Seminar 7: Calculating energy

Biophysics

Course 2022-2023

Alberto Meseguer
Irene Acero

Calculating energy of a protein-protein interaction

To calculate energies we will follow the following formula:

$$\Delta G^{A-B} = \Delta G_{elect}{}^{A-B} + \Delta G_{vdw}{}^{A-B} + \Delta G_{Solv}{}^{A-B} - \Delta G_{Solv}{}^{A} - \Delta G_{Solv}{}^{B}$$

Calculating energy of a protein-protein interaction

To calculate energies we will follow the following formula:

$$\Delta G^{A-B} = \Delta G_{elect}{}^{A-B} + \Delta G_{vdw}{}^{A-B} + \Delta G_{Solv}{}^{A-B} - \Delta G_{Solv}{}^{A} - \Delta G_{Solv}{}^{B}$$

Electrostatics of the interaction

Van Der Waals of the interaction

Solvation of the interaction

Solvation of the individual subunits

Calculating energy of a protein-protein interaction

To calculate energies we will follow the following formula:

$$\Delta G^{A-B} = \Delta G_{elect}^{A-B} + \Delta G_{vdw}^{A-B} + \Delta G_{Solv}^{A-B} - \Delta G_{Solv}^{A} - \Delta G_{Solv}^{B}$$

We will work with partial charges (charges that are not integers).
Therefore, H-bonds will be included into electrostatic interactions.

Electrostatics of the interaction

Van Der Waals of the interaction

Solvation of the interaction

Solvation of the individual subunits

Starting files we give to you

To calculate the energies of the system you can start from the jupyter notebook and complementary files we give you

In [1]:
```
import argparse
import sys
import os
import math

from Bio.PDB.PDBParser import PDBParser
from Bio.PDB.NACCESS import NACCESS_atomic
from Bio.PDB.NeighborSearch import NeighborSearch
from Bio.PDB.PDBIO import PDBIO, Select
```

This are functions that you will need to import the parameters for VanderWaals or the residue library:

In [3]:
```
class ResiduesDataLib():
    def __init__(self, fname):
        self.residue_data = {}
        try:
            fh = open(fname, "r")
        except OSError:
            print("#ERROR while loading library file (", fname, ")")
            sys.exit(2)
        for line in fh:
            if line[0] == '#':
                continue
            data = line.split()
            r = Residue(data)
            self.residue_data[r.id] = r
        self.nres = len(self.residue_data)

    def get_params(self, resid, atid):
        atom_id = resid + ':' + atid
        if atom_id in self.residue_data:
            return self.residue_data[atom_id]
```

Starting files we give to you

In the complementary files we are using to set the parameters of the atoms you will see that we are assuming partial charges

#Res	Atom	Type	Charge
ALA	N	N	-0.4157
ALA	H	HN	0.2719
ALA	CA	C	0.0337
ALA	HA	H	0.0823
ALA	CB	C	-0.1825
ALA	HB1	H	0.0603
ALA	HB2	H	0.0603
ALA	HB3	H	0.0603
ALA	C	C	0.5973
ALA	O	O	-0.5679

Starting files we give to you

In the complementary files we are using to set the parameters of the atoms you will see that we are assuming partial charges

#Res	Atom	Type	Charge
ALA	N	N	-0.4157
ALA	H	HN	0.2719
ALA	CA	C	0.0337
ALA	HA	H	0.0823
ALA	CB	C	-0.1825
ALA	HB1	H	0.0603
ALA	HB2	H	0.0603
ALA	HB3	H	0.0603
ALA	C	C	0.5973
ALA	O	O	-0.5679

Electronegative atoms such as Nitrogen or Oxygen have partially negative charges

↓

Since they are more electronegative, they pull harder from electrons making them sligtly negative

↓

As a consequence of this, atoms bonded to Nitrogen or Oxygen are slightly positive

Understand the code you have to use

Some of the parts of the code that we will use are in Josep Lluis' GitHub. There are links to his GitHub in the document of the project.

README.md

BioPhysics

Source codes and data for BioPhysics course on Bioinformatics

forcefield.py

Module for forcefield parameters management

data/vdwprm

Simple vdw parameters (based on AMBER ff using AutoDock compatible atom types)

External dependencies Bio.PDB.PDBParser (Biopython)

Understand the code you have to use

The add_atom_parameters will provide parameters to each of the atoms so that we can use them to calculate electrostatic or Van Der Waals energies.

```
def add_atom_parameters(st, res_lib, ff_params):
    ''' Adds parameters from libraries to atom .xtra field
        For not recognized atoms, issues a warning and put default parameters
    '''
    for at in st.get_atoms():
        resname = at.get_parent().get_resname()
        params = res_lib.get_params(resname, at.id)
        if not params:
            print(at)
            #print("WARNING: residue/atom pair not in library ("+atom_id(at) + ')')
            at.xtra['atom_type'] = at.element
            at.xtra['charge'] = 0
        else:
            at.xtra['atom_type'] = params.at_type
            at.xtra['charge'] = params.charge
        at.xtra['vdw'] = ff_params.at_types[at.xtra['atom_type']]
```

```
add_atom_parameters(st, residue_library, ff_params)
```

Understand the code you have to use

Use the elec_int function to calculate the electrostatic interactions between two atoms at a distance r

```
def elec_int(at1, at2, r):
    '''Electrostatic interaction energy between two atoms at r distance'''
    return 332.16 * at1.xtra['charge'] * at2.xtra['charge'] / MH_diel(r) / r
```

This function represents the following formula:

Electrostatic interaction:

$$E_{elec\,ij} = 332.16\,\frac{q_i q_j}{\varepsilon r_{ij}}$$

Understand the code you have to use

Use the elec_int function to calculate the electrostatic interactions between two atoms at a distance r

```
def elec_int(at1, at2, r):
    '''Electrostatic interaction energy between two atoms at r distance'''
    return 332.16 * at1.xtra['charge'] * at2.xtra['charge'] / MH_diel(r) / r
```

See that the function to calculate the dielectric constant is included in the formula

This function represents the following formula:

Electrostatic interaction:

$$E_{elec\,ij} = 332.16\,\frac{q_i q_j}{\varepsilon r_{ij}}$$

Understand the code you have to use

Use the vdw_int function to calculate the Van Der Waals interactions between two atoms at a distance r

```
def vdw_int(at1, at2, r):
    '''Vdw interaction energy between two atoms'''
    eps12 = math.sqrt(at1.xtra['vdw'].eps * at2.xtra['vdw'].eps)
    sig12_2 = at1.xtra['vdw'].sig * at2.xtra['vdw'].sig
    return 4 * eps12 * (sig12_2**6/r**12 - sig12_2**3/r**6)
```

This function represents the following formula:

Vdw interaction:

$$Evdw_{ij} = 4\epsilon_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right)$$

Understand the code you have to use

Use the calc_solvation function to calculate the energy for one residue (res)

```
def calc_solvation(st, res):
    '''Solvation energy based on ASA'''
    solv = 0.
    solv_ala = 0.
    for at in res.get_atoms():
        if 'EXP_NACCESS' not in at.xtra:
            continue
        s = float(at.xtra['EXP_NACCESS'])* at.xtra['vdw'].fsrf
        solv += s
    return solv
```

This function represents the following formula:

$$\Delta G = \sum_{AtTypes} \sigma_i \, ASA_i$$

Understand the code you have to use

Use the calc_int_energies function to calculate the electrostatic and Van Der Waals energies for one residue (res) inside one structure (st)

```
def calc_int_energies(st, res):
    '''Returns interaction energies (residue against other chains)
        for all atoms
    '''
    elec = 0.
    vdw = 0.

    for at1 in res.get_atoms():
        for at2 in st.get_atoms():
        # skip same chain atom pairs
            if at2.get_parent().get_parent() != res.get_parent():
                r = at1 - at2
                e = elec_int(at1, at2, r)
                elec += e
                e = vdw_int(at1, at2, r)
                vdw += e

    return elec,vdw
```

Use Josep Lluis' code

If you use the functions we showed you, you can calculate the different components of the following formula

$$\Delta G^{A-B} = \Delta G_{elect}^{A-B} + \Delta G_{vdw}^{A-B} + \Delta G_{Solv}^{A-B} - \Delta G_{Solv}^{A} - \Delta G_{Solv}^{B}$$

Electrostatics of the interaction

Van Der Waals of the interaction

Solvation of the interaction

Solvation of the individual subunits

How to implement the alanine scanning

Use the code you made to calculate the energies of the system when replacing each residue of the interface by an alanine

How to implement the alanine scanning

Use the code you made to calculate the energies of the system when replacing each residue of the interface by an alanine

First tip:

```
#Possible Atom names that correspond to Ala atoms"
ala_atoms = {'N', 'H', 'CA', 'HA', 'C', 'O', 'CB', 'HB', 'HB1', 'HB2', 'HB3', 'HA1', 'HA2', 'HA3'}
```

How to implement the alanine scanning

Use the code you made to calculate the energies of the system when replacing each residue of the interface by an alanine

First tip:

```
#Possible Atom names that correspond to Ala atoms"
ala_atoms = {'N', 'H', 'CA', 'HA', 'C', 'O', 'CB', 'HB', 'HB1', 'HB2', 'HB3', 'HA1', 'HA2', 'HA3'}
```

Second tip:

What are the possible atom names for other amino acids??

How to implement the alanine scanning

Use the code you made to calculate the energies of the system when replacing each residue of the interface by an alanine

First tip:

```
#Possible Atom names that correspond to Ala atoms"
ala_atoms = {'N', 'H', 'CA', 'HA', 'C', 'O', 'CB', 'HB', 'HB1', 'HB2', 'HB3', 'HA1', 'HA2', 'HA3'}
```

Second tip:
What are the possible atom names for other amino acids??

Third tip:
What atoms are present in alanine but not in the other amino acids??