```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
#
import sys
import os
import glob
import warnings
import copy

import numpy
import pickle
import time
import datetime

from collections import UserList

import pandas as pd
from tqdm import tqdm
from numpy import cos

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

from proteusPy import *

from proteusPy.DisulfideExceptions import *

from proteusPy.turtle3D import *
from proteusPy.proteusGlobals import PDB_DIR, MODEL_DIR, ORIENT_SIDECHAIN
from proteusPy.DisulfideGlobals import *

class DisulfideList(UserList):
    def __init__(self, iterable, id):
        self.pdb_id = id
        super().__init__(self.validate_ss(item) for item in iterable)

    def __getitem__(self, item):
        if isinstance(item, slice):
            indices = range(*item.indices(len(self.data)))
            name = self.data[0].pdb_id
            sublist = [self.data[i] for i in indices]
            return DisulfideList(sublist, name)
        return UserList.__getitem__(self, item)

    def __setitem__(self, index, item):
```



```python
        self.data[index] = self.validate_ss(item)

    def insert(self, index, item):
        self.data.insert(index, self.validate_ss(item))

    def append(self, item):
        self.data.append(self.validate_ss(item))

    def extend(self, other):
        if isinstance(other, type(self)):
            self.data.extend(other)
        else:
            self.data.extend(self._validate_ss(item) for item in other)

    def validate_ss(self, value):
        if isinstance(value, (proteusPy.disulfide.Disulfide)):
            return value
        raise TypeError(f"Disulfide object expected, got
{type(value).__name__}")

    def set_id(self, value):
        self.pdb_id = value

    def get_id(self):
        return self.pdb_id

class DisulfideLoader():
    '''
    This class loads .pkl files created from the DisulfideExtractor()
routine
    and initializes itself with their contents. The Disulfide objects are
contained
    in a DisulfideList object and Dict. This makes it possible to access
the disulfides by
    array index or PDB structure ID.\n

    Example:
        from Disulfide import DisulfideList, Disulfide, DisulfideLoader

        SS1 = DisulfideList([],'All_SS')
        SS2 = DisulfideList([], '4yys')

        PDB_SS = DisulfideLoader()
        SS1 = PDB_SS[0]         <-- returns a Disulfide object at index 0
        SS2 = PDB_SS['4yys']    <-- returns a DisulfideList containing all
disulfides for 4yys
        SS3 = PDB_SS[:10]       <-- returns a DisulfideList containing the
slice
```

```
    '''

    def __init__(self, verbose=True, modeldir=MODEL_DIR,
picklefile=SS_PICKLE_FILE,
                 pickle_dict_file=SS_DICT_PICKLE_FILE,
                 torsion_file=SS_TORSIONS_FILE):
        self.ModelDir = modeldir
        self.PickleFile = f'{modeldir}{picklefile}'
        self.PickleDictFile = f'{modeldir}{pickle_dict_file}'
        self.TorsionFile = f'{modeldir}{torsion_file}'
        self.SSList = DisulfideList([], 'ALL_PDB_SS')
        #self.SSList = []
        self.SSDict = {}
        self.TorsionDF = pd.DataFrame()
        self.TotalDisulfides = 0
        self.IDList = []

        # 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']
        SS_df = pd.DataFrame(columns=df_cols, index=['source'])
        _SSList = DisulfideList([], 'ALL_PDB_SS')

        idlist = []
        if verbose:
            print(f'Reading disulfides from: {self.PickleFile}')
        with open(self.PickleFile, 'rb') as f:
            self.SSList = pickle.load(f)

        self.TotalDisulfides = len(self.SSList)

        if verbose:
            print(f'Disulfides Read: {self.TotalDisulfides}')
            print(f'Reading disulfide dict from: {self.PickleDictFile}')

        with open(self.PickleDictFile, 'rb') as f:
            self.SSDict = pickle.load(f)
            for key in self.SSDict:
                idlist.append(key)
            self.IDList = idlist.copy()
            totalSS_dict = len(self.IDList)

        if verbose:
            print(f'Reading Torsion DF {self.TorsionFile}.')

        self.TorsionDF = pd.read_csv(self.TorsionFile)

```



```
        if verbose:
            print(f'Read torsions DF.')
            print(f'PDB IDs parsed: {totalSS_dict}')
            print(f'Total Space Used: {sys.getsizeof(self.SSList) +
sys.getsizeof(self.SSDict) + sys.getsizeof(self.TorsionDF)} bytes.')
        return

    # overload __getitem__ to handle slicing and indexing
    def __getitem__(self, item):
        if isinstance(item, slice):
            indices = range(*item.indices(len(self.SSList)))
            # return [self.SSList[i] for i in indices]
            name = self.SSList[0].pdb_id
            sublist = [self.SSList[i] for i in indices]
            return DisulfideList(sublist, name)

        if isinstance(item, int):
            if (item < 0 or item >= self.TotalDisulfides):
                mess = f'DisulfideDataLoader error. Index {item} out of
range 0-{self.TotalDisulfides - 1}'
                raise DisulfideException(mess)
            else:
                return self.SSList[item]

        try:
            res = self.SSDict[item]
        except KeyError:
            mess = f'! Cannot find key {item} in SSBond dict!'
            raise DisulfideException(mess)
        return res

    def __setitem__(self, index, item):
        self.SSList[index] = self.validate_ss(item)

    def getlist(self):
        return self.SSList.copy()

    def getdict(self) -> dict:
        return copy.deepcopy(self.SSDict)

    def getTorsions(self):
        return copy.deepcopy(self.TorsionDF)

    def validate_ss(self, value):
        if isinstance(value, (Disulfide)):
            return value
        raise TypeError(
            f"Disulfide object expected, got {type(value).__name__}"
```

```
        )


# float init for class
_FLOAT_INIT = -999.9

# tqdm progress bar width
_PBAR_COLS = 100

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

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))

def parse_ssbond_header_rec(ssbond_dict: dict) -> list:
    '''
    Parse the SSBOND dict returned by parse_pdb_header.
    NB: Requires EGS-Modified BIO.parse_pdb_header.py

    Arguments:
        ssbond_dict: the input SSBOND dict
    Returns: a list of tuples representing the proximal, distal residue
             ids for the disulfide.

    '''
    disulfide_list = []
    for ssb in ssbond_dict.items():
        disulfide_list.append(ssb[1])

    return disulfide_list

#
# function reads a comma separated list of PDB IDs and download the
corresponding
# .ent files to the PDB_DIR global.
```



```
# Used to download the list of proteins containing at least one SS bond
# with the ID list generated from: http://www.rcsb.org/
#

def DownloadDisulfides(pdb_home=PDB_DIR, model_home=MODEL_DIR,
                       verbose=False, reset=False) -> None:
    '''
    Function reads a comma separated list of PDB IDs and downloads them
    to the pdb_home path.

    Used to download the list of proteins containing at least one SS bond
    with the ID list generated from: http://www.rcsb.org/
    '''

    start = time.time()
    donelines = []
    SS_done = []
    ssfile = None

    cwd = os.getcwd()
    os.chdir(pdb_home)

    pdblist = PDBList(pdb=pdb_home, verbose=verbose)
    ssfilename = f'{model_home}{SS_ID_FILE}'
    print(ssfilename)

    # list of IDs containing >1 SSBond record
    try:
        ssfile = open(ssfilename)
        Line = ssfile.readlines()
    except Exception:
        raise DisulfideIOException(f'Cannot open file: {ssfile}')

    for line in Line:
        entries = line.split(',')

    print(f'Found: {len(entries)} entries')
    completed = {'xxx'} # set to keep track of downloaded

    # file to track already downloaded entries.
    if reset==True:
        completed_file = open(f'{model_home}ss_completed.txt', 'w')
        donelines = []
        SS_DONE = []
    else:
        completed_file = open(f'{model_home}ss_completed.txt', 'w+')
        donelines = completed_file.readlines()

```

```
277     if len(donelines) > 0:
278         for dl in donelines[0]:
279             # create a list of pdb id already downloaded
280             SS_done = dl.split(',')
281 
282     count = len(SS_done) - 1
283     completed.update(SS_done) # update the completed set with what's
    … downloaded
284 
285     # Loop over all entries,
286     pbar = tqdm(entries, ncols=_PBAR_COLS)
287     for entry in pbar:
288         pbar.set_postfix({'Entry': entry})
289         if entry not in completed:
290             if pdblist.retrieve_pdb_file(entry, file_format='pdb',
    … pdir=pdb_home):
291                 completed.update(entry)
292                 completed_file.write(f'{entry},')
293                 count += 1
294 
295     completed_file.close()
296 
297     end = time.time()
298     elapsed = end - start
299 
300     print(f'Overall files processed: {count}')
301     print(f'Complete. Elapsed time: {datetime.timedelta(seconds=elapsed)}
    … (h:m:s)')
302     os.chdir(cwd)
303     return
304 
305 def build_torsion_df(SSList: DisulfideList):
306     # create a dataframe with the following columns for the disulfide
    … conformations extracted from the structure
307     df_cols = ['source', 'ss_id', 'proximal', 'distal', 'chi1', 'chi2',
    … 'chi3', 'chi4', 'chi5', 'energy']
308     SS_df = pd.DataFrame(columns=df_cols)
309 
310     pbar = tqdm(SSList, ncols=_PBAR_COLS, miniters=400000)
311     for ss in pbar:
312         #pbar.set_postfix({'ID': ss.name}) # update the progress bar
313         new_row = [ss.pdb_id, ss.name, ss.proximal, ss.distal, ss.chi1,
    … ss.chi2, ss.chi3, ss.chi4, ss.chi5, ss.energy]
314         # add the row to the end of the dataframe
315         SS_df.loc[len(SS_df.index)] = new_row.copy() # deep copy
316 
317     return SS_df.copy()
318 
```



```
319 def DisulfideExtractor(numb=-1, verbose=False, quiet=False,
    … pdbdir=PDB_DIR,
320                         modeldir=MODEL_DIR, picklefile=SS_PICKLE_FILE,
321                         torsionfile=SS_TORSIONS_FILE,
    … problemfile=PROBLEM_ID_FILE,
322                         dictfile=SS_DICT_PICKLE_FILE) -> None:
323     '''
324     This function creates .pkl files needed for the DisulfideLoader class.
    … The Disulfide
325     objects are contained in a DisulfideList object and Dict within these
    … files.
326     In addition, .csv files containing all of the torsions for the
    … disulfides and
327     problem IDs are written.
328 
329     Arguments:
330         numb:           number of entries to process, defaults to all
331         verbose:        more messages
332         quiet:          turns of DisulfideConstruction warnings
333         pdbdir:         path to PDB files
334         modeldir:       path to resulting .pkl files
335         picklefile:     name of the disulfide .pkl file
336         torsionfile:    name of the disulfide torsion file .csv created
337         problemfile:    name of the .csv file containing problem ids
338         dictfile:       name of the .pkl file
339 
340     Example:
341         from proteusPy.Disulfide import DisulfideExtractor,
    … DisulfideLoader, DisulfideList
342 
343         DisulfideExtractor(numb=500, pdbdir=PDB_DIR, verbose=False,
    … quiet=True)
344 
345         SS1 = DisulfideList([],'All_SS')
346         SS2 = DisulfideList([], '4yys')
347 
348         PDB_SS = DisulfideLoader()
349         SS1 = PDB_SS[0]         <-- returns a Disulfide object at index 0
350         SS2 = PDB_SS['4yys']    <-- returns a DisulfideList containing all
    … disulfides for 4yys
351         SS3 = PDB_SS[:10]       <-- returns a DisulfideList containing the
    … slice
352     '''
353 
354     entrylist = []
355     problem_ids = []
356     bad = 0
357 
```

```
    # we use the specialized list class DisulfideList to contain our disulfides
    # we'll use a dict to store DisulfideList objects, indexed by the structure ID
    All_ss_dict = {}
    All_ss_list = []

    start = time.time()
    cwd = os.getcwd()

    # Build a list of PDB files in PDB_DIR that are readable. These files were downloaded
    # via the RCSB web query interface for structures containing >= 1 SS Bond.

    os.chdir(pdbdir)

    ss_filelist = glob.glob(f'*.ent')
    tot = len(ss_filelist)

    if verbose:
        print(f'PDB Directory {pdbdir} contains: {tot} files')

    # the filenames are in the form pdb{entry}.ent, I loop through them and extract
    # the PDB ID, with Disulfide.name_to_id(), then add to entrylist.

    for entry in ss_filelist:
        entrylist.append(name_to_id(entry))

    # 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']
    #SS_df = pd.DataFrame(columns=df_cols)
    df_cols = ['source', 'ss_id', 'proximal', 'distal', 'chi1', 'chi2', 'chi3', 'chi4', 'chi5', 'energy']
    SS_df = pd.DataFrame(columns=df_cols)

    # define a tqdm progressbar using the fully loaded entrylist list. If numb is passed then
    # only do the last numb entries.
    if numb > 0:
        pbar = tqdm(entrylist[:numb], ncols=_PBAR_COLS)
    else:
        pbar = tqdm(entrylist, ncols=_PBAR_COLS)

    # loop over ss_filelist, create disulfides and initialize them
```



```
    for entry in pbar:
        pbar.set_postfix({'ID': entry, 'Bad': bad}) # update the progress bar

        # returns an empty list if none are found.
        sslist = DisulfideList([], entry)
        sslist = load_disulfides_from_id(entry, model_numb=0, verbose=verbose, quiet=quiet, pdb_dir=pdbdir)
        if len(sslist) > 0:
            for ss in sslist:
                All_ss_list.append(ss)
                new_row = [ss.pdb_id, ss.name, ss.proximal, ss.distal, ss.chi1, ss.chi2, ss.chi3, ss.chi4, ss.chi5, ss.energy]
                # add the row to the end of the dataframe
                SS_df.loc[len(SS_df.index)] = new_row.copy() # deep copy

            All_ss_dict[entry] = sslist
        else:
            # at this point I really shouldn't have any bad non-parsible file
            bad += 1
            problem_ids.append(entry)
            os.remove(f'pdb{entry}.ent')

    if bad > 0:
        if verbose:
            print(f'Found and removed: {len(problem_ids)} problem structures.')
        prob_cols = ['id']
        problem_df = pd.DataFrame(columns=prob_cols)
        problem_df['id'] = problem_ids

        print(f'Saving problem IDs to file: {modeldir}{problemfile}')
        problem_df.to_csv(f'{modeldir}{problemfile}')
    else:
        if verbose:
            print('No problems found.')

    # dump the all_ss array of disulfides to a .pkl file. ~520 MB.
    fname = f'{modeldir}{picklefile}'
    if True:
        print(f'Saving {len(All_ss_list)} Disulfides to file: {fname}')

    with open(fname, 'wb+') as f:
        pickle.dump(All_ss_list, f)

    # dump the all_ss array of disulfides to a .pkl file. ~520 MB.
    dict_len = len(All_ss_dict)
```

```
440     fname = f'{modeldir}{dictfile}'
441
442     if True:
443         print(f'Saving {len(All_ss_dict)} Disulfide-containing PDB IDs to
    file: {fname}')
444
445     with open(fname, 'wb+') as f:
446         pickle.dump(All_ss_dict, f)
447
448     fname = f'{modeldir}{torsionfile}'
449     if True:
450         print(f'Saving torsions to file: {fname}')
451
452     SS_df.to_csv(fname)
453
454     end = time.time()
455     elapsed = end - start
456
457     print(f'Disulfide Extraction complete! Elapsed time:
    {datetime.timedelta(seconds=elapsed)} (h:m:s)')
458
459     # return to original directory
460     os.chdir(cwd)
461     return
462
463
464 # NB - this only works with the EGS modified version of
    BIO.parse_pdb_header.py
465 def load_disulfides_from_id(struct_name: str,
466                             pdb_dir = '.',
467                             model_numb = 0,
468                             verbose = False,
469                             quiet=False,
470                             dbg = False) -> list:
471     '''
472     Loads all Disulfides by PDB ID and initializes the Disulfide objects.
473     Assumes the file is downloaded in the pdb_dir path.
474
475     NB: Requires EGS-Modified BIO.parse_pdb_header.py
476
477     Arguments:
478         struct_name: the name of the PDB entry.
479
480         pdb_dir: path to the PDB files, defaults to PDB_DIR
481
482         model_numb: model number to use, defaults to 0 for single
483         structure files.
484
```



```
485         verbose: print info while parsing
486
487     Returns: a list of Disulfide objects initialized from the file.
488     Example:
489       Assuming the PDB_DIR has the pdb5rsa.ent file in place calling:
490
491       SS_list = []
492       SS_list = load_disulfides_from_id('5rsa', verbose=True)
493
494       loads the Disulfides from the file and initialize the disulfide
    objects, returning
495       them in the result. '''
496
497     i = 1
498     proximal = distal = -1
499     SSList = DisulfideList([], struct_name)
500     _chaina = None
501     _chainb = None
502
503     parser = PDBParser(PERMISSIVE=True)
504
505     # Biopython uses the Structure -> Model -> Chain hierarchy to organize
506     # structures. All are iterable.
507
508     structure = parser.get_structure(struct_name,
    file=f'{pdb_dir}pdb{struct_name}.ent')
509     model = structure[model_numb]
510
511     if verbose:
512         print(f'-> load_disulfide_from_id() - Parsing structure:
    {struct_name}:')
513
514     ssbond_dict = structure.header['ssbond'] # NB: this requires the
    modified code
515
516     # list of tuples with (proximal distal chaina chainb)
517     ssbonds = parse_ssbond_header_rec(ssbond_dict)
518
519     with warnings.catch_warnings():
520         if quiet:
521             #warnings.filterwarnings("ignore",
    category=DisulfideConstructionWarning)
522             warnings.filterwarnings("ignore")
523         for pair in ssbonds:
524             # in the form (proximal, distal, chain)
525             proximal = pair[0]
526             distal = pair[1]
527             chain1_id = pair[2]
```

```
528             chain2_id = pair[3]
529
530             if not proximal.isnumeric() or not distal.isnumeric():
531                 mess = f'Cannot parse SSBond record (non-numeric IDs):
…   {struct_name} Prox: {proximal} {chain1_id} Dist: {distal} {chain2_id},
…   ignoring.'
532                 warnings.warn(mess, DisulfideConstructionWarning)
533                 continue
534             else:
535                 proximal = int(proximal)
536                 distal = int(distal)
537
538             _chaina = model[chain1_id]
539             _chainb = model[chain2_id]
540
541             if (_chaina is None) or (_chainb is None):
542                 mess = f' -> NULL chain(s): {struct_name}: {proximal}
…   {chain1_id} - {distal} {chain2_id}, ignoring!'
543                 warnings.warn(mess, DisulfideConstructionWarning)
544                 continue
545
546             if (chain1_id != chain2_id):
547                 if verbose:
548                     print(f' -> Cross Chain SS for: Prox: {proximal}
…   {chain1_id} Dist: {distal} {chain2_id}')
549                     pass # was break
550
551             try:
552                 prox_res = _chaina[proximal]
553                 dist_res = _chainb[distal]
554
555             except KeyError:
556                 mess = f'Cannot parse SSBond record (KeyError):
…   {struct_name} Prox:  {proximal} {chain1_id} Dist: {distal} {chain2_id},
…   ignoring!'
557                 warnings.warn(mess, DisulfideConstructionWarning)
558                 continue
559
560             # make a new Disulfide object, name them based on proximal and
…   distal
561             # initialize SS bond from the proximal, distal coordinates
562
563             if _chaina[proximal].is_disordered() or
…   _chainb[distal].is_disordered():
564                 mess = f'Disordered chain(s): {struct_name}: {proximal}
…   {chain1_id} - {distal} {chain2_id}, ignoring!'
565                 warnings.warn(mess, DisulfideConstructionWarning)
566                 continue
```



```
567             else:
568                 if verbose:
569                     print(f' -> SSBond: {i}: {struct_name}: {proximal}
…   {chain1_id} - {distal} {chain2_id}')
570                 ssbond_name =
…   f'{struct_name}_{proximal}{chain1_id}_{distal}{chain2_id}'
571                 new_ss = Disulfide(ssbond_name)
572                 new_ss.initialize_disulfide_from_chain(_chaina, _chainb,
…   proximal, distal)
573                 SSList.append(new_ss)
574         i += 1
575     return SSList
576
577 def check_header_from_file(filename: str,
578                            model_numb = 0,
579                            verbose = False,
580                            dbg = False) -> bool:
581
582     '''
583     Loads all Disulfides by PDB ID and initializes the Disulfide objects.
584     Assumes the file is downloaded in the pdb_dir path.
585
586     NB: Requires EGS-Modified BIO.parse_pdb_header.py
587
588     Arguments:
589         struct_name: the name of the PDB entry.
590
591         pdb_dir: path to the PDB files, defaults to PDB_DIR
592
593         model_numb: model number to use, defaults to 0 for single
594         structure files.
595
596         verbose: print info while parsing
597
598     Returns: a list of Disulfide objects initialized from the file.
599     Example:
600       Assuming the PDB_DIR has the pdb5rsa.ent file in place calling:
601
602       SS_list = []
603       SS_list = load_disulfides_from_id('5rsa', verbose=True)
604
605       loads the Disulfides from the file and initialize the disulfide
…   objects, returning
606       them in the result. '''
607
608     i = 1
609     proximal = distal = -1
610     SSList = []
```

```
    _chaina = None
    _chainb = None

    parser = PDBParser(PERMISSIVE=True)

    # Biopython uses the Structure -> Model -> Chain hierarchy to organize
    # structures. All are iterable.

    structure = parser.get_structure('tmp', file=filename)
    struct_name = structure.get_id()

#    structure = parser.get_structure(struct_name,
file=f'{pdb_dir}pdb{struct_name}.ent')
    model = structure[model_numb]

    if verbose:
        print(f'-> check_header_from_file() - Parsing file: {filename}:')

    ssbond_dict = structure.header['ssbond'] # NB: this requires the
modified code

    # list of tuples with (proximal distal chaina chainb)
    ssbonds = parse_ssbond_header_rec(ssbond_dict)

    for pair in ssbonds:
        # in the form (proximal, distal, chain)

        proximal = pair[0]
        distal = pair[1]

        if not proximal.isnumeric() or not distal.isnumeric():
            if verbose:
                print(f' ! Cannot parse SSBond record (non-numeric IDs):
{struct_name} Prox:  {proximal} {chain1_id} Dist: {distal} {chain2_id}')
            continue # was pass
        else:
            proximal = int(proximal)
            distal = int(distal)

        chain1_id = pair[2]
        chain2_id = pair[3]

        _chaina = model[chain1_id]
        _chainb = model[chain2_id]

        if (chain1_id != chain2_id):
            if verbose:
                print(f' -> Cross Chain SS for: Prox: {proximal}
```



```
{chain1_id} Dist: {distal} {chain2_id}')
                pass # was break

        try:
            prox_res = _chaina[proximal]
            dist_res = _chainb[distal]
        except KeyError:
            print(f' ! Cannot parse SSBond record (KeyError):
{struct_name} Prox:  <{proximal}> {chain1_id} Dist: <{distal}>
{chain2_id}')
            continue

        # make a new Disulfide object, name them based on proximal and
distal
        # initialize SS bond from the proximal, distal coordinates
        if (_chaina is not None) and (_chainb is not None):
            if _chaina[proximal].is_disordered() or
_chainb[distal].is_disordered():
                continue
            else:
                if verbose:
                    print(f' -> SSBond: {i}: {struct_name}: {proximal}
{chain1_id} - {distal} {chain2_id}')
        else:
            if dbg:
                print(f' -> NULL chain(s): {struct_name}: {proximal}
{chain1_id} - {distal} {chain2_id}')
        i += 1
    return True

def check_header_from_id(struct_name: str,
                             pdb_dir = '.',
                             model_numb = 0,
                             verbose = False,
                             dbg = False) -> bool:
    '''
    Loads all Disulfides by PDB ID and initializes the Disulfide objects.
    Assumes the file is downloaded in the pdb_dir path.

    NB: Requires EGS-Modified BIO.parse_pdb_header.py

    Arguments:
        struct_name: the name of the PDB entry.

        pdb_dir: path to the PDB files, defaults to PDB_DIR

        model_numb: model number to use, defaults to 0 for single
        structure files.
```

```
697
698         verbose: print info while parsing
699
700     Returns: True if the proximal and distal residues are CYS and there
…   are no cross-chain SS bonds
701
702     Example:
703       Assuming the PDB_DIR has the pdb5rsa.ent file in place calling:
704
705       SS_list = []
706       goodfile = check_header_from_id('5rsa', verbose=True)
707
708     '''
709
710     parser = PDBParser(PERMISSIVE=True, QUIET=True)
711     structure = parser.get_structure(struct_name,
…   file=f'{pdb_dir}pdb{struct_name}.ent')
712     model = structure[0]
713
714     ssbond_dict = structure.header['ssbond'] # NB: this requires the
…   modified code
715
716     bondlist = []
717     i = 0
718
719     # get a list of tuples containing the proximal, distal residue IDs for
…   all SSBonds in the chain.
720     bondlist = parse_ssbond_header_rec(ssbond_dict)
721
722     if len(bondlist) == 0:
723         if (verbose):
724             print(f'-> check_header_from_id(): no bonds found in
…   bondlist.')
725         return False
726
727     for pair in bondlist:
728         # in the form (proximal, distal, chain)
729         proximal = pair[0]
730         distal = pair[1]
731         chain1 = pair[2]
732         chain2 = pair[3]
733
734         chaina = model[chain1]
735         chainb = model[chain2]
736
737         try:
738             prox_residue = chaina[proximal]
739             dist_residue = chainb[distal]
```



```
740
741             prox_residue.disordered_select("CYS")
742             dist_residue.disordered_select("CYS")
743
744             if prox_residue.get_resname() != 'CYS' or
…   dist_residue.get_resname() != 'CYS':
745                 if (verbose):
746                     print(f'build_disulfide() requires CYS at both
…   residues: {prox_residue.get_resname()} {dist_residue.get_resname()}')
747                 return False
748         except KeyError:
749             if (dbg):
750                 print(f'Keyerror: {struct_name}: {proximal} {chain1} -
…   {distal} {chain2}')
751                 return False
752
753         if verbose:
754             print(f' -> SSBond: {i}: {struct_name}: {proximal} {chain1} -
…   {distal} {chain2}')
755
756         i += 1
757     return True
758
759 # Class defination for a structure-based Disulfide Bond.
760
761 class Disulfide:
762     """
763     The Disulfide Bond is characterized by the atomic coordinates N, Cα,
…   Cβ, C', Sγ
764     for both residues, the dihedral angles X1 - X5 for the disulfide bond
…   conformation,
765     a name, proximal resiude number and distal residue number, and
…   conformational energy.
766     All atomic coordinates are represented by the BIO.PDB.Vector class.
767     """
768     def __init__(self, name="SSBOND"):
769         """
770         Initialize the class. All positions are set to the origin. The
…   optional string name may be passed.
771         """
772         self.name = name
773         self.proximal = -1
774         self.distal = -1
775         self.energy = _FLOAT_INIT
776         self.proximal_chain = str('')
777         self.distal_chain = str('')
778         self.pdb_id = str('')
779         self.proximal_residue_fullid = str('')
```

```
780         self.distal_residue_fullid = str('')
781         self.PERMISSIVE = bool(True)
782         self.QUIET = bool(True)
783 
784         # global coordinates for the Disulfide, typically as returned from
    … the PDB file
785         self.n_prox = Vector(0,0,0)
786         self.ca_prox = Vector(0,0,0)
787         self.c_prox = Vector(0,0,0)
788         self.o_prox = Vector(0,0,0)
789         self.cb_prox = Vector(0,0,0)
790         self.sg_prox = Vector(0,0,0)
791         self.sg_dist = Vector(0,0,0)
792         self.cb_dist = Vector(0,0,0)
793         self.ca_dist = Vector(0,0,0)
794         self.n_dist = Vector(0,0,0)
795         self.c_dist = Vector(0,0,0)
796         self.o_dist = Vector(0,0,0)
797 
798         # local coordinates for the Disulfide, computed using the Turtle3D
    … in
799         # Orientation #1 these are generally private.
800 
801         self._n_prox = Vector(0,0,0)
802         self._ca_prox = Vector(0,0,0)
803         self._c_prox = Vector(0,0,0)
804         self._o_prox = Vector(0,0,0)
805         self._cb_prox = Vector(0,0,0)
806         self._sg_prox = Vector(0,0,0)
807         self._sg_dist = Vector(0,0,0)
808         self._cb_dist = Vector(0,0,0)
809         self._ca_dist = Vector(0,0,0)
810         self._n_dist = Vector(0,0,0)
811         self._c_dist = Vector(0,0,0)
812         self._o_dist = Vector(0,0,0)
813 
814         # Dihedral angles for the disulfide bond itself, set to
    … _FLOAT_INIT
815         self.chi1 = _FLOAT_INIT
816         self.chi2 = _FLOAT_INIT
817         self.chi3 = _FLOAT_INIT
818         self.chi4 = _FLOAT_INIT
819         self.chi5 = _FLOAT_INIT
820 
821         # I initialize an array for the torsions which will be used for
    … comparisons
822         self.dihedrals = numpy.array((_FLOAT_INIT, _FLOAT_INIT,
    … _FLOAT_INIT, _FLOAT_INIT, _FLOAT_INIT), "d")
```



```
823 
824     def reset(self):
825         self.__init__(self)
826 
827     # comparison operators, used for sorting. keyed to SS bond energy
828     def __lt__(self, other):
829         if isinstance(other, Disulfide):
830             return self.energy < other.energy
831 
832     def __le__(self, other):
833         if isinstance(other, Disulfide):
834             return self.energy <= other.energy
835 
836     def __gt__(self, other):
837         if isinstance(other, Disulfide):
838             return self.energy > other.energy
839 
840     def __ge__(self, other):
841         if isinstance(other, Disulfide):
842             return self.energy >= other.energy
843 
844     def __eq__(self, other):
845         if isinstance(other, Disulfide):
846             return self.energy == other.energy
847 
848     def __ne__(self, other):
849         if isinstance(other, Disulfide):
850             return self.energy != other.energy
851 
852     # repr functions. The class is large, so I split it up into sections
853     def repr_ss_info(self):
854         """
855         Representation for the Disulfide class
856         """
857         s1 = f'<Disulfide {self.name} SourceID: {self.pdb_id} Proximal:
    … {self.proximal} {self.proximal_chain} Distal: {self.distal}
    … {self.distal_chain}'
858         return s1
859 
860     def repr_ss_coords(self):
861         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'
862         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'
863         stot = f'{s2} {s3}'
864         return stot
```

```
865
866     def repr_ss_conformation(self):
867         s4 = f'Conformation: (X1-X5):  {self.chi1:.3f}°, {self.chi2:.3f}°,
{self.chi3:.3f}°, {self.chi4:.3f}° {self.chi5:.3f}° '
868         s5 = f'Energy: {self.energy:.3f} kcal/mol'
869         stot = f'{s4} {s5}'
870         return stot
871
872     def repr_ss_local_coords(self):
873         """
874         Representation for the Disulfide class, internal coordinates.
875         """
876         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'
877         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'
878         stot = f'{s2i} {s3i}'
879         return stot
880
881     def repr_ss_chain_ids(self):
882         return(f'Proximal Chain fullID: <{self.proximal_residue_fullid}>
Distal Chain fullID: <{self.distal_residue_fullid}>')
883
884     def __repr__(self):
885         """
886         Representation for the Disulfide class
887         """
888
889         s1 = self.repr_ss_info()
890         res = f'{s1}>'
891         return res
892
893     def pprint(self):
894         """
895         pretty print general info for the Disulfide
896         """
897
898         s1 = self.repr_ss_info()
899         s4 = self.repr_ss_conformation()
900         res = f'{s1} {s4}>'
901         return res
902
903     def pprint_all(self):
904         """
905         pretty print all info for a Disulfide
906         """
```



```
907
908         s1 = self.repr_ss_info() + '\n'
909         s2 = self.repr_ss_coords()
910         s3 = self.repr_ss_local_coords()
911         s4 = self.repr_ss_conformation()
912         s5 = self.repr_chain_ids()
913         res = f'{s1} {s5} {s2} {s3} {s4} >'
914         return res
915
916     def _handle_SS_exception(self, message):
917         """Handle exception (PRIVATE).
918
919         This method catches an exception that occurs in the Disulfide
920         object (if PERMISSIVE), or raises it again, this time adding the
921         PDB line number to the error message.
922         """
923         # message = "%s at line %i." % (message)
924         message = f'{message}'
925
926         if self.PERMISSIVE:
927             # just print a warning - some residues/atoms may be missing
928             warnings.warn(
929                 "DisulfideConstructionException: %s\n"
930                 "Exception ignored.\n"
931                 "Some atoms may be missing in the data structure."
932                 % message,
933                 DisulfideConstructionWarning,
934             )
935         else:
936             # exceptions are fatal - raise again with new message
(including line nr)
937             raise DisulfideConstructionException(message) from None
938
939     def print_compact(self):
940         return(f'{self.repr_ss_info()} {self.repr_ss_conformation()}')
941
942     def repr_conformation(self):
943         return(f'{self.repr_ss_conformation()}')
944
945     def repr_coords(self):
946         return(f'{self.repr_ss_coords()}')
947
948     def repr_internal_coords(self):
949         return(f'{self.repr_ss_local_coords()}')
950
951     def repr_chain_ids(self):
952         return(f'{self.repr_ss_chain_ids()}')
953
```

```
    def set_permissive(self, perm: bool) -> None:
        self.PERMISSIVE = perm

    def get_permissive(self) -> bool:
        return self.PERMISIVE

    def set_quiet(self, perm: bool) -> None:
        self.QUIET = perm

    def get_quiet(self) -> bool:
        return self.QUIET

    def get_full_id(self):
        return((self.proximal_residue_fullid, self.distal_residue_fullid))

    def initialize_disulfide_from_chain(self, chain1, chain2, proximal,
distal):
        '''
        Initialize a new Disulfide object with atomic coordinates from the
proximal and
        distal coordinates, typically taken from a PDB file.

        Arguments:
            chain1: list of Residues in the model, eg: chain = model['A']
            chain2: list of Residues in the model, eg: chain = model['A']
            proximal: proximal residue sequence ID
            distal: distal residue sequence ID

        Returns: none. The internal state is modified.
        '''

        id = chain1.get_full_id()[0]

        self.pdb_id = id

        # create a new Disulfide object
        chi1 = chi2 = chi3 = chi4 = chi5 = _FLOAT_INIT

        prox = int(proximal)
        dist = int(distal)

        prox_residue = chain1[prox]
        dist_residue = chain2[dist]

        if (prox_residue.get_resname() != 'CYS' or
dist_residue.get_resname() != 'CYS'):
            print(f'build_disulfide() requires CYS at both residues:
{prox} {prox_residue.get_resname()} {dist} {dist_residue.get_resname()}
```



```
Chain: {prox_residue.get_segid()}')

        # set the objects proximal and distal values
        self.set_resnum(proximal, distal)

        self.proximal_chain = chain1.get_id()
        self.distal_chain = chain2.get_id()

        self.proximal_residue_fullid = prox_residue.get_full_id()
        self.distal_residue_fullid = dist_residue.get_full_id()


        # grab the coordinates for the proximal and distal residues as
vectors so we can do math on them later

        # proximal residue
        if self.QUIET:
            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)

```

```
1040         # calculate and set the disulfide dihedral angles
1041         self.chi1 = numpy.degrees(calc_dihedral(n1, ca1, cb1, sg1))
1042         self.chi2 = numpy.degrees(calc_dihedral(ca1, cb1, sg1, sg2))
1043         self.chi3 = numpy.degrees(calc_dihedral(cb1, sg1, sg2, cb2))
1044         self.chi4 = numpy.degrees(calc_dihedral(sg1, sg2, cb2, ca2))
1045         self.chi5 = numpy.degrees(calc_dihedral(sg2, cb2, ca2, n2))
1046 
1047         # calculate and set the SS bond torsional energy
1048         self.compute_disulfide_torsional_energy()
1049 
1050         # compute and set the local coordinates
1051         self.compute_local_disulfide_coords()
1052 
1053     def set_chain_id(self, chain_id):
1054         self.chain_id = chain_id
1055 
1056     def set_positions(self, n_prox: Vector, ca_prox: Vector, c_prox:
    … Vector,
1057                       o_prox: Vector, cb_prox: Vector, sg_prox: Vector,
1058                       n_dist: Vector, ca_dist: Vector, c_dist: Vector,
1059                       o_dist: Vector, cb_dist: Vector, sg_dist: Vector):
1060         '''
1061         Sets the atomic positions for all atoms in the disulfide bond.
1062         Arguments:
1063             n_prox
1064             ca_prox
1065             c_prox
1066             o_prox
1067             cb_prox
1068             sg_prox
1069             n_distal
1070             ca_distal
1071             c_distal
1072             o_distal
1073             cb_distal
1074             sg_distal
1075         Returns: None
1076         '''
1077 
1078         # deep copy
1079         self.n_prox = n_prox.copy()
1080         self.ca_prox = ca_prox.copy()
1081         self.c_prox = c_prox.copy()
1082         self.o_prox = o_prox.copy()
1083         self.cb_prox = cb_prox.copy()
1084         self.sg_prox = sg_prox.copy()
1085         self.sg_dist = sg_dist.copy()
1086         self.cb_dist = cb_dist.copy()
```



```
1087         self.ca_dist = ca_dist.copy()
1088         self.n_dist = n_dist.copy()
1089         self.c_dist = c_dist.copy()
1090         self.o_dist = o_dist.copy()
1091 
1092     def set_conformation(self, chi1, chi2, chi3, chi4, chi5):
1093         '''
1094         Sets the 5 dihedral angles chi1 - chi5 for the Disulfide object
    … and computes the torsional energy.
1095 
1096         Arguments: chi, chi2, chi3, chi4, chi5 - Dihedral angles in
    … degrees (-180 - 180) for the Disulfide conformation.
1097         Returns: None
1098         '''
1099 
1100         self.chi1 = chi1
1101         self.chi2 = chi2
1102         self.chi3 = chi3
1103         self.chi4 = chi4
1104         self.chi5 = chi5
1105         self.dihedrals = list([chi1, chi2, chi3, chi4, chi5])
1106         self.compute_disulfide_torsional_energy()
1107 
1108     def set_name(self, namestr="Disulfide"):
1109         '''
1110         Sets the Disulfide's name
1111         Arguments: (str)namestr
1112         Returns: none
1113         '''
1114 
1115         self.name = namestr
1116 
1117     def set_resnum(self, proximal, distal):
1118         '''
1119         Sets the Proximal and Distal Residue numbers for the Disulfide
1120         Arguments:
1121             Proximal: Proximal residue number
1122             Distal: Distal residue number
1123         Returns: None
1124         '''
1125 
1126         self.proximal = proximal
1127         self.distal = distal
1128 
1129     def compute_disulfide_torsional_energy(self):
1130         '''
1131         Compute the approximate torsional energy for the Disulfide's
    … conformation.
```

```
        Arguments: chi1, chi2, chi3, chi4, chi5 - the dihedral angles for
… the Disulfide
        Returns: Energy (kcal/mol)
        '''
        # @TODO find citation for the ss bond energy calculation
        chi1 = self.chi1
        chi2 = self.chi2
        chi3 = self.chi3
        chi4 = self.chi4
        chi5 = self.chi5

        energy = 2.0 * (cos(torad(3.0 * chi1)) + cos(torad(3.0 * chi5)))
        energy += cos(torad(3.0 * chi2)) + cos(torad(3.0 * chi4))
        energy += 3.5 * cos(torad(2.0 * chi3)) + 0.6 * cos(torad(3.0 *
… chi3)) + 10.1

        self.energy = energy

    def compute_local_disulfide_coords(self):
        """
        Compute the internal coordinates for a properly initialized
… Disulfide Object.
        Arguments: SS initialized Disulfide object
        Returns: None, modifies internal state of the input
        """

        turt = Turtle3D('tmp')
        # get the coordinates as numpy.array for Turtle3D use.
        n = self.n_prox.get_array()
        ca = self.ca_prox.get_array()
        c = self.c_prox.get_array()
        cb = self.cb_prox.get_array()
        o = self.o_prox.get_array()
        sg = self.sg_prox.get_array()

        sg2 = self.sg_dist.get_array()
        cb2 = self.cb_dist.get_array()
        ca2 = self.ca_dist.get_array()
        c2 = self.c_dist.get_array()
        n2 = self.n_dist.get_array()
        o2 = self.o_dist.get_array()

        turt.orient_from_backbone(n, ca, c, cb, ORIENT_SIDECHAIN)

        # internal (local) coordinates, stored as Vector objects
        # to_local returns numpy.array objects

        self._n_prox = Vector(turt.to_local(n))
```



```
        self._ca_prox = Vector(turt.to_local(ca))
        self._c_prox = Vector(turt.to_local(c))
        self._o_prox = Vector(turt.to_local(o))
        self._cb_prox = Vector(turt.to_local(cb))
        self._sg_prox = Vector(turt.to_local(sg))

        self._n_dist = Vector(turt.to_local(n2))
        self._ca_dist = Vector(turt.to_local(ca2))
        self._c_dist = Vector(turt.to_local(c2))
        self._o_dist = Vector(turt.to_local(o2))
        self._cb_dist = Vector(turt.to_local(cb2))
        self._sg_dist = Vector(turt.to_local(sg2))

    def build_disulfide_model(self, turtle: Turtle3D):
        """
        Build a model Disulfide based on the internal dihedral angles.
        Routine assumes turtle is in orientation #1 (at Ca, headed toward
        Cb, with N on left), builds disulfide, and updates the object's
… internal
        coordinate state. It also adds the distal protein backbone,
        and computes the disulfide conformational energy.

        Arguments: turtle: Turtle3D object properly oriented for the
… build.
        Returns: None. The Disulfide object's internal state is updated.
        """

        tmp = Turtle3D('tmp')
        tmp.copy_coords(turtle)

        n = Vector(0, 0, 0)
        ca = Vector(0, 0, 0)
        cb = Vector(0, 0, 0)
        c = Vector(0, 0, 0)

        self.ca_prox = tmp._position
        tmp.schain_to_bbone()
        n, ca, cb, c = build_residue(tmp)

        self.n_prox = n
        self.ca_prox = ca
        self.c_prox = c

        tmp.bbone_to_schain()
        tmp.move(1.53)
        tmp.roll(self.chi1)
        tmp.yaw(112.8)
        self.cb_prox = tmp._position
```

```
1223
1224            tmp.move(1.86)
1225            tmp.roll(self.chi2)
1226            tmp.yaw(103.8)
1227            self.sg_prox = tmp._position
1228
1229            tmp.move(2.044)
1230            tmp.roll(self.chi3)
1231            tmp.yaw(103.8)
1232            self.sg_dist = tmp._position
1233
1234            tmp.move(1.86)
1235            tmp.roll(self.chi4)
1236            tmp.yaw(112.8)
1237            self.cb_dist = tmp._position
1238
1239            tmp.move(1.53)
1240            tmp.roll(self.chi5)
1241            tmp.pitch(180.0)
1242            tmp.schain_to_bbone()
1243            n, ca, cb, c = build_residue(tmp)
1244
1245            self.n_dist = n
1246            self.ca_dist = ca
1247            self.c_dist = c
1248
1249            self.compute_torsional_energy()
1250
1251    # Class defination ends
1252
1253    # End of file
1254
```