PyRosetta集群安装与使用简介

分享人：杨皓

2022.11.4

上海科技大学 ShanghaiTech University | 高性能计算

SIAIS ShanghaiTech University 免疫化学研究所 Shanghai Institute for Advanced Immunochemical Studies (SIAIS)

- PyRosetta简介
- PyRosetta集群安装
- PyRosetta使用简介
- 案例展示（Fastrelax，全原子/粗粒度蛋白对接）

- PyRosetta简介
- PyRosetta集群安装
- PyRosetta使用简介
- 案例展示（全原子/粗粒度蛋白对接）

PyRosetta简介

蛋白质结构的计算建模和分析

Rosetta使用方式

- Command Line Interface（命令行交互）
- PyRosetta
- PyRosetta Toolkit
- RosettaScripts
- GUIs（图形用户界面）
- Servers（服务器）

Rosetta协议（Protocols）

RosettaAbinitio	De novo protein structure prediction.
RosettaDesign	Identifies low free energy sequences for target protein backbones.
Rosetta Design PyMol Plugin	PyMOL plugin A user-friendly interface for submitting Protein Design simulations using RosettaDesign.
RosettaDock	Predicts the structure of a protein-protein complex from the individual structures of the monomer components.
RosettaAntibody*	Predicts antibody Fv region structures. See PubMed 24519881 and 19062174.
RosettaFragments	Generates fragment libraries for use by Rosetta ab initio in building protein structures.
RosettaNMR	Incorporates NMR data into the basic Rosetta protocol to accelerate the process of NMR structure prediction
RosettaDNA	For the design of proteins that interact with specified DNA sequences.
RosettaRNA	Fragment assembly of RNA.
RosettaLigand	Small molecule - protein docking
RosettaSymmetry	Enforcing symmetry in Rosetta
RosettaEnzdes	Enzyme design
RosettaMembrane	Membrane protein ab initio modeling
RosettaDDG*	Estimating the impact of sequence changes on protein stability
RosettaScripts	An xml-based scripting language for control of modeling trajectories. Supports all major Rosetta functionalities
RosettaSnugDock*	Enables docking an antibody Fv region to an antigen and allows backbone flexibility in the paratope.
RosettaMultigraft	Performs matching, backbone, and side chain grafting of functional motifs onto scaffold proteins.
Rosetta FlexPepDock	Peptide-protein docking
Rosetta ERRASER	Remodeling RNA crystallographic models with electron density constraint.
RosettaVIP	Stabilize a protein by identifying and filling voids
RosettaMatdes	Materials design

- PyRosetta简介
- PyRosetta集群安装
- PyRosetta使用简介
- 案例展示（全原子/粗粒度蛋白对接）

PyRosetta集群安装

- Rosetta可以通过源文件安装，也可以通过conda安装
- conda的安装可参考高性能计算中心常见问题（https://it.shanghaitech.edu.cn/8852/list.htm）

（1）conda的安装简要说明：

1.下载conda安装包

```
wget https://repo.anaconda.com/archive/Anaconda3-2022.05-Linux-x86_64.sh
bash Anaconda3-2022.05-Linux-x86_65.sh
```

2.首次安装需要输入

```
conda init
```

3.如果提示 command not found，需要在~/.bashrc中加入conda的环境变量(CONDA_PATH需要换成你自己安装的位置)

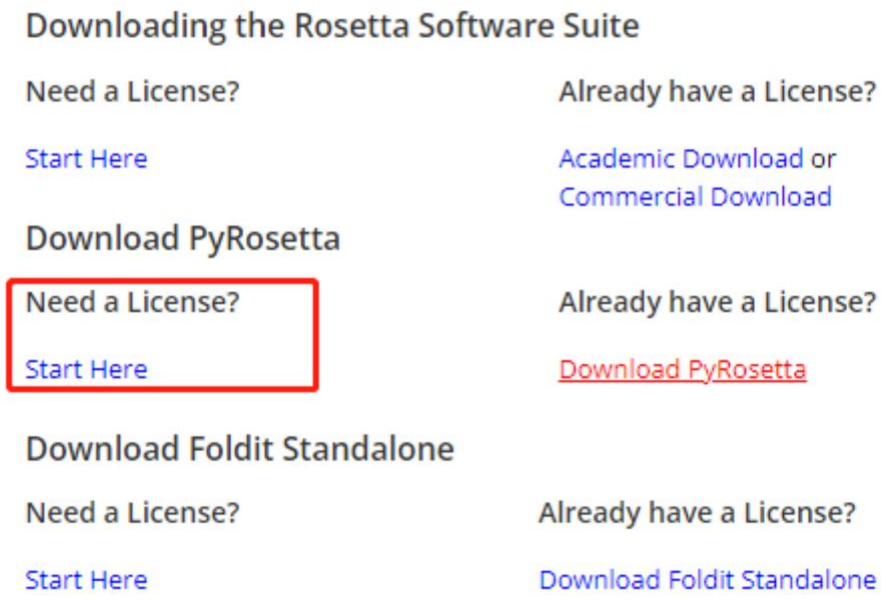
```
export PATH=CONDA_PATH/anaconda3/bin:$PATH
```

4.然后重新执行 conda init，然后按照提示重新打开界面。这样在命令行前面就会多出(base)的环境 提示

PyRosetta集群安装

（2）在 Rosetta 上申请 license

https://www.rosettacommons.org/software/license-and-download



申请成功后，会获得用户名和密码

（3）使用上一步获得的用户名和密码，加入到channel中

(在~/.condarc中写入以下内容)

```
channels:
  - https://USERNAME:PASSWORD@conda.graylab.jhu.edu
  - defaults
```

其中的USERNAME和PASSWORD需要替换成你所获得的用户名和密码

（4）创建一个新的conda虚拟环境，（一般来说，一个项目需要建立一个虚拟环境，避免环境带 来的兼容问题）

```
conda create -n ENV_NAME
```

这里的ENV_NAME可以替换成你设置的环境名称。如下创建了一个名为pyrosetta的环境，并且指 定python版本为3.8（接下页）

```
(base) [        @hpc-login ~]$ conda create -n pyrosetta python=3.8
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##
```

PyRosetta集群安装

（接上页）创建环境后需要激活环境

```
conda activate ENV_NAME
```

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate pyrosetta
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) [      hpc-login ~]$ conda activate pyrosetta
(pyrosetta) [      hpc-login ~]$
```

（5）直接安装pyrosetta

```
conda install pyrosetta
```

（6）安装完成后可以使用conda list命令查看安装的包

（7）Pyrosetta的教程可参考PyRosetta – Tutorials（https://www.pyrosetta.org/documentation/tutorials），里面有关于蛋白质折叠，优化，对接的教程，但是注意，其格式为jupyter（.ipynb），在集群上运行时，需要变为python脚本（.py）并提交PBS进行计算。

验证pyrosetta是否安装成功

```
import pyrosetta; pyrosetta.init()
```

```
(pyrosetta) [      node50 setup]$ python
Python 3.8.13 | packaged by conda-forge | (default, Mar 25 2022, 06:04:10)
[GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pyrosetta; pyrosetta.init()
PyRosetta-4 2022 [Rosetta PyRosetta4.Release.python38.linux 2022.26+release.cd16
be2910907fd8044a25e8aaf705b9474af667 2022-06-29T22:22:46] retrieved from: http:/
/www.pyrosetta.org
(C) Copyright Rosetta Commons Member Institutions. Created in JHU by Sergey Lysk
ov and PyRosetta Team.
core.init: Checking for fconfig files in pwd and ./rosetta/flags
core.init: Rosetta version: PyRosetta4.Release.python38.linux r323 2022.26+relea
se.cd16be2 cd16be2910907fd8044a25e8aaf705b9474af667 http://www.pyrosetta.org 202
2-06-29T22:22:46
core.init: command: PyRosetta -ex1 -ex2aro -database
software/PyRosetta4.Release.python38.linux.release-323/setup/pyrosetta/database
basic.random.init_random_generator: 'RNG device' seed mode, using '/dev/urandom'
, seed=1599812435 seed_offset=0 real_seed=1599812435
basic.random.init_random_generator: RandomGenerator:init: Normal mode, seed=1599
812435 RG_type=mt19937
>>>
```

PyRosetta集群安装

使用安装包安装

- 首先在https://graylab.jhu.edu/download/PyRosetta4/archive/release/上找到对应python版本的 pyrosetta，使用wget下载，如

```
wget
https://graylab.jhu.edu/download/PyRosetta4/archive/release/PyRosetta4.Release.python3
8.linux/PyRosetta4.Release.python38.linux.release-323.tar.bz2
```

- 解压（下面的version要改成你自己下载的版本）

```
tar -vjxf PyRosetta-<version>.tar.bz2
```

- 然后在一个conda的新环境中安装

```
cd PyRosetta-<version>/setup && python setup.py install
```

- PyRosetta简介
- PyRosetta集群安装
- PyRosetta使用简介
- 案例展示（全原子/粗粒度蛋白对接）

PyRosetta使用简介

基本用法

- PDB文件可在 RCSB PDB(https://www.rcsb.org/)上进行下载
- 使用shell下载，e.g. 下载 5t3j.pdb文件，在shell中输入：

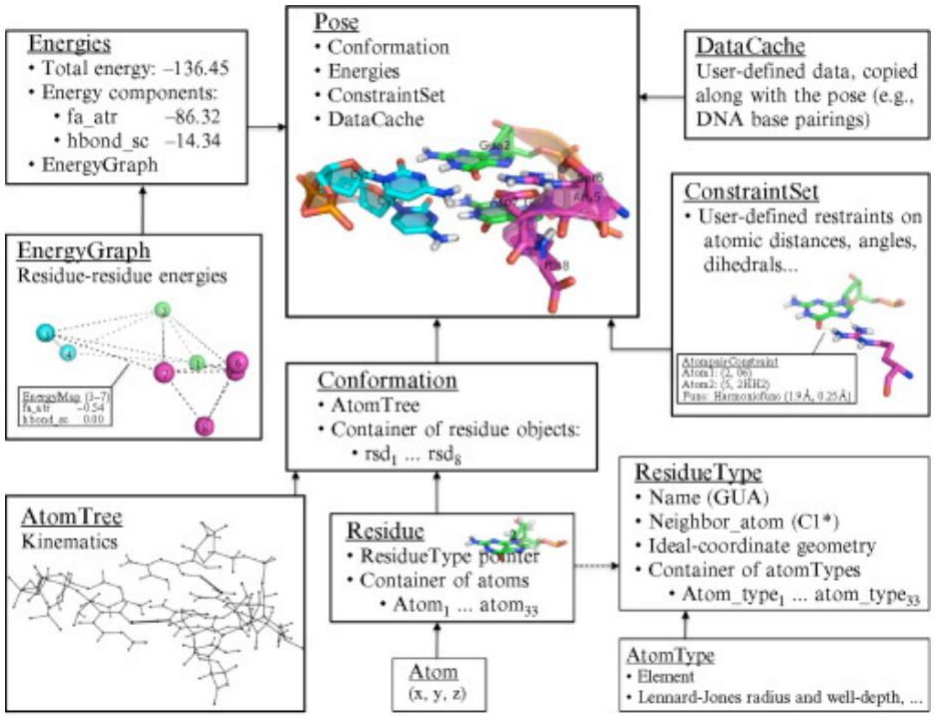
```
wget https://files.rcsb.org/download/5tj3.pdb
```

PyRosetta初始化

```
# 初始化
from pyrosetta import *
init()

#一些pdb文件中可能含有非标准氨基酸，建议使用如下命令预处理
from pyrosetta.toolbox import cleanATOM
cleanATOM("5tj3.pdb") # 需要先下载pdb文件，本行命令会生成5tj3.clean.pdb
```

Pose

PyRosetta中，蛋白的每一个构象对应一个Pose。



PyRosetta使用简介

- pose读取
- pose信息读取
- pose导出为PDB

```
# pose的初始化方式主要有以下4种：
# pose_from_pdb
# pose_from_sequence
# pose_from_silent
# pose_from_rcsb

# 以读取5tj3.clean.pdb为例
pose = pose_from_pdb('5tj3.clean.pdb')

# pose的常用方法
pose.sequence() # 蛋白序列
pose.total_residue() # 蛋白序列的长度
pose.dump_pdb('5tj3.out.pdb') # 导出为pdb
```

pdb_info

- PyRosetta中的Pose对残基进行了重新编号，从1开始，并且忽略了链的信息。
- pdb_info() 函数。

```
# 常用的转换函数
# pdb2pose
# pose2pdb
pose_pdbinfo = pose.pdb_info()
# 获得pdb中，A链、编号为24的残基，在pose中的编号，如果不存在该氨基酸，返回0
pose_number = pose_pdbinfo.pdb2pose(chain='A',res=24)
# 获得pose编号为1的残基在pdb中的编号及链名
pdb_number = pose_pbdinfo.pose2pdb(res=1)

# pdb的基本信息，包括链及始末残基的编号
pose_pdbinfo.short_desc()

# pdb的晶体信息
pose_pdbinfo.crystinfo()
# 第一个残基，第一个原子的bfactor，occupancy。（pose编号）
pose_pdbinfo.bfactor(res=1, atom_index=1)
pose_pdbinfo.occupancy(res=1, atom_index=1)
```

PyRosetta使用简介

residue

- 读取指定residue
- 获取res的信息

```
# 获取pose编号为1的氨基酸，注意从1开始
res1 = pose.residue(1)

res1.annotated_name() # 残基名
res1.name1() # 单字母缩写
res1.name3() # 三字母缩写
res1.natoms() # 残基1的原子总数

# 判断残基1的各种属性
res1.is_polar()
res1.is_aromatic()
res1.is_charged()
res1.is_DNA()
res1.is_protein()
res1.is_aramid()
```

conformation

- 获取键长、键角、坐标和距离等

```
# 使用AtomID获取原子的编号（索引）
from pyrosetta.rosetta.core.id import AtomID
atom1 = AtomID(atomno_in=1, rsd_in=3)  # 3号残基的第一个原子
atom2 = AtomID(atomno_in=2, rsd_in=3)  # 3号残基的第二个原子
atom3 = AtomID(atomno_in=3, rsd_in=3)  # 3号残基的第三个原子
atom4 = AtomID(atomno_in=4, rsd_in=3)  # 3号残基的第四个原子

# 键长
bond_length = pose.conformation().bond_length(atom1, atom2)
# 键角
bond_angle = pose.conformation().bond_angle(atom1, atom2, atom3)

# 一些基本的角度可以直接获取
# pose编号3的残基的phi，psi和omega角
phi = pose.phi(3)
psi = pose.psi(3)
omega = pose.psi(3)
pose.residue(3).chi_atoms() # 残基3的chi角数量
pose.chi(1, 3) # 残基3的chi1角

# 坐标信息
res1.xyz('N') # 残基1的N原子的坐标，等价于下一行
res1.atom('N').xyz()
res1.xyz(1) # 残基1的第一个原子的坐标，等价于下一行
res1.atom(1).xyz()

# 残基1的原子1和原子2的距离
(res1.xyz(1)-res1.xyz(2)).norm()
```

PyRosetta使用简介

score function

- Energies：Cullen force，Lennard-Jones potential
- Constrain set

```
# 获取pose编号为1的氨基酸，注意从1开始
res1 = pose.residue(1)

res1.annotated_name() # 残基名
res1.name1() # 单字母缩写
res1.name3() # 三字母缩写
res1.natoms() # 残基1的原子总数

# 判断残基1的各种属性
res1.is_polar()
res1.is_aromatic()
res1.is_charged()
res1.is_DNA()
res1.is_protein()
res1.is_aramid()
```

- 自定义打分函数对pose打分

```
# 初始化打分函数
# 全原子模型
sfxn = create_score_function('ref2015')
# 或者 sfxn = get_score_function(is_fullatom=True)

# 粗粒度模型
sfxn1 = create_score_function('cen_std')
# 或者 sfxn = get_score_function(is_fullatom=True)

#自定义打分函数
sfxn2 = ScoreFunction()

from pyrosetta.rosetta.core.scoring import ScoreType
from pyrosetta.rosetta.core.scoring import fa_atr, fa_rep

# 重设能量项的权重
sfxn.set_weight(fa_atr, 1.0) # 吸引项权重设为1
sfxn.set_weight(fa_rep, 1.0) # 排斥项权重设为1
score.set_weight(ScoreType.atom_pair_constraint, 1.0) # 原子对约束权重设为1

# 使用constraint_generator添加约束
# eg. 设置N端和C端之间原子的约束
from pyrosetta.rosetta.protocols.constraint_generator import
TerminiConstraintGenerator
termin_cst = TerminiConstraintGenerator()
termin_cst.set_min_distance(8)
termin_cst.set_max_distance(20)
termin_cst.set_sd(1.0)

# 将约束加入到pose中
from pyrosetta.rosetta.protocols.constraint_generator import AddConstraints
add_cst = AddConstraints()
add_cst.add_generator(termin_cst)
add_cst.apply(pose)

sfxn(pose) # 计算pose的整体打分
pose.energies() # pose的energies的部分的得分
```

案例一：FastRelax

- 初始化及导入蛋白
- 设置打分函数
- 确定蛋白的移动部分（主链和侧链）
- 设置relax的参数（打分函数、迭代次数等）
- 运行relax
- 导出pdb结构

```
#PBS -N test
#PBS -l nodes=1:ppn=8
#PBS -S /bin/bash
#PBS -j oe
#PBS -q amdnode
#PBS -m b
#PBS -l walltime=360:00:00

cd $PBS_O_WORKDIR

python  test.py
~
```

执行命令

```
# 初始化及导入蛋白
from pyrosetta import *
from pyrosetta.toolbox import cleanATOM
init()
cleanATOM('5tj3.pdb')
pose = pose_from_pdb("5tj3.clean.pdb")

# 设置打分函数
sf = create_score_function("ref2015")
sf.set_weight(rosetta.core.scoring.atom_pair_constraint, 5)
sf.set_weight(rosetta.core.scoring.dihedral_constraint, 1)
sf.set_weight(rosetta.core.scoring.angle_constraint, 1)

# 确定蛋白的移动部分
mmap = MoveMap()
mmap.set_bb(True)  # 可移动主链
mmap.set_chi(True) # 可移动侧链
mmap.set_jump(True) # 可蛋白之间可相对移动

# 设置relax的参数
relax = rosetta.protocols.relax.FastRelax()
relax.set_scorefxn(sf)
relax.max_iter(200)
relax.dualspace(True)
relax.set_movemap(mmap)

# 对蛋白进行relax
relax.apply(pose)

# 导出蛋白，保存为5tj3.out.pdb
pose.dump_pdb('5tj3.out.pdb')
```

案例二：Low Resolution Docking

- 初始化及导入蛋白
- 将蛋白转为粗粒度模型并保存（用于后续分析）
- 设置打分函数
- 设置对接参数（移动蛋白等）
- 运行docking
- 保存docking结构
- 比较对接前后变化

提交pbs运行！

执行命令

```
# 初始化及导入蛋白
from pyrosetta import *
from pyrosetta.teaching import *
from pyrosetta.toolbox import cleanATOM
from pyrosetta.rosetta.protocols.docking import setup_foldtree
init()
cleanATOM('1v74.pdb')
pose = pose_from_pdb('1v74.clean.pdb')

# 将蛋白转为粗粒度模型
cen_switch = SwitchResidueTypeSetMover("centroid")
cen_switch.apply(pose)

# 保存初始粗粒度模型，用来与生成的模型对比
starting_cen_pose = pose.clone()

# 设置打分函数
sfxn_low = create_score_function("interchain_cen")

# 设置蛋白对参数，"A_B"表示让B移动来与A对接，Vector1([1])表示为两条链创造一个虚拟链接的点
setup_foldtree(pose, "A_B", Vector1([1]))

# 设置docking参数，jump_num表示虚拟链接的数量
jump_num =1
dock_lowres = DockingLowRes(sfxn_low, jump_num)

# 对蛋白对进行对接
dock_lowres.apply(pose)

# 保存对接模型
pose.dump_pdb('1v74.lowres.pdb')

# 查看对接前后的变化
print(CA_rmsd(pose, starting_cen_pose))
```

案例三：High Resolution Docking

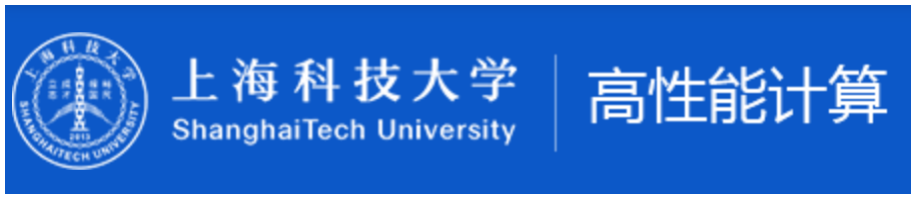
- 初始化及导入蛋白
- 首先进行一次粗粒度对接
- 恢复对接出来的粗粒度模型decoy的侧链
- 设置全原子对接的打分函数
- 运行docking
- 保存docking结构

提交pbs运行！

执行命令

```
# 初始化及导入蛋白
from pyrosetta import *
from pyrosetta.teaching import *
from pyrosetta.toolbox import cleanATOM
from pyrosetta.rosetta.protocols.docking import setup_foldtree
init()
cleanATOM('1v74.pdb')
pose = pose_from_pdb('1v74.clean.pdb')

# 首先进行一次low resolution docking, 获得decoy
starting_pose = pose.clone()
cen_switch = SwitchResidueTypeSetMover("centroid")
cen_switch.apply(pose)

jump_num =1
sfxn_low = create_score_function("interchain_cen")
dock_lowres = DockingLowRes(sfxn_low, jump_num)
setup_foldtree(pose, "A_B", Vector1([1]))
dock_lowres.apply(pose)

# 将原始模型的侧链移植到产生的粗粒度decoy上
recover_sidechains = ReturnSidechainMover(starting_pose)
recover_sidechains.apply(pose)

# 设置docking参数
sfxn_high = create_score_function("ref2015.wts", "docking")
dock_hires = DockMCMProtocol()
dock_hires.set_scorefxn(sfxn_high)
dock_hires.set_partners("A_B")

# docking
dock_hires.apply(pose)

# 保存对接模型
pose.dump_pdb('1v74.hires.pdb')
```

谢谢大家


上海科技大学
ShanghaiTech University
高性能计算


SIAIS
ShanghaiTech
University
免疫化学研究所
Shanghai Institute for Advanced Immunochemical Studies (SIAIS)