Docking Python Documentation

Release 0.3.0-rc

Samuel Murail

Nov 24, 2021

Contents:

CHAPTER 1

Docking Python

Docking_py is a python library allowing a simplified use of the Smina, vina, qvina2 and qvinaw docking software. Docking_py can be easily automatize and scripted.

- Free software: GNU General Public License v2 (GPLv2)
- Documentation: https://docking-py.readthedocs.io.

1.1 Features

- Prepare receptors and ligands.
- **Launch docking using:**
  - Autodock with or without **GPU** acceleration
  - Vina
  - Smina
  - Qvina2
  - Qvinaw

1.2 Citing this work

If you use the code or data in this package, please cite:

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 1. Get sources from the GithubRepo

The sources for Docking Python can be downloaded from the GithubRepo.

You can either clone the public repository:

```
$ git clone git://github.com/samuelmurail/docking_py
```

Or download the tarball:

```
$ curl -OJL https://github.com/samuelmurail/docking_py/tarball/master
```

Once you have a copy of the source, switch to the `docking_py` directorie.

```
cd docking_py
```

2.2 2. Create Conda Environment

You need to create a conda environment to be able to use:

- vina
- smina
- qvina2 and qvinaw
- MGLTools for `prepare_ligand4.py` and `prepare_receptor4.py` scripts.
- Autodock with or without GPU *support*

Use *conda en create* to create it using the `.conda.yml` file. You can overide the environmnent name using the option `--name YOUR_NAME`.

```
$ conda env create -f .conda.yml
```

If you use a linux OS and have a GPU card, you could try the `autodock-gpu` version:

```
$ conda env create -f .conda_gpu.yml
```

This will create an environmnet called `docking` or `docking_gpu` (or the name you defined). You will then, need to activate the environmnent:

```
$ conda activate docking
```

2.3 3. Install docking_py

Once you have a copy of the source and have create a conda encironment, you can install it with:

```
$ python setup.py install
```

2.4 4. Test Installation

To test the installation, simply use `pytest`:

```
$ pytest
==================================== test session starts␣
↪====================================
platform linux -- Python 3.8.2, pytest-5.4.2, py-1.9.0, pluggy-0.13.1
rootdir: /home/murail/Documents/Code/docking_py, inifile: pytest.ini
plugins: cov-2.10.1
collected 13 items

docking_py/docking.py .......                                                ␣
↪[ 53%]
docking_py/tests/test_docking_py.py ......                                   ␣
↪[100%]

============================== 13 passed, 1 warning in 21.18s␣
↪==============================
```

CHAPTER 3

Usage

To explain the usage of `docking_py`, we will use a redocking procedure

The same project should be launched from the `docking` conda environment:

```
$ conda activate docking
```

To use Docking Python in a project:

[1]:
```
from pdb_manip_py import pdb_manip
```

3.1 Extract Ligand coordinates with pdb_manip_py

First you need to extract the ligand coordinates, we will use the `1hsg.pdb` PDB file and extract the coordinates of L-735,524 an inhibitor of the HIV proteases (resname `MK1`) using the `pdb_manip_py` library (Installed with `docking_py`):

[2]:
```
# Create a Coor object
coor_1hsg = pdb_manip.Coor()
coor_1hsg.get_PDB('1hsg', 'data/1hsg.pdb')
```

[3]:
```
# Select res_name MK1
lig_coor = coor_1hsg.select_part_dict(selec_dict={'res_name': ['MK1']})
# Save the ligand coordinates
lig_coor.write_pdb('data/lig.pdb')
```

[17]:
```
view_lig = lig_coor.view
view_lig
```

```
NGLWidget()
```

[20]:
```
# Unecessary, only need to nglview online:
IFrame(src='../_static/lig.html', width=800, height=300)
```

[20]:
```
<IPython.lib.display.IFrame at 0x7f70533bf370>
```

3.2 Extract Receptor coordinates with pdb_manip_py

Then you need to extract the receptor coordinates, we will use the `1hsg.pdb` PDB file and extract the coordinates of the HIV II protease using the `pdb_manip_py` library:

[21]:
```
# Keep only the amino acids
rec_coor = coor_1hsg.select_part_dict(selec_dict={'res_name': pdb_manip.PROTEIN_RES})
rec_coor.write_pdb('data/rec.pdb')
```

[22]:
```
view_rec = rec_coor.view
view_rec
```

```
NGLWidget()
```

[25]:
```
# Unecessary, only need to nglview online:
IFrame(src='../_static/rec.html', width=800, height=300)
```

[25]:
```
<IPython.lib.display.IFrame at 0x7f70531c5970>
```

3.3 Prepare Ligand and receptor structures

You need to create a `Docking` object, and the use the functions `prepare_ligand()` and `prepare_receptor()`:

[8]:
```
from docking_py import docking

test_dock = docking.Docking('test', lig_pdb='data/lig.pdb', rec_pdb='data/rec.pdb')
test_dock.prepare_ligand()
```

```
python2.5 ../../../../../../miniconda3/envs/docking/bin/prepare_ligand4.py -l lig.pdb␣
↪-B none -A hydrogens -o lig.pdbqt
```

[9]:
```
test_dock.prepare_receptor()
```

```
python2.5 ../../../../../miniconda3/envs/docking/bin/prepare_receptor4.py -r data/rec.
↪pdb -A checkhydrogens -o data/rec.pdbqt
```

3.4 Launch docking calculation

Launch the docking:

[10]:
```
test_dock.run_docking(out_pdb='test_dock.pdb',
                      num_modes=10,
                      energy_range=10,
                      exhaustiveness=16,
                      dock_bin='smina')
```

```
Grid points: None
```

```
smina --ligand data/lig.pdbqt --receptor data/rec.pdbqt --log test_dock_log.txt --num_
↪modes 10 --exhaustiveness 16 --energy_range 10 --out test_dock.pdb --size_x 66.00 --
↪size_y 81.00 --size_z 83.00 --center_x 16.07 --center_y 26.49 --center_z 3.77
```

3.5 Analysis

Extract affinity and RMSD to crystal structure:

[11]:
```
rmsd_list = test_dock.compute_dock_rmsd(test_dock.lig_pdbqt)
```

```
File name doesn't finish with .pdb read it as .pdb anyway
```

[12]:
```
rmsd_list
```

[12]:
```
[0.6172348337545442,
 4.523207300135602,
 11.579705736330263,
 9.904196947759067,
 10.692842899809198,
 10.975378963844483,
 12.19258827074875,
 10.207969165313932,
 9.394261151362569,
 12.029979500398163]
```

[26]:
```
view_dock = test_dock.view_dock(ref_pdb="data/1hsg.pdb")
view_dock
```

```
NGLWidget(max_frame=9)
```

[30]:
```
# Unecessary, only need to nglview online:
IFrame(src='../_static/dock.html', width=800, height=300)
```

[30]:
```
<IPython.lib.display.IFrame at 0x7f70532c21c0>
```

[15]:
```
test_dock.affinity
```

[15]:
```
{1: {'affinity': -11.9, 'rmsd_low': 0.0, 'rmsd_high': 0.0},
 2: {'affinity': -10.6, 'rmsd_low': 2.288, 'rmsd_high': 4.387},
 3: {'affinity': -9.3, 'rmsd_low': 3.55, 'rmsd_high': 11.574},
 4: {'affinity': -8.8, 'rmsd_low': 5.812, 'rmsd_high': 9.719},
 5: {'affinity': -8.7, 'rmsd_low': 5.959, 'rmsd_high': 10.368},
 6: {'affinity': -8.7, 'rmsd_low': 3.265, 'rmsd_high': 10.921},
 7: {'affinity': -8.4, 'rmsd_low': 3.702, 'rmsd_high': 12.258},
 8: {'affinity': -8.3, 'rmsd_low': 5.468, 'rmsd_high': 9.968},
 9: {'affinity': -8.2, 'rmsd_low': 5.679, 'rmsd_high': 9.289},
 10: {'affinity': -8.1, 'rmsd_low': 7.058, 'rmsd_high': 11.97}}
```

[ ]:

CHAPTER 4

docking_py

4.1 docking_py package

4.1.1 Subpackages

4.1.2 Submodules

4.1.3 docking_py.cli module

Console script for docking_py.

docking_py.cli.**main**()
Console script for docking_py.

4.1.4 docking_py.docking module

Include the Docking class

class docking_py.docking.**Docking**(*name*, *lig_pdb=None*, *rec_pdb=None*, *lig_pdbqt=None*, *rec_pdbqt=None*, *log_level=20*)

Bases: object

Docking encapsulation class.

This class can be used to launch vina, smina, qvina and qvinaw.

Parameters

- **name** (*str*) – generic name of the system
- **lig_pdb** (*str, optional*) – path of the ligand coordinate file (.pdb)
- **rec_pdb** (*str, optional*) – path of the receptor coordinate file (.pdb)
- **lig_pdbqt** (*str, optional*) – path of the ligand coordinate file (.pdbqt)

- **rec_pdbqt** (*str, optional*) – path of the receptor coordinate file (.pdbqt)
- **dock_pdb** (*str, optional*) – path of the docking ligand coordinate file (.pdb)
- **dock_log** (*str, optional*) – path of the docking log file (.log)

align_receptor(*ref_pdb, chain_ref=['A'], chain_rec=['A']*)
Align self.rec_pdb to ref_pdb.

Example

```
>>> pdb_manip.show_log()
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> dock_4yob = Docking(name='4yob')
>>> dock_4yob.extract_receptor(os.path.join(TEST_PATH, '4yob.pdb'),␣
↪TEST_OUT, {'res_name': pdb_manip.PROTEIN_RES})        #doctest: +ELLIPSIS
Succeed to read file ...4yob.pdb ,  916 atoms found
Succeed to save file ...4yob_rec.pdb
>>> dock_4yob.align_receptor(os.path.join(TEST_PATH, '1hsg.pdb'))
Succeed to read file .../4yob_rec.pdb ,  760 atoms found
Succeed to read file .../1hsg.pdb ,  1686 atoms found
PQITLWKRPIVTIKIGGQLKEALLNTGADDTVFEEVNLPGRWKPKLIGGIGGFVKVRQYDQVPIEICGHKVIGTVLVGPT
******|**|**************|*******|**||********|*******|*******|␣
↪*******|*********
PQITLWQRPLVTIKIGGQLKEALLDTGADDTVLEEMSLPGRWKPKMIGGIGGFIKVRQYDQILIEICGHKAIGTVLVGPT
<BLANKLINE>
PTNVIGRNLMTQIGCTLNF
*|*|*****|*********
PVNIIGRNLLTQIGCTLNF
<BLANKLINE>
Succeed to save file ...4yob_rec.pdb
>>> coor_holo = pdb_manip.Coor(os.path.join(TEST_PATH, '1hsg.pdb'))
↪#doctest: +ELLIPSIS
Succeed to read file ...1hsg.pdb ,  1686 atoms found
>>> coor_rec = pdb_manip.Coor(dock_4yob.rec_pdb) #doctest: +ELLIPSIS
Succeed to read file ...4yob_rec.pdb ,  760 atoms found
>>> rmsd = coor_rec.compute_rmsd_to(coor_holo,        selec_dict={'name': ['CA
↪'], 'chain':['A']})
>>> print('RMSD after alignement is {:.2f} Å'.format(rmsd))
RMSD after alignement is 1.50 Å
```

compute_dock_rmsd(*ref_lig_pdb*, *selec_dict={}*)
Compute RMSD from docking pdb to `ref_lig_pdb`. By default use all atoms for RMSD calculation. To use only Calpha atoms define `selec_dict={'name':['CA']}`.

Parameters

- **ref_lig_pdb** (*str*) – PDB reference file
- **selec_dict** (*dict, optional, default={}*) – Selection for RMSD calculation

Returns RMSD list

Return type list

display()
Display defined attribute of the Docking object.

dock_log

dock_pdb

dock_xml

extract_affinity()
Extract affinity from the docking `.log` file.

Returns Affinity and RMSD informations as a dictionnary

Return type dict

extract_autodock_pdb_affinity(*out_pdb*, *reorder=True*)
Extract pdb models from the the autodock log files.

CPU version

extract_autodock_pdb_affinity2(*out_pdb*, *reorder=True*)
Extract pdb models from the the autodock log files. Makes use of the xml generated by the gpu version.

GPU version

extract_lig_rec_pdb(*coor_in*, *folder_out*, *rec_select_dict*, *lig_select_dict*)

- Extract receptor and ligand coordinates from a coor file
- remove alternative location
- Keep only amino acid residues
- Save both coordinates and add it in the object

Parameters

- **pdb_id** (*str*) – PDB ID
- **rec_chain** (*list of str*) – Chain(s) of the receptor
- **lig_chain** (*list of str*) – Chain(s) of the ligand

Object field(s) changed:

- self.rec_pdb
- self.lig_pdb

Example

```
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> dock_1hsg = Docking(name='1hsg')
>>> dock_1hsg.extract_lig_rec_pdb(os.path.join(TEST_PATH, '1hsg.pdb'),
↪TEST_OUT, {'res_name': pdb_manip.PROTEIN_RES}, {'res_name': 'MK1'})
↪#doctest: +ELLIPSIS
Succeed to read file ...1hsg.pdb ,  1686 atoms found
Succeed to save file ...1hsg_rec.pdb
Succeed to save file ...1hsg_input_lig.pdb
>>> coor_lig = pdb_manip.Coor(dock_1hsg.lig_pdb) #doctest: +ELLIPSIS
Succeed to read file ...1hsg_input_lig.pdb ,  45 atoms found
>>> coor_rec = pdb_manip.Coor(dock_1hsg.rec_pdb) #doctest: +ELLIPSIS
Succeed to read file ...1hsg_rec.pdb ,  1514 atoms found
```

extract_ligand(*coor_in*, *folder_out*, *lig_select_dict*)

- Extract ligand coordinates
- remove alternative location
- Save coordinates and add it in the object

Object field(s) changed:

- self.lig_pdb

Example

```
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> dock_1hsg = Docking(name='1hsg')
>>> dock_1hsg.extract_ligand(os.path.join(TEST_PATH, '1hsg.pdb'),        TEST_
↪OUT, {'res_name': 'MK1'})        #doctest: +ELLIPSIS
Succeed to read file ...1hsg.pdb ,  1686 atoms found
Succeed to save file ...1hsg_lig.pdb
>>> coor_lig = pdb_manip.Coor(dock_1hsg.lig_pdb) #doctest: +ELLIPSIS
Succeed to read file ...1hsg_lig.pdb ,  45 atoms found
```

extract_receptor(*coor_in*, *folder_out*, *rec_select_dict*)

- Extract receptor coordinates
- remove alternative location
- Keep only amino acid residues
- align structure on ref
- Save coordinates and add it in the object

Parameters

- **pdb_id** (*str*) – PDB ID
- **ref_pdb** (*str*) – Reference coordinates file
- **rec_chain** (*list of str*) – Chain(s) of the receptor
- **rec_chain** – Chain(s) of the reference file

Object field(s) changed:

- self.rec_pdb

Example

```
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> dock_1hsg = Docking(name='1hsg')
>>> dock_1hsg.extract_receptor(os.path.join(TEST_PATH, '1hsg.pdb'),
↪TEST_OUT, {'res_name': pdb_manip.PROTEIN_RES})        #doctest: +ELLIPSIS
Succeed to read file ...1hsg.pdb ,  1686 atoms found
Succeed to save file ...1hsg_rec.pdb
>>> coor_rec = pdb_manip.Coor(dock_1hsg.rec_pdb) #doctest: +ELLIPSIS
Succeed to read file ...1hsg_rec.pdb ,  1514 atoms found
```

get_gridfld()
Get `gridfld` from the `.gpf` file.

gpf

lig_pdb

lig_pdbqt

log_to_pdb(*out_pdb*)
Read autodock log, extract ligand model coordinates and extract affinities.

Parameters **out_pdb** (*str*) – output pdb file

> **Warning:** Difference between GPU and CPU version of autodock logs. Torsional Free Energy is not computed with GPU version.

prepare_grid(*out_folder*, *gpf_out_prefix=None*, *spacing=0.375*, *grid_npts=None*, *center=None*, *check_file_out=True*)
Grid preparation

Launch the `prepare_gpf4.py` command from MGLToolsPackage. And `autogrid4`.

prepare_ligand(*lig_pdbqt=None*, *rigid=False*, *center=False*, *random_rot=False*, *check_file_out=True*)
Ligand preparation to *pdbqt* format using the *prepare_ligand4.py* command. Can center the ligand, could be usefull with autodock (issues when x,y,z > 100 Å).

Parameters

- **lig_pdbqt** (*str, optional, default=None*) – output name
- **rigid** (*bool, optional, default=False*) – Flag to define if ligand is rigid
- **center** (*bool, optional, default=False*) – Flag to define if ligand have to centered
- **check_file_out** (*bool, optional, default=True*) – flag to check or not if file has already been created. If the file is present then the command break.

Object requirement(s):

- self.lig_pdb

Object field(s) changed:

- self.lig_pdbqt

Example

```
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> coor_1hsg = pdb_manip.Coor(os.path.join(TEST_PATH, '1hsg.pdb'))
↪#doctest: +ELLIPSIS
Succeed to read file ...tests/input/1hsg.pdb ,  1686 atoms found
>>> lig_coor = coor_1hsg.select_part_dict(            selec_dict={'res_name':
↪'MK1'})
>>> lig_atom_num = lig_coor.num
>>> print('Ligand has {} atoms'.format(lig_atom_num))
Ligand has 45 atoms
>>> out_lig = os.path.join(TEST_OUT,'lig.pdb')
>>> lig_coor.write_pdb(out_lig) #doctest: +ELLIPSIS
Succeed to save file .../lig.pdb
>>> test_dock = Docking('test', lig_pdb=out_lig)
>>> test_dock.prepare_ligand() #doctest: +ELLIPSIS
python2... .../prepare_ligand4.py -l lig.pdb -B none -A hydrogens -o lig.pdbqt
>>> coor_lig = pdb_manip.Coor(test_dock.lig_pdbqt)        #doctest: +ELLIPSIS
File name doesn't finish with .pdb read it as .pdb anyway
Succeed to read file .../lig.pdbqt ,  50 atoms found
```

(continues on next page)

(continued from previous page)

```
>>> test_dock.display() #doctest: +ELLIPSIS
name          : test
lig_pdb       : .../lig.pdb
lig_pdbqt     : .../lig.pdbqt
ref_lig_pdb   : .../lig.pdb
```

prepare_receptor(*rec_pdbqt=None*, *check_file_out=True*)

Receptor preparation to *pdbqt* format using the *prepare_receptor4.py* command.

Parameters

- **rec_pdbqt** (*str, optional, default=None*) – output name
- **check_file_out** (*bool, optional, default=True*) – flag to check or not if file has already been created. If the file is present then the command break.

Object requirement(s):

- self.rec_pdb

Object field(s) changed:

- self.rec_pdbqt

Example

```
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> coor_1hsg = pdb_manip.Coor(os.path.join(TEST_PATH, '1hsg.pdb'))
↪#doctest: +ELLIPSIS
Succeed to read file .../1hsg.pdb ,  1686 atoms found
>>> # Keep only amino acid
>>> rec_coor = coor_1hsg.select_part_dict(selec_dict={'res_name': pdb_manip.
↪PROTEIN_RES})
>>> out_rec = os.path.join(TEST_OUT,'rec.pdb')
>>> rec_coor.write_pdb(out_rec) #doctest: +ELLIPSIS
Succeed to save file .../rec.pdb
>>> rec_atom_num = rec_coor.num
>>> print('Receptor has {} atoms'.format(rec_atom_num))
Receptor has 1514 atoms
>>> test_dock = Docking('test', rec_pdb=out_rec)
>>> test_dock.prepare_receptor() #doctest: +ELLIPSIS
python2... .../prepare_receptor4.py -r .../rec.pdb -A checkhydrogens -o .../
↪rec.pdbqt
>>> coor_rec = pdb_manip.Coor(test_dock.rec_pdbqt)        #doctest: +ELLIPSIS
File name doesn't finish with .pdb read it as .pdb anyway
Succeed to read file .../rec.pdbqt ,  1844 atoms found
>>> test_dock.display() #doctest: +ELLIPSIS
name          : test
rec_pdb       : .../rec.pdb
rec_pdbqt     : .../rec.pdbqt
```

random_rot_ligand()

- Do a random rotation on ligand

Object field(s) changed:

- self.lig_pdb

Example

```
>>> TEST_OUT = str(getfixture('tmpdir'))
>>> dock_1hsg = Docking(name='1hsg')
>>> dock_1hsg.extract_ligand(os.path.join(TEST_PATH, '1hsg.pdb'),        TEST_
↪OUT, {'res_name': 'MK1'})        #doctest: +ELLIPSIS
Succeed to read file ...1hsg.pdb ,  1686 atoms found
Succeed to save file ...1hsg_lig.pdb
>>> coor_lig = pdb_manip.Coor(dock_1hsg.lig_pdb) #doctest: +ELLIPSIS
Succeed to read file ...1hsg_lig.pdb ,  45 atoms found
>>> com_before = coor_lig.center_of_mass()
>>> dock_1hsg.random_rot_ligand()
Succeed to read file ...1hsg_lig.pdb ,  45 atoms found
Succeed to save file ...1hsg_lig.pdb
>>> coor_lig = pdb_manip.Coor(dock_1hsg.lig_pdb) #doctest: +ELLIPSIS
Succeed to read file ...1hsg_lig.pdb ,  45 atoms found
>>> com_after = coor_lig.center_of_mass()
>>> print('Same center of mass after rotation :{}'.format(com_before==com_
↪after))
Same center of mass after rotation :[False False False]
```

..warning: The function overwrite lig_pdb coordinates.

rec_com()
Get center of mass of the receptor pdb file.

rec_grid(*buffer_space=30*, *spacing=1.0*)
Compute grid from the receptor pdb file.

rec_pdb

rec_pdbqt

ref_lig_pdb

run_autodock(*out_folder*, *dock_out_prefix=None*, *dock_log=None*, *dock_pdb=None*, *nrun=10*, *check_file_out=True*, *GPU=True*)
Run autodock with cpu or gpu if available

run_autodock_cpu(*out_folder*, *dock_out_prefix=None*, *dock_log=None*, *dock_pdb=None*, *dock_xml=None*, *dpf_out=None*, *nrun=10*, *param_list=[]*, *check_file_out=True*)

1. Launch the `prepare_dpf4.py` command from MGLToolsPackage.
2. Launch `autodock4`

This requires a gpf and associated map files, and ligand pdbqt It creates pose pdb + xml + dlg + dpf and smina like _log.txt files

run_autodock_docking(*out_pdb*, *log=None*, *prepare_grid=True*, *num_modes=100*, *center=None*, *spacing=0.375*, *grid_size=None*, *grid_max_points=None*, *check_file_out=True*, *GPU=True*)
Run docking using autodock.

Parameters

- **out_pdb** (*str*) – PDB output name
- **log** (*str, optional, default=None*) – Log ouput name
- **prepare_grid** (*bool, optional, default=True*) – perform grid setup

- **num_modes** (*int, optional, default=100*) – maximum number of binding modes to generate
- **center** (*list, optional, default=None*) – coordinate of the center (x, y, z, Angstroms)
- **grid_size** (*list, optional, default=None*) – size in the docking box (x, y, z, Angstroms)
- **grid_max_points** (*int, optional, default=None*) – max number of grid points per dimension (256 for GPU)
- **check_file_out** (*bool, optional, default=True*) – flag to check or not if file has already been created. If the file is present then the command break.

Object requirement(s):

- self.lig_pdbqt
- self.rec_pdbqt

Object field(s) changed:

- self.dock_pdb
- self.dock_log

Example

run_autodock_gpu(*out_folder*, *dock_out_prefix=None*, *dock_log=None*, *dock_pdb=None*, *dock_xml=None*, *nrun=10*, *check_file_out=True*)

Autodock GPU arguments:

mandatory: -ffile ./input/1stp/derived/1stp_protein.maps.fld -lfile ./input/1stp/derived/1stp_ligand.pdbqt

opyional: -nrun # LGA runs 1 -nev # Score evaluations (max.) per LGA run 2500000 -ngen # Generations (max.) per LGA run 27000 -lsmet Local-search method sw (Solis-Wets) -lsit # Local-search iterations (max.) 300 -psize Population size 150 -mrat Mutation rate 2 (%) -crat Crossover rate 80 (%) -lsrat Local-search rate 6 (%) -trat Tournament (selection) rate 60 (%) -resnam Name for docking output log "docking" -hsym Handle symmetry in RMSD calc. 1

This requires a gpf and associated map files, and ligand pdbqt It creates pose pdb + xml + dlg and smina like _log.txt files

> **Warning:** Difference between GPU and CPU version of autodock logs. Torsional Free Energy is not computed with GPU version.

run_docking(*out_pdb*, *log=None*, *dock_bin='vina'*, *num_modes=100*, *energy_range=10*, *exhaustiveness=16*, *cpu=None*, *seed=None*, *autobox=False*, *center=None*, *grid_size=None*, *min_rmsd_filter=None*, *scoring=None*, *check_file_out=True*)

Run docking using vina, qvina, qvinaw or smina.

Parameters

- **out_pdb** (*str*) – PDB output name
- **log** (*str, optional, default=None*) – Log ouput name
- **dock_bin** (*str, optional, default='vina'*) – Docking software name ('vina', 'qvina', 'qvinaw', 'smina')

- **num_modes** (*int, optional, default=100*) – maximum number of binding modes to generate
- **energy_range** (*int, optional, default=10*) – maximum energy difference between the best binding mode and the worst one displayed (kcal/mol)
- **exhaustiveness** (*int, optional, default=16*) – exhaustiveness of the global search (roughly proportional to time): 1+
- **cpu** (*int, optional, default=None*) – the number of CPUs to use (the default is to try to detect the number of CPUs or, failing that, use 1)
- **seed** (*int, optional, default=None*) – explicit random seed
- **autobox** (*bool, optional, default=False*) – Flag to use ligand to define the docking box
- **center** (*list, optional, default=None*) – coordinate of the center (x, y, z, Angstroms)
- **grid_size** (*list, optional, default=None*) – size in the docking box (x, y, z, Angstroms)
- **check_file_out** (*bool, optional, default=True*) – flag to check or not if file has already been created. If the file is present then the command break.

Object requirement(s):

- self.lig_pdbqt
- self.rec_pdbqt

Object field(s) changed:

- self.dock_pdb
- self.dock_log

Example

view_dock(*ref_pdb=None*)
Return a *nglview* object to view the object coordinates in a jupyter notebook with the module `nglview`.

MDAnalysis module is required.

write_out_affinities(*fn*, *affinities*)
Save affinities in a file

`docking_py.docking.`**set_log_level**(*level=20*)
setup log verbose level

`docking_py.docking.`**show_log**()
To use only with Doctest !!! Redirect logger output to sys.stdout

4.1.5 Module contents

Top-level package for Docking Python.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/samuelmurail/docking_py/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Docking Python could always use more documentation, whether as part of the official Docking Python docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/samuelmurail/docking_py/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *docking_py* for local development.

1. Fork the *docking_py* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/docking_py.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv docking_py
$ cd docking_py/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 docking_py tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/samuelmurail/docking_py/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_docking_py
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Samuel Murail, Université de Paris <samuel.murail@u-paris.fr>

6.2 Contributors

- Pierre Tuffery, INSERM
- Damien Espana

We are open to any contribution.

CHAPTER 7

History

7.1 0.3.0 (2021-15-11)

Docking_py module has been published in the reference *SeamDock* article

- Murail S, de Vries SJ, Rey J, Moroy G and Tufféry P. *SeamDock: An Interactive and Collaborative Online Docking Resource to Assist Small Compound Molecular Docking.* **Front. Mol. Biosci. (2021)**. 8:716466. doi: 10.3389/fmolb.2021.716466.

7.2 0.1.0 (2020-04-15)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

V

W