Факультет биоинженерии и биоинформатики,
Московский государственный университет имени М. В. Ломоносова

Python и 3D биоинформатика

Биоинформатика, 4 курс ФББ МГУ, осенний семестр
Злобин А. С., alexander.zlobin@fbb.msu.ru

Зачем нам Python?

Не всегда все можно и нужно сделать вручную. Например

- Найти атом с самым высоким B-фактором. Не просматривать же весь PDB файл глазами?
- Посчитать число водородных связей у каждого остатка в системе. Крайне легко запутаться и/или что-то пропустить
- Сравнить число водородных связей с окружением и площадь контакта у всех фрагментов, кристаллизованных с протеазой Sars-Cov-2
- Посчитать, как меняется паттерн контактов определенного участка белка в динамике (представленной как набор из ~1000 PDB-файлов)

Во всех этих случаях нам поможет

- Умение программировать на Python
- Знание пакетов, которые могут избавить от разработки собственных велосипедов в большом количестве случаев

Пакет ProDy

Установка: `pip install prody` (на kodomo уже стоит)

Туториалы от разработчиков Мануал от разработчиков

Мы будем знакомиться с функциями постепенно, по мере надобности для конкретных практикумов.

Допбаллы: разберите один из инструментов на Python для работы со структурными данными помимо ProDy и напишите по нему туториал на русском языке, понятный вашим текущим и будущим коллегам.
Что можно взять:

- PyMol API
- MDAnalysis
- mdtraj

Можно найти что-то свое, но убедитесь, что
а) проект в настоящее время продолжает развиваться (есть коммиты за этот год/свежие версии)
б) решает релевантные для нашего курса задачи

Загрузка информации о белке

```
[1]: import prody as pd
```

```
[2]: P1 = pd.parsePDB('6xmk') # Так он скачает файл самостоятельно
```

```
[3]: P2 = pd.parsePDB('/home/domain/zlobin/Downloads/2ifr.pdb') # А так вчитает файл с диска
```

А что внутри?

```
[6]: P1
```

```
[6]: <AtomGroup: 6xmk (4862 atoms)>
```

```
[7]: P2
```

```
[7]: <AtomGroup: 2ifr (1713 atoms)>
```

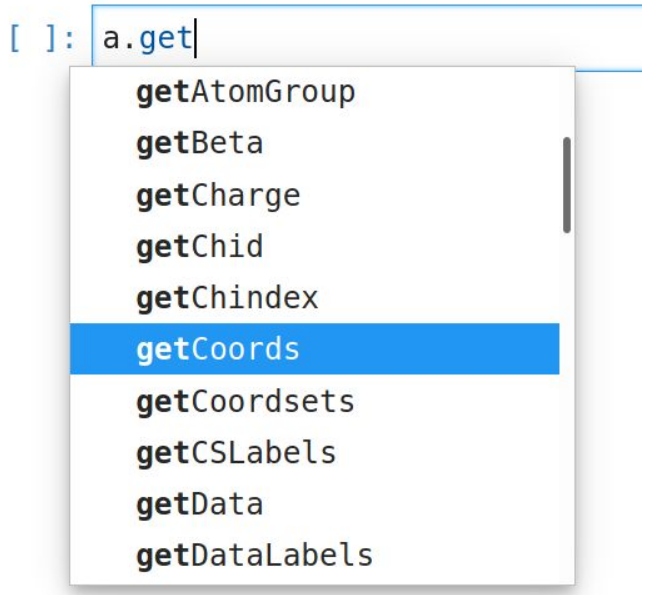
AtomGroup – это класс для набора атомов. У атомов есть различные атрибуты.

Можно итерировать

```
[8]: for a in P1:
         print(a)
```

```
Atom N (index 0)
Atom CA (index 1)
Atom C (index 2)
Atom O (index 3)
Atom CB (index 4)
Atom CG (index 5)
Atom CD (index 6)
Atom NE (index 7)
Atom CZ (index 8)
Atom NH1 (index 9)
Atom NH2 (index 10)
Atom N (index 11)
Atom CA (index 12)
Atom C (index 13)
Atom O (index 14)
Atom CB (index 15)
Atom CG (index 16)
```

Атом



Jupyter notebook и TAB =
не нужно ничего запоминать

Можно вытащить всю обычную информацию об атоме, содержащуюся в PDB:

getBeta()	B-фактор
getChid()	Имя цепи
getCoords()	Координаты
getElement()	Химический элемент
getIndex()	Индекс (нумерация с 0)
getMass()	Масса
getName()	Имя
getOccupancy()	Населенность
getResindex()	Индекс остатка
getResname()	Имя остатка
getResnum()	Номер остатка

Группы атомов

ProDy имеет свой синтаксис выделения групп атомов, местами похожий, местами отличающийся от такового в PyMol. В целом возможностей у ProDy сильно больше. Не нужно пытаться их все запомнить, лучше держать ссылку на документацию под рукой.

Пример различий:

	PyMol	**ProDy**
Номер остатка	resi	resnum
Перечисления	resi 1-10+15-20+33+44+55	resnum 1 to 20 15 to 20 33 44 55

```
[3]: my_selection = P1.select('resname QYS and chain A')

[4]: my_selection

[4]: <Selection: 'resname QYS and chain A' from 6xmk (31 atoms)>
```

О группах атомов можно получать ту же информацию, что и об одном. В этом случае название метода имеет в конце -s: не **getName()**, а **getNames()**

```
[5]: my_selection.getBetas()
```

```
[5]: array([60.55, 47.39, 47.33, 46.5 , 33.97, 33.47, 24.92, 24.93, 29.23,
            33.3 , 32.21, 31.81, 25.11, 23.01, 24.  , 26.67, 30.13, 27.99,
            26.74, 37.37, 45.42, 70.72, 66.91, 27.04, 24.34, 27.35, 31.69,
            26.97, 29.07, 33.94, 27.42])
```

```
[6]: type(my_selection.getBetas())
```

```
[6]: numpy.ndarray
```

Используя numpy можно получить значение среднего B-фактора атомов остатка в одну строчку:

```
[7]: import numpy as np
     np.mean(my_selection.getBetas())
```

```
[7]: 34.75806451612904
```

Методы pd.calcXXX()

Можно легко вычислить положение центра координат и центра масс любого объекта (выделения, самого белка, набора атомов, остатка...):

```
[8]: geometric_center = pd.calcCenter(my_selection)
     geometric_center
```

```
[8]: array([10.33380645, 23.71848387, 25.78812903])
```

```
[9]: mass_center = pd.calcCenter(my_selection, weights=my_selection.getMasses())
     mass_center
```

```
[9]: array([10.25742246, 23.12263636, 25.91179679])
```

Можно рассчитать расстояние в ангстремах между двумя точками:

```
[10]: protein_mass_center = pd.calcCenter(P1, weights=P1.getMasses())
      distance = pd.calcDistance(mass_center, protein_mass_center)
      distance
```

```
[10]: 23.20693090772202
```

Не забывайте про возможности ? и TAB!

```
[68]: ?pd.calcCenter
```

```
Signature: pd.calcCenter(atoms, weights=None)
Docstring:
Returns geometric center of *atoms*.  If *weights* is given it must
be a flat array with length equal to number of atoms.  Mass center of
atoms can be calculated by setting weights equal to atom masses, i.e.
``weights=atoms.getMasses()``.
File:      /usr/local/lib/python3.6/dist-packages/prody/measure/measure.
py
Type:      function
```

```
[ ]: pd.calc
```

```
calcPercentIdentities
calcPerturbResponse
calcPhi
calcProjection
calcPsi
calcRankorder
calcRMSD
calcRMSF
calcShannonEntropy
calcSignatureCollectivity
```

Итерирование по остаткам и первое задание

Для итерирования по остаткам нужен метод **iterResidues()**

[26]:
```
for residue in P1.iterResidues():
    print(residue.getChain(),
          residue.getResname(),
          residue.getResnum())
```

```
Chain A ARG 4
Chain A LYS 5
Chain A MET 6
Chain A ALA 7
Chain A PHE 8
Chain A PRO 9
```

Тут я итерирую по обеим цепям

[28]:
```
for residue in P1['B'].iterResidues():
    print(residue.getChain(),
          residue.getResname(),
          residue.getResnum())
```

```
Chain B SER 1
Chain B GLY 2
Chain B PHE 3
Chain B ARG 4
Chain B LYS 5
Chain B MET 6
Chain B ALA 7
```

Тут я итерирую только по цепи B

[29]:
```
mean_betas = []

for residue in P1.iterResidues():
    mean_beta = np.mean(residue.getBetas())
    mean_betas.append([residue, mean_beta])
```

Остаток с наименьшим средним B-фактором – триптофан 31 цепи B

[31]:
```
sorted(mean_betas, key=lambda x:x[1])[0]
```

[31]:
```
[<Residue: TRP 31 from Chain B from 6xmk (14 atoms)>, 19.382857142857144]
```

```
[29]: mean_betas = []

      for residue in P1.iterResidues():
          mean_beta = np.mean(residue.getBetas())
          mean_betas.append([residue, mean_beta])
```

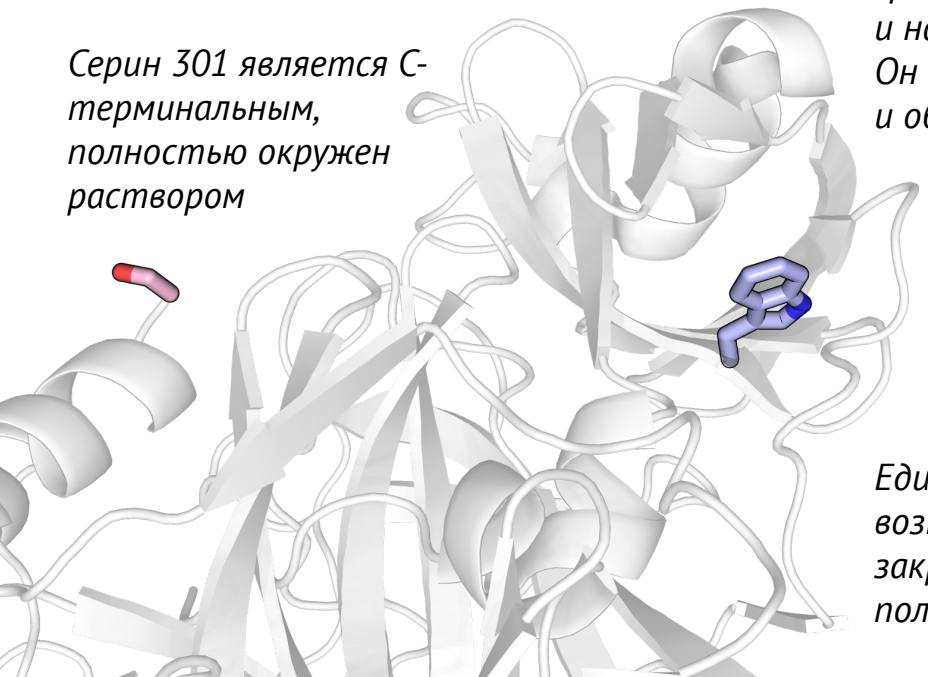
Остаток с наименьшим средним B-фактором – триптофан 31 цепи B

```
[31]: sorted(mean_betas, key=lambda x:x[1])[0]

[31]: [<Residue: TRP 31 from Chain B from 6xmk (14 atoms)>, 19.382857142857144]
```

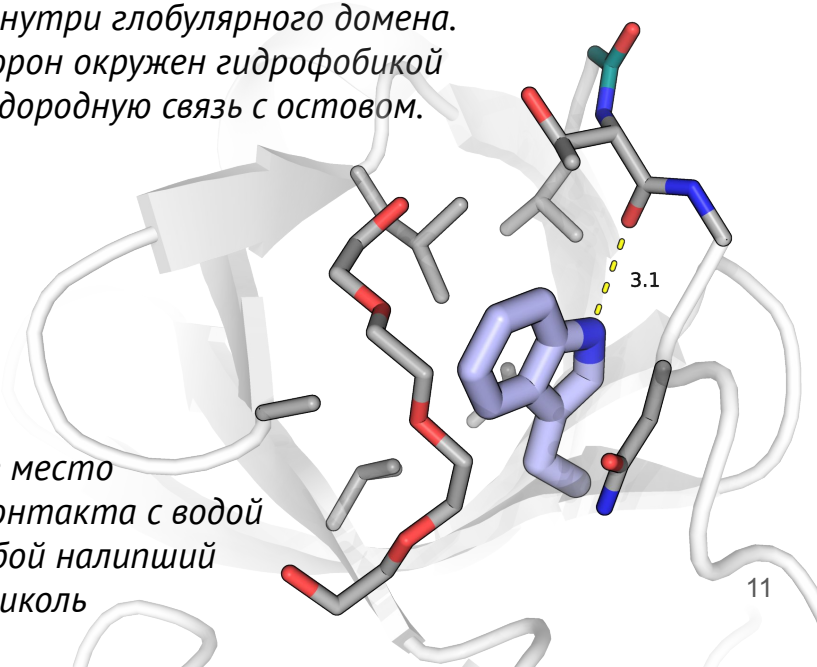
Остаток с наибольшим средним B-фактором – серин 301 цепи A

```
[32]: sorted(mean_betas, key=lambda x:x[1], reverse=True)[0]

[32]: [<Residue: SER 301 from Chain A from 6xmk (6 atoms)>, 69.35333333333334]
```

Серин 301 является C-терминальным, полностью окружен раствором



Триптофан 31 входит в бета-слой и находится внутри глобулярного домена. Он со всех сторон окружен гидрофобикой и образует водородную связь с остовом.

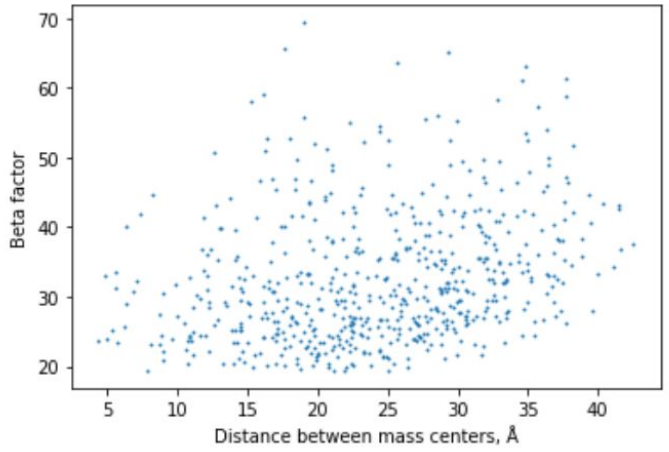


Единственное место возможного контакта с водой закрывает собой налипший полиэтиленгликоль

Второе задание

[46]:
```
protein_mass_center = pd.calcCenter(P1, weights=P1.getMasses())

distances = []
betas = []

for residue in P1.iterResidues():
    if 'CA' in residue.getNames(): # Хочу рассматривать только аминокислоты белка
        beta = np.mean(residue.getBetas())
        betas.append(beta)

        mass_center = pd.calcCenter(residue, weights=residue.getMasses())
        d = pd.calcDistance(protein_mass_center, mass_center)
        distances.append(d)
```

[47]:
```
fig,ax = plt.subplots()
ax.scatter(distances,betas,s=1)
ax.set_xlabel('Distance between mass centers, Å')
ax.set_ylabel('Beta factor')
fig.show()
```



Всякое разное

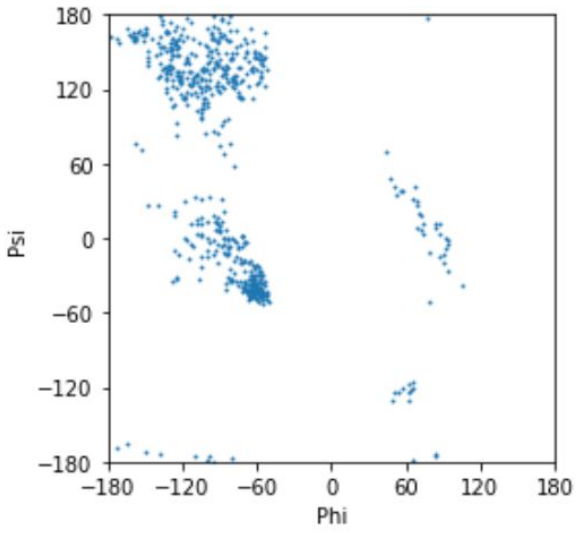
Карта Рамачандрана

[37]:
```
phis = []
psis = []
for residue in P1.iterResidues():
    try:
        phi = pd.calcPhi(residue)
        psi = pd.calcPsi(residue)
        phis.append(phi)
        psis.append(psi)
    except ValueError as e: # We skip all terminal residues
        print(e)
```

```
ARG 4 is a terminal residue
SER 301 and SER 1 does not seem to be connected
SER 1 and SER 301 does not seem to be connected
QYS 401 is not an amino acid
QYS 401 does not have N atom
QYS 401 is not an amino acid
QYS 401 is not an amino acid
PG4 402 is not an amino acid
HOH 501 is not an amino acid
HOH 502 is not an amino acid
HOH 503 is not an amino acid
```

Вот так, например, можно построить карту Рамачандрана для любого белка

[44]:
```
fig,ax = plt.subplots()
ax.scatter(phis,psis,s=1)
ax.set_aspect('equal')
ax.set_xlabel('Phi')
ax.set_ylabel('Psi')
ax.set_xlim(-180,180)
ax.set_ylim(-180,180)
ax.set_xticks(np.arange(-180,181,60))
ax.set_yticks(np.arange(-180,181,60))
fig.show()
```



Солевые мостики

Условно, конечно, заряженными могут быть еще и концы цепей и гистидин, а некоторые аспартаты и глутаматы могут нести протон.

[99]:
```
positive = P1.select('(resname LYS and name NZ) or (resname ARG and name CZ)')
negative = P1.select('(resname ASP and name CG) or (resname GLU and name CD)')
```

[102]:
```
for atom1 in positive:
    for atom2 in negative:
        if pd.calcDistance(atom1, atom2) <= 5:
            res1 = (atom1.getChid(),
                    atom1.getResname(),
                    atom1.getResnum())
            res2 = (atom2.getChid(),
                    atom2.getResname(),
                    atom2.getResnum())
            print(res1, res2)
```

```
('A', 'LYS', 5) ('A', 'GLU', 288)
('A', 'ARG', 40) ('A', 'ASP', 187)
('A', 'ARG', 60) ('A', 'ASP', 56)
('A', 'LYS', 90) ('A', 'ASP', 34)
('A', 'ARG', 105) ('A', 'ASP', 176)
('A', 'ARG', 131) ('A', 'ASP', 197)
('A', 'ARG', 131) ('A', 'ASP', 289)
('A', 'ARG', 131) ('A', 'GLU', 290)
('B', 'ARG', 40) ('B', 'ASP', 187)
('B', 'LYS', 90) ('B', 'ASP', 34)
('B', 'ARG', 105) ('B', 'ASP', 176)
('B', 'ARG', 131) ('B', 'ASP', 197)
('B', 'ARG', 131) ('B', 'ASP', 289)
('B', 'ARG', 131) ('B', 'GLU', 290)
```