pgn_build

November 8, 2022

[1]:
```
# PROTEIN GRAPH NETWORK: MAKE ATOM LIST
import numpy as np

'''
MAKE ATOM LIST
    - PARAM:
        - pdb_file_path = path to new imported .pdb file
        - hetatm_name_list = list of all target HETATM residues, with structure␣
 ↪RESIDUE_NAME-CHAIN_ID, i.e. IRE-A
    - Makes 2D array with the following column index assignments:
        0 - index (0 to end, consecutive)
        1 - atom_number (number from original .pdb file, not necessarily␣
 ↪consecutive)
        2 - atom_name (unique atom name)
        3 - atom_type (array structured with [B]ackbone and [R]esidual first,␣
 ↪then [H]etatm and [W]ater)
        4 - residue_number
        5 - residue_name (3-letter abbreviation)
        6 - chain_id (A, B, C, etc.)
        7-9 - atomic x_coord, y_coord, z_coord
        10 - element (symbol)
    - RETURN:
        - atom_list = 2D array with atom information on each row
'''
def make_atom_list(pdb_file_path,hetatm_name_list):
    print('\n[running make_atom_list]')

     # get PDB file contents
    try:
        with open(pdb_file_path,'r') as pdb_file:
            pdb = pdb_file.readlines()
    except:
        raise Exception(f'  ERROR: error opening PDB file: [{pdb_file_path}];␣
 ↪invalid PDB file path')

    # check inputs
    if len(pdb)==0: # check if PDB file is valid
```

```
        raise Exception(f'  ERROR: error reading PDB file; file is empty; check␣
↪PDB file path')
    try:
        hetatm_name_list = list(map(str,hetatm_name_list))
    except:
        raise Exception(f'  ERROR: error reading list of all HETATM residue␣
↪names: [{hetatm_name_list}]')
    for hetatm_name in hetatm_name_list:
        if '-' not in hetatm_name:
            raise Exception(f'  ERROR: error reading HETATM residue name:␣
↪[{hetatm_name}]; must follow structure RESIDUE_NAME-CHAIN_ID, i.e. IRE-A')

    # make 2D array and fill
    try:
        maxlen = len(pdb) # maximum number of atoms cannot be larger than␣
↪number of lines in PDB
        atom_list = np.empty([maxlen,11],dtype=object) # preallocate atom array

        index = 0
        residue_position = 0 # position relative to the start of a residue␣
↪(index 0)
        prev_residue_number = -1

        for line in pdb:
            if line[0:6].upper()=='ATOM  ': # find protein backbone and residue␣
↪atom lines in PDB and add to list
                atom_number = int(line[6:11])
                atom_name = line[12:16].replace(' ','').upper()
                atom_type = '' # adjusted below
                residue_number = int(line[22:26])
                residue_name = line[17:20].replace(' ','').upper()
                chain_id = line[21].upper()
                x_coord = float(line[30:38])
                y_coord = float(line[38:46])
                z_coord = float(line[46:54])
                element = line[76:78].replace(' ','').upper()

                alt_loc = line[16].upper()
                occupancy = float(line[54:60])

                if (occupancy>0.5) or (occupancy==0.5 and alt_loc=='A'): # for␣
↪atoms wtih alternate locations, choose location with higher occupancy
                    if residue_number==prev_residue_number: # mark residue atoms
                        residue_position += 1
                    else:
                        residue_position = 0
```

```
                    prev_residue_number = residue_number
                    if residue_position<4 and␣
↪atom_name==['N','CA','C','O'][residue_position]:
                        atom_type = 'B' # mark backbone atoms
                    else:
                        atom_type = 'R' # mark residue atoms

                    atom_list[index,:] = [index,atom_number,atom_name,atom_type,
                                          residue_number,residue_name,chain_id,
                                          x_coord,y_coord,z_coord,element]
                    index += 1

        for line in pdb:
            if line[0:6].upper()=='HETATM': # find hetatm and water atom lines␣
↪in PDB and add to list
                atom_number = int(line[6:11])
                atom_name = line[12:16].replace(' ','').upper()
                atom_type = 'H' # mark hetatm atoms (adjusted below)
                residue_number = int(line[22:26])
                residue_name = line[17:20].replace(' ','').upper()
                chain_id = line[21].upper()
                x_coord = float(line[30:38])
                y_coord = float(line[38:46])
                z_coord = float(line[46:54])
                element = line[76:78].replace(' ','').upper()

                alt_loc = line[16].upper()
                occupancy = float(line[54:60])

                if (f'{residue_name}-{chain_id}' in hetatm_name_list) and␣
↪((occupancy>0.5) or (occupancy==0.5 and (alt_loc=='A' or alt_loc==' '))): #␣
↪for atoms wtih alternate locations, choose location with higher occupancy
                    if residue_name=='HOH': # mark water atoms
                        atom_type = 'W'

                    atom_list[index,:] = [index,atom_number,atom_name,atom_type,
                                          residue_number,residue_name,chain_id,
                                          x_coord,y_coord,z_coord,element]
                    index += 1

        atom_list = atom_list[0:index,:] # crop atom array to remove unfilled␣
↪rows
    except Exception as e:
        raise Exception(f'  ERROR: error making atom list; check PDB data␣
↪structure; PDB line and error message:\n{line}\n{e}')
```

```
    if not np.array_equal(atom_list[:,0],np.array(range(len(atom_list[:,0])))):␣
 ↪# check indices are consecutive
        print('  WARNING: error in parsing PDB; indices not consecutive␣
 ↪indicating missing atom record')

    return atom_list


# PROTEIN GRAPH NETWORK: MAKE DISTANCE MATRIX
import numpy as np

'''
MAKE DISTANCE MATRIX
    - PARAM:
        - atom_list = 2D array with atom information on each row
    - Makes 2D square array of pairwise distances between all atoms (indexed␣
 ↪along row and column axes) in Angstroms
    - RETURN:
        - dist_matrix = 2D array with atom distance information at each element
'''
def make_distance_matrix(atom_list):
    print('\n[running make_distance_matrix]')

    if type(atom_list)!=np.ndarray:
        raise Exception(f'  ERROR: atom list must be a Numpy ndarray')

    n = len(atom_list[:,0]) # total number of atoms

    try:
        xc = np.zeros([n,1]) # preallocate column vectors for coordinates
        yc = np.zeros([n,1])
        zc = np.zeros([n,1])

        xc[:,0] = atom_list[:,7] # fill column vectors for coordinates
        yc[:,0] = atom_list[:,8]
        zc[:,0] = atom_list[:,9]

        xyzdiff = [(xc-(xc.T)),(yc-(yc.T)),(zc-(zc.T))] # calculate pairwise␣
 ↪differences between coordinates

        dist_matrix = np.linalg.norm(xyzdiff,axis=0) # calculate Euclidean norm␣
 ↪to get distances
    except Exception as e:
        raise Exception(f'  ERROR: error creating distance matrix; check atom␣
 ↪list; error message:\n{e}')
```

```
    return dist_matrix


# PROTEIN GRAPH NETWORK: MAKE CONNECTION MATRIX
import numpy as np

'''
MAKE CONNECTION MATRIX
    - PARAM:
        - atom_list = 2D array with atom information on each row
    - Preallocates 2D square array of bond and interaction types between atoms␣
 ↪(indexed along row and column axes):
        - BCS = backbone covalent single bond
        - BCD = backbone covalent double bond
        - RCS = residue covalent single bond
        - RCD = residue covalent double bond
        - HCS = hetatm covalent single bond
        - HCD = hetatm covalent double bond
        - HCT = hetatm covalent triple bond
        - HHH = hydrogen interaction
        - HPO = hydrophobic interaction
    - RETURN:
        - conn_matrix = 2D array with bond information at each element
'''
def make_connection_matrix(atom_list):
    print('\n[running make_connection_matrix]')

    if type(atom_list)!=np.ndarray:
        raise Exception(f'  ERROR: atom list must be a Numpy ndarray')

    n = len(atom_list[:,0]) # total number of atoms
    conn_matrix = np.empty([n,n],dtype=object) # create connection matrix

    return conn_matrix


# PROTEIN GRAPH NETWORK: FILL PROTEIN BONDS
import numpy as np

'''
FILL PROTEIN BONDS
    - PARAM:
        - conn_matrix = 2D array with bond information at each element
        - atom_list = 2D array with atom information on each row
    - Adds protein backbone (BCS, BCD) and residue (RCS, RCD) bonds to␣
 ↪connection matrix
    - RETURN:
```

```
        - conn_matrix = 2D array with bond information at each element
'''
def fill_protein_bonds(conn_matrix,atom_list):
    print('\n[running fill_protein_bonds]')

    if type(conn_matrix)!=np.ndarray:
        raise Exception(f'  ERROR: connection matrix must be a square Numpy␣
 ↪ndarray')
    if type(atom_list)!=np.ndarray:
        raise Exception(f'  ERROR: atom list must be a Numpy ndarray')

    n = len(atom_list[:,0]) # total number of atoms
    if np.shape(conn_matrix)!=(n,n):
        raise Exception('  ERROR: improperly-sized connection matrix or atom␣
 ↪list; connection matrix dimensions do not match atom list length')

    try:
        n_prev = -1 # records index of previous amino acid backbone N (for␣
 ↪backbone bonding)
        ca_prev = -1 # records index of previous amino acid backbone CA (for␣
 ↪residue bonding)
        c_prev = -1 # records index of previous amino acid backbone C (for␣
 ↪backbone bonding)
        aaid_prev = -1 # records previous residue_number

        i = 0
        while i<n:

            # 1. BACKBONE (PDB ATOM ORDER: N,CA,C,O)
            if atom_list[i,3]=='B' and atom_list[i,2]=='N': # N
                if n_prev>=0 and atom_list[c_prev,2]=='C' and␣
 ↪atom_list[c_prev,3]=='B' and atom_list[i,4]==(atom_list[n_prev,4]+1) and␣
 ↪atom_list[i,6]==atom_list[n_prev,6]: # checks if backbone segemnt connects␣
 ↪to an adjacent segment (accounts for missing amino acids)
                    conn_matrix[n_prev+2,i] = 'BCS' # C-N backbone bond
                n_prev = i # update N index
                backbone_count = 0

                ca_prev = -1
                c_prev = -1

                if atom_list[i+1,3]=='B' and atom_list[i+1,2]=='CA': # CA
                    conn_matrix[i,i+1] = 'BCS' # N-CA backbone bond
                    ca_prev = i+1 # update CA index
                    backbone_count += 1
                    if atom_list[i+2,3]=='B' and atom_list[i+2,2]=='C': # C
```

```
                        conn_matrix[i+1,i+2] = 'BCS' # CA-C backbone bond
                        c_prev = i+2 # update C index
                        backbone_count += 1
                        if atom_list[i+3,3]=='B' and atom_list[i+3,2]=='O': # O
                            conn_matrix[i+2,i+3] = 'BCD' # C=O backbone bond
                            backbone_count += 1

                if (backbone_count+1)<4:
                    print(f'  WARNING: missing backbone atom(s) in residue␣
↪[{str(atom_list[i,5]).upper()}-{atom_list[i,6]} #{atom_list[i,4]}]')

                i += backbone_count # skip to next possible amino acid backbone␣
↪N

            # 2. RESIDUES (PDB GREEK ALPHABET ORDER: B,G,D,E,Z,H)
            elif atom_list[i,3]=='R':
                if atom_list[i,4]!=aaid_prev: # checks that atom is part of new␣
↪residue
                    amino = str(atom_list[i,5]).upper() # gets residue name
                    complete_res = False # condition to check if residue is␣
↪complete

                    if amino=='ALA': # 1 alanine - CB
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            complete_res = True

                    elif amino=='VAL': # 2 valine - CB CG1 CG2
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG1':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='CG2':
                                    conn_matrix[i,i+2] = 'RCS'
                                    complete_res = True

                    elif amino=='ILE': # 3 isoleucine - CB CG1 CG2 CD1
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG1':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='CG2':
                                    conn_matrix[i,i+2] = 'RCS'
                                    if atom_list[i+3,2]=='CD1':
                                        conn_matrix[i+1,i+3] = 'RCS'
                                        complete_res = True
```

```
                    elif amino=='LEU': # 4 leucine - CB CG CD1 CD2
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='CD1':
                                    conn_matrix[i+1,i+2] = 'RCS'
                                    if atom_list[i+3,2]=='CD2':
                                        conn_matrix[i+1,i+3] = 'RCS'
                                        complete_res = True

                    elif amino=='MET': # 5 methionine - CB CG SD CE
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='SD':
                                    conn_matrix[i+1,i+2] = 'RCS'
                                    if atom_list[i+3,2]=='CE':
                                        conn_matrix[i+2,i+3] = 'RCS'
                                        complete_res = True

                    elif amino=='PHE': # 6 phenylalanine - CB CG CD1 CD2 CE1␣
↪CE2 CZ
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='CD1':
                                    conn_matrix[i+1,i+2] = 'RCD'
                                    if atom_list[i+3,2]=='CD2':
                                        conn_matrix[i+1,i+3] = 'RCS'
                                        if atom_list[i+4,2]=='CE1':
                                            conn_matrix[i+2,i+4] = 'RCS'
                                            if atom_list[i+5,2]=='CE2':
                                                conn_matrix[i+3,i+5] = 'RCD'
                                                if atom_list[i+6,2]=='CZ':
                                                    conn_matrix[i+4,i+6] = 'RCD'
                                                    conn_matrix[i+5,i+6] = 'RCS'
                                                    complete_res = True

                    elif amino=='TYR': # 7 tyrosine - CB CG CD1 CD2 CE1 CE2 CZ␣
↪OH
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG':
                                conn_matrix[i,i+1] = 'RCS'
```

```
                                if atom_list[i+2,2]=='CD1':
                                    conn_matrix[i+1,i+2] = 'RCD'
                                    if atom_list[i+3,2]=='CD2':
                                        conn_matrix[i+1,i+3] = 'RCS'
                                        if atom_list[i+4,2]=='CE1':
                                            conn_matrix[i+2,i+4] = 'RCS'
                                            if atom_list[i+5,2]=='CE2':
                                                conn_matrix[i+3,i+5] = 'RCD'
                                                if atom_list[i+6,2]=='CZ':
                                                    conn_matrix[i+4,i+6] = 'RCD'
                                                    conn_matrix[i+5,i+6] = 'RCS'
                                                    if atom_list[i+7,2]=='OH':
                                                        conn_matrix[i+6,i+7] =␣
↪'RCS'
                                                        complete_res = True

                    elif amino=='TRP': # 8 tryptophan - CB CG CD1 CD2 NE1 CE2␣
↪CE3 CZ2 CZ3 CH2
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='CD1':
                                    conn_matrix[i+1,i+2] = 'RCD'
                                    if atom_list[i+3,2]=='CD2':
                                        conn_matrix[i+1,i+3] = 'RCS'
                                        if atom_list[i+4,2]=='NE1':
                                            conn_matrix[i+2,i+4] = 'RCS'
                                            if atom_list[i+5,2]=='CE2':
                                                conn_matrix[i+3,i+5] = 'RCS'
                                                if atom_list[i+6,2]=='CE3':
                                                    conn_matrix[i+3,i+6] = 'RCD'
                                                    conn_matrix[i+4,i+6] = 'RCS'
                                                    if atom_list[i+7,2]=='CZ2':
                                                        conn_matrix[i+5,i+7] =␣
↪'RCD'
                                                        if␣
↪atom_list[i+8,2]=='CZ3':
                                                            ␣
↪conn_matrix[i+6,i+8] = 'RCS'
                                                            if␣
↪atom_list[i+9,2]=='CH2':
                                                                ␣
↪conn_matrix[i+7,i+9] = 'RCS'
                                                                ␣
↪conn_matrix[i+8,i+9] = 'RCD'
```

```
                                                    complete_res =␣
↪True

                    elif amino=='CYS': # 9 cysteine - CB SG
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='SG':
                                conn_matrix[i,i+1] = 'RCS'
                                complete_res = True

                    elif amino=='GLY': # 10 glycine - [NONE]
                        complete_res = True

                    elif amino=='PRO': # 11 proline - CB CG CD
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='CD':
                                    conn_matrix[i+1,i+2] = 'RCS'
                                    conn_matrix[ca_prev-1,i+2] = 'RCS'
                                    complete_res = True

                    elif amino=='SER': # 12 serine - CB OG
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='OG':
                                conn_matrix[i,i+1] = 'RCS'
                                complete_res = True

                    elif amino=='THR': # 13 threonine - CB OG1 CG2
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='OG1':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='CG2':
                                    conn_matrix[i,i+2] = 'RCS'
                                    complete_res = True

                    elif amino=='ASN': # 14 asparagine - CB CG OD1 ND2
                        if atom_list[i,2]=='CB':
                            conn_matrix[ca_prev,i] = 'RCS'
                            if atom_list[i+1,2]=='CG':
                                conn_matrix[i,i+1] = 'RCS'
                                if atom_list[i+2,2]=='OD1':
                                    conn_matrix[i+1,i+2] = 'RCD'
                                    if atom_list[i+3,2]=='ND2':
```

```
                            conn_matrix[i+1,i+3] = 'RCS'
                            complete_res = True

                elif amino=='GLN': # 15 glutamine - CB CG CD OE1 NE2
                    if atom_list[i,2]=='CB':
                        conn_matrix[ca_prev,i] = 'RCS'
                        if atom_list[i+1,2]=='CG':
                            conn_matrix[i,i+1] = 'RCS'
                            if atom_list[i+2,2]=='CD':
                                conn_matrix[i+1,i+2] = 'RCS'
                                if atom_list[i+3,2]=='OE1':
                                    conn_matrix[i+2,i+3] = 'RCD'
                                    if atom_list[i+4,2]=='NE2':
                                        conn_matrix[i+2,i+4] = 'RCS'
                                        complete_res = True

                elif amino=='ARG': # 16 arginine - CB CG CD NE CZ NH1 NH2
                    if atom_list[i,2]=='CB':
                        conn_matrix[ca_prev,i] = 'RCS'
                        if atom_list[i+1,2]=='CG':
                            conn_matrix[i,i+1] = 'RCS'
                            if atom_list[i+2,2]=='CD':
                                conn_matrix[i+1,i+2] = 'RCS'
                                if atom_list[i+3,2]=='NE':
                                    conn_matrix[i+2,i+3] = 'RCS'
                                    if atom_list[i+4,2]=='CZ':
                                        conn_matrix[i+3,i+4] = 'RCS'
                                        if atom_list[i+5,2]=='NH1':
                                            conn_matrix[i+4,i+5] = 'RCS'
                                            if atom_list[i+6,2]=='NH2':
                                                conn_matrix[i+4,i+6] = 'RCD'
                                                complete_res = True

                elif amino=='HIS': # 17 histidine - CB CG ND1 CD2 CE1 NE2
                    if atom_list[i,2]=='CB':
                        conn_matrix[ca_prev,i] = 'RCS'
                        if atom_list[i+1,2]=='CG':
                            conn_matrix[i,i+1] = 'RCS'
                            if atom_list[i+2,2]=='ND1':
                                conn_matrix[i+1,i+2] = 'RCS'
                                if atom_list[i+3,2]=='CD2':
                                    conn_matrix[i+1,i+3] = 'RCD'
                                    if atom_list[i+4,2]=='CE1':
                                        conn_matrix[i+2,i+4] = 'RCD'
                                        if atom_list[i+5,2]=='NE2':
                                            conn_matrix[i+3,i+5] = 'RCS'
                                            conn_matrix[i+4,i+5] = 'RCS'
```

```
                                    complete_res = True

            elif amino=='LYS': # 18 lysine - CB CG CD CE NZ
                if atom_list[i,2]=='CB':
                    conn_matrix[ca_prev,i] = 'RCS'
                    if atom_list[i+1,2]=='CG':
                        conn_matrix[i,i+1] = 'RCS'
                        if atom_list[i+2,2]=='CD':
                            conn_matrix[i+1,i+2] = 'RCS'
                            if atom_list[i+3,2]=='CE':
                                conn_matrix[i+2,i+3] = 'RCS'
                                if atom_list[i+4,2]=='NZ':
                                    conn_matrix[i+3,i+4] = 'RCS'
                                    complete_res = True

            elif amino=='ASP': # 19 aspartic acid - CB CG OD1 OD2
                if atom_list[i,2]=='CB':
                    conn_matrix[ca_prev,i] = 'RCS'
                    if atom_list[i+1,2]=='CG':
                        conn_matrix[i,i+1] = 'RCS'
                        if atom_list[i+2,2]=='OD1':
                            conn_matrix[i+1,i+2] = 'RCS'
                            if atom_list[i+3,2]=='OD2':
                                conn_matrix[i+1,i+3] = 'RCD'
                                complete_res = True

            elif amino=='GLU': # 20 glutamic acid - CB CG CD OE1 OE2
                if atom_list[i,2]=='CB':
                    conn_matrix[ca_prev,i] = 'RCS'
                    if atom_list[i+1,2]=='CG':
                        conn_matrix[i,i+1] = 'RCS'
                        if atom_list[i+2,2]=='CD':
                            conn_matrix[i+1,i+2] = 'RCS'
                            if atom_list[i+3,2]=='OE1':
                                conn_matrix[i+2,i+3] = 'RCS'
                                if atom_list[i+4,2]=='OE2':
                                    conn_matrix[i+2,i+4] = 'RCD'
                                    complete_res = True

            elif amino=='SEC': # 21 selenocysteine (rare) - CB SEG (?)
                if atom_list[i,2]=='CB':
                    conn_matrix[ca_prev,i] = 'RCS'
                    if atom_list[i+1,2]=='SEG':
                        conn_matrix[i,i+1] = 'RCS'
                        complete_res = True

            # elif amino=='PYL': # 22 pyrrolysine (rare) (?)
```

```
                    else:
                        print(f'  WARNING: unrecognized residue␣
 ↪[{amino}-{atom_list[i,6]} #{atom_list[i,4]}]')

                    if complete_res==False:
                        print(f'  WARNING: incomplete residue␣
 ↪[{amino}-{atom_list[i,6]} #{atom_list[i,4]}]')

                    aaid_prev = atom_list[i,4] # update last amino acid segment␣
 ↪id

            elif atom_list[i,3]=='H' or atom_list[i,3]=='W':
                pass # ignore HETATM and water bonds
            else:
                print(f'  WARNING: non-covalently-bonded atom␣
 ↪[{atom_list[i,2]}-{atom_list[i,3]} #{atom_list[i,0]}] in residue␣
 ↪[{str(atom_list[i,5]).upper()}-{atom_list[i,6]} #{atom_list[i,4]}]')
            i += 1
    except Exception as e:
        raise Exception(f'  ERROR: error adding protein bonds to connection␣
 ↪matrix; check atom list; error message:\n{e}')

    return conn_matrix


# PROTEIN GRAPH NETWORK: FILL LIGAND BONDS
import numpy as np

'''
FILL LIGAND BONDS
    - PARAM:
        - conn_matrix = 2D array with bond information at each element
        - atom_list = 2D array with atom information on each row
        - ligand_bonds = 2D array with ligand bond information on each row
    - Adds ligand (HCS, HCD, HCT) and ligand-protein (HHH, HPO) bonds to␣
 ↪connection matrix
    - RETURN:
        - conn_matrix = 2D array with bond information at each element
'''
def fill_ligand_bonds(conn_matrix,atom_list,ligand_bonds):
    print('\n[running fill_ligand_bonds]')

    if type(conn_matrix)!=np.ndarray:
        raise Exception(f'  ERROR: connection matrix must be a square Numpy␣
 ↪ndarray')
```

```
    if type(atom_list)!=np.ndarray:
        raise Exception(f'  ERROR: atom list must be a Numpy ndarray')
    if type(ligand_bonds) not in [np.ndarray,list]:
        raise Exception(f'  ERROR: ligand bond information must be a list or␣
↪Numpy ndarray')

    n = len(atom_list[:,0]) # total number of atoms
    if np.shape(conn_matrix)!=(n,n):
        raise Exception('  ERROR: improperly-sized connection matrix or atom␣
↪list; connection matrix dimensions do not match atom list length')

    try:
        for line in ligand_bonds:
            a_residue_name = line[0].upper()
            a_residue_number = int(line[1])
            a_chain_id = line[2].upper()
            a_atom_name = line[3].upper()

            b_residue_name = line[5].upper()
            b_residue_number = int(line[6])
            b_chain_id = line[7].upper()
            b_atom_name = line[8].upper()

            bond_type = line[4]

            if a_residue_number in list(atom_list[:,4]):
                try:
                    a_ind = np.where((atom_list[:,5]==a_residue_name) &␣
↪(atom_list[:,4]==a_residue_number) &
                                     (atom_list[:,6]==a_chain_id) & (atom_list[:
↪,2]==a_atom_name))[0][0]
                except:
                    print(f'  WARNING: error parsing ligplot bond:␣
↪[{a_atom_name} in {a_residue_name}-{a_chain_id} #{a_residue_number}] with␣
↪{bond_type} bond to [{b_atom_name} in {b_residue_name}-{b_chain_id}␣
↪#{b_residue_number}]')
                    break
            else:
                print(f'  WARNING: residue in ligand {bond_type} bond␣
↪information not found in PDB: [{str(a_residue_name).upper()}-{a_chain_id}␣
↪#{a_residue_number}]')
                continue

            if b_residue_number in list(atom_list[:,4]):
                try:
```

```
                    b_ind = np.where((atom_list[:,5]==b_residue_name) &␣
↪(atom_list[:,4]==b_residue_number) &
                                     (atom_list[:,6]==b_chain_id) & (atom_list[:
↪,2]==b_atom_name))[0][0]
                except:
                    print(f'  ERROR: error parsing ligplot bond: [{a_atom_name}␣
↪in {a_residue_name}-{a_chain_id} #{a_residue_number}] with {bond_type} bond␣
↪to [{b_atom_name} in {b_residue_name}-{b_chain_id} #{b_residue_number}]')
                    break
            else:
                print(f'  WARNING: residue in ligand {bond_type} bond␣
↪information not found in PDB: [{str(b_residue_name).upper()}-{b_chain_id}␣
↪#{b_residue_number}]')
                continue

            if a_ind>b_ind:
                conn_matrix[b_ind,a_ind] = bond_type
            elif b_ind>a_ind:
                conn_matrix[a_ind,b_ind] = bond_type
    except Exception as e:
        raise Exception(f'  ERROR: error adding ligand bonds to connection␣
↪matrix; check atom list and ligand bond array; error message:\n{e}')

    return conn_matrix


# PROTEIN GRAPH NETWORK: MAKE BOND LIST
import numpy as np

'''
MAKE BOND LIST
    - PARAM:
        - conn_matrix = 2D array with bond information at each element
        - atom_list = 2D array with atom information on each row
        - dist_matrix = 2D array with atom distance information at each element
    - Makes 2D array with the following column index assignments:
        0 - bond_index (0 to end, consecutive)
        1 - a_index (atom A index for bond)
        2 - b_index (atom B index for bond)
        3 - bond_type (BCS, BCD, RCS, RCD, HCS, HCD, HCT, HHH, HPO)
        4 - bond_length (distance)

        5 - a_atom_number (numbering from original .pdb file, not necessarily␣
↪consecutive)
        6 - a_atom_name (unique atom name)
        7 - a_atom_type ([B]ackbone, [R]esidual, [H]etatm, [W]ater)
```

```
        8 - a_residue_number
        9 - a_residue_name (3-letter abbreviation)
        10 - a_chain_id (A, B, C, etc.)
        11-13 - atomic a_x_coord, a_y_coord, a_z_coord
        14 - a_element (symbol)

        15 - b_atom_number (numbering from original .pdb file, not necessarily␣
 ↪consecutive)
        16 - b_atom_name (unique atom name)
        17 - b_atom_type ([B]ackbone, [R]esidual, [H]etatm, [W]ater)
        18 - b_residue_number
        19 - b_residue_name (3-letter abbreviation)
        20 - b_chain_id (A, B, C, etc.)
        21-23 - atomic b_x_coord, b_y_coord, b_z_coord
        24 - b_element (symbol)
    - RETURN:
        - bond_list = 2D array with bond information on each row
'''
def make_bond_list(conn_matrix,atom_list,dist_matrix):
    print('\n[running make_bond_list]')

    if type(conn_matrix)!=np.ndarray:
        raise Exception(f'  ERROR: connection matrix must be a square Numpy␣
 ↪ndarray')
    if type(atom_list)!=np.ndarray:
        raise Exception(f'  ERROR: atom list must be a Numpy ndarray')
    if type(dist_matrix)!=np.ndarray:
        raise Exception(f'  ERROR: distance matrix must be a square Numpy␣
 ↪ndarray')

    n = len(atom_list[:,0]) # total number of atoms
    if np.shape(conn_matrix)!=(n,n):
        raise Exception(f'  ERROR: improperly-sized connection matrix or atom␣
 ↪list; connection matrix dimensions [{np.shape(conn_matrix)}] do not match␣
 ↪atom list length [{n}]')

    if np.size(dist_matrix)==0 or np.shape(dist_matrix)!=(n,n):
        print(f'  WARNING: improperly-sized distance matrix; distance matrix␣
 ↪dimensions [{np.shape(dist_matrix)}] do not match connection matrix␣
 ↪dimensions [{np.shape(dist_matrix)}]; setting bond lengths/distances to␣
 ↪zero')
        dist_matrix = np.zeros([n,n])

    try:
        n = len(atom_list[:,0])

        map_idxs = (np.where(np.array(np.ravel(conn_matrix))!=None))[0]
```

```
bond_index = np.array(range(len(map_idxs)))
a_index = np.repeat(range(n),n)[map_idxs]
b_index = np.tile(range(n),n)[map_idxs]
bond_type = np.array(np.ravel(conn_matrix))[map_idxs]
bond_length = np.array(np.ravel(dist_matrix))[map_idxs]

a_atom_number = atom_list[a_index,1]
a_atom_name = atom_list[a_index,2]
a_atom_type = atom_list[a_index,3]
a_residue_number = atom_list[a_index,4]
a_residue_name = atom_list[a_index,5]
a_chain_id = atom_list[a_index,6]
a_x_coord = atom_list[a_index,7]
a_y_coord = atom_list[a_index,8]
a_z_coord = atom_list[a_index,9]
a_element = atom_list[a_index,10]

b_atom_number = atom_list[b_index,1]
b_atom_name = atom_list[b_index,2]
b_atom_type = atom_list[b_index,3]
b_residue_number = atom_list[b_index,4]
b_residue_name = atom_list[b_index,5]
b_chain_id = atom_list[b_index,6]
b_x_coord = atom_list[b_index,7]
b_y_coord = atom_list[b_index,8]
b_z_coord = atom_list[b_index,9]
b_element = atom_list[b_index,10]

bond_list = np.empty([len(map_idxs),25],dtype=object)

bond_list[:,0] = bond_index
bond_list[:,1] = a_index
bond_list[:,2] = b_index
bond_list[:,3] = bond_type
bond_list[:,4] = bond_length

bond_list[:,5] = a_atom_number
bond_list[:,6] = a_atom_name
bond_list[:,7] = a_atom_type
bond_list[:,8] = a_residue_number
bond_list[:,9] = a_residue_name
bond_list[:,10] = a_chain_id
bond_list[:,11] = a_x_coord
bond_list[:,12] = a_y_coord
bond_list[:,13] = a_z_coord
bond_list[:,14] = a_element
```

```
        bond_list[:,15] = b_atom_number
        bond_list[:,16] = b_atom_name
        bond_list[:,17] = b_atom_type
        bond_list[:,18] = b_residue_number
        bond_list[:,19] = b_residue_name
        bond_list[:,20] = b_chain_id
        bond_list[:,21] = b_x_coord
        bond_list[:,22] = b_y_coord
        bond_list[:,23] = b_z_coord
        bond_list[:,24] = b_element

    except Exception as e:
        raise Exception(f'  ERROR: error making bond list; check distance␣
↪matrix, connection matrix, and atom list; error message:\n{e}')

    for bond in bond_list:
        if bond[1]>bond[2]:
            print(f'  WARNING: improper bond indices detected: [{bond}]; atom B␣
↪index should always be greater than atom A index')

        bond_length = bond[4]
        bond_type = bond[3]
        if bond_type in ['BCS','BCD','RCS','RCD','HCS','HCD','HCT'] and␣
↪bond_length>4:
            print(f'  WARNING: long [{bond_type}] bond detected (> 4 Angstroms):
↪ [{bond}]')
        elif bond_type in ['HHH'] and bond_length>4:
            print(f'  WARNING: long [{bond_type}] bond detected (> 4 Angstroms):
↪ [{bond}]')
        if bond_type in ['VDW'] and bond_length>4:
            print(f'  WARNING: long [{bond_type}] bond detected (> 4 Angstroms):
↪ [{bond}]')

    return bond_list
```

[ ]: