pgn_import

November 8, 2022

[7]:
```python
# PROTEIN GRAPH NETWORK: GET PDB
import os
import numpy as np
import requests

'''
GET PDB
    - PARAM:
        - pdb_input = input [XXXX] PDB code (if using url) or file path ending␣
 ↪in [/XXXX[...].pdb]
        - pdb_url = PDB url to download .pdb file, with 'XXXX' in place of␣
 ↪4-character PDB code
        - output_path = path to output folder
    - Imports .pdb file from either a file path or url
    - Saves .pdb file to the output path
    - Prints the set of unique HETATM molecules
    - RETURN:
        - pdb_file_path = path to new imported .pdb file
'''
def get_pdb(pdb_input,pdb_url,output_path):
    print(f'\n[running get_pdb for [{pdb_input}]]')

    # check inputs
    if type(pdb_input)!=str:
        raise Exception(f'  ERROR: error reading PDB input code or file path:␣
 ↪[{pdb_input}]; must be a string')
    if type(pdb_url)!=str:
        raise Exception(f'  ERROR: error reading PDB url: [{pdb_url}]; must be␣
 ↪a string')

    if type(output_path)!=str or ' ' in output_path:
        raise Exception(f'  ERROR: error reading output path: [{output_path}];␣
 ↪must be a string without spaces')
    try:
        if not os.path.exists(output_path):
            os.makedirs(output_path)
            print(f'  NOTE: creating output path: [{output_path}]')
    except:
        raise Exception(f'  ERROR: error reading output path: [{output_path}]')

    # import .pdb file from url or file path and save to output path
    if len(pdb_input)==4 and len(pdb_url)>4: # using PDB url
        pdb_code = pdb_input.lower()
        url_idx = pdb_url.upper().find('XXXX') # index to insert PDB code
        if url_idx==-1:
            raise Exception(f'  ERROR: error reading PDB url: [{pdb_url}];␣
↪invalid PDB 4-char code or url')
        try:
            pdb_input_file_url = pdb_url[:url_idx] + pdb_code +␣
↪pdb_url[url_idx+4:] # create complete url to PDB
            pdb_file = requests.get(pdb_input_file_url)
        except Exception as e:
            raise Exception(f'  ERROR: error opening PDB url; invalid PDB␣
↪4-char code or url, or bad network connection; error message:\n{e}')
        if str(pdb_file)=='<Response [404]>':
            raise Exception(f'  ERROR: error opening PDB url:␣
↪[{pdb_input_file_url}]; response [404] returned')
        try:
            pdb_text = pdb_file.text
            pdb_lines = pdb_text.splitlines()
        except Exception as e:
            raise Exception(f'  ERROR: error formatting PDB contents; error␣
↪message:\n{e}')

        pdb_file_path = f'{output_path}/{pdb_code}.pdb'
        open(pdb_file_path,'w').write(pdb_text) # write PDB to a new file in␣
↪output path
        print(f'  NOTE: saving PDB file to output path: [{pdb_file_path}]')

    elif pdb_input[-4:]=='.pdb' and len(pdb_input)>=8: # using PDB file path
        pdb_code = pdb_input.split('/')[-1][0:4].lower()
        pdb_input_file_name = pdb_input.split('/')[-1]
        pdb_input_file_path = pdb_input
        try:
            with open(pdb_input_file_path,'r') as pdb_file:
                pdb_lines = pdb_file.readlines()
            pdb_text = ''.join(pdb_lines)
        except Exception as e:
            raise Exception(f'  ERROR: error opening PDB url:␣
↪[{pdb_input_file_url}]; invalid PDB file path; error message:\n{e}')

        pdb_file_path = f'{output_path}/{pdb_input_file_name}.pdb'
        open(pdb_file_path,'w').write(pdb_text) # write PDB to a new file in␣
↪output path
        print(f'  NOTE: saving PDB file to output path: [{pdb_file_path}]')

    else:
        raise Exception(f'  ERROR: error opening PDB url or file; invalid PDB␣
↪4-char code, url, or file path')

    pdb = pdb_lines

    if len(pdb)==0: # check if PDB file is valid
        raise Exception(f'  ERROR: error reading PDB file; file is empty; check␣
↪PDB 4-char code, url, or file path')

    # find set of unique HETATM molecules
    try:
        hetatm_name_list = []
        hetatm_name_list_with_residue_number = []
        for line in pdb:
            if line[0:6].upper()=='HETATM':
                hetatm_name = f"{line[17:20].replace(' ','').upper()}-{line[21].
↪upper()}"
                hetatm_name_list.append(hetatm_name)
                hetatm_name_with_residue_number = f"{line[17:20].replace('␣
↪','').upper()}-{line[21].upper()} (#{str(int(line[22:26]))})" # structure:␣
↪RESIDUE_NAME-CHAIN_ID (#RESIDUE_NUMBER), i.e. IRE-A (#1101)
                hetatm_name_list_with_residue_number.
↪append(hetatm_name_with_residue_number)
        hetatm_name_list_with_residue_number =␣
↪list(set(hetatm_name_list_with_residue_number))
        hetatm_name_list = list(set(hetatm_name_list))

        hetatm_name_list_without_water = []
        water_count = 0
        for hetatm_name in hetatm_name_list_with_residue_number:
            if hetatm_name[0:3]!='HOH':
                hetatm_name_list_without_water.append(hetatm_name) # remove␣
↪water molecules from HETATM set
            elif hetatm_name[0:3]=='HOH':
                water_count += 1 # count number of water molecules
        hetatm_name_list_without_water = ', '.
↪join(hetatm_name_list_without_water)
        print(f'\n  NOTE: set of non-protein HETATM molecules:␣
↪[{hetatm_name_list_without_water}], with [{water_count}] water molecules')
    except Exception as e:
        raise Exception(f' ERROR: error finding set of non-protein HETATM␣
↪molecules; check PDB data structure; error message:\n{e}')

    return pdb_file_path,hetatm_name_list


# PROTEIN GRAPH NETWORK: GET LIGPLOT
import os
import numpy as np

'''
GET LIGPLOT
    - PARAM:
        - pdb_file_path = path to new imported .pdb file
        - hetatm_name = name of target HETATM residue, with structure␣
↪RESIDUE_NAME-CHAIN_ID, i.e. IRE-A
        - ligplot_program_path = path to LigPlot program
        - ligplot_prm_file_path = path to LigPlot .prm file
        - het_group_dictionary_file_path = path to Het Group Dictionary .cif␣
↪file
        - hbplus_params = 4 float parameters for hbplus thresholds
        - output_path = path to output folder
    - Runs LigPlot shell script, which saves outputs to an output folder
    - RETURN:
        - ligplot_hhb_file_path = ligplot.hhb file path (HHH bonds)
        - ligplot_nnb_file_path = ligplot.nnb file path (HPO bonds)
        - hbadd_bonds_file_path = hbadd.bonds file path (HC bonds)
'''
def␣
↪get_ligplot(pdb_file_path,hetatm_name,ligplot_program_path,ligplot_prm_file_path,het_group_
↪
    print(f'\n[running get_ligplot for [{hetatm_name}]]')

    # 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')
```

```
    if not os.path.exists(ligplot_program_path):
        raise Exception(f'  ERROR: cannot find LigPlot program path/folder:␣
↪[{ligplot_program_path}]')
    if not os.path.exists(ligplot_prm_file_path):
        raise Exception(f'  ERROR: cannot find ligplot.prm file:␣
↪[{ligplot_prm_file_path}]')
    if not os.path.exists(het_group_dictionary_file_path):
        print(f'  WARNING: cannot find Het Group Dictionary .cif file:␣
↪[{het_group_dictionary_file_path}]; running without Het Group Dictionary')
    if not os.path.exists(f'{ligplot_program_path}/hbadd'):
        raise Exception(f'  ERROR: cannot find hbadd program file in LigPlot␣
↪program path: [{ligplot_program_path}/hbadd]')
    if not os.path.exists(f'{ligplot_program_path}/hbplus'):
        raise Exception(f'  ERROR: cannot find hbplus program file in LigPlot␣
↪program path: [{ligplot_program_path}/hbplus]')
    if not os.path.exists(f'{ligplot_program_path}/ligplot'):
        raise Exception(f'  ERROR: cannot find ligplot program file in LigPlot␣
↪program path: [{ligplot_program_path}/ligplot]')

    try:
        hbplus_params = list(map(float,hbplus_params))
    except:
        raise Exception(f'  ERROR: error reading hbplus parameters:␣
↪[{hbplus_params}]; must be a list of 4 floats')
    if len(hbplus_params)!=4:
        raise Exception(f'  ERROR: error reading hbplus parameters:␣
↪[{hbplus_params}]; must be a list of 4 floats')

    if type(output_path)!=str or ' ' in output_path:
        raise Exception(f'  ERROR: error reading output path: [{output_path}];␣
↪must be a string without spaces')
    try:
        if not os.path.exists(output_path):
            os.makedirs(output_path)
            print(f'  NOTE: creating output path: [{output_path}]')
    except:
        raise Exception(f'  ERROR: error reading output path: [{output_path}]')

    ligplot_output_path = f'{output_path}/LigPlot-{hetatm_name}'
    try:
        if not os.path.exists(ligplot_output_path):
            os.makedirs(ligplot_output_path)
            print(f'  NOTE: creating LigPlot output path:␣
↪[{ligplot_output_path}]')
    except:
        raise Exception(f'  ERROR: error creating LigPLot output path:␣
↪[{ligplot_output_path}]')

    hetatm_name = str(hetatm_name)
    if hetatm_name[0:3]=='HOH': # ignore HOH water HETATM names
        raise Exception(f'  ERROR: unrecognized PDB HETATM: [{hetatm_name}];␣
↪cannot run LigPlot on individual water molecules')

    hbadd_input = []
    try:
        found_hetatm = False
        for line in pdb:
            if line[0:6].upper()=='HETATM' and (f"{line[17:20].replace(' ','').
↪upper()}-{line[21].upper()}")==hetatm_name:
                hetatm_number = int(line[22:26]) # find HETATM residue number␣
↪and chain id
                hetatm_chain_id = line[21].upper()
                hbadd_input.append(line)
                found_hetatm = True
    except Exception as e:
        raise Exception(f'  ERROR: error reading PDB lines; check PDB data␣
↪structure; PDB line and error message:\n{line}\n{e}')
    if found_hetatm==False:
        raise Exception(f'  ERROR: unrecognized PDB HETATM: [{hetatm_name}]')

    hbadd_input = ''.join(hbadd_input)
    open(f'{ligplot_output_path}/hbadd_input.pdb','w').write(hbadd_input) #␣
↪create modified .pdb for hbadd that only contains HETATM records for the␣
↪target residue

    # run LigPlot shell script (hbadd, hbplus x2, ligplot)
    try:
        print('\n            \n')
        run_state = os.system(f'''
        cd {ligplot_output_path}
        {ligplot_program_path}/hbadd {ligplot_output_path}/hbadd_input.pdb␣
↪{het_group_dictionary_file_path} -wkdir ./
        {ligplot_program_path}/hbplus {pdb_file_path} -f ./hbplus.rc -L -h␣
↪{hbplus_params[0]} -d {hbplus_params[1]} -wkdir ./
        {ligplot_program_path}/hbplus {pdb_file_path} -f ./hbplus.rc -L -h␣
↪{hbplus_params[2]} -d {hbplus_params[3]} -N -wkdir ./
        {ligplot_program_path}/ligplot {pdb_file_path} {str(hetatm_number)}␣
↪{str(hetatm_number)} {hetatm_chain_id} -prm {ligplot_prm_file_path} -wkdir ./
        ''')
        print('\n            \n')
    except Exception as e:
        raise Exception(f' ERROR: error running LigPlot; error message:\n{e}')
    if run_state!=0:
        raise Exception(' ERROR: error running LigPlot')
    print(f' NOTE: LigPlot run completed for [{hetatm_name}]]')

    # get ligplot.hhb, ligplot.nnb, hbadd.bonds file paths
    ligplot_hhb_file_path = f"{ligplot_output_path}/ligplot.hhb"
    if not os.path.exists(ligplot_hhb_file_path):
        print(f' WARNING: cannot find [{ligplot_hhb_file_path}]; possible␣
↪LigPlot error')

    ligplot_nnb_file_path = f"{ligplot_output_path}/ligplot.nnb"
    if not os.path.exists(ligplot_nnb_file_path):
        print(f' WARNING: cannot find [{ligplot_nnb_file_path}]; possible␣
↪LigPlot error')

    hbadd_bonds_file_path = f"{ligplot_output_path}/hbadd.bonds"
    if not os.path.exists(hbadd_bonds_file_path):
        print(f' WARNING: cannot find [{hbadd_bonds_file_path}]; possible␣
↪LigPlot error')

    return ligplot_hhb_file_path,ligplot_nnb_file_path,hbadd_bonds_file_path


# PROTEIN GRAPH NETWORK: FORMAT LIGPLOT
import os
import numpy as np

'''
FORMAT LIGPLOT
    - PARAM:
        - file_path = ligplot.hhb, ligplot.nnb, or hbadd.bonds file path
        - hetatm_name = name of target HETATM residue, with structure␣
↪RESIDUE_NAME-CHAIN_ID, i.e. IRE-A
    - Formats LigPlot output files ligplot.hhb, ligplot.nnb, or hbadd.bonds to␣
↪2D array with the following column index assignments:
        - 0 - a_residue_name (3-letter abbreviation)
        - 1 - a_residue_number
        - 2 - a_chain_id (A, B, C, etc.)
        - 3 - a_atom_name (unique atom name)
        - 4 - bond_type (HCS, HCD, HCT, HHH, HPO)
        - 5 - b_residue_name (3-letter abbreviation)
        - 6 - b_residue_number
        - 7 - b_chain_id (A, B, C, etc.)
        - 8 - b_atom_name (unique atom name)
    - RETURN:
        - ligand_bonds = 2D array with ligand bond information on each row
```

```
'''
def format_ligplot(file_path,hetatm_name):
    print(f'\n[running format_ligplot for [{file_path}] for [{hetatm_name}]]')

    # check inputs
    if not os.path.exists(file_path):
        raise Exception(f' ERROR: cannot find [{file_path}] file')
    file_name = file_path.rsplit('/', 1)[-1] # get file name from end of file␣
↪path
    if file_name not in ['ligplot.hhb','ligplot.nnb','hbadd.bonds']:
        raise Exception(f' ERROR: unrecognized file: [{file_path}]; can only␣
↪format [/ligplot.hhb], [/ligplot.nnb], or [/hbadd.bonds] files')

    # parse LigPlot bond information
    with open(file_path,'r') as ligplot_file:
        ligplot_lines = ligplot_file.readlines()
    nlig = len(ligplot_lines)

    hetatm_name = str(hetatm_name)
    ligand_bonds = []

    if file_name in ['ligplot.hhb','ligplot.nnb']:
        if file_name=='ligplot.hhb':
            bond_type = 'HHH'
        elif file_name=='ligplot.nnb':
            bond_type = 'HPO'

        for line in ligplot_lines:
            line = line.rstrip() # removes line breaks
            if (not str.isdigit(line[6:10].replace(' ',''))) and (not str.
↪isdigit(line[27:31].replace(' ',''))): # check if data structure is valid
                print(f' WARNING: ignoring [/{file_name}] line: [{line}];␣
↪unrecognized data structure')
                continue
            try:
                a_residue_name = line[0:3].replace(' ','').upper()
                a_residue_number = int(line[6:10])
                a_chain_id = line[4].upper()
                a_atom_name = line[12:16].replace(' ','').upper()

                b_residue_name = line[21:24].replace(' ','').upper()
                b_residue_number = int(line[27:31])
                b_chain_id = line[25].upper()
                b_atom_name = line[33:37].replace(' ','').upper()
            except:
                print(f' WARNING: error parsing [/{file_name}] line: [{line}];␣
↪unrecognized data structure')
```

```
                continue

            if f'{a_residue_name}-{a_chain_id}'==hetatm_name or␣
↪f'{b_residue_name}-{b_chain_id}'==hetatm_name: # check if bond is for target␣
↪residue
                ligand_bonds.
↪append([a_residue_name,a_residue_number,a_chain_id,a_atom_name,bond_type,b_residue_name,b_r
            else:
                print(f'  WARNING: ignoring [/{file_name}] line: [{line}]; does␣
↪not include target residue')
                continue

    elif file_name=='hbadd.bonds':
        for line in ligplot_lines:
            line = line.rstrip() # removes line breaks
            if (not str.isdigit(line[15:19].replace(' ',''))) and (not str.
↪isdigit(line[42:46].replace(' ',''))): # check if data structure is valid
                print(f'  WARNING: ignoring [/{file_name}] line: [{line}];␣
↪unrecognized data structure')
                continue
            try:
                a_residue_name = line[11:14].replace(' ','').upper()
                a_residue_number = int(line[15:19])
                a_chain_id = line[21].upper()
                a_atom_name = line[6:10].replace(' ','').upper()

                b_residue_name = line[38:41].replace(' ','').upper()
                b_residue_number = int(line[42:46])
                b_chain_id = line[48].upper()
                b_atom_name = line[33:37].replace(' ','').upper()

                ab_bond_type = str(line[52:56]).upper()
            except:
                print(f'  WARNING: error parsing [/{file_name}] line: [{line}];␣
↪unrecognized data structure')
                continue

            if ab_bond_type=='SING':
                bond_type = 'HCS'
            elif ab_bond_type=='DOUB':
                bond_type = 'HCD'
            elif ab_bond_type=='TRIP':
                bond_type = 'HCT'
            else:
                print(f'  WARNING: error parsing [/hbadd.bonds] line: [{line}];␣
↪unrecognized bond type')
```

```
                continue

            if f'{a_residue_name}-{a_chain_id}'==hetatm_name or␣
↪f'{b_residue_name}-{b_chain_id}'==hetatm_name: # check if bond is for target␣
↪residue
                ligand_bonds.
↪append([a_residue_name,a_residue_number,a_chain_id,a_atom_name,bond_type,b_residue_name,b_r
            else:
                print(f' WARNING: ignoring [/{file_name}] line: [{line}]; does␣
↪not include target residue')
                continue

    ligand_bonds = np.array(ligand_bonds,dtype=object) # convert 2D list to 2D␣
↪array

    return ligand_bonds
```

[ ]: