```python
# Implementation for a Disulfide Bond Class object.
# Based on the original C/C++ implementation by Eric G. Suchanek
# Part of the program Proteus, a program for the analysis and modeling of
# protein structures, with an emphasis on disulfide bonds.
# Author: Eric G. Suchanek, PhD
# Last revision: 12/18/22
# Cα Cβ Sγ

import math
import numpy

from proteusPy import *

from proteusPy.atoms import *
from proteusPy.DisulfideExceptions import *
from proteusPy.DisulfideGlobals import *
from proteusPy.proteusGlobals import *

from Bio.PDB import Select, Vector, PDBParser
from Bio.PDB.vectors import calc_dihedral

import pyvista as pv

# float init for class
_FLOAT_INIT = -999.9

# tqdm progress bar width
_PBAR_COLS = 100

# make a colormap in vector space from starting color to
# ending color

def cmap_vector(strtc, endc, steps):
    # make a colormap in vector space
    # starting and ending colors

    newcol = numpy.zeros(shape=3)
    cdir = numpy.zeros(shape=3)
    res = numpy.zeros(shape=(steps,3))

    # color direction vector and length
    cdir = endc - strtc
    clen = math.dist(strtc, endc)
    cdir /= clen # normalize direction

    # delta along color vector
    dlta = clen / steps
    for i in range(steps):
```

```
49          newcol = strtc + cdir * i * dlta
50          res[i] = newcol
51      return res
52
53  # DisulfideList class definition.
54  # I extend UserList to handle lists of Disulfide objects.
55  # Indexing and slicing are supported, sorting is based on energy
56
57  class DisulfideList(UserList):
58      '''
59      Class provides a sortable list for Disulfide objects.
60      Indexing and slicing are supported, and normal list operations like
…   .insert, .append and .extend.
61      The DisulfideList object must be initialized with an iterable (tuple,
…   list) and a name.
62
63      Example:
64          from proteusPy.disulfide import DisulfideList, Disulfide,
…   DisulfideLoader
65
66          # make some empty disulfides
67          ss1 = Disulfide('ss1')
68          ss2 = Disulfide('ss2')
69
70          # make a DisulfideList containing ss1, named 'tmp'
71          sslist = DisulfideList([ss1], 'tmp')
72          sslist.append(ss2)
73
74          # load the PDB Disulfide database
75          PDB_SS = None
76          PDB_SS = DisulfideLoader(verbose=True, modeldir=MODELS)
77
78          # extract a disulfide with typical index
79          ss1 = PDB_SS[0]
80          print(f'{ss1.pprint_all()}')
81
82          # grab a subset via slicing
83          subset = DisulfideList(PDB_SS[0:10],'subset')
84      '''
85
86      def __init__(self, iterable, id):
87          self.pdb_id = id
88          super().__init__(self.validate_ss(item) for item in iterable)
89
90      def __getitem__(self, item):
91          if isinstance(item, slice):
92              indices = range(*item.indices(len(self.data)))
93              name = self.data[0].pdb_id
```

```
94              sublist = [self.data[i] for i in indices]
95              return DisulfideList(sublist, name)
96          return UserList.__getitem__(self, item)
97
98      def __setitem__(self, index, item):
99          self.data[index] = self.validate_ss(item)
100
101     def insert(self, index, item):
102         self.data.insert(index, self.validate_ss(item))
103
104     def append(self, item):
105         self.data.append(self.validate_ss(item))
106
107     def extend(self, other):
108         if isinstance(other, type(self)):
109             self.data.extend(other)
110         else:
111             self.data.extend(self._validate_ss(item) for item in other)
112
113     def validate_ss(self, value):
114         if isinstance(value, (Disulfide)):
115             return value
116         raise TypeError(f"Disulfide object expected, got
… {type(value).__name__}")
117
118     def set_id(self, value):
119         self.pdb_id = value
120
121     def get_id(self):
122         return self.pdb_id
123
124     def display(self, style='bs'):
125         '''
126             Create a pyvista Plotter object with linked four windows for
… CPK, ball and stick,
127             wireframe and surface displays for the Disulfide.
128             Argument:
129                 self
130             Returns:
131                 None. Updates internal object.
132         '''
133
134         ssList = self.data
135         tot_ss = len(ssList) # number off ssbonds
136
137         cols = 2
138         rows = (tot_ss + 1) // cols
139         near_range = -10.0
```

```
140         far_range = 10.0
141         i = 0
142 
143         pl = pv.Plotter(window_size=WINSIZE, shape=(rows, cols))
144         pl.camera.clipping_range = (near_range, far_range)
145         pl.add_camera_orientation_widget()
146 
147         for r in range(rows):
148             for c in range(cols):
149                 pl.subplot(r,c)
150                 if i < tot_ss:
151                     pl.enable_anti_aliasing('msaa')
152                     #pl.view_isometric()
153                     pl.add_axes()
154                     ss = ssList[i]
155                     src = ss.pdb_id
156                     enrg = ss.energy
157                     title = f'{src}:
… {ss.proximal}{ss.proximal_chain}-{ss.distal}{ss.distal_chain} Energy:
… {enrg:.2f} kcal/mol'
158 
159                     pl.add_title(title=title, font_size=FONTSIZE)
160                     pl = render_disulfide(ss, pl, style=style)
161                     pl.camera_position = CAMERA_POS
162                 i += 1
163 
164         pl.link_views()
165         pl.camera.zoom(.65)
166 
167         pl.show()
168         pl.close()
169         return
170 
171 # Class definition for a Disulfide bond.
172 class Disulfide:
173     """
174     This class provides an object representing a physical disulfide bond
… that is either
175     extracted from the RCSB protein databank or built using the
… proteusPy.Turtle3D
176     The Disulfide Bond is characterized by the atomic coordinates N, Cα,
… Cβ, C', Sγ
177     for both residues, the dihedral angles X1 - X5 for the disulfide bond
… conformation,
178     a name, proximal resiude number and distal residue number, and
… conformational energy.
179     All atomic coordinates are represented by the BIO.PDB.Vector class.
… The class uses the
```

```
180     internal methods to initialize dihedral angles and approximate energy
    … upon initialization.
181
182     """
183     def __init__(self, name="SSBOND"):
184         """
185         Initialize the class. All positions are set to the origin. The
    … optional string name may be passed.
186         """
187         self.name = name
188         self.proximal = -1
189         self.distal = -1
190         self.energy = _FLOAT_INIT
191         self.proximal_chain = str('')
192         self.distal_chain = str('')
193         self.pdb_id = str('')
194         self.proximal_residue_fullid = str('')
195         self.distal_residue_fullid = str('')
196         self.PERMISSIVE = bool(True)
197         self.QUIET = bool(True)
198         self.ca_distance = _FLOAT_INIT
199         self.torsion_array = numpy.array((_FLOAT_INIT, _FLOAT_INIT,
    … _FLOAT_INIT, _FLOAT_INIT, _FLOAT_INIT))
200         self.pvplotter = None
201
202         # global coordinates for the Disulfide, typically as returned from
    … the PDB file
203         self.n_prox = Vector(0,0,0)
204         self.ca_prox = Vector(0,0,0)
205         self.c_prox = Vector(0,0,0)
206         self.o_prox = Vector(0,0,0)
207         self.cb_prox = Vector(0,0,0)
208         self.sg_prox = Vector(0,0,0)
209         self.sg_dist = Vector(0,0,0)
210         self.cb_dist = Vector(0,0,0)
211         self.ca_dist = Vector(0,0,0)
212         self.n_dist = Vector(0,0,0)
213         self.c_dist = Vector(0,0,0)
214         self.o_dist = Vector(0,0,0)
215
216         # local coordinates for the Disulfide, computed using the Turtle3D
    … in
217         # Orientation #1 these are generally private.
218
219         self._n_prox = Vector(0,0,0)
220         self._ca_prox = Vector(0,0,0)
221         self._c_prox = Vector(0,0,0)
222         self._o_prox = Vector(0,0,0)
```

```
223         self._cb_prox = Vector(0,0,0)
224         self._sg_prox = Vector(0,0,0)
225         self._sg_dist = Vector(0,0,0)
226         self._cb_dist = Vector(0,0,0)
227         self._ca_dist = Vector(0,0,0)
228         self._n_dist = Vector(0,0,0)
229         self._c_dist = Vector(0,0,0)
230         self._o_dist = Vector(0,0,0)
231 
232         # Dihedral angles for the disulfide bond itself, set to
… _FLOAT_INIT
233         self.chi1 = _FLOAT_INIT
234         self.chi2 = _FLOAT_INIT
235         self.chi3 = _FLOAT_INIT
236         self.chi4 = _FLOAT_INIT
237         self.chi5 = _FLOAT_INIT
238 
239         # I initialize an array for the torsions which will be used for
… comparisons
240         self.dihedrals = numpy.array((_FLOAT_INIT, _FLOAT_INIT,
… _FLOAT_INIT, _FLOAT_INIT, _FLOAT_INIT), "d")
241 
242     def internal_coords(self) -> numpy.array:
243         res_array = numpy.zeros(shape=(6,3))
244 
245         res_array = numpy.array((
246             self._n_prox.get_array(),
247             self._ca_prox.get_array(),
248             self._c_prox.get_array(),
249             self._o_prox.get_array(),
250             self._cb_prox.get_array(),
251             self._sg_prox.get_array(),
252             self._n_dist.get_array(),
253             self._ca_dist.get_array(),
254             self._c_dist.get_array(),
255             self._o_dist.get_array(),
256             self._cb_dist.get_array(),
257             self._sg_dist.get_array(),
258         ))
259         return res_array
260 
261     def cofmass(self) -> numpy.array:
262         res = numpy.zeros(shape=(6,3))
263         res = self.internal_coords()
264         return res.mean(axis=0)
265 
266     def internal_coords_res(self, resnumb) -> numpy.array:
267         res_array = numpy.zeros(shape=(6,3))
```

```
268
269         if resnumb == self.proximal:
270             res_array = numpy.array((
271                 self._n_prox.get_array(),
272                 self._ca_prox.get_array(),
273                 self._c_prox.get_array(),
274                 self._o_prox.get_array(),
275                 self._cb_prox.get_array(),
276                 self._sg_prox.get_array(),
277             ))
278             return res_array
279         elif resnumb == self.distal:
280             res_array = numpy.array((
281                 self._n_dist.get_array(),
282                 self._ca_dist.get_array(),
283                 self._c_dist.get_array(),
284                 self._o_dist.get_array(),
285                 self._cb_dist.get_array(),
286                 self._sg_dist.get_array(),
287             ))
288             return res_array
289         else:
290             mess = f'-> Disulfide.internal_coords(): Invalid argument.
… Unable to find residue: {resnumb} '
291             raise DisulfideConstructionWarning(mess)
292
293     def reset(self):
294         self.__init__(self)
295
296     def display(self, single=True, style='bs'):
297         src = self.pdb_id
298         enrg = self.energy
299         title = f'{src}:
… {self.proximal}{self.proximal_chain}-{self.distal}{self.distal_chain}
… Energy: {enrg:.2f} kcal/mol'
300
301         near_range = -10.0
302         far_range = 10.0
303         src = self.pdb_id
304         fullname = self.name
305         enrg = self.energy
306         title = f'{src}:
… {self.proximal}{self.proximal_chain}-{self.distal}{self.distal_chain}
… Energy: {enrg:.2f} kcal/mol'
307
308         near_range = -10.0
309         far_range = 10.0
310
```

```
311         pl = pv.Plotter(window_size=WINSIZE)
312         pl.camera.clipping_range = (near_range, far_range)
313         pl.add_title(title=title, font_size=FONTSIZE)
314         pl.enable_anti_aliasing('msaa')
315         pl.add_axes()
316         pl.add_camera_orientation_widget()
317 
318         if single:
319             pl = pv.Plotter(window_size=WINSIZE)
320             pl = render_disulfide(self, pl, style=style)
321             pl.camera_position = CAMERA_POS
322             pl.camera.zoom(.4)
323             pl.link_views()
324 
325         else:
326             pl = pv.Plotter(window_size=WINSIZE, shape=(2,2))
327             pl.subplot(0,0)
328             pl.add_axes()
329             pl.add_title(title=title, font_size=FONTSIZE)
330             pl = render_disulfide(self, pl, style='cpk')
331 
332             pl.subplot(0,1)
333             pl.add_axes()
334             pl.add_title(title=title, font_size=FONTSIZE)
335             pl = render_disulfide(self, pl, style='sb')
336 
337             pl.subplot(1,0)
338             pl.add_axes()
339             pl.add_title(title=title, font_size=FONTSIZE)
340             pl = render_disulfide(self, pl, style='plain')
341 
342         pl.show()
343         pl.close()
344 
345     # comparison operators, used for sorting. keyed to SS bond energy
346     def __lt__(self, other):
347         if isinstance(other, Disulfide):
348             return self.energy < other.energy
349 
350     def __le__(self, other):
351         if isinstance(other, Disulfide):
352             return self.energy <= other.energy
353 
354     def __gt__(self, other):
355         if isinstance(other, Disulfide):
356             return self.energy > other.energy
357 
358     def __ge__(self, other):
```

```
359         if isinstance(other, Disulfide):
360             return self.energy >= other.energy
361 
362     def __eq__(self, other):
363         if isinstance(other, Disulfide):
364             return self.energy == other.energy
365 
366     def __ne__(self, other):
367         if isinstance(other, Disulfide):
368             return self.energy != other.energy
369 
370     # repr functions. The class is large, so I split it up into sections
371     def repr_ss_info(self):
372         """
373         Representation for the Disulfide class
374         """
375         s1 = f'<Disulfide {self.name} SourceID: {self.pdb_id} Proximal:
… {self.proximal} {self.proximal_chain} Distal: {self.distal}
… {self.distal_chain}'
376         return s1
377 
378     def repr_ss_coords(self):
379         s2 = f'\nProximal Coordinates:\n   N: {self.n_prox}\n   Cα:
… {self.ca_prox}\n   C: {self.c_prox}\n   O: {self.o_prox}\n   Cβ:
… {self.cb_prox}\n   Sγ: {self.sg_prox}\n\n'
380         s3 = f'Distal Coordinates:\n   N: {self.n_dist}\n   Cα:
… {self.ca_dist}\n   C: {self.c_dist}\n   O: {self.o_dist}\n   Cβ:
… {self.cb_dist}\n   Sγ: {self.sg_dist}\n\n'
381         stot = f'{s2} {s3}'
382         return stot
383 
384     def repr_ss_conformation(self):
385         s4 = f'Conformation: (X1-X5):  {self.chi1:.3f}°, {self.chi2:.3f}°,
… {self.chi3:.3f}°, {self.chi4:.3f}° {self.chi5:.3f}° '
386         s5 = f'Energy: {self.energy:.3f} kcal/mol'
387         stot = f'{s4} {s5}'
388         return stot
389 
390     def repr_ss_local_coords(self):
391         """
392         Representation for the Disulfide class, internal coordinates.
393         """
394         s2i = f'Proximal Internal Coordinates:\n   N: {self._n_prox}\n
… Cα: {self._ca_prox}\n   C: {self._c_prox}\n   O: {self._o_prox}\n   Cβ:
… {self._cb_prox}\n   Sγ: {self._sg_prox}\n\n'
395         s3i = f'Distal Internal Coordinates:\n   N: {self._n_dist}\n   Cα:
… {self._ca_dist}\n   C: {self._c_dist}\n   O: {self._o_dist}\n   Cβ:
… {self._cb_dist}\n   Sγ: {self._sg_dist}\n'
```

```
396         stot = f'{s2i} {s3i}'
397         return stot
398 
399     def repr_ss_chain_ids(self):
400         return(f'Proximal Chain fullID: <{self.proximal_residue_fullid}>
… Distal Chain fullID: <{self.distal_residue_fullid}>')
401 
402     def __repr__(self):
403         """
404         Representation for the Disulfide class
405         """
406 
407         s1 = self.repr_ss_info()
408         res = f'{s1}>'
409         return res
410 
411     def pprint(self):
412         """
413         pretty print general info for the Disulfide
414         """
415 
416         s1 = self.repr_ss_info()
417         s4 = self.repr_ss_conformation()
418         res = f'{s1} {s4}>'
419         return res
420 
421     def pprint_all(self):
422         """
423         pretty print all info for a Disulfide
424         """
425 
426         s1 = self.repr_ss_info() + '\n'
427         s2 = self.repr_ss_coords()
428         s3 = self.repr_ss_local_coords()
429         s4 = self.repr_ss_conformation()
430         s5 = self.repr_chain_ids()
431         res = f'{s1} {s5} {s2} {s3} {s4} >'
432         return res
433 
434     def _handle_SS_exception(self, message):
435         """Handle exception (PRIVATE).
436 
437         This method catches an exception that occurs in the Disulfide
438         object (if PERMISSIVE), or raises it again, this time adding the
439         PDB line number to the error message.
440         """
441         # message = "%s at line %i." % (message)
442         message = f'{message}'
```

```
443
444         if self.PERMISSIVE:
445             # just print a warning - some residues/atoms may be missing
446             warnings.warn(
447                 "DisulfideConstructionException: %s\n"
448                 "Exception ignored.\n"
449                 "Some atoms may be missing in the data structure."
450                 % message,
451                 DisulfideConstructionWarning,
452             )
453         else:
454             # exceptions are fatal - raise again with new message
…  (including line nr)
455             raise DisulfideConstructionException(message) from None
456
457     def print_compact(self):
458         return(f'{self.repr_ss_info()} {self.repr_ss_conformation()}')
459
460     def repr_conformation(self):
461         return(f'{self.repr_ss_conformation()}')
462
463     def repr_coords(self):
464         return(f'{self.repr_ss_coords()}')
465
466     def repr_internal_coords(self):
467         return(f'{self.repr_ss_local_coords()}')
468
469     def repr_chain_ids(self):
470         return(f'{self.repr_ss_chain_ids()}')
471
472     def set_permissive(self, perm: bool) -> None:
473         self.PERMISSIVE = perm
474
475     def get_permissive(self) -> bool:
476         return self.PERMISIVE
477
478     def set_quiet(self, perm: bool) -> None:
479         self.QUIET = perm
480
481     def get_quiet(self) -> bool:
482         return self.QUIET
483
484     def get_full_id(self):
485         return((self.proximal_residue_fullid, self.distal_residue_fullid))
486
487     def initialize_disulfide_from_chain(self, chain1, chain2, proximal,
…  distal):
488         '''
```

```
489         Initialize a new Disulfide object with atomic coordinates from the
…  proximal and
490         distal coordinates, typically taken from a PDB file.
491
492         Arguments:
493             chain1: list of Residues in the model, eg: chain = model['A']
494             chain2: list of Residues in the model, eg: chain = model['A']
495             proximal: proximal residue sequence ID
496             distal: distal residue sequence ID
497
498         Returns: none. The internal state is modified.
499         '''
500
501         id = chain1.get_full_id()[0]
502
503         self.pdb_id = id
504
505         # create a new Disulfide object
506         chi1 = chi2 = chi3 = chi4 = chi5 = _FLOAT_INIT
507
508         prox = int(proximal)
509         dist = int(distal)
510
511         prox_residue = chain1[prox]
512         dist_residue = chain2[dist]
513
514         if (prox_residue.get_resname() != 'CYS' or
…  dist_residue.get_resname() != 'CYS'):
515             print(f'build_disulfide() requires CYS at both residues:
…  {prox} {prox_residue.get_resname()} {dist} {dist_residue.get_resname()}
…  Chain: {prox_residue.get_segid()}')
516
517         # set the objects proximal and distal values
518         self.set_resnum(proximal, distal)
519
520         self.proximal_chain = chain1.get_id()
521         self.distal_chain = chain2.get_id()
522
523         self.proximal_residue_fullid = prox_residue.get_full_id()
524         self.distal_residue_fullid = dist_residue.get_full_id()
525
526
527         # grab the coordinates for the proximal and distal residues as
…  vectors so we can do math on them later
528
529         # proximal residue
530         if self.QUIET:
531             warnings.filterwarnings("ignore",
```

```
category=DisulfideConstructionWarning)
        try:
            n1 = prox_residue['N'].get_vector()
            ca1 = prox_residue['CA'].get_vector()
            c1 = prox_residue['C'].get_vector()
            o1 = prox_residue['O'].get_vector()
            cb1 = prox_residue['CB'].get_vector()
            sg1 = prox_residue['SG'].get_vector()

        except Exception:
            raise DisulfideConstructionWarning(f"Invalid or missing coordinates for proximal residue {proximal}") from None

        # distal residue
        try:
            n2 = dist_residue['N'].get_vector()
            ca2 = dist_residue['CA'].get_vector()
            c2 = dist_residue['C'].get_vector()
            o2 = dist_residue['O'].get_vector()
            cb2 = dist_residue['CB'].get_vector()
            sg2 = dist_residue['SG'].get_vector()

        except Exception:
            raise DisulfideConstructionWarning(f"Invalid or missing coordinates for proximal residue {distal}") from None

        # update the positions and conformation
        self.set_positions(n1, ca1, c1, o1, cb1, sg1, n2, ca2, c2, o2, cb2, sg2)

        # calculate and set the disulfide dihedral angles
        self.chi1 = numpy.degrees(calc_dihedral(n1, ca1, cb1, sg1))
        self.chi2 = numpy.degrees(calc_dihedral(ca1, cb1, sg1, sg2))
        self.chi3 = numpy.degrees(calc_dihedral(cb1, sg1, sg2, cb2))
        self.chi4 = numpy.degrees(calc_dihedral(sg1, sg2, cb2, ca2))
        self.chi5 = numpy.degrees(calc_dihedral(sg2, cb2, ca2, n2))

        self.ca_distance = distance3d(self.ca_prox, self.ca_dist)
        self.torsion_array = numpy.array((self.chi1, self.chi2, self.chi3, self.chi4, self.chi5))

        # calculate and set the SS bond torsional energy
        self.compute_torsional_energy()

        # compute and set the local coordinates
        self.compute_local_coords()

    def set_chain_id(self, chain_id):
```

```
575         self.chain_id = chain_id
576
577     def set_positions(self, n_prox: Vector, ca_prox: Vector, c_prox:
…  Vector,
578                       o_prox: Vector, cb_prox: Vector, sg_prox: Vector,
579                       n_dist: Vector, ca_dist: Vector, c_dist: Vector,
580                       o_dist: Vector, cb_dist: Vector, sg_dist: Vector):
581         '''
582         Sets the atomic positions for all atoms in the disulfide bond.
583         Arguments:
584             n_prox
585             ca_prox
586             c_prox
587             o_prox
588             cb_prox
589             sg_prox
590             n_distal
591             ca_distal
592             c_distal
593             o_distal
594             cb_distal
595             sg_distal
596         Returns: None
597         '''
598
599         # deep copy
600         self.n_prox = n_prox.copy()
601         self.ca_prox = ca_prox.copy()
602         self.c_prox = c_prox.copy()
603         self.o_prox = o_prox.copy()
604         self.cb_prox = cb_prox.copy()
605         self.sg_prox = sg_prox.copy()
606         self.sg_dist = sg_dist.copy()
607         self.cb_dist = cb_dist.copy()
608         self.ca_dist = ca_dist.copy()
609         self.n_dist = n_dist.copy()
610         self.c_dist = c_dist.copy()
611         self.o_dist = o_dist.copy()
612
613     def set_conformation(self, chi1, chi2, chi3, chi4, chi5):
614         '''
615         Sets the 5 dihedral angles chi1 - chi5 for the Disulfide object
…  and computes the torsional energy.
616
617         Arguments: chi, chi2, chi3, chi4, chi5 - Dihedral angles in
…  degrees (-180 - 180) for the Disulfide conformation.
618         Returns: None
619         '''
```

```
620
621         self.chi1 = chi1
622         self.chi2 = chi2
623         self.chi3 = chi3
624         self.chi4 = chi4
625         self.chi5 = chi5
626         self.dihedrals = list([chi1, chi2, chi3, chi4, chi5])
627         self.compute_torsional_energy()
628
629     def set_name(self, namestr="Disulfide"):
630         '''
631         Sets the Disulfide's name
632         Arguments: (str)namestr
633         Returns: none
634         '''
635
636         self.name = namestr
637
638     def set_resnum(self, proximal, distal):
639         '''
640         Sets the Proximal and Distal Residue numbers for the Disulfide
641         Arguments:
642             Proximal: Proximal residue number
643             Distal: Distal residue number
644         Returns: None
645         '''
646
647         self.proximal = proximal
648         self.distal = distal
649
650     def TorsionDistance(p1: Vector, p2: Vector):
651         '''
652         Calculate the 5D Euclidean distance for 2 Disulfide torsion_vector
… objects. This is used
653         to compare Disulfide Bond torsion angles to determine their
… torsional
654         'distance'.
655
656         Arguments: p1, p2 Vector objects of dimensionality 5 (5D)
657         Returns: Distance
658         '''
659         _p1 = p1.get_array()
660         _p2 = p2.get_array()
661         if (len(_p1) != 5 or len(_p2) != 5):
662             raise ProteusPyWarning("--> distance5d() requires vectors of
… length 5!")
663         d = math.dist(_p1, _p2)
664         return d
```

```
665
666     def compute_torsional_energy(self):
667         '''
668         Compute the approximate torsional energy for the Disulfide's
…   conformation.
669         Arguments: chi1, chi2, chi3, chi4, chi5 - the dihedral angles for
…   the Disulfide
670         Returns: Energy (kcal/mol)
671         '''
672         # @TODO find citation for the ss bond energy calculation
673         chi1 = self.chi1
674         chi2 = self.chi2
675         chi3 = self.chi3
676         chi4 = self.chi4
677         chi5 = self.chi5
678
679         energy = 2.0 * (cos(torad(3.0 * chi1)) + cos(torad(3.0 * chi5)))
680         energy += cos(torad(3.0 * chi2)) + cos(torad(3.0 * chi4))
681         energy += 3.5 * cos(torad(2.0 * chi3)) + 0.6 * cos(torad(3.0 *
…   chi3)) + 10.1
682
683         self.energy = energy
684
685     def compute_local_coords(self):
686         """
687         Compute the internal coordinates for a properly initialized
…   Disulfide Object.
688         Arguments: SS initialized Disulfide object
689         Returns: None, modifies internal state of the input
690         """
691
692         turt = Turtle3D('tmp')
693         # get the coordinates as numpy.array for Turtle3D use.
694         n = self.n_prox.get_array()
695         ca = self.ca_prox.get_array()
696         c = self.c_prox.get_array()
697         cb = self.cb_prox.get_array()
698         o = self.o_prox.get_array()
699         sg = self.sg_prox.get_array()
700
701         sg2 = self.sg_dist.get_array()
702         cb2 = self.cb_dist.get_array()
703         ca2 = self.ca_dist.get_array()
704         c2 = self.c_dist.get_array()
705         n2 = self.n_dist.get_array()
706         o2 = self.o_dist.get_array()
707
708         turt.orient_from_backbone(n, ca, c, cb, ORIENT_SIDECHAIN)
```

```
709
710         # internal (local) coordinates, stored as Vector objects
711         # to_local returns numpy.array objects
712
713         self._n_prox = Vector(turt.to_local(n))
714         self._ca_prox = Vector(turt.to_local(ca))
715         self._c_prox = Vector(turt.to_local(c))
716         self._o_prox = Vector(turt.to_local(o))
717         self._cb_prox = Vector(turt.to_local(cb))
718         self._sg_prox = Vector(turt.to_local(sg))
719
720         self._n_dist = Vector(turt.to_local(n2))
721         self._ca_dist = Vector(turt.to_local(ca2))
722         self._c_dist = Vector(turt.to_local(c2))
723         self._o_dist = Vector(turt.to_local(o2))
724         self._cb_dist = Vector(turt.to_local(cb2))
725         self._sg_dist = Vector(turt.to_local(sg2))
726
727     def build_model(self, turtle: Turtle3D):
728         """
729         Build a model Disulfide based on the internal dihedral angles.
730         Routine assumes turtle is in orientation #1 (at Ca, headed toward
731         Cb, with N on left), builds disulfide, and updates the object's
  … internal
732         coordinate state. It also adds the distal protein backbone,
733         and computes the disulfide conformational energy.
734
735         Arguments: turtle: Turtle3D object properly oriented for the
  … build.
736         Returns: None. The Disulfide object's internal state is updated.
737         """
738
739         tmp = Turtle3D('tmp')
740         tmp.copy_coords(turtle)
741
742         n = Vector(0, 0, 0)
743         ca = Vector(0, 0, 0)
744         cb = Vector(0, 0, 0)
745         c = Vector(0, 0, 0)
746
747         self.ca_prox = tmp._position
748         tmp.schain_to_bbone()
749         n, ca, cb, c = build_residue(tmp)
750
751         self.n_prox = n
752         self.ca_prox = ca
753         self.c_prox = c
754
```

```
755         tmp.bbone_to_schain()
756         tmp.move(1.53)
757         tmp.roll(self.chi1)
758         tmp.yaw(112.8)
759         self.cb_prox = tmp._position
760 
761         tmp.move(1.86)
762         tmp.roll(self.chi2)
763         tmp.yaw(103.8)
764         self.sg_prox = tmp._position
765 
766         tmp.move(2.044)
767         tmp.roll(self.chi3)
768         tmp.yaw(103.8)
769         self.sg_dist = tmp._position
770 
771         tmp.move(1.86)
772         tmp.roll(self.chi4)
773         tmp.yaw(112.8)
774         self.cb_dist = tmp._position
775 
776         tmp.move(1.53)
777         tmp.roll(self.chi5)
778         tmp.pitch(180.0)
779         tmp.schain_to_bbone()
780         n, ca, cb, c = build_residue(tmp)
781 
782         self.n_dist = n
783         self.ca_dist = ca
784         self.c_dist = c
785 
786         self.compute_torsional_energy()
787 
788 # Class defination ends
789 import copy
790 
791 class DisulfideLoader():
792     '''
793     This class loads .pkl files created from the ExtractDisulfides()
    … routine
794     and initializes itself with their contents. The Disulfide objects are
    … contained
795     in a DisulfideList object and Dict. This makes it possible to access
    … the disulfides by
796     array index or PDB structure ID.\n
797 
798     Example:
799         from proteusPy.disulfide import DisulfideList, Disulfide,
```

```
799…DisulfideLoader
800
801         SS1 = DisulfideList([],'tmp1')
802         SS2 = DisulfideList([],'tmp2')
803
804         PDB_SS = DisulfideLoader()
805         SS1 = PDB_SS[0]         <-- returns a Disulfide object at index 0
806         SS2 = PDB_SS['4yys']    <-- returns a DisulfideList containing all
…disulfides for 4yys
807         SS3 = PDB_SS[:10]       <-- returns a DisulfideList containing the
…slice
808     '''
809
810     def __init__(self, verbose=True, modeldir=MODEL_DIR,
…picklefile=SS_PICKLE_FILE,
811                  pickle_dict_file=SS_DICT_PICKLE_FILE,
812                  torsion_file=SS_TORSIONS_FILE):
813         self.ModelDir = modeldir
814         self.PickleFile = f'{modeldir}{picklefile}'
815         self.PickleDictFile = f'{modeldir}{pickle_dict_file}'
816         self.TorsionFile = f'{modeldir}{torsion_file}'
817         self.SSList = DisulfideList([], 'ALL_PDB_SS')
818         self.SSDict = {}
819         self.TorsionDF = pd.DataFrame()
820         self.TotalDisulfides = 0
821         self.IDList = []
822
823         # create a dataframe with the following columns for the disulfide
…conformations extracted from the structure
824         df_cols = ['source', 'ss_id', 'proximal', 'distal', 'chi1',
…'chi2', 'chi3', 'chi4', 'chi5', 'energy']
825         SS_df = pd.DataFrame(columns=df_cols, index=['source'])
826         _SSList = DisulfideList([], 'ALL_PDB_SS')
827
828         idlist = []
829         if verbose:
830             print(f'Reading disulfides from: {self.PickleFile}')
831         with open(self.PickleFile, 'rb') as f:
832             self.SSList = pickle.load(f)
833
834         self.TotalDisulfides = len(self.SSList)
835
836         if verbose:
837             print(f'Disulfides Read: {self.TotalDisulfides}')
838             print(f'Reading disulfide dict from: {self.PickleDictFile}')
839
840         with open(self.PickleDictFile, 'rb') as f:
841             self.SSDict = pickle.load(f)
```

```
842             for key in self.SSDict:
843                 idlist.append(key)
844             self.IDList = idlist.copy()
845             totalSS_dict = len(self.IDList)
846 
847         if verbose:
848             print(f'Reading Torsion DF {self.TorsionFile}.')
849 
850         self.TorsionDF = pd.read_csv(self.TorsionFile)
851 
852         if verbose:
853             print(f'Read torsions DF.')
854             print(f'PDB IDs parsed: {totalSS_dict}')
855             print(f'Total Space Used: {sys.getsizeof(self.SSList) +
… sys.getsizeof(self.SSDict) + sys.getsizeof(self.TorsionDF)} bytes.')
856         return
857 
858     # overload __getitem__ to handle slicing and indexing
859     def __getitem__(self, item):
860         if isinstance(item, slice):
861             indices = range(*item.indices(len(self.SSList)))
862             # return [self.SSList[i] for i in indices]
863             name = self.SSList[0].pdb_id
864             sublist = [self.SSList[i] for i in indices]
865             return DisulfideList(sublist, name)
866 
867         if isinstance(item, int):
868             if (item < 0 or item >= self.TotalDisulfides):
869                 mess = f'DisulfideDataLoader error. Index {item} out of
… range 0-{self.TotalDisulfides - 1}'
870                 raise DisulfideException(mess)
871             else:
872                 return self.SSList[item]
873 
874         try:
875             res = self.SSDict[item]
876         except KeyError:
877             mess = f'! Cannot find key {item} in SSBond dict!'
878             raise DisulfideException(mess)
879         return res
880 
881     def __setitem__(self, index, item):
882         self.SSList[index] = self.validate_ss(item)
883 
884     def getlist(self):
885         return self.SSList.copy()
886 
887     def getdict(self) -> dict:
```

```
888         return copy.deepcopy(self.SSDict)
889
890     def getTorsions(self):
891         return copy.deepcopy(self.TorsionDF)
892
893     def validate_ss(self, value):
894         if isinstance(value, (Disulfide)):
895             return value
896         raise TypeError(f"Disulfide object expected, got
{type(value).__name__}")
897
898     def copy(self):
899         return copy.deepcopy(self)
900
901     def display_overlay(self, pdbid: str):
902         '''
903         Render all disulfides for a given PDB ID overlaid in stick mode
against
904         a common coordinate frames. This allows us to see all of the
disulfides
905         at one time in a single view. Colors vary smoothy between bonds.
906
907         Arguments:
908             PDB_SS: DisulfideLoader object initialized with the database.
909             pdbid: the actual PDB id string
910
911         Returns: None.
912         '''
913
914         ssbonds = self[pdbid]
915
916         tot_ss = len(ssbonds) # number off ssbonds
917         title = f'Disulfides for {pdbid}: ({tot_ss} total)'
918
919         pl = pv.Plotter(window_size=WINSIZE)
920         pl.add_title(title=title, font_size=FONTSIZE)
921         pl.enable_anti_aliasing('msaa')
922
923         # pl.view_isometric()
924         pl.add_camera_orientation_widget()
925         pl.camera_position = CAMERA_POS
926         pl.add_axes()
927
928         # make a colormap in vector space
929         # starting and ending colors
930         strtc = numpy.array([.1, .1, 1])
931         endc = numpy.array([.95, .1, .15])
932         mycol = cmap_vector(strtc, endc, tot_ss)
```

```
933
934         i = 0
935         for ss in ssbonds:
936             pl = render_disulfide(ss, pl, style='st', bondcolor=mycol[i])
937             i += 1
938
939         pl.camera.zoom(.4)
940         pl.show()
941         pl.close()
942         return
943
944     def display(self, style='bs'):
945         '''
946         Display the Disulfides
947         Argument:
948             self
949         Returns:
950             None. Updates internal object.
951         '''
952
953         FONTSIZE = 10 # fontsize
954
955         ssList = self.SSList
956         tot_ss = len(ssList) # number off ssbonds
957
958         cols = 2
959         rows = (tot_ss + 1) // cols
960         near_range = -10.0
961         far_range = 10.0
962         i = 0
963
964         pl = pv.Plotter(window_size=WINSIZE, shape=(rows, cols))
965         pl.camera.clipping_range = (near_range, far_range)
966         pl.add_camera_orientation_widget()
967
968         for r in range(rows):
969             for c in range(cols):
970                 pl.subplot(r,c)
971                 if i < tot_ss:
972                     pl.enable_anti_aliasing('msaa')
973                     #pl.view_isometric()
974                     pl.add_axes()
975                     ss = ssList[i]
976                     src = ss.pdb_id
977                     enrg = ss.energy
978                     title = f'{src}:
… {ss.proximal}{ss.proximal_chain}-{ss.distal}{ss.distal_chain} Energy:
… {enrg:.2f} kcal/mol'
```

```
                    pl.add_title(title=title, font_size=FONTSIZE)
                    pl = render_disulfide(ss, pl, style=style)
                    pl.camera_position = CAMERA_POS
                i += 1

        pl.link_views()
        pl.camera.zoom(.65)

        pl.show()
        pl.close()
        return

# class ends

class CysSelect(Select):
    def accept_residue(self, residue):
        if residue.get_name() == 'CYS':
            return True
        else:
            return False

def distance3d(p1: Vector, p2: Vector):
    '''
    Calculate the 3D Euclidean distance for 2 Vector objects

    Arguments: p1, p2 Vector objects of dimensionality 3 (3D)
    Returns: Distance
    '''
    _p1 = p1.get_array()
    _p2 = p2.get_array()
    if (len(_p1) != 3 or len(_p2) != 3):
        raise ProteusPyWarning("--> distance3d() requires vectors of
length 3!")
    d = math.dist(_p1, _p2)
    return d

def name_to_id(fname: str):
    '''return an entry id for filename pdb1crn.ent -> 1crn'''
    ent = fname[3:-4]
    return ent

def torad(deg):
    return(numpy.radians(deg))

def todeg(rad):
    return(numpy.degrees(rad))

```

```
1026 def parse_ssbond_header_rec(ssbond_dict: dict) -> list:
1027     '''
1028     Parse the SSBOND dict returned by parse_pdb_header.
1029     NB: Requires EGS-Modified BIO.parse_pdb_header.py
1030 
1031     Arguments:
1032         ssbond_dict: the input SSBOND dict
1033     Returns: a list of tuples representing the proximal, distal residue
1034              ids for the disulfide.
1035 
1036     '''
1037     disulfide_list = []
1038     for ssb in ssbond_dict.items():
1039         disulfide_list.append(ssb[1])
1040 
1041     return disulfide_list
1042 
1043 #
1044 # function reads a comma separated list of PDB IDs and download the
  … corresponding
1045 # .ent files to the PDB_DIR global.
1046 # Used to download the list of proteins containing at least one SS bond
1047 # with the ID list generated from: http://www.rcsb.org/
1048 #
1049 
1050 def DownloadDisulfides(pdb_home=PDB_DIR, model_home=MODEL_DIR,
1051                        verbose=False, reset=False) -> None:
1052     '''
1053     Function reads a comma separated list of PDB IDs and downloads them
1054     to the pdb_home path.
1055 
1056     Used to download the list of proteins containing at least one SS bond
1057     with the ID list generated from: http://www.rcsb.org/
1058     '''
1059 
1060     start = time.time()
1061     donelines = []
1062     SS_done = []
1063     ssfile = None
1064 
1065     cwd = os.getcwd()
1066     os.chdir(pdb_home)
1067 
1068     pdblist = PDBList(pdb=pdb_home, verbose=verbose)
1069     ssfilename = f'{model_home}{SS_ID_FILE}'
1070     print(ssfilename)
1071 
1072     # list of IDs containing >1 SSBond record
```

```
1073     try:
1074         ssfile = open(ssfilename)
1075         Line = ssfile.readlines()
1076     except Exception:
1077         raise DisulfideIOException(f'Cannot open file: {ssfile}')
1078 
1079     for line in Line:
1080         entries = line.split(',')
1081 
1082     print(f'Found: {len(entries)} entries')
1083     completed = {'xxx'} # set to keep track of downloaded
1084 
1085     # file to track already downloaded entries.
1086     if reset==True:
1087         completed_file = open(f'{model_home}ss_completed.txt', 'w')
1088         donelines = []
1089         SS_DONE = []
1090     else:
1091         completed_file = open(f'{model_home}ss_completed.txt', 'w+')
1092         donelines = completed_file.readlines()
1093 
1094     if len(donelines) > 0:
1095         for dl in donelines[0]:
1096             # create a list of pdb id already downloaded
1097             SS_done = dl.split(',')
1098 
1099     count = len(SS_done) - 1
1100     completed.update(SS_done) # update the completed set with what's
…  downloaded
1101 
1102     # Loop over all entries,
1103     pbar = tqdm(entries, ncols=_PBAR_COLS)
1104     for entry in pbar:
1105         pbar.set_postfix({'Entry': entry})
1106         if entry not in completed:
1107             if pdblist.retrieve_pdb_file(entry, file_format='pdb',
…  pdir=pdb_home):
1108                 completed.update(entry)
1109                 completed_file.write(f'{entry},')
1110                 count += 1
1111 
1112     completed_file.close()
1113 
1114     end = time.time()
1115     elapsed = end - start
1116 
1117     print(f'Overall files processed: {count}')
1118     print(f'Complete. Elapsed time: {datetime.timedelta(seconds=elapsed)}
```

```
(h:m:s)')
    os.chdir(cwd)
    return

def build_torsion_df(SSList: DisulfideList) -> pd.DataFrame:
    # create a dataframe with the following columns for the disulfide conformations extracted from the structure
    df_cols = ['source', 'ss_id', 'proximal', 'distal', 'chi1', 'chi2', 'chi3', 'chi4', 'chi5', 'energy', 'ca_distance']
    SS_df = pd.DataFrame(columns=df_cols)

    pbar = tqdm(SSList, ncols=_PBAR_COLS, miniters=400000)
    for ss in pbar:
        #pbar.set_postfix({'ID': ss.name}) # update the progress bar

        new_row = [ss.pdb_id, ss.name, ss.proximal, ss.distal, ss.chi1, ss.chi2,
                   ss.chi3, ss.chi4, ss.chi5, ss.energy, ss.ca_distance]
        # add the row to the end of the dataframe
        SS_df.loc[len(SS_df.index)] = new_row.copy() # deep copy

    return SS_df.copy()

def ExtractDisulfides(numb=-1, verbose=False, quiet=False, pdbdir=PDB_DIR,
                      modeldir=MODEL_DIR, picklefile=SS_PICKLE_FILE,
                      torsionfile=SS_TORSIONS_FILE, problemfile=PROBLEM_ID_FILE,
                      dictfile=SS_DICT_PICKLE_FILE) -> None:
    '''
    This function creates .pkl files needed for the DisulfideLoader class. The Disulfide
    objects are contained in a DisulfideList object and Dict within these files.
    In addition, .csv files containing all of the torsions for the disulfides and
    problem IDs are written.

    Arguments:
        numb:           number of entries to process, defaults to all
        verbose:        more messages
        quiet:          turns of DisulfideConstruction warnings
        pdbdir:         path to PDB files
        modeldir:       path to resulting .pkl files
        picklefile:     name of the disulfide .pkl file
        torsionfile:    name of the disulfide torsion file .csv created
        problemfile:    name of the .csv file containing problem ids
        dictfile:       name of the .pkl file

```

```
1159        Example:
1160            from proteusPy.Disulfide import ExtractDisulfides,
    … DisulfideLoader, DisulfideList
1161
1162            ExtractDisulfides(numb=500, pdbdir=PDB_DIR, verbose=False,
    … quiet=True)
1163
1164            SS1 = DisulfideList([],'All_SS')
1165            SS2 = DisulfideList([], '4yys')
1166
1167            PDB_SS = DisulfideLoader()
1168            SS1 = PDB_SS[0]         <-- returns a Disulfide object at index 0
1169            SS2 = PDB_SS['4yys']    <-- returns a DisulfideList containing all
    … disulfides for 4yys
1170            SS3 = PDB_SS[:10]       <-- returns a DisulfideList containing the
    … slice
1171        '''
1172
1173        entrylist = []
1174        problem_ids = []
1175        bad = 0
1176
1177        # we use the specialized list class DisulfideList to contain our
    … disulfides
1178        # we'll use a dict to store DisulfideList objects, indexed by the
    … structure ID
1179        All_ss_dict = {}
1180        All_ss_list = []
1181
1182        start = time.time()
1183        cwd = os.getcwd()
1184
1185        # Build a list of PDB files in PDB_DIR that are readable. These files
    … were downloaded
1186        # via the RCSB web query interface for structures containing >= 1 SS
    … Bond.
1187
1188        os.chdir(pdbdir)
1189
1190        ss_filelist = glob.glob(f'*.ent')
1191        tot = len(ss_filelist)
1192
1193        if verbose:
1194            print(f'PDB Directory {pdbdir} contains: {tot} files')
1195
1196        # the filenames are in the form pdb{entry}.ent, I loop through them
    … and extract
1197        # the PDB ID, with Disulfide.name_to_id(), then add to entrylist.
```

```
1198
1199        for entry in ss_filelist:
1200            entrylist.append(name_to_id(entry))
1201
1202        # create a dataframe with the following columns for the disulfide
…   conformations extracted from the structure
1203
1204        df_cols = ['source', 'ss_id', 'proximal', 'distal', 'chi1', 'chi2',
…   'chi3', 'chi4', 'chi5', 'energy', 'ca_distance']
1205        SS_df = pd.DataFrame(columns=df_cols)
1206
1207        # define a tqdm progressbar using the fully loaded entrylist list. If
…   numb is passed then
1208        # only do the last numb entries.
1209        if numb > 0:
1210            pbar = tqdm(entrylist[:numb], ncols=_PBAR_COLS)
1211        else:
1212            pbar = tqdm(entrylist, ncols=_PBAR_COLS)
1213
1214        # loop over ss_filelist, create disulfides and initialize them
1215        for entry in pbar:
1216            pbar.set_postfix({'ID': entry, 'Bad': bad}) # update the progress
…   bar
1217
1218            # returns an empty list if none are found.
1219            sslist = DisulfideList([], entry)
1220            sslist = load_disulfides_from_id(entry, model_numb=0,
…   verbose=verbose, quiet=quiet, pdb_dir=pdbdir)
1221            if len(sslist) > 0:
1222                for ss in sslist:
1223                    All_ss_list.append(ss)
1224                    new_row = [ss.pdb_id, ss.name, ss.proximal, ss.distal,
…   ss.chi1, ss.chi2, ss.chi3, ss.chi4, ss.chi5, ss.energy, ss.ca_distance]
1225                    # add the row to the end of the dataframe
1226                    SS_df.loc[len(SS_df.index)] = new_row.copy() # deep copy
1227
1228                All_ss_dict[entry] = sslist
1229            else:
1230                # at this point I really shouldn't have any bad non-parsible
…   file
1231                bad += 1
1232                problem_ids.append(entry)
1233                os.remove(f'pdb{entry}.ent')
1234
1235        if bad > 0:
1236            prob_cols = ['id']
1237            problem_df = pd.DataFrame(columns=prob_cols)
1238            problem_df['id'] = problem_ids
```

```
1239
1240            print(f'Found and removed: {len(problem_ids)} problem
    structures.')
1241            print(f'Saving problem IDs to file: {modeldir}{problemfile}')
1242
1243            problem_df.to_csv(f'{modeldir}{problemfile}')
1244        else:
1245            if verbose:
1246                print('No problems found.')
1247
1248        # dump the all_ss array of disulfides to a .pkl file. ~520 MB.
1249        fname = f'{modeldir}{picklefile}'
1250        print(f'Saving {len(All_ss_list)} Disulfides to file: {fname}')
1251
1252        with open(fname, 'wb+') as f:
1253            pickle.dump(All_ss_list, f)
1254
1255        # dump the all_ss array of disulfides to a .pkl file. ~520 MB.
1256        dict_len = len(All_ss_dict)
1257        fname = f'{modeldir}{dictfile}'
1258
1259        print(f'Saving {len(All_ss_dict)} Disulfide-containing PDB IDs to
    file: {fname}')
1260
1261        with open(fname, 'wb+') as f:
1262            pickle.dump(All_ss_dict, f)
1263
1264        # save the torsions
1265        fname = f'{modeldir}{torsionfile}'
1266        print(f'Saving torsions to file: {fname}')
1267
1268        SS_df.to_csv(fname)
1269
1270        end = time.time()
1271        elapsed = end - start
1272
1273        print(f'Disulfide Extraction complete! Elapsed time:
    {datetime.timedelta(seconds=elapsed)} (h:m:s)')
1274
1275        # return to original directory
1276        os.chdir(cwd)
1277        return
1278
1279    def check_chains(pdbid, pdbdir, verbose=True):
1280        '''Returns True if structure has multiple chains of identical length,
    False otherwise'''
1281
1282        parser = PDBParser(PERMISSIVE=True)
```

```
1283     structure = parser.get_structure(pdbid,
…  file=f'{pdbdir}pdb{pdbid}.ent')
1284     ssbond_dict = structure.header['ssbond'] # dictionary of tuples with
…  SSBond prox and distal
1285 
1286     if verbose:
1287         print(f'ssbond dict: {ssbond_dict}')
1288 
1289     same = False
1290     model = structure[0]
1291     chainlist = model.get_list()
1292 
1293     if len(chainlist) > 1:
1294         chain_lens = []
1295         if verbose:
1296             print(f'multiple chains. {chainlist}')
1297         for chain in chainlist:
1298             chain_length = len(chain.get_list())
1299             chain_id = chain.get_id()
1300             if verbose:
1301                 print(f'Chain: {chain_id}, length: {chain_length}')
1302             chain_lens.append(chain_length)
1303 
1304         if numpy.min(chain_lens) != numpy.max(chain_lens):
1305             same = False
1306             if verbose:
1307                 print(f'chain lengths are unequal: {chain_lens}')
1308         else:
1309             same = True
1310             if verbose:
1311                 print(f'Chains are equal length, assuming the same.
…  {chain_lens}')
1312     return(same)
1313 
1314 # NB - this only works with the EGS modified version of
…  BIO.parse_pdb_header.py
1315 def load_disulfides_from_id(struct_name: str,
1316                             pdb_dir = '.',
1317                             model_numb = 0,
1318                             verbose = False,
1319                             quiet=False,
1320                             dbg = False) -> list:
1321     '''
1322     Loads all Disulfides by PDB ID and initializes the Disulfide objects.
1323     Assumes the file is downloaded in the pdb_dir path.
1324 
1325     NB: Requires EGS-Modified BIO.parse_pdb_header.py
1326 
```

```
1327     Arguments:
1328         struct_name: the name of the PDB entry.
1329 
1330         pdb_dir: path to the PDB files, defaults to PDB_DIR
1331 
1332         model_numb: model number to use, defaults to 0 for single
1333         structure files.
1334 
1335         verbose: print info while parsing
1336 
1337     Returns: a list of Disulfide objects initialized from the file.
1338     Example:
1339       Assuming the PDB_DIR has the pdb5rsa.ent file in place calling:
1340 
1341       SS_list = []
1342       SS_list = load_disulfides_from_id('5rsa', verbose=True)
1343 
1344       loads the Disulfides from the file and initialize the disulfide
…  objects, returning
1345       them in the result. '''
1346 
1347     i = 1
1348     proximal = distal = -1
1349     SSList = DisulfideList([], struct_name)
1350     _chaina = None
1351     _chainb = None
1352 
1353     parser = PDBParser(PERMISSIVE=True)
1354 
1355     # Biopython uses the Structure -> Model -> Chain hierarchy to organize
1356     # structures. All are iterable.
1357 
1358     structure = parser.get_structure(struct_name,
…  file=f'{pdb_dir}pdb{struct_name}.ent')
1359     model = structure[model_numb]
1360 
1361     if verbose:
1362         print(f'-> load_disulfide_from_id() - Parsing structure:
…  {struct_name}:')
1363 
1364     ssbond_dict = structure.header['ssbond'] # NB: this requires the
…  modified code
1365 
1366     # list of tuples with (proximal distal chaina chainb)
1367     ssbonds = parse_ssbond_header_rec(ssbond_dict)
1368 
1369     with warnings.catch_warnings():
1370         if quiet:
```

```
1371            #warnings.filterwarnings("ignore",
    … category=DisulfideConstructionWarning)
1372            warnings.filterwarnings("ignore")
1373        for pair in ssbonds:
1374            # in the form (proximal, distal, chain)
1375            proximal = pair[0]
1376            distal = pair[1]
1377            chain1_id = pair[2]
1378            chain2_id = pair[3]
1379
1380            if not proximal.isnumeric() or not distal.isnumeric():
1381                mess = f' -> Cannot parse SSBond record (non-numeric IDs):
    … {struct_name} Prox: {proximal} {chain1_id} Dist: {distal} {chain2_id},
    … ignoring.'
1382                warnings.warn(mess, DisulfideConstructionWarning)
1383                continue
1384            else:
1385                proximal = int(proximal)
1386                distal = int(distal)
1387
1388            if proximal == distal:
1389                mess = f' -> Cannot parse SSBond record (proximal ==
    … distal): {struct_name} Prox: {proximal} {chain1_id} Dist: {distal}
    … {chain2_id}, ignoring.'
1390                warnings.warn(mess, DisulfideConstructionWarning)
1391                continue
1392
1393            _chaina = model[chain1_id]
1394            _chainb = model[chain2_id]
1395
1396            if (_chaina is None) or (_chainb is None):
1397                mess = f' -> NULL chain(s): {struct_name}: {proximal}
    … {chain1_id} - {distal} {chain2_id}, ignoring!'
1398                warnings.warn(mess, DisulfideConstructionWarning)
1399                continue
1400
1401            if (chain1_id != chain2_id):
1402                if verbose:
1403                    mess = (f' -> Cross Chain SS for: Prox: {proximal}
    … {chain1_id} Dist: {distal} {chain2_id}')
1404                    warnings.warn(mess, DisulfideConstructionWarning)
1405                    pass # was break
1406
1407            try:
1408                prox_res = _chaina[proximal]
1409                dist_res = _chainb[distal]
1410
1411            except KeyError:
```

```
1412                    mess = f'Cannot parse SSBond record (KeyError):
…  {struct_name} Prox:  {proximal} {chain1_id} Dist: {distal} {chain2_id},
…  ignoring!'
1413                    warnings.warn(mess, DisulfideConstructionWarning)
1414                    continue
1415
1416                # make a new Disulfide object, name them based on proximal and
…  distal
1417                # initialize SS bond from the proximal, distal coordinates
1418
1419                if _chaina[proximal].is_disordered() or
…  _chainb[distal].is_disordered():
1420                    mess = f'Disordered chain(s): {struct_name}: {proximal}
…  {chain1_id} - {distal} {chain2_id}, ignoring!'
1421                    warnings.warn(mess, DisulfideConstructionWarning)
1422                    continue
1423                else:
1424                    if verbose:
1425                        print(f' -> SSBond: {i}: {struct_name}: {proximal}
…  {chain1_id} - {distal} {chain2_id}')
1426                    ssbond_name =
…  f'{struct_name}_{proximal}{chain1_id}_{distal}{chain2_id}'
1427                    new_ss = Disulfide(ssbond_name)
1428                    new_ss.initialize_disulfide_from_chain(_chaina, _chainb,
…  proximal, distal)
1429                    SSList.append(new_ss)
1430            i += 1
1431        return SSList
1432
1433    def check_header_from_file(filename: str,
1434                               model_numb = 0,
1435                               verbose = False,
1436                               dbg = False) -> bool:
1437
1438        '''
1439        Loads all Disulfides by PDB ID and initializes the Disulfide objects.
1440        Assumes the file is downloaded in the pdb_dir path.
1441
1442        NB: Requires EGS-Modified BIO.parse_pdb_header.py
1443
1444        Arguments:
1445            struct_name: the name of the PDB entry.
1446
1447            pdb_dir: path to the PDB files, defaults to PDB_DIR
1448
1449            model_numb: model number to use, defaults to 0 for single
1450            structure files.
1451
```

```
1452        verbose: print info while parsing
1453
1454    Returns: a list of Disulfide objects initialized from the file.
1455    Example:
1456      Assuming the PDB_DIR has the pdb5rsa.ent file in place calling:
1457
1458      SS_list = []
1459      SS_list = load_disulfides_from_id('5rsa', verbose=True)
1460
1461      loads the Disulfides from the file and initialize the disulfide
…  objects, returning
1462      them in the result. '''
1463
1464    i = 1
1465    proximal = distal = -1
1466    SSList = []
1467    _chaina = None
1468    _chainb = None
1469
1470    parser = PDBParser(PERMISSIVE=True)
1471
1472    # Biopython uses the Structure -> Model -> Chain hierarchy to organize
1473    # structures. All are iterable.
1474
1475    structure = parser.get_structure('tmp', file=filename)
1476    struct_name = structure.get_id()
1477
1478    model = structure[model_numb]
1479
1480    if verbose:
1481        print(f'-> check_header_from_file() - Parsing file: {filename}:')
1482
1483    ssbond_dict = structure.header['ssbond'] # NB: this requires the
…  modified code
1484
1485    # list of tuples with (proximal distal chaina chainb)
1486    ssbonds = parse_ssbond_header_rec(ssbond_dict)
1487
1488    for pair in ssbonds:
1489        # in the form (proximal, distal, chain)
1490        proximal = pair[0]
1491        distal = pair[1]
1492
1493        if not proximal.isnumeric() or not distal.isnumeric():
1494            if verbose:
1495                print(f' ! Cannot parse SSBond record (non-numeric IDs):
…  {struct_name} Prox:  {proximal} {chain1_id} Dist: {distal} {chain2_id}')
1496            continue # was pass
```

```
1497            else:
1498                proximal = int(proximal)
1499                distal = int(distal)
1500
1501            chain1_id = pair[2]
1502            chain2_id = pair[3]
1503
1504            _chaina = model[chain1_id]
1505            _chainb = model[chain2_id]
1506
1507            if (chain1_id != chain2_id):
1508                if verbose:
1509                    print(f' -> Cross Chain SS for: Prox: {proximal}
… {chain1_id} Dist: {distal} {chain2_id}')
1510                    pass # was break
1511
1512            try:
1513                prox_res = _chaina[proximal]
1514                dist_res = _chainb[distal]
1515            except KeyError:
1516                print(f' ! Cannot parse SSBond record (KeyError):
… {struct_name} Prox:  <{proximal}> {chain1_id} Dist: <{distal}>
… {chain2_id}')
1517                continue
1518
1519            # make a new Disulfide object, name them based on proximal and
… distal
1520            # initialize SS bond from the proximal, distal coordinates
1521            if (_chaina is not None) and (_chainb is not None):
1522                if _chaina[proximal].is_disordered() or
… _chainb[distal].is_disordered():
1523                    continue
1524                else:
1525                    if verbose:
1526                       print(f' -> SSBond: {i}: {struct_name}: {proximal}
… {chain1_id} - {distal} {chain2_id}')
1527            else:
1528                if dbg:
1529                    print(f' -> NULL chain(s): {struct_name}: {proximal}
… {chain1_id} - {distal} {chain2_id}')
1530            i += 1
1531        return True
1532
1533    def check_header_from_id(struct_name: str,
1534                             pdb_dir = '.',
1535                             model_numb = 0,
1536                             verbose = False,
1537                             dbg = False) -> bool:
```

```
1538     '''
1539     Loads all Disulfides by PDB ID and initializes the Disulfide objects.
1540     Assumes the file is downloaded in the pdb_dir path.
1541 
1542     NB: Requires EGS-Modified BIO.parse_pdb_header.py
1543 
1544     Arguments:
1545         struct_name: the name of the PDB entry.
1546 
1547         pdb_dir: path to the PDB files, defaults to PDB_DIR
1548 
1549         model_numb: model number to use, defaults to 0 for single
1550         structure files.
1551 
1552         verbose: print info while parsing
1553 
1554     Returns: True if the proximal and distal residues are CYS and there
… are no cross-chain SS bonds
1555 
1556     Example:
1557       Assuming the PDB_DIR has the pdb5rsa.ent file in place calling:
1558 
1559       SS_list = []
1560       goodfile = check_header_from_id('5rsa', verbose=True)
1561 
1562     '''
1563 
1564     parser = PDBParser(PERMISSIVE=True, QUIET=True)
1565     structure = parser.get_structure(struct_name,
… file=f'{pdb_dir}pdb{struct_name}.ent')
1566     model = structure[0]
1567 
1568     ssbond_dict = structure.header['ssbond'] # NB: this requires the
… modified code
1569 
1570     bondlist = []
1571     i = 0
1572 
1573     # get a list of tuples containing the proximal, distal residue IDs for
… all SSBonds in the chain.
1574     bondlist = parse_ssbond_header_rec(ssbond_dict)
1575 
1576     if len(bondlist) == 0:
1577         if (verbose):
1578             print(f'-> check_header_from_id(): no bonds found in
… bondlist.')
1579         return False
1580 
```

```
1581     for pair in bondlist:
1582         # in the form (proximal, distal, chain)
1583         proximal = pair[0]
1584         distal = pair[1]
1585         chain1 = pair[2]
1586         chain2 = pair[3]
1587 
1588         chaina = model[chain1]
1589         chainb = model[chain2]
1590 
1591         try:
1592             prox_residue = chaina[proximal]
1593             dist_residue = chainb[distal]
1594 
1595             prox_residue.disordered_select("CYS")
1596             dist_residue.disordered_select("CYS")
1597 
1598             if prox_residue.get_resname() != 'CYS' or
…   dist_residue.get_resname() != 'CYS':
1599                 if (verbose):
1600                     print(f'build_disulfide() requires CYS at both
…   residues: {prox_residue.get_resname()} {dist_residue.get_resname()}')
1601                 return False
1602         except KeyError:
1603             if (dbg):
1604                 print(f'Keyerror: {struct_name}: {proximal} {chain1} -
…   {distal} {chain2}')
1605                 return False
1606 
1607         if verbose:
1608             print(f' -> SSBond: {i}: {struct_name}: {proximal} {chain1} -
…   {distal} {chain2}')
1609 
1610         i += 1
1611     return True
1612 
1613 # Using pyVista to render Disulfide Bonds.
1614 
1615 from proteusPy.atoms import BOND_RADIUS, FONTSIZE
1616 WINSIZE = (1200, 1200)
1617 CAMERA_POS = ((0, 0, -10), (0,0,0), (0,1,0))
1618 
1619 def render_disulfide(ss: Disulfide, pvplot: pv.Plotter(), style='cpk',
1620                      bondcolor='brown', bs_scale=.2, spec=.6, specpow=4) ->
…   pv.Plotter:
1621     '''
1622     Update the passed pyVista plotter() object with the mesh data for the
…   input Disulfide Bond
```

```
1623        Arguments:
1624            pvp: pyvista.Plotter() object
1625            style: 'bs', 'st', 'cpk', 'plain': Whether to render as CPK,
    … ball-and-stick or stick.
1626            Bonds are colored by atom color, unless 'plain' is specified.
1627        Returns:
1628            Updated pv.Plotter() object.
1629        '''
1630
1631        cyl_radius = BOND_RADIUS
1632        coords = ss.internal_coords()
1633        cofmass = ss.cofmass()
1634
1635        # translate to cofmass frame
1636        for i in range(12):
1637            coords[i] = coords[i] - cofmass
1638
1639        atoms = ('N', 'C', 'C', 'O', 'C', 'SG', 'N', 'C', 'C', 'O', 'C', 'SG')
1640        pvp = pvplot
1641
1642        # bond connection table with atoms in the specific order shown above:
1643        # returned by ss.get_internal_coords()
1644        bond_conn = numpy.array(
1645            [
1646                [0, 1], # n-ca
1647                [1, 2], # ca-c
1648                [2, 3], # c-o
1649                [1, 4], # ca-cb
1650                [4, 5], # cb-sg
1651                [6, 7], # n-ca
1652                [7, 8], # ca-c
1653                [8, 9], # c-o
1654                [7, 10], # ca-cb
1655                [10, 11], #cb-sg
1656                [5, 11]   #sg -sg
1657            ])
1658
1659        # colors for the bonds. Index into ATOM_COLORS array
1660        bond_split_colors = numpy.array(
1661            [
1662                ('N', 'C'),
1663                ('C', 'C'),
1664                ('C', 'O'),
1665                ('C', 'C'),
1666                ('C', 'SG'),
1667                ('N', 'C'),
1668                ('C', 'C'),
1669                ('C', 'O'),
```

```
1670                ('C', 'C'),
1671                ('C', 'SG'),
1672                ('SG', 'SG')
1673            ]
1674        )
1675
1676        if style=='cpk':
1677            i = 0
1678            for atom in atoms:
1679                rad = ATOM_RADII_CPK[atom]
1680                pvp.add_mesh(pv.Sphere(center=coords[i], radius=rad),
  … color=ATOM_COLORS[atom], smooth_shading=True, specular=spec,
  … specular_power=specpow)
1681                i += 1
1682
1683        elif style == 'bs': # ball and stick
1684            i = 0
1685            for atom in atoms:
1686                rad = ATOM_RADII_CPK[atom] * bs_scale
1687                pvp.add_mesh(pv.Sphere(center=coords[i], radius=rad),
  … color=ATOM_COLORS[atom], smooth_shading=True, specular=spec,
  … specular_power=specpow)
1688                i += 1
1689
1690            # work through connectivity and colors
1691            for i in range(len(bond_conn)):
1692                bond = bond_conn[i]
1693
1694                # get the indices for the origin and destination atoms
1695                orig = bond[0]
1696                dest = bond[1]
1697
1698                col = bond_split_colors[i]
1699                orig_col = ATOM_COLORS[col[0]]
1700                dest_col = ATOM_COLORS[col[1]]
1701
1702                # get the coords
1703                prox_pos = coords[orig]
1704                distal_pos = coords[dest]
1705
1706                # compute a direction vector
1707                direction = distal_pos - prox_pos
1708
1709                # and vector length. divide by 2 since split bond
1710                height = math.dist(prox_pos, distal_pos) / 2.0
1711
1712                origin1 = prox_pos + 0.25 * direction # the cylinder origin is
  … actually in the middle so we translate
```

```
1713            origin2 = prox_pos + 0.75 * direction # the cylinder origin is
…  actually in the middle so we translate
1714
1715            cyl1 = pv.Cylinder(origin1, direction, radius=cyl_radius,
…  height=height)
1716            cyl2 = pv.Cylinder(origin2, direction, radius=cyl_radius,
…  height=height)
1717
1718            pvp.add_mesh(cyl1, color=orig_col)
1719            pvp.add_mesh(cyl2, color=dest_col)
1720
1721        elif style == 'sb': # splitbonds
1722            i = 0
1723
1724            for i in range(len(bond_conn)):
1725                bond = bond_conn[i]
1726                orig = bond[0]
1727                dest = bond[1]
1728
1729                col = bond_split_colors[i]
1730                orig_col = ATOM_COLORS[col[0]]
1731                dest_col = ATOM_COLORS[col[1]]
1732
1733                prox_pos = coords[orig]
1734                distal_pos = coords[dest]
1735                direction = distal_pos - prox_pos
1736                height = math.dist(prox_pos, distal_pos) / 2.0
1737
1738                origin = prox_pos + 0.25 * direction # the cylinder origin is
…  actually in the middle so we translate
1739                origin2 = prox_pos + 0.75 * direction # the cylinder origin is
…  actually in the middle so we translate
1740
1741                cap1 = pv.Sphere(center=prox_pos, radius=cyl_radius)
1742                cap2 = pv.Sphere(center=distal_pos, radius=cyl_radius)
1743
1744                cyl1 = pv.Cylinder(origin, direction, radius=cyl_radius,
…  height=height)
1745                cyl2 = pv.Cylinder(origin2, direction, radius=cyl_radius,
…  height=height)
1746
1747                pvp.add_mesh(cyl1, color=orig_col)
1748                pvp.add_mesh(cyl2, color=dest_col)
1749                pvp.add_mesh(cap1, color=orig_col)
1750                pvp.add_mesh(cap2, color=dest_col)
1751
1752        else: # plain
1753            for i in range(len(bond_conn)):
```

```
1754                bond = bond_conn[i]
1755                orig = bond[0]
1756                dest = bond[1]
1757                prox_pos = coords[orig]
1758                distal_pos = coords[dest]
1759                direction = distal_pos - prox_pos
1760                height = math.dist(prox_pos, distal_pos)
1761                origin = prox_pos + 0.5 * direction # the cylinder origin is
…   actually in the middle so we translate
1762
1763                cap1 = pv.Sphere(center=prox_pos, radius=cyl_radius)
1764                cap2 = pv.Sphere(center=distal_pos, radius=cyl_radius)
1765                cyl = pv.Cylinder(origin, direction, radius=cyl_radius,
…   height=height)
1766
1767                pvp.add_mesh(cap1, color=bondcolor)
1768                pvp.add_mesh(cap2, color=bondcolor)
1769                pvp.add_mesh(cyl, color=bondcolor)
1770        return pvp
1771
1772    # End of file
1773
```