_align* in *zdiff.py* will create these automatically. Usage is zdiff\_align(matrix, output\_dir), where matrix is the name of the matrix to be used, and output\_dir is the name of a folder, which will be created if it does not already exist, to put the alignments. It does not support, at the moment, giving extra parameters to ClustalW. However, by modifying this function, and the "align" and "call\_clustalw" functions of the "alignments" module (found in "modules/alignments"), this could be added fairly easily: see the Biopython documentation for Bio.Align.Applications.ClustalwCommandline.

**The next steps are calculating the zdiff from the alignments, and colorizing a protein from the zdiffs. However I have not yet finisehd the code for carrying out these functions.**

**I fixed the naming scheme of the sequences in the alignments produced by *zdiff\_align*. However, the resulting alignments are different sizes than the original ones; slightly smaller. This creeps me out, there's no reason that this should be the case. The new alignments are in the folder "gonnet\_aligned", the old ones in "messed up names".**

**Created the zdiff's for the gonnet alignments. They have not yet been tested. The command used to create them is**

calc\_all("gonnet\_aligned", "gonnet\_zdiff", "zdiff of structures aligned with gonnet matrix.\nDefault ClustalW settings.\nSee log entry from October 1st 2012 for more details.")

**Here's the code used to generate them. Much of it is repeated from the last entry, though it's now better commented. The *calc\_all* subroutine is new.**

**from** Bio.PDB **import** PDBParser

**import** warnings

**from** Bio **import** AlignIO

**import** os

**import** re

**import** alignments

**from** sundries **import** CIDict

# The guts that it runs on

**def** z(residue):

'''Returns the z coordinate of a residue object's Calpha.'''

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

**class** Gap(object):

'''Represents a gap in a Position object.'''

**pass**

**class** Position(object):

**def** \_\_init\_\_(self, pairs):

self.residues = CIDict(pairs)

**def** zdiff(self, template\_id, unknown\_id):

template\_res = self.residues[template\_id]

unknown\_res = self.residues[unknown\_id]

**if** type(template\_res) **is** Gap **or** type(unknown\_res) **is** Gap:

**return** None

**else**:

**return** z(template\_res) - z(unknown\_res)

**def** resi(self, stru\_id):

'''Return resi of position in specified structure'''

# Get the residue of the specified structure that is in this

# position in the alignment.

target = self.residues[stru\_id]

# It might actually be a gap; return None of it is

**if** type(target) **is** Gap:

**return** None

# Otherwise, return the id of the residue. It's a Biopython

# Residue object, so this is done with its get\_id() method.

**else**:

**return** self.residues[stru\_id].get\_id()[1]

**class** NotFoundError(Exception):

**pass**

**class** Zdiff(object):

**def** \_\_init\_\_(self, \*stru\_seq\_pairlist):

pos\_inputs = list()

**for** structure, sequence **in** stru\_seq\_pairlist:

residues = structure.get\_residues()

pairs\_for\_pos = list()

**for** letter **in** sequence:

**if** letter == '-':

seq\_unit = Gap()

**else**:

seq\_unit = residues.next()

pairs\_for\_pos.append((structure.get\_id(), seq\_unit))

pos\_inputs.append(pairs\_for\_pos)

self.positions = [Position(id\_res\_pairlist) \

**for** id\_res\_pairlist **in** zip(\*pos\_inputs)]

**def** get(self, template\_id, unknown\_id, resi, start=0):

**for** pos **in** self.positions[start:]:

**if** pos.resi(unknown\_id) == resi:

**return** pos.zdiff(template\_id, unknown\_id)

**raise** NotFoundError('resi {0} of {1} not found' \

.format(resi, unknown\_id))

**def** report(self, template\_id, unknown\_id):

output = (pos.zdiff(template\_id, unknown\_id) \

**for** pos **in** self.positions)

**return** filter(lambda x: x **is** **not** None, output)

**def** resi\_report(self, template\_id, unknown\_id):

output = ((pos.resi(template\_id),

pos.zdiff(template\_id, unknown\_id)) \

**for** pos **in** self.positions)

**return** filter(lambda x: x[1] **is** **not** None, output)

**def** zdiff\_align(matrix, output\_dir):

walk = list(os.walk('comparison structures'))

# Get the names of the clusters:

# [0][1] is the list of foldernames in the rootmost directory

clusters = walk[0][1]

**for** path, folder\_list, file\_list **in** walk[1:]:

# Establish what cluster we're talking about

**for** name **in** clusters:

**if** name **in** path:

cluster = name

**for** filename **in** file\_list:

# Align each pair of an HHOMP structure, and the structure in

# our dataset that was matched to the same cluster, with

# all the sequences of the cluster

match = re.match("(....) aligned to daniel's (....)\.pdb",

filename)

**if** match **is** None:

**continue**

# Retrieve from the regex match the pdbid of the protein in

# HHOMP:

hhomp\_pdbid = match.group(1)

# Retrieve from the regex match the pdbid of the protein in

# our dataset that was mapped to this cluster:

our\_pdbid = match.group(2)

# Retrieve the path names of the PDB files representing both

hhomp\_path = path + '/' + match.group(0)

our\_path = path + '/aligned\_{0}.pdb'.format(our\_pdbid)

# Make the alignment

# Create the output directory if it does not already exist

**try**:

# This will not successfully make the directory if

# nested directories would have to be created

os.mkdir(output\_dir)

**except** OSError:

# This is the error you'd get if the directory already

# exists

# So, our work is done, don't worry about it:

**pass**

output\_path = output\_dir + '/{}, {} as target, {} as template'\

.format(cluster, hhomp\_pdbid,

our\_pdbid)

alignments.align(output\_path, cluster, matrix,

hhomp\_path, our\_path)

**def** calc(template\_name, target\_name, template\_structure\_filename,

target\_structure\_filename, alignment\_filename,

write\_to, comment, format\_='clustal'):

'''Calculate zdiffs from an alignment and write them to a file.

    template\_name and target\_name: the id's of the template and target

        sequences in the alignment

    template\_structure\_filename and target\_structure\_filename: where to

        find the PDB-format structure files of the structures of the

        template and target

    alignment\_filename: where to find an alignment containing both the

        template and the target

    write\_to: where to put the resulting file of zdiffs

    comment: a note to put in the comments of the zdiff files. Record

        the scientific meaning of these zdiffs so they can still be useful

        as your memories about them fade.

    '''

# Open relevant files

**try**:

template\_structure\_file = open(template\_structure\_filename, 'r')

target\_structure\_file = open(target\_structure\_filename, 'r')

alignment\_file = open(alignment\_filename, 'r')

# Load structures with Biopython's PDB file parser

# Daniel's aligned structures are missing some inessential

# information, and as a consequence the parser gives thousands

# of warnings. Gotta ignore these.

with warnings.catch\_warnings():

warnings.simplefilter('ignore')

template\_structure = PDBParser().\

get\_structure(template\_name,

template\_structure\_file)

target\_structure = PDBParser().\

get\_structure(target\_name,

target\_structure\_file)

**except** Exception:

**print**('problems opening template structure file ' \

+ template\_structure\_filename)

**print**('and target structure file ' + target\_structure\_filename)

**print**('and alignment file ' + alignment\_filename)

**raise**

**finally**:

**for** i **in** (template\_structure\_file, target\_structure\_file,

alignment\_file):

i.close()

# Open alignment using Biopython's parser

alignment = AlignIO.read(alignment\_filename, format\_)

# Find the template and target sequences in the alignment

**for** seq\_record **in** alignment:

**if** seq\_record.id == template\_name:

templ\_seq = seq\_record

**if** seq\_record.id == target\_name:

targ\_seq = seq\_record

# Calculate zdiff

results = Zdiff((template\_structure, templ\_seq),

(target\_structure, targ\_seq))

# Write results to a file

with open(write\_to, 'w') as o:

# Write some coments so I know which zdiff file this is

o.write('# Template: ')

o.write(template\_name + ' (' + template\_structure\_filename + ')\n')

o.write('# Target: ')

o.write(target\_name + ' (' + target\_structure\_filename + ')\n')

o.write('# Alignment: ' + alignment\_filename + '\n')

**for** line **in** comment.split('\n'):

o.write('# ' + line + '\n')

# Write the actual data

# Weird quirk of the zdiff objects - before giving a report, it

# requires the id's of the structures. I was planning on making

# a zdiff object support more than two structures, so you give

# it the id's to tell it which pair you want, out of all those

# in the object. I may even have implemented this.

**for** resi, zdiff **in** results.resi\_report(template\_name, target\_name):

o.write(str(resi) + ', ' + str(zdiff) + '\n')

**def** calc\_all(input\_dir, output\_dir, comment, format\_='clustal'):

'''Calculate zdiffs for all alignments in a given folder. Add to

    each zdiff the given comment (multiline comments will still work).

    There are some harsh requirements on the names and contents

    of the alignments. The names must be of the form

    '{clustername}, {pdbid} as target, {pdbid} as template', followed by

    whatever file extension. The format must be whatever is given in the

    "format\_" keyword argument (clustal by default). The sequence of the

    target protein (the one of which a model is being made) must be

    in the alignment with the id "target\_{pdbid}", and the sequence

    of the protein whose structure is to be used as a template must

    be present with the id "template\_{pdbid}".'''

# Make the directory output\_dir

# Does nothing if it already exists

# Does nothing if more than one directory would have to be created,

# leading to errors downstream

# This is bad coding. You never want to catch an exception in a

# way such taht two different errors get handled the same way,

# when there are two different appropriate responses. I may fix

# this later.

**try**:

os.mkdir(output\_dir)

**except** OSError:

**pass**

**for** alignment\_filename **in** os.listdir(input\_dir):

# Check if it's a zdiff alignment, and if it is, retrieve the name

# of the cluster, the PDBID of the target to be modelled, and the

# PDBID of the template

match = re.match(r"(.\*?), (....) as target, (....) as template",

alignment\_filename)

**if** match **is** None:

**continue**

**else**:

cluster = match.group(1)

target = match.group(2)

template = match.group(3)

# Find the pdb files. Need z values to calculate zdiffs.

target\_path = "comparison structures/" \

+ "{}/{} aligned to daniel's {}.pdb".format(cluster,

target,

template)

template\_path = "comparison structures/" \

+ "{}/aligned\_{}.pdb".format(cluster, template)

# Construct the path and filename of the zdiff file to be created

zdiff\_path = output\_dir + '/' + match.group(0) + '.zdiff'

# Create the zdiffs

# Remember: "template" and "target" are PDBID's

calc('template\_'+template, 'target\_'+target, template\_path,

target\_path, input\_dir + '/' + alignment\_filename, zdiff\_path,

comment, format\_=format\_)

**First, I need to know, of the residue numbers in the zdiff files: are they residue numbers of the target or the template structure? Target I would think; that's how I would write it now.**

**However, interpreting my uncommented code from the Zdiff object, it seems that they are actually from the template structure!**

**def** resi\_report(self, template\_id, unknown\_id):

output = ((pos.resi(template\_id),

pos.zdiff(template\_id, unknown\_id)) \

**for** pos **in** self.positions)

**return** filter(lambda x: x[1] **is** **not** None, output)

**That code needs to be commented, and the resi's should be from the target structure. However, I can go ahead and test now, and fix that later.**

**Hand-calculated zdiffs for alignment "gonnet\_aligned/cluster99, 2OMF as target, 2J1N as template":**

## ****October 5, 2012****

To examine the relevance of the zdiff calculations, I checked how many sequences, in each cluster, had sequence identity with the template that is at least the sequence identity between the zdiff comparison structure and the template. I called this number the "coverage": the number of sequences about which the zdiff calculation has something interesting to say. Really, this is an underestimate, since there are sequences with similar but lower sequence identity with the template.

Here are the results:

Examining cluster 18

Identity between zdiff test protein 1BY5 and template 3EFM is 0.132639791938

Number of sequences closer to the template than the protein for which zdiff was calculated: 338

Total number of sequences in the cluster: 533

Fractional coverage: 0.636022514071

Examining cluster 71

Identity between zdiff test protein 1T16 and template 3BS0 is 0.124210526316

Number of sequences closer to the template than the protein for which zdiff was calculated: 108

Total number of sequences in the cluster: 196

Fractional coverage: 0.55612244898

Examining cluster 73

Identity between zdiff test protein 2MPR and template 1A0S is 0.190889370933

Number of sequences closer to the template than the protein for which zdiff was calculated: 19

Total number of sequences in the cluster: 53

Fractional coverage: 0.377358490566

Examining cluster 99

Identity between zdiff test protein 1PHO and template 2J1N is 0.560224089636

Number of sequences closer to the template than the protein for which zdiff was calculated: 18

Total number of sequences in the cluster: 79

Fractional coverage: 0.240506329114

Examining cluster 99

Identity between zdiff test protein 2OMF and template 2J1N is 0.556473829201

Number of sequences closer to the template than the protein for which zdiff was calculated: 18

Total number of sequences in the cluster: 79

Fractional coverage: 0.240506329114

The code I used to generate this is below.

It is too bad that the available structures in cluster 99 are so similar to the template structure. Fractional coverage is low even for very dissimilar structures.

Looking at cluster18 as a typical example, here are the closest ten sequences (the last ten entries from a list generated by "identities\_with\_template", code below):

[('gi|114046494|ref|Y', 0.29918032786885246),

('gi|68542704|ref|ZP', 0.3011049723756906),

('gi|70730679|ref|YP', 0.30958549222797926),

('gi|94497748|ref|ZP', 0.3106796116504854),

('gi|21244095|ref|NP', 0.31275720164609055),

('gi|71737812|ref|YP', 0.3131578947368421),

('gi|50086162|ref|YP', 0.3287101248266297),

('gi|34557768|ref|NP', 0.34584450402144773),

('gi|33593446|ref|NP', 0.776566757493188),

('template\_3EFM', 1.0)]

So the closest sequences in the cluster are pretty close. A random amino acid is more likely than not to be different, but you could tell just looking at them that they're related.

Here are the ten most distant: (the first ten entries of *identities\_with\_template* output):

[('gi|43921118|gb|EAG', 0.04113924050632911),

('gi|43774910|gb|EAF', 0.04328358208955224),

('gi|44004810|gb|EAG', 0.04354136429608128),

('gi|43273969|gb|EAD', 0.05055292259083728),

('gi|43255458|gb|EAC', 0.05221518987341772),

('gi|42963161|gb|EAB', 0.05348460291734198),

('gi|43964796|gb|EAG', 0.05384615384615385),

('gi|44235318|gb|EAH', 0.055714285714285716),

('gi|43926731|gb|EAG', 0.0571870170015456),

('gi|43549608|gb|EAE', 0.057488653555219364)]

Very little sequence identity.

In cluster18, which I'm examining, the sequence identity between the template and the zdiff test protein is about 13%. Coverage is 63%. But what is considered "covered" extends down to 12%, coverage shoots up to 86%. Here, *x* is the list of (sequence name, identity with template) pairs returned by *identities\_with\_template* for cluster18:

In: x[72]

Out: ('gi|11230853|gb|AAG', 0.12029161603888214)

In: (len(x)-72) / len(x)

Out: 0.864915572233

Here's the code for the functions I've been using for this:

**from** \_\_future\_\_ **import** division

**import** Bio.AlignIO

**import** glob

**import** re

**def** identity(seq1, seq2):

'''Return fraction identical between two sequences'''

# Check that the sequences are the same length - this will always

# be the case for aligned sequences

**if** len(seq1) != len(seq2):

**raise** ValueError('Sequences must be the same length')

# Count number of positions that are the same, and number of positions

# that are different. Skip positions in which both are gaps.

same = 0

different = 0

**for** top, bottom **in** zip(seq1, seq2):

# Skip a pair of gaps

**if** top == '-' **and** bottom == '-':

**continue**

**else**:

**if** top == bottom:

same += 1

**elif** top != bottom:

different += 1

# Return fraction identical - number between zero and 1

**return** same / (same + different)

**class** AmbiguousMarker(Exception):

'''Raised by find\_in\_alignment when a marker is found in the names

    or identifiers of two or more sequences of an alignment'''

**pass**

**class** MeaninglessMarker(Exception):

'''Raised by find\_in\_alignment when a marker is not found in the names

    or identifiers of any sequence in an alignment'''

**pass**

**def** find\_in\_alignment(alignment, marker):

marked\_seq = None

**for** seq **in** alignment:

# I don't really know the difference between names and id's, so

# check both:

**if** marker **in** seq.name **or** marker **in** seq.id:

# There should only be one

**if** marked\_seq **is** **not** None:

**raise** AmbiguousMarker('More than one sequence has "'\

+ marker + '" in its '\

+ 'name or id')

**else**:

marked\_seq = seq

# There should be more than zero

**if** marked\_seq **is** None:

**raise** MeaninglessMarker('No sequence has {0} in its name or id'\

.format(marker))

**return** marked\_seq

**def** identities\_with\_template(alignment, template\_identifier):

'''Return a list of each sequence identifier, along with its sequence

    identity with the sequence of a template structure. The template

    structure's sequence must be in the alignment, and must be the only

    sequence with the string given as the second argument contained within

    its name or id.

    Returned list is sorted from lowest sequence identity to highest.

    Raises a AmbiguousMarker exception if more than one sequence is

    identified as a template, and a MeaninglessMarker exception of no

    sequences are.'''

# Find the template sequence

template\_seq = find\_in\_alignment(alignment, template\_identifier)

# Make a list of sequence identities with the template

identities = list()

**for** seq **in** alignment:

identities.append((seq.id, identity(template\_seq, seq)))

**return** sorted(identities, key=lambda x: x[1])

**def** identity\_of\_pair(alignment, marker1, marker2):

'''Given an alignment, return the sequence identity of

    two sequences that contain, in their names or id's, the given markers.

    Raise an AmbiguousMarker exception if either marker matches two

    or more sequences, and a MeaninglessMarker exception if either marker

    matches no sequence.'''

# Find the sequences

seq1 = find\_in\_alignment(alignment, marker1)

seq2 = find\_in\_alignment(alignment, marker2)

# Return the sequence identity as a number from 0 to 1

**return** identity(seq1, seq2)

Here's the code for the specific script that, using these functions, calculated how many sequences in each cluster were closer to the template than the zdiff comparison protein:

# Find out how many sequences in each cluster are closer to the

# template than the protein used in the zdiff comparison

**for** filename **in** glob.glob('gonnet\_aligned/cluster\*'):

# Discover the cluster number, target, and template in the alignment

info\_re = r"cluster(\d\d), (....) as target, (....) as template"

clusternum, target, template = re.search(info\_re, filename).groups()

# Parse the alignment

alignment = Bio.AlignIO.read(filename, 'clustal')

# Announce which alignment is being examined

**print**('Examining cluster ' + str(clusternum))

# Report the sequence identity between the target and the template

temp\_targ\_id = identity\_of\_pair(alignment, 'target', 'template')

**print**('Identity between zdiff test protein {} and template {} is {}'\

.format(target, template, temp\_targ\_id))

# Count how many have at least this much sequence identity with the

# template

# identities\_with\_template returns a list sorted from lowest to

# highest, so this reduces to the problem of finding out how far in

# the list is the target

id\_list = identities\_with\_template(alignment, 'template')

# Search through just the names of the sequences

**for** index, name **in** enumerate(x[0] **for** x **in** id\_list):

**if** 'target' **in** name:

target\_index = index

**break**

number\_above = len(id\_list) - target\_index

# Fractional coverage is number above plus one over total. The plus

# one is to include the target sequence itself, which obviously is part

# of the coverage.

fractional\_coverage = (number\_above+1) / len(alignment)

# Report the results

**print**('Number of sequences closer to the template than the protein '

+ 'for which zdiff was calculated: ' + str(number\_above))

**print**('Total number of sequences in the cluster: '\

+ str(len(alignment)))

**print**('Fractional coverage: ' + str(fractional\_coverage))

# Print a blank line as a separator before the results for the next

# cluster

**print**('')

Alright, now to hand-calculate a few zdiff's to make sure my code isn't completely insane. As noted in the last entry, it's a little insane: the zdiff's are associated with the residue number of the *template* structure, not of the protein being modeled! However, keeping this in mind, I am going to check if it's working.

**Hand-calculated zdiffs for alignment "gonnet\_aligned/cluster99, 2OMF as target, 2J1N as template":**

I'm opening the alignment in ClustalX, and deleting all sequences except for those of the template and the target.

I opened "gonnet\_zdiff\cluster99, 2OMF as target, 2J1N as template.zdiff" in a text editor.

In PyMOL, I opened Daniel's aligned structure of 2J1N, "comparison structures\cluster99\aligned\_2J1N.pdb". I also opened the structure of 2OMF that, and this is important, that I have previously structurally aligned to Daniel's 2J1N structure: " cluster99\2OMF aligned to daniel's 2J1N.pdb"

The first five residues of the proteins are aligned with each other. I'll check the zdiff on these first.

In PyMOL's sequence viewer, I selected the first five residues of 2J1N, and entered the following command, receiving the following output:

PyMOL>iterate\_state 1, sele & n. ca, print(str(resi) + ': ' + str(z))

1: -12.1529998779

2: -14.2550001144

3: -14.0089998245

4: -15.4919996262

5: -18.2070007324

IterateState: iterated over 5 atom coordinate states.

Then, I did the same for 2OMF.

PyMOL>iterate\_state 1, sele & n. ca, print(str(resi) + ': ' + str(z))

1: -11.9630002975

2: -14.1680002213

3: -14.0349998474

4: -15.5559997559

5: -18.2000007629

IterateState: iterated over 5 atom coordinate states.

These differences are: -.19, -.09, .02, .07, 2.71

From the zdiff file, "gonnet\_zdiff\cluster99, 2OMF as target, 2J1N as template.zdiff":

1, -0.19

2, -0.0869999

3, 0.026

4, 0.0640001

5, -0.00699997

They're all right on target, except for the sign of 4, and 5 is way off. I obviously miscalculted number five, looking back at the numbers from PyMOL. I don't know about four, but I don't care about sign errors, and it was probably me, not the program.

That check was for a part before there are any gaps. If there's some screwup happening due to the gaps, I wouldn't see it here.

The last time I checked zdiff's by hand, I wrote a wonderful little script called *find\_seq*, which gives the residue number of the first instance of a specified 5-residue sequence. Its overly specific nature and inflexibility remind me of a wizard's spell for some reason, like from Dungeons and Dragons I guess. The fact that it's completely uncommented complete's the effect that it's compied from the arcane notation in a wizard's spellbook.

**from** collections **import** deque

**from** sundries **import** one\_letter

stored.one\_letter = one\_letter

**def** find\_seq(target):

stored.resi\_resn = list()

cmd.iterate('n. ca', 'stored.resi\_resn.append((resi, one\_letter[resn]))')

last\_five = deque(stored.resi\_resn[:5])

**for** resi, resn **in** stored.resi\_resn[5:]:

**if** ''.join([x[1] **for** x **in** last\_five]) == target:

**return** resi

**else**:

last\_five.popleft()

last\_five.append((resi, resn))

**return** ''.join([x[1] **for** x **in** last\_five])

About halfway through the sequence, after plenty of gaps, there's this segment (copying by sight from the alignment):

2J1N: S-DNFM  
2OMF: YSDDFF

With *find\_seq*, I found and highlighted those five residues of the template 2J1N, checked the surrounding residues against the alignment just in case this five-letter segment repeats. Then, I found their z-coordinates, as before:

PyMOL>print(find\_seq('SDNFM'))

122

PyMOL>iterate\_state 1, sele & n. ca, print(str(resi) + ': ' + str(z))

117: 11.1780004501

118: 12.8280000687

119: 13.7419996262

120: 12.8979997635

121: 9.18000030518

IterateState: iterated over 5 atom coordinate states.

Same procedure for the target 2OMF:

PyMOL>print(find\_seq('YSDDF'))

129

PyMOL>iterate\_state 1, sele & n. ca, print(str(resi) + ': ' + str(z))

124: 11.5970001221

125: 11.1590003967

126: 12.6750001907

127: 13.5450000763

128: 12.0129995346

129: 8.5389995575

IterateState: iterated over 6 atom coordinate states.

These should start in the same place in the alignment, so 117 in template should correspond to 124 in target. Just to check that I started them in the right place, I'm going to check if the residue type of 117 is serine, and the residue type of 124 is tyrosine:

PyMOL>iterate i. 117 & n. ca & aligned\_2J1N, print(resn)

SER

Iterate: iterated over 1 atoms.  
PyMOL>iterate i. 124 & n. ca & 2OMF\_aligned\_to\_daniel\_s\_2J1N, print(resn)

TYR

Iterate: iterated over 1 atoms.

And they are. Since the zdiff files currently associate the zdiff with residue numbers in the template rather than the target, these should correspond to numbers 117 to 121 in the zdiff file. The second position is a gap in the template, so I should ignore residue 125 in the target when hand-calculating the zdiff's.

And the numbers I get are: -.43, .15, .2, .89, .64

From the .zdiff file:

117, -0.419

118, 0.153

119, 0.197

120, 0.885

121, 0.641001

Pretty much the same. As of now I believe these calculations.

I modified the zdiff generating code to use the residue number of the target, not the template, protein.

Here's the code of the modified, somewhat better documented Zdiff class:

**class** Zdiff(object):

**def** \_\_init\_\_(self, \*stru\_seq\_pairlist):

'''Initialize with any number of pairs (structure, sequence)

        where "structure" is a Biopython.PDB structure and

        "sequence" is the sequence of that structure, with gaps,

        taken from a multiple sequence alignment'''

pos\_inputs = list()

**for** structure, sequence **in** stru\_seq\_pairlist:

residues = structure.get\_residues()

pairs\_for\_pos = list()

**for** letter **in** sequence:

**if** letter == '-':

seq\_unit = Gap()

**else**:

seq\_unit = residues.next()

pairs\_for\_pos.append((structure.get\_id(), seq\_unit))

pos\_inputs.append(pairs\_for\_pos)

self.positions = [Position(id\_res\_pairlist) \

**for** id\_res\_pairlist **in** zip(\*pos\_inputs)]

**def** get(self, template\_id, unknown\_id, resi, start=0):

**for** pos **in** self.positions[start:]:

**if** pos.resi(unknown\_id) == resi:

**return** pos.zdiff(template\_id, unknown\_id)

# Raise exception if the target was not found, and the

# return statement never reached:

**raise** NotFoundError('resi {0} of {1} not found' \

.format(resi, unknown\_id))

**def** report(self, template\_id, unknown\_id):

'''Return a list of zdiffs; use resi\_report for a list with

        residue numbers included'''

output = (pos.zdiff(template\_id, unknown\_id) \

**for** pos **in** self.positions)

**return** filter(lambda x: x **is** **not** None, output)

**def** resi\_report(self, template\_id, unknown\_id):

'''Return a list of pairs, (resi, zdiff), where "resi" is

        a residue number in the sequence whose structure is being

        predicted. Residues that are gapped in either sequence are

        not included.'''

output = ((pos.resi(target\_id),

pos.zdiff(template\_id, unknown\_id)) \

**for** pos **in** self.positions)

**return** filter(lambda x: x[1] **is** **not** None, output)