# Scripting system

Zarek S. Siegel

March 15, 2016

# 1 Summary

## 1.1 Description

High platforms team leverage enthusiastically high infrastructures utilize convergence functionalized action capital. Than growth multimedia viral alternative emerging infrastructures e-enable sucking fashion capital niches standards. Extensible "organic" team re-engineer reinvent quality leading-edge paradigms cultivate infrastructures energistically dynamic. Art holisticly covalent leverage initiatives enterprise interoperable empowerment actualize collaborative invested chains utilize expedite corporate. Improvements potentialities results energistically streamline completely positioning brand adaptive team visualize of holistic infrastructures.

## 1.2 File Structure

- NOTES:
  - ***bolded-italicized*** files/directories need to be created by the user
  - **bolded** files/directories/names need to be modified by the user
  - *italicized* files and directories are created by scripts
  - `plain` files and directories do not (and should not) by modified
  - `$ command` indicates a command line command in the terminal
  - directories and files (and parts of file names) in lower case must included exactly as indicated
  - directories and files in all caps should be named appropriately to the proteins and dockings in question

- ***base_dir*** – The root of the system is the base directory, containing all other files (probably best not to put anything else in this folder but what's indicated below)
  - `Readme.md` –This file
  - ***Dockings.csv*** – A spreadsheet containing master docking parameters
  - ***Gridboxes.csv*** – A spreadsheet specifying all the grid box parameters
  - `ligsets/` –A directory containing all the sets of ligands
    - ***ligsets/LIGSET/*** – for each set, the there should be a directory within the `ligsets/` directory, whose name is the name of the ligand set. Substitute `LIGSET` for some appropriate name (e.g. `ligsets/my_awesome_ligset/`)
      - ***ligsets/LIGSET/pdbqts/*** –a directory containing a PDBQT file for each ligand in the set (whose name is `LIG.pdbqt`, with `LIG` being exactly the same as in `ligsets/LIGSET/LIGSET_list.txt`) [may be scripted later]
      - ***ligsets/LIGSET/LIGSET_list.txt*** –a text file containing a list of all the ligands (one on each line) and nothing else
        - you can create this easily on the command line using: `$ cd base_dir/ligsets/; for l in $(ls LIGSET/pdbqts | sed 's/.pdbqt//'); do echo $l >> LIGSET/LIGSET_list.txt ; done` (appropriately substituting `base_dir` and `LIGSET`, and assuming the PDBQTs are already made)
      - [optional/preliminary] `ligsets/LIGSET/LIGSET.cdxml/` or `ligsets/LIGSET/LIGSET.mol/`– one file to show all the ligands in one page for presentations, PDFmaking and such.
      - [optional/preliminary] `/mols/` and `ligsets/LIGSET/pdbs/` – directories for preparing the initial PDBQT files. Will either be optional, or more scripts will be written.
  - `parameters_csvs/` –A directory containing small CSV files specifying the docking parameters for each individual docking (needed for scripts, at least as of now). Generated by `scripts/write_params_csv.R` from information in `Gridboxes.csv`
    - They will be named **`parameters_csvs/DOCKING_parameters.csv`**, where `DOCKING` is the docking ID
  - `vina_submit_shs/` – A directory containing the submission files for

Vina jobs on the (Wesleyan) cluster. Generated by `write_vina_submit` function of `docking_data_assembly.py`

- **`vina_submit_shs/vina_submit_DOCKING.sh`** –for a docking of 20 models or less, a single submission script is written (and submitted using `$ bsub < vina_submit_DOCKING.sh`, see submission instructions below)
- **`vina_submits_DOCKING/`** –dockings of more than 20 models need to be submitted with multiple scripts (because Vina will not generate more than 20 poses). In this case, the `write_vina_submit` function will create a directory called `vina_submits_DOCKING/` containing $n$ scripts **`vina_submit_DOCKING.1.sh`** , **`vina_submit_DOCKING.2.sh`** through **`vina_submit_DOCKING.n.sh`** . This script is set up to write each $n$ submission scripts, where each of which script has a model number of 20 and $n$ is the number of models divided by 20. Therefore, if greater than 20, the number of models should always be a multiple of twenty, or things will get messed up. (These are submitted used `$ for s in $(ls vina_submits_DOCKING); do bsub < $s; done$`, see instructions below)

◦ ***`PROTEIN/`*** , etc. – A directory for *each* protein, whose name is the name of the protein, the reference name/abbreviation used throughout, it just needs to be consistent (e.g. I primarily dock the proteins HepI and p300 and have the directories `hepi/` and `p300/` in my `base_dir/`)

- ***`PROTEIN/PROTEIN.pdbqt`*** – the PDBQT file to be used for docking (`PROTEIN` must be *exactly* the same for the directory/file names and in the `Dockings.csv` and `Gridboxes.csv` entries for the protein's dockings) [may be scripted later, from `PROTEIN.pdb`]
- [optional/preliminary] `PROTEIN.pdb` – the original PDB file
- **`PROT/DOCKING/`** – for every docking, there should be a directory whose name in the docking ID (the same as in `Dockings.csv`). Note: the user shouldn't make this folder, it is made by `vina_submit_DOCKING.sh`.
  - This will eventually contain
- ***`PROT/binding_sites/`*** – a directory containing binding site PDBs:
  - To do binding site scoring by residues contacted (which is detected by the AutoDockTools script `process_VinaResult.py`),

There must be one or more `PROT/binding_sites/BINDING_SITE.pdb` files. These are subsets of the original `PROTEIN.pdb` (I originally created mine by grabbing the residues within $5\AA$ of the bound ligands that came with the crystal structure, but it could be done many other ways).

- `scripts/` – all the scripts needed to use this set up
  - `write_params_csv.R` – writes `DOCK_parameters.csv` using information in `Dockings.csv` and `Gridboxes.csv`
  - `load_parameters.sh` – loads parameters from `DOCKING_parameters.csv` (used in the next script)
  - `separate_vina_results.sh` – runs Vina result PDBQTs through the AutoDockTools script `process_VinaResult.py`, which separates the poses into separate files and extracts the receptor contacts. The resulting files ended up in `DOCKING/processed_pdbqts/`, named `DOCKING_LIGAND_mMODEL.pdbqt`, where MODEL is the particular pose represented by the file. (A docking with $l$ ligands and with Vina set to produce $m$ models will therefore end up having $l \times m$ files after this script runs.)
  - `cleanup_processed_vina_results.sh` – cleans up processed PDBQTs (`DOCKING/processed_pdbqts/DOCKING_LIGAND_mMODEL.pdbqt`) and converts then to PDBs using the AutoDockTools script `pdbqt_to_pdb.py`
  - `parse_pdb.py` – defines a object class called `Pdb` for parsing PDB and PDBQT files for their 3D coordinates and in the case of processed Vina results, their binding energy, protein contacts, and other data generated by Vina. (Necessary for `docking_data_assembly.py`.)
  - `aiad_icpd.py` – defines functions to calculate the AIAD (averaged inter–atomic distance) and ICPD (inter–centerpoint distance) between two `Pdb` objects, two useful parameters for determining where a pose is binding on a protein and clustering poses together based on proximity. (Necessary for these functions in `docking_data_assembly.py`.)
  - `docking_data_assembly.py` – defines an object class called `Docking` for preparation and analysis of dockings. Contains several important functions:
    - `write_vina_submit` – prepares Vina job submission scripts
    - `assemble_dic` – assembles a data dictionary that contains all

the mined data from the Vina results
    - `score_binding_sites` – scores each pose for the proportion of residues contacted in each reference binding site
    - `assess_all_resis` – uses binding scores to determine a True/-False for each pose binding in at each binding site (based on a threshold score, currently 0.1 or 10%)
    - (`aiad_icpd_binding_sites`) – calculates AIAD and ICPD scores for each pose compared to each binding site
    - `write_alldata_csv` – writes the data dictionary to a CSV file called `DOCKING_alldata.csv`
    - `cluster_poses` [`prepare_clustering_csv.py`] – calculates AIAD scores for every pose compared to every other pose, for cluster analysis later on
- `pre_and_post_control.py` – links to all of the above scripts to coordinate their function, providing the global variables required to run this system on different computers.
- **Other scripts that may or may not be added:**
    - A script to add an entry to `Dockings.csv` or `Gridboxes.csv` with use input

## 1.3 Complete listing of included files

- **Notes**:
    - ... indicates more of the same kind of file or directory
    - .py files (except for the control script) may have a compiled .pyc file with them
    - (files) are optional (for now)
    - [files] are works in progress and may not be included ultimately
    - [[files]] need to be made

```
/path/to/base_dir/
  Readme.md
  Dockings.csv
  Gridboxes.csv
  ligsets/
    LIGSET1/
      LIGSET1_list.txt
      (LIGSET1.cdxml)
      (mols/...)
      (pdbs/...)
```

```
          pdbqts/
            LIG1_1.pdbqt
            LIG1_2.pdbqt
            LIG1_3.pdbqt
            ...
        LIGSET2/
          LIGSET1_list.txt
          (mols/...)
          (pdbs/...)
          pdbqts/
            LIG2_1.pdbqt
            LIG2_2.pdbqt
            LIG2_3.pdbqt
            ...
        ...
      vina_submit_shs/
        vina_submit_A1.sh
        vina_submits_A2/
          vina_submit_A2.1.sh
          vina_submit_A2.2.sh
          vina_submit_A2.3.sh
          ...
        vina_submit_B1.sh
        ...
      PROT_A/
        (PROT_A.pdb)
        PROT_A.pdbqt
        binding_sites/
          BINDING_SITE_ALPHA1.pdb
          BINDING_SITE_ALPHA2.pdb
          BINDING_SITE_ALPHA3.pdb
          ...
        A1/
        A2/
        A3/
        ...
      PROT_B/
        (PROT_B.pdb)
        PROT_B.pdbqt
        binding_sites/
          BINDING_SITE_BETA1.pdb
          ...
        B1/
        B2/
        ...
```

```
      ...
      scripts/
        [new_grid_or_dock_entry.R]
        [[write_ligset_list_txt.sh]]
        load_parameters.sh
        [[ligand, protein preparation]]
        separate_vina_results.sh
        cleanup_processed_vina_results.sh
        parse_pdb.py
        aiad_icpd.py
        [[prepare_clustering_csv.py]]
        docking_data_assembly.py
        pre_and_post_control.py
        [[post_docking_graphs.R]]
        [[clustering_graphs.R]]
        [[R cript to select poses to view in PyMol]]
        [[Py script to load PyMol sessions from lists]]
```

### 1.3.1 After docking post-processing:

```
      PROTEIN/
        (PROTEIN.pdb)
        PROTEIN.pdbqt
        binding_sites/
          SITE1.pdb
          SITE1.pdb
          SITE1.pdb
          ...
        DOCKING/
          result_pdbqts/
            DOCKING_LIG1_results.pdbqt
            DOCKING_LIG2_results.pdbqt
            DOCKING_LIG3_results.pdbqt
            ...
          processed_pdbqts/
            DOCKING_LIG1_m1.pdbqt
            DOCKING_LIG1_m2.pdbqt
            DOCKING_LIG1_m3.pdbqt
            ...
            DOCKING_LIG2_m1.pdbqt
            DOCKING_LIG2_m2.pdbqt
            DOCKING_LIG2_m3.pdbqt
```

```
            ...
            DOCKING_LIG3_m1.pdbqt
            DOCKING_LIG3_m2.pdbqt
            DOCKING_LIG3_m3.pdbqt
            ...
          processed_pdbs/
            DOCKING_LIG1_m1.pdb
            DOCKING_LIG1_m2.pdb
            DOCKING_LIG1_m3.pdb
            ...
            DOCKING_LIG2_m1.pdb
            DOCKING_LIG2_m2.pdb
            DOCKING_LIG2_m3.pdb
            ...
            DOCKING_LIG3_m1.pdb
            DOCKING_LIG3_m2.pdb
            DOCKING_LIG3_m3.pdb
            ...
          DOCKING_alldata.csv
          DOCKING.p
          DOCKING_clustering.csv [[DOCKING_pose_pose_aiads.csv]]
          [[DOCKING_best_aiad_pairs.csv]]
          [[graphs/]]
            [[...graphs...]]
```

## 1.4 Example Dockings.csv file

```
Docking ID,Date,Protein,Ligset,Grid box,Exhaustiveness,Number of Models,Number
     of CPUs,Notes
A1,20160301,PROTA,LIGS1,AAS,20,10,2,looking at active size of protein A
A2,20160308,PROTA,LIGS2,AWP,50,400,4,high volume docking of whole protein A
B1,20160308,PROTB,LIGS3,BWP,8,20,1,initial docking of whole protein B
```

## 1.5 Example Gridboxes.csv file

```
Gridbox Name,Protein,Size in x-dimension,Size in y-dimension,Size in z-
    dimension,Center in x-dimension,Center in y-dimension,Center in z-
    dimension,Notes
```

| Docking ID | Date | Protein | Ligset | Grid box | Exhaustiveness | Number of Models | Number of CPUs | Notes |
|---|---|---|---|---|---|---|---|---|
| A1 | 20160301 | PROTA | LIGS1 | AAS | 20 | 10 | 2 | looking at active size of protein A |
| A2 | 20160308 | PROTA | LIGS2 | AWP | 50 | 400 | 4 | high volume docking of whole protein A |
| B1 | 20160308 | PROTB | LIGS3 | BWP | 8 | 20 | 1 | initial docking of whole protein B |

```
AAS,PROTA,60,72,88,41.89,2.69,-1.85,active site of protein A
AWP,PROTA,126,126,126,41.89,2.69,-1.85,all of protein A
BWP,PROTB,126,126,126,4.89,-5.27,12.0,all of protein B
```

| Gridbox Name | Protein | Box size (x) | Box size (y) | Box size (z) | Box center (x) | Box center (y) | Box center (z) | Notes |
|---|---|---|---|---|---|---|---|---|
| AAS | PROTA | 60 | 72 | 88 | 41.89 | 2.69 | -1.85 | active site of protein A |
| AWP | PROTA | 126 | 126 | 126 | 41.89 | 2.69 | -1.85 | all of protein A |
| BWP | PROTB | 126 | 126 | 126 | 4.89 | -5.27 | 12.0 | all of protein B |

# 2 Scripts

## 2.1 `constants.py`

### 2.1.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
### Module-wide constants
# (c) Zarek Siegel
# v1 3/11/16

base_dir="/Users/zarek/GitHub/TaylorLab/zvina"
cluster_base_dir="/home/zsiegel"
ADT_dir="/Library/MGLTools/latest/MGLToolsPckgs/AutoDockTools"
MGL_py_bin="/Library/MGLTools/latest/bin/pythonsh"
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/constants.py

## 2.2 `new_grid_or_dock_entry.py`

### 2.2.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/usr/bin/env python

### Write a new entry to Dockings.csv or Gridboxes.csv
# (c) Zarek Siegel
# v1 3/10/16
# v1.1 3/11/16

import csv, re, argparse, time, datetime
from constants import *

class Timestamp():
  def __init__(self):
    time_obj = time.time()
    self.display = datetime.datetime.fromtimestamp(time_obj).strftime('%Y-%m-%
    d %H:%M:%S')
    self.eightdigit = datetime.datetime.fromtimestamp(time_obj).strftime('%Y%m
    %d')

def new_docking_entry():
  # Hello
  print("\n\tWelcome! This script will add an entry to Dockings.csv\n")

  # Dockings.csv columns:
  # Docking ID
  # Date
  # Protein
  # Protein File
  # Ligset
  # Gridbox
  # Exhaustiveness
```

```
29  # Number of Models
30  # Number of CPUs
31  # Notes
32
33  # Going through each variable, taking user input
34  dock_input = raw_input("\t\tDocking identifier: ")
35  print("\t\t>>> dock set to: {}\n".format(dock_input))
36
37  current_time = Timestamp()
38  date_input = raw_input("\t\tDate (enter 'd' to default to {}): ".format(
      current_time.eightdigit))
39  if date_input == "d": date_input = current_time.eightdigit
40  print("\t\t>>> date set to: {}\n".format(date_input))
41
42  prot_input = raw_input("\t\tProtein: ")
43  print("\t\t>>> prot set to: {}\n".format(prot_input))
44
45  prot_file_input = raw_input("\t\tSpecific protein file (without the .pdbqt):
       ")
46  print("\t\t>>> prot_file set to: {}\n".format(prot_file_input))
47
48  ligset_input = raw_input("\t\tLigset identifier: ")
49  print("\t\t>>> ligset set to: {}\n".format(ligset_input))
50
51  box_input = raw_input("\t\tGridbox identifier: ")
52  print("\t\t>>> box set to: {}\n".format(box_input))
53
54  exhaust_input = raw_input("\t\tExhaustiveness: ")
55  print("\t\t>>> exhaust set to: {}\n".format(exhaust_input))
56
57  n_models_input = raw_input("\t\tNumber of models: ")
58  print("\t\t>>> n_models set to: {}\n".format(n_models_input))
59
60  n_cpus_input = raw_input("\t\tNumber of CPUs: ")
61  print("\t\t>>> n_cpus set to: {}\n".format(n_cpus_input))
62
63  notes_input = raw_input("\t\tAny notes: ")
64  if notes_input == "":
65    notes_input = "Entered by new_grid_or_dock_entry.py {}".format(
      current_time.display)
66  else:
67    notes_input = "{} (Entered by new_grid_or_dock_entry.py {})".format(
68      notes_input, current_time.display)
69  print("\t\t>>> notes set to: {}\n".format(notes_input))
70
```

```
  # New row as a dictionary
  new_row = {
    'Docking ID' : dock_input,
    'Date' : date_input,
    'Protein' : prot_input,
    'Protein File' : prot_file_input,
    'Ligset' : ligset_input,
    'Gridbox' : box_input,
    'Exhaustiveness' : exhaust_input,
    'Number of Models' : n_models_input,
    'Number of CPUs' : n_cpus_input,
    'Notes': notes_input
  }

  # Print confirmation of all entered variabled
  print("\t>>> The new row will be\n\n\
    Docking ID: {dock}\n\
    Date: {date}\n\
    Protein: {prot}\n\
    Protein File: {prot_file}\n\
    Ligset: {ligset}\n\
    Gridbox: {box}\n\
    Exhaustiveness: {exhaust}\n\
    Number of Models: {n_models}\n\
    Number of CPUs: {n_cpus}\n\
    Notes: {notes}\n".format(
              dock = dock_input,
              date = date_input,
              prot = prot_input,
              prot_file = prot_file_input,
              ligset = ligset_input,
              box = box_input,
              exhaust = exhaust_input,
              n_models = n_models_input,
              n_cpus = n_cpus_input,
              notes = notes_input
            )
  )

  # Don't write row without confirmation
  proceed = raw_input("\tWrite this as a new docking entry? [y/n] ")
#   print("\t{}".format(new_row))

  # If "y" is entered, write the row, otherwise don't
  if proceed == "y":
```

```
    dockings_csv = "{b_d}/Dockings.csv".format(b_d=base_dir)
    dockings_headers = ["Docking ID", "Date", "Protein", "Protein File",
      "Ligset", "Gridbox", "Exhaustiveness", "Number of Models",
      "Number of CPUs", "Notes"]
    with open(dockings_csv, 'a') as f:
      appender = csv.DictWriter(f, fieldnames=dockings_headers)
      appender.writerow(new_row)
    print("\n\t>>> New row appended to Dockings.csv:\n\topen {}\n".format(
    dockings_csv))
  else:
    print("\n\t>>> No docking entry written\n")

def new_gridbox_entry():
  # Hello
  print("\n\tWelcome! This script will add an entry to Gridboxes.csv\n")

  # Gridboxes.csv columns:
  # Gridbox Name
  # Protein File
  # Size in x-dimension
  # Size in y-dimension
  # Size in z-dimension
  # Center in x-dimension
  # Center in y-dimension
  # Center in z-dimension
  # Notes

  # Going through each variable, taking user input
  box_input = raw_input("\t\tGridbox Name: ")
  print("\t\t>>> box set to: {}\n".format(box_input))

  prot_file_input = raw_input("\t\tSpecific protein file (without the .pdbqt):
     ")
  print("\t\t>>> prot_file set to: {}\n".format(prot_file_input))

  box_size_x_input = raw_input("\t\tSize in x-dimension: ")
  print("\t\t>>> box_size_x set to: {}\n".format(box_size_x_input))

  box_size_y_input = raw_input("\t\tSize in y-dimension: ")
  print("\t\t>>> box_size_y set to: {}\n".format(box_size_y_input))

  box_size_z_input = raw_input("\t\tSize in z-dimension: ")
  print("\t\t>>> box_size_z set to: {}\n".format(box_size_z_input))

  box_center_x_input = raw_input("\t\tCenter in x-dimension: ")
```

```
159   print("\t\t>>> box_center_x set to: {}\n".format(box_center_x_input))
160
161   box_center_y_input = raw_input("\t\tCenter in y-dimension: ")
162   print("\t\t>>> box_center_y set to: {}\n".format(box_center_y_input))
163
164   box_center_z_input = raw_input("\t\tCenter in z-dimension: ")
165   print("\t\t>>> box_center_z set to: {}\n".format(box_center_z_input))
166
167   current_time = Timestamp()
168   notes_input = raw_input("\t\tAny notes: ")
169   if notes_input == "":
170     notes_input = "Entered by new_grid_or_dock_entry.py {}".format(
      current_time.display)
171   else:
172     notes_input = "{} (Entered by new_grid_or_dock_entry.py {})".format(
173       notes_input, current_time.display)
174   print("\t\t>>> notes set to: {}\n".format(notes_input))
175
176   # New row as a dictionary
177   new_row = {
178     'Gridbox Name' : box_input,
179     'Protein File' : prot_file_input,
180     'Size in x-dimension' : box_size_x_input,
181     'Size in y-dimension' : box_size_y_input,
182     'Size in z-dimension' : box_size_z_input,
183     'Center in x-dimension' : box_center_x_input,
184     'Center in y-dimension' : box_center_y_input,
185     'Center in z-dimension' : box_center_z_input,
186     'Notes' : notes_input
187   }
188
189   # Print confirmation of all entered variabled
190   print("\t>>> The new row will be\n\n\
191     Gridbox Name: {box}\n\
192     Protein File: {prot_file}\n\
193     Size in x-dimension: {box_size_x}\n\
194     Size in y-dimension: {box_size_y}\n\
195     Size in z-dimension: {box_size_z}\n\
196     Center in x-dimension: {box_center_x}\n\
197     Center in y-dimension: {box_center_y}\n\
198     Center in z-dimension: {box_center_z}\n\
199     Notes: {notes}\n".format(
200               box = box_input,
201               prot_file = prot_file_input,
202               box_size_x = box_size_x_input,
```

```
              box_size_y = box_size_y_input,
              box_size_z = box_size_z_input,
              box_center_x = box_center_x_input,
              box_center_y = box_center_y_input,
              box_center_z = box_center_z_input,
              notes = notes_input
            )
  )

  # Don't write row without confirmation
  proceed = raw_input("\tWrite this as a new grid box entry? [y/n] ")

  # If "y" is entered, write the row, otherwise don't
  if proceed == "y":
    gridboxes_csv = "{b_d}/Gridboxes.csv".format(b_d=base_dir)
    gridboxes_headers = ["Gridbox Name", "Protein File", "Size in x-dimension",
      "Size in y-dimension", "Size in z-dimension", "Center in x-dimension",
      "Center in y-dimension", "Center in z-dimension", "Notes"]
    with open(gridboxes_csv, 'a') as f:
      appender = csv.DictWriter(f, fieldnames=gridboxes_headers)
      appender.writerow(new_row)
    print("\n\t>>> New row appended to Gridboxes.csv:\n\topen {}\n".format(
    gridboxes_csv))
  else:
    print("\n\t>>> No grid box entry written\n")

# Stuff below is commented because this script is being used as a module

# def main():
#   parser = argparse.ArgumentParser(
#     description='Write a new entry to Dockings.csv or Gridboxes.csv')
#
#   parser.add_argument('-b', '--base_dir', metavar='BASE_DIR', type=str,
#     nargs=1,
#     help='The base directory containing Docking.csv and Gridboxes.csv')
#   parser.add_argument('-d', '--new_docking', action='store_true', default=
#     False,
#     help='New set of docking parameters (written to Dockings.csv)')
#   parser.add_argument('-g', '--new_gridbox', action='store_true', default=
#     False,
#     help='New set of grid box parameters (written to Gridboxes.csv)')
#
#   args = vars(parser.parse_args())
#   global base_dir
```

```
#   base_dir = str(args['base_dir'][0])
#   new_docking = args['new_docking']
#   new_gridbox = args['new_gridbox']
#
#   if new_docking: new_docking_entry()
#   elif new_gridbox: new_gridbox_entry()
#
# if __name__ == "__main__": main()
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/new_grid_or_dock_entry.py

## 2.3 `load_parameters.sh`

### 2.3.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/bin/bash

### Print parameters (originally load_parameters.sh)
# (c) Zarek Siegel
# v1 3/4/16
# v1.1 3/5/16
# v2 3/5/16
# v3 3/11/16

# Docking ID as required argument
# dock=$1

# Set scripts directory to the directory containing this script
scripts_dir="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
# Set base directory to the one containing scripts_dir
base_dir="$( cd $scripts_dir && cd .. )"
# Source # AutoDockTools Directory and MGLTools Python binary paths from
    constants.py
source $scripts_dir/constants.py
# CSV file with docking parameters
dockings_csv=$base_dir/Dockings.csv

# Define a function for looking
function look_up {
  parameter=$1 # argument taken is the column header
  cat $dockings_csv | # look in Dockings.csv
  # AWK script to look up parameter for docking in the CSV
  awk -v dock="$dock" -v parameter="$parameter" \
    'BEGIN{
      FS=","; # CSV
```

```
      dock_row=""; # declare global variables
      parameter_field="";
    }
    {
      if ($1 == dock) {
        dock_row=NR; # determine which row to look in
      }
    }
    NR==1{
        for (f=1; f<=NF; f++) {
          {
            if ($f == parameter) {
              parameter_field=f; # determine which row to look in
            }
          }
        }
    }
    NR==dock_row{print $parameter_field} # output the intersection
  '
}

# Source all relevant parameters
dock=$( look_up "Docking ID" )
date=$( look_up "Date" )
prot=$( look_up "Protein" )
prot_file=$( look_up "Protein File" )
ligset=$( look_up "Ligset" )
box=$( look_up "Gridbox" )
exhaust=$( look_up "Exhaustiveness" )
n_models=$( look_up "Number of Models" )
n_cpus=$( look_up "Number of CPUs" )

# Print all parameters
export dock=$dock
export date=$date
export prot=$prot
export prot_file=$prot_file
export ligset=$ligset
export box=$box
export exhaust=$exhaust
export n_models=$n_models
export n_cpus=$n_cpus
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/load_parameters.sh

## 2.4 `separate_vina_results.sh`

### 2.4.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/bin/bash

### process_VinaResult and a bit of organization
# (c) Zarek Siegel
# v1 3/5/16
# v1.2 3/6/16
# v2 3/6/16 (batch separation)
# v3 3/11/16

### Required input
dock=$1
# Set scripts directory to the directory containing this script
scripts_dir="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
# Set base directory to the one containing scripts_dir
base_dir="$( cd $scripts_dir && cd .. )"
# Source # AutoDockTools Directory and MGLTools Python binary paths from
    constants.py
source $scripts_dir/constants.py
# Location of process_VinaResult.py
pvr_py="$ADT_dir/Utilities24/process_VinaResult.py"

# Retrieve the parameters for this docking
source $base_dir/scripts/load_parameters.sh $dock

# Exit if already done
if [ -e $base_dir/$prot/$dock/processed_pdbqts/ ]; then
  echo "  ! Results already separated"
  echo "    ($prot/$dock/processed_pdbqts/ exists),"
  echo "  -> exiting this step"
  exit 1
```

```
fi

# Retrieve ligset list
ligset_list_txt=$base_dir/ligsets/$ligset/$ligset\_list.txt
ligset_list=$(for l in $(cat $ligset_list_txt); do echo $l; done)

# Relevant directories
result_pdbqts_dir=$base_dir/$prot/$dock/result_pdbqts/
processed_pdbqts_dir=$base_dir/$prot/$dock/processed_pdbqts/

# Create a directory for processed files
mkdir $processed_pdbqts_dir

# The actual process_VinaResult step
receptor_pdbqt=$base_dir/$prot/$prot_file.pdbqt
batch_size=20
# No batches
n_models=$(echo $n_models | sed 's/[^0-9]//')
if [[ "n_models" -le "$batch_size" ]]; then
  for lig in $ligset_list; do
    result_pdbqt=$result_pdbqts_dir/$dock\_$lig\_results.pdbqt
    processed_pdbqt_stem=$processed_pdbqts_dir/$dock\_$lig\_m
    $MGL_py_bin $pvr_py -r $receptor_pdbqt \
              -f $result_pdbqt \
              -o $processed_pdbqt_stem
    echo "  processed ligand $lig"
  done
# Batches
elif [[ "n_models" -gt "$batch_size" ]]; then
  n_batches=$(bc <<< "$n_models / $batch_size")
  for ((b=1;b<=$n_batches;b++)); do
    echo "  processing batch $b"
    for lig in $ligset_list; do
      result_pdbqt=$result_pdbqts_dir/$dock\.$b\_$lig\_results.pdbqt
      processed_pdbqt_stem=$processed_pdbqts_dir/$dock\.$b\_$lig\_m
      $MGL_py_bin $pvr_py -r $receptor_pdbqt \
                -f $result_pdbqt \
                -o $processed_pdbqt_stem
      # Rename the processed pdbqts
      for ((m=1;m<=$batch_size;m++)); do
        old_processed_pdbqt=$processed_pdbqts_dir/$dock\.$b\_$lig\_m$m.pdbqt
        new_m=$(bc <<< "(( $b - 1 ) * $batch_size ) + $m")
        new_processed_pdbqt=$processed_pdbqts_dir/$dock\_$lig\_m$new_m.pdbqt
        mv $old_processed_pdbqt $new_processed_pdbqt
      done
```

```
      echo "    processed ligand $lig"
    done
  done
else
  echo "! ! ! Error in batch processing (n_models is weird)"
fi

# *** check for results, prot.pdb, params
# *** check if already pvr'd
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/separate_vina_results.sh

## 2.5 `cleanup_processed_vina_results.sh`

### 2.5.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/bin/bash

### Converting and cleaning up processed vina result pdbqts
# (c) Zarek Siegel
# v1 3/5/16
# v1.2 3/6/16

### Required input
dock=$1
# Set scripts directory to the directory containing this script
scripts_dir="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
# Set base directory to the one containing scripts_dir
base_dir="$( cd $scripts_dir && cd .. )"
# Source # AutoDockTools Directory and MGLTools Python binary paths from
    constants.py
source $scripts_dir/constants.py
# Location of pdbqt_to_pdb
q2b_py="$ADT_dir/Utilities24/pdbqt_to_pdb.py"

# Retrieve the parameters for this docking
source $base_dir/scripts/load_parameters.sh $dock

# Relevant directories
processed_pdbqts_dir=$base_dir/$prot/$dock/processed_pdbqts
cleanedup_processed_pdbqts_dir=$base_dir/$prot/$dock/
    cleanedup_processed_pdbqts
processed_pdbs_dir=$base_dir/$prot/$dock/processed_pdbs

# Check if already done
if [ -d $processed_pdbs_dir ]; then
```

```
  echo "  ! Results already cleaned up (processed_pdbs exists), exiting this
    step"
  exit 1
fi

# Create a directory for cleaned up files and pdb converts
mkdir $cleanedup_processed_pdbqts_dir
mkdir $processed_pdbs_dir

# Retrieve ligset list
ligset_list_txt=$base_dir/ligsets/$ligset/$ligset\_list.txt
ligset_list=$(for l in $(cat $ligset_list_txt); do echo $l; done)

# The clean-up step
for lig in $ligset_list; do
  for ((m=1;m<=$n_models;m++)); do
    processed_pdbqt=$processed_pdbqts_dir/$dock\_$lig\_m$m.pdbqt
    cleanedup_processed_pdbqt=$cleanedup_processed_pdbqts_dir/$dock\_$lig\_m$m
    .pdbqt
    processed_pdb=$processed_pdbs_dir/$dock\_$lig\_m$m.pdb

    # The clean-up step
    cat $processed_pdbqt | \
      sed 's/^\(HETATM..........\)...../\1LIG L/g' \
      > $cleanedup_processed_pdbqt

    # The PDBQT > PDB Conversion step
    $MGL_py_bin $q2b_py -f $cleanedup_processed_pdbqt \
              -o $processed_pdb

    echo "---> processed ligand $lig model $m"
  done
done

# Overwrite pre-clean-up pvr'd pdbqts with cleaned up ones
rm -rf $processed_pdbqts_dir
mv $cleanedup_processed_pdbqts_dir $processed_pdbqts_dir
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/cleanup_processed_vina_results.sh

## 2.6 `parse_pdb.py`

### 2.6.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/usr/bin/env python

### Parsing data from processed pdbqt result files
# (c) Zarek Siegel
# v1 3/5/16

import re

### A class for residues and residue atoms
#   (input is a string of form 'RES123' or 'RES123_A1')

class Residue:
  def __init__(self, str):
    # String
    self.str = str
    # Atom & Residue String
    if re.search(r'^[A-Z]+[0-9]+$', self.str):
      self.atom = None
      self.res_str = self.str
    elif re.search(r'^[A-Z]+[0-9]+_.+$', self.str):
      self.atom = re.sub(r'^[A-Z]+[0-9]+_', '', self.str)
      self.res_str = re.sub(r'_[A-Z0-9]+$', '', self.str)
    else: self.atom = None
    # Residue Index
    self.resi = re.sub(r'^[A-Z]+|_?[^_]*$', '', self.str)
    try:
      self.resi = int(self.resi)
    except ValueError:
      self.resi = None
    # Residue Name
```

```
    self.resn = re.sub(r'[0-9]+_?[^_]*$', '', self.str)
    # Dictionary of Props
    self.dic = {'str' : self.str, 'res_str' : self.res_str,
      'resi' : self.resi, 'resn' : self.resn, 'atom' : self.atom}
  def __str__(self):
    return self.str

### A class for molecules, including ones with data from process_VinaResult.py
#   (input is a .pdb or .pdbqt file address which may or may not be pvr'd)

class Pdb:
  def get_pdb_coords(self):
    _coords = []
    for line in self.pdb_lines:
      if re.search('HETATM|ATOM', line) or (re.search('ATOM', line)):
        _dic = {
          'atomi' : int(line[6:11]),
          'atomn' : line[12:16].replace(" ", ""),
          'resn' : line[17:20].replace(" ", ""),
          'resi' : line[22:26].replace(" ", ""),
          'x' : float(line[30:38].replace(" ", "")),
          'y' : float(line[38:46].replace(" ", "")),
          'z' : float(line[46:54].replace(" ", "")),
          'xyz' : (float(line[30:38].replace(" ", "")),
            float(line[38:46].replace(" ", "")),
            float(line[46:54].replace(" ", "")) ),
          'atom_type' : line[76:78].replace(" ", ""),
          'charge' : line[78:80].replace(" ", "") # element
        }
        _coords.append(_dic)
    self.coords = _coords

  def get_pdbqt_coords(self):
    _coords = []
    for line in self.pdb_lines:
      if re.search('HETATM|ATOM', line) or (re.search('ATOM', line)):
        _dic = {
          'atomi' : int(line[6:11]),
          'atomn' : line[12:16].replace(" ", ""),
          'resn' : line[17:20].replace(" ", ""),
          'resi' : line[22:26].replace(" ", ""),
          'x' : float(line[30:38].replace(" ", "")),
          'y' : float(line[38:46].replace(" ", "")),
          'z' : float(line[46:54].replace(" ", "")),
          'xyz' : (float(line[30:38].replace(" ", "")),
```

```
            float(line[38:46].replace(" ", "")),
            float(line[46:54].replace(" ", "")) ),
          'charge' : line[70:76].replace(" ", ""), # partial charge
          'atom_type' : line[77:79].replace(" ", "") # AD4 atom type
        }
        _coords.append(_dic)
    self.coords = _coords

  def mine_pvr_data(self): # mine data from pvrd file
    contacts = []
    for line in self.pdb_lines:
      # Binding Energy
      if re.search('REMARK VINA RESULT: ', line):
        self.E = re.sub( r'^REMARK VINA RESULT:[ ]+|[ ]+[^ ]+[ ]+[^ ]+$' ,
          r'' , line.replace('\n', '')) # [23:31].replace(" ", ""))
        self.E = float(self.E)
      # RMSD Lower Bound
      if re.search('REMARK VINA RESULT: ', line):
        self.rmsd_lb = re.sub( r'^REMARK VINA RESULT:[ ]+[^ ]+[ ]+|[ ]+[^ ]+$'
    ,
          r'' , line.replace('\n', ''))
        self.rmsd_lb = float(self.rmsd_lb)
      # RMSD Upper Bound
      if re.search('REMARK VINA RESULT: ', line):
        self.rmsd_ub = re.sub( r'^REMARK VINA RESULT:[ ]+[^ ]+[ ]+[ ]+[^ ]+' ,
          r'' , line.replace('\n', ''))
        self.rmsd_ub = float(self.rmsd_ub)
      # Ligand Efficiency (whatever that means...)
      if re.search('USER  AD>  ligand efficiency', line):
        self.pvr_effic = re.sub( r'USER  AD>  ligand efficiency' ,
          r'' , line.replace('\n', ''))
        self.pvr_effic = float(self.pvr_effic)
      # Model Number
      if re.search(r'USER  AD> .+ of .+ MODELS', line):
        self.pvr_model = re.sub( r'USER  AD>| of [0-9]+ MODELS' ,
          r'' , line.replace('\n', ''))
        self.pvr_model = int(self.pvr_model)
      # Torsional Degrees of Freesom
      if re.search('REMARK .+ active torsions:', line):
        self.torsdof = re.sub( r'REMARK|active torsions:' ,
          r'' , line.replace('\n', ''))
        self.torsdof = int(self.torsdof)
      # Number of Contacts
      if re.search('USER  AD> macro_close_ats:', line):
        self.macro_close_ats = re.sub( r'USER  AD> macro_close_ats:' ,
```

```
          r'' , line.replace('\n', ''))
        self.macro_close_ats = int(self.macro_close_ats)
      # Contacts
      if re.search(r'^USER  AD> [^ ]+:[^ ]+:[^ ]+:[^ ,]+$', line):
        contacts.append(line.replace('\n', '').replace('USER  AD> ', ''))

    # Contacts Processing
    self.pvr_resis_objs = []
    self.pvr_resis = []
    self.pvr_resis_atoms = []
    for c in contacts:
      self.pvr_resis_objs.append(Residue(re.sub(r'^[^:]+:[^:]+:', '',
        c).replace(':', '_')))

    for r in self.pvr_resis_objs:
      if r.atom != None:
        self.pvr_resis_atoms.append(r.str)
        self.pvr_resis.append(r.res_str)
      else:
        self.pvr_resis_atoms.append(None)
        self.pvr_resis.append(r.res_str)

    self.pvr_resis_objs = list(set(self.pvr_resis_objs)) # remove duplicates
    self.pvr_resis = list(set(self.pvr_resis))
    self.pvr_resis_atoms = list(set(self.pvr_resis_atoms))

    self.pvr_data = {
      'E' : self.E,
      'rmsd_ub' : self.rmsd_ub,
      'rmsd_lb' : self.rmsd_lb,
      'pvr_resis' : self.pvr_resis,
      'pvr_resis_atoms' : self.pvr_resis_atoms,
      'pvr_resis_objs' : self.pvr_resis_objs,
      'torsdof' : self.torsdof,
      'macro_close_ats' : self.macro_close_ats,
      'pvr_model' : self.pvr_model
    }

  def get_types(self):
    # Detect if its been through ADT process_VinaResult.py
    self.is_pvrd = False # by default
    for line in self.pdb_lines:
      if re.search('REMARK VINA RESULT: ', line):
        self.mine_pvr_data()
        self.is_pvrd = True
```

```
165
166     # Determine file type PDB/PDBQT (they are slightly different)
167     if self.pdb_file_in[-5:] == 'pdbqt':
168       self.get_pdbqt_coords()
169       self.file_type = 'pdbqt'
170     elif self.pdb_file_in[-3:] == 'pdb':
171       self.get_pdb_coords()
172       self.file_type = 'pdb'
173     else:
174       print("!!! BAD FILETYPE !!!")
175
176   def __init__(self, pdb_file_in):
177     # Specify input file
178     self.pdb_file_in = pdb_file_in
179     # Try to read it, else error
180     try:
181       pdb_file_open = open(pdb_file_in)
182       with pdb_file_open as f:
183         self.pdb_lines = f.readlines()
184     except IOError:
185       print("! ! ! IOError")
186       pass
187     # Determine if PDB or PDBQT, and whether it has been through
      process_VinaResult.py
188     self.get_types() # this also mines the actual data
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/parse_pdb.py

## 2.7 `aiad_icpd.py`

### 2.7.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/usr/bin/env python

### Calculating average inter-atomic distance
# (c) Zarek Siegel
# v1 3/6/16

from parse_pdb import *
from math import sqrt
from numpy import mean

class Molecule():
  def list_coords(self):
    self.coord_triples = []
    for atom in self.pdb.coords:
      self.coord_triples.append(atom['xyz'])

  def get_centerpoint(self):
    x_coords = []
    y_coords = []
    z_coords = []
    for triple in self.coord_triples:
      x_coords.append(triple[0])
      y_coords.append(triple[1])
      z_coords.append(triple[2])
    self.centerpoint = (mean(x_coords), mean(y_coords), mean(z_coords))

  def __init__(self, pdb):
    self.pdb = pdb
    self.list_coords()
    self.get_centerpoint()
```

```
31
32 def threeD_distance(triple1, triple2):
33   x1 = triple1[0]
34   y1 = triple1[1]
35   z1 = triple1[2]
36   x2 = triple2[0]
37   y2 = triple2[1]
38   z2 = triple2[2]
39   distance = sqrt( ((x2 - x1)**2) + ((y2 - y1)**2) + ((z2 - z1)**2) )
40   return distance
41
42 def caclulate_aiad(pdb1, pdb2):
43   molc1 = Molecule(pdb1)
44   molc2 = Molecule(pdb2)
45   dist_list = []
46   for triple1 in molc1.coord_triples:
47     dists_from_t1 = []
48     for triple2 in molc2.coord_triples:
49       dist = threeD_distance(triple1, triple2)
50       dists_from_t1.append(dist)
51     dist_list.append(min(dists_from_t1))
52   return mean(dist_list)
53
54 def calculate_icpd(pdb1, pdb2):
55   molc1 = Molecule(pdb1)
56   molc2 = Molecule(pdb2)
57   return threeD_distance(molc1.centerpoint, molc2.centerpoint)
58
59 def main():
60   # m1_p = Pdb("/Users/zarek/lab/Docking/p300/p27/res_pdbqts_cleaned/p27_s3_m4
       .pdbqt")
61 #   m2_p = Pdb("/Users/zarek/lab/Docking/p300/p27/res_pdbqts_cleaned/
       p27_s2_m142.pdbqt")
62 #   m1 = Molecule(m1_p)
63 #   m2 = Molecule(m2_p)
64 #   caclulate_aiad(m1_p, m2_p)
65   pass
66
67 if __name__ == "__main__": main()
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/aiad_icpd.py

## 2.8 `docking_data_assembly.py`

### 2.8.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/usr/bin/env python

### Putting together all parsed data from processed vina results
# (c) Zarek Siegel
# v1 3/5/16 (as assemble_alldata.py)
# v2 3/6/16

from __future__ import print_function
import csv, re, os, subprocess
import cPickle as pickle
from constants import *
from parse_pdb import *
from aiad_icpd import *

### A class for docking data, for sourcing, reading, and analyzing
class Docking():
  # Basic docking parameters are contained in CSV file in .../base/
    parameters_csvs/
  def load_parameters(self):
    # Basic docking parameters are stored in base_dir/Dockings.csv
    dockings_csv = "{}/Dockings.csv".format(base_dir)
    with open(dockings_csv) as f:
      reader = csv.DictReader(f)
      for row in reader:
        if row['Docking ID'] == self.dock:
          self.dockings_csv_row = row
    # Source the parameters as class attributes
    self.prot = self.dockings_csv_row['Protein']
    self.prot_file = self.dockings_csv_row['Protein File']
    self.ligset = self.dockings_csv_row['Ligset']
```

```
    self.box = self.dockings_csv_row['Gridbox']
    self.exhaust = self.dockings_csv_row['Exhaustiveness']
    self.n_models = int(self.dockings_csv_row['Number of Models'])
    self.n_cpus = self.dockings_csv_row['Number of CPUs']

    # Grid box parameters are stored in base_dir/Gridboxes.csv
    gridboxes_csv = "{}/Gridboxes.csv".format(base_dir)
    with open(gridboxes_csv) as f:
      reader = csv.DictReader(f)
      for row in reader:
        if row['Gridbox Name'] == self.box and row['Protein File'] == self.
    prot_file:
          self.gridboxes_csv_row = row

    # Source the grid box parameters as class attributes
    self.box_center_x = self.gridboxes_csv_row['Center in x-dimension']
    self.box_center_y = self.gridboxes_csv_row['Center in y-dimension']
    self.box_center_z = self.gridboxes_csv_row['Center in z-dimension']
    self.box_size_x = self.gridboxes_csv_row['Size in x-dimension']
    self.box_size_y = self.gridboxes_csv_row['Size in y-dimension']
    self.box_size_z = self.gridboxes_csv_row['Size in z-dimension']

    # Some useful directories
    self.prot_dir = "{b_d}/{p}".format(b_d=base_dir, p=self.prot)
    self.dock_dir = "{b_d}/{p}/{d}".format(b_d=base_dir, p=self.prot, d=self.
    dock)

    print("---> Loaded docking parameters")

  # Retrieve the list of ligands in the set
  def get_ligset_list(self):
    ligset_list_txt = "{b_d}/ligsets/{ls}/{ls}_list.txt".format(
      b_d=base_dir,ls=self.ligset)
    ligset_list_txt_open = open(ligset_list_txt, 'r')
    with ligset_list_txt_open as f:
      self.ligset_list = f.read().splitlines()
    print("---> Retrieved ligset list")

  # Write the script for submission of vina jobs on the cluster
  def write_vina_submit(self):
    template =  "#BSUB -q hp12\n\
#BSUB -n {n_cpus}\n\
#BSUB -J vina_{subdock}\n\
\n\
# Text file with list of ligands (one on each line)\n\
```

```
ligset_list_txt={cluster_base_dir}/ligsets/{ligset}/{ligset}_list.txt\n\
\n\
# Create the docking and output directories\n\
mkdir {cluster_base_dir}/{prot}/{dock}/\n\
mkdir {cluster_base_dir}/{prot}/{dock}/result_pdbqts\n\
\n\
# Generate the list of ligands\n\
ligset_list=$(for l in $(cat $ligset_list_txt); do echo $l; done)\n\
\n\
# Vina command\n\
for lig in $ligset_list; do\n\
  /share/apps/autodock/autodock_vina_1_1_2_linux_x86/bin/vina \\\n\
  --receptor {cluster_base_dir}/{prot}/{prot_file}.pdbqt \\\n\
  --ligand {cluster_base_dir}/ligsets/{ligset}/pdbqts/$lig.pdbqt \\\n\
  --out {cluster_base_dir}/{prot}/{dock}/result_pdbqts/{subdock}_$lig\_results
    .pdbqt \\\n\
  --center_x {box_center_x} \\\n\
  --center_y {box_center_y} \\\n\
  --center_z {box_center_z} \\\n\
  --size_x {box_size_x} \\\n\
  --size_y {box_size_y} \\\n\
  --size_z {box_size_z} \\\n\
  --cpu {n_cpus} \\\n\
  --num_modes {n_models} \\\n\
  --exhaustiveness {exhaust}\n\
done".format(
      cluster_base_dir = cluster_base_dir,
      n_cpus = self.n_cpus,
      prot = self.prot,
      prot_file = self.prot_file,
      ligset = self.ligset,
      box_center_x = self.box_center_x,
      box_center_y = self.box_center_y,
      box_center_z = self.box_center_z,
      box_size_x = self.box_size_x,
      box_size_y = self.box_size_y,
      box_size_z = self.box_size_z,
      exhaust = self.exhaust,
      n_models = '{n_models}',
      dock = '{dock}',
      subdock = '{subdock}'
    )
    # (no need for batch submission)
    if self.n_models <= 20:
      template_filled = template.format(
```

```
        n_models = self.n_models, dock = self.dock, subdock = self.dock)
      vina_submit_sh = "{b_d}/vina_submit_shs/vina_submit_{d}.sh".format(
        b_d=base_dir, d=dock)
      with open(vina_submit_sh, 'w') as f:
        f.write(template_filled)
      print("---> Vina submission script for docking h11 has been created. It
    can be found at:")
      print("\t{}".format(vina_submit_sh))
    # (batch submission)
    elif self.n_models > 20:
      vina_submits_dir = "{b_d}/vina_submit_shs/vina_submits_{d}/".format(
        b_d=base_dir, d=dock)
      subprocess.call(['mkdir', vina_submits_dir])
      n_batches = self.n_models / 20
      for b in range(1, n_batches + 1):
        subdock = "{d}.{b}".format(d = dock, b = b)
        template_filled = template.format(
          n_models = 20, dock = self.dock, subdock = subdock)
        vina_submit_sh = "{v_s_d}/vina_submit_{sd}.sh".format(
          v_s_d = vina_submits_dir, sd = subdock)
        with open(vina_submit_sh, 'w') as f:
          f.write(template_filled)
      print("---> Vina submission scripts for docking h11 have been created.
    They can be found in:")
      print("\t{}".format(vina_submits_dir))
    else: print("! ! ! bad n_models")

  ### AFTER DOCKING

  # Mine Vina result for data (actual mining is in parse_pdb.py, acting via
    the Pdb class)
  def assemble_dic(self):
    self.data_dic = {}
    self.keys = []
    for lig in self.ligset_list:
      for m in range(1, int(self.parameters['n_models'])+1):
        processed_pdbqt = "{d_d}/processed_pdbqts/{d}_{lig}_m{m}.pdbqt".format
    (
          d_d=self.dock_dir, d=dock, lig=lig, m=m)
        pose = Pdb(processed_pdbqt)
        key = "{}_{}_m{}".format(dock, lig, m)
        try:
          pose_dic = {
            'key' : key,
            'E' : pose.E,
```

```
            'rmsd_ub' : pose.rmsd_ub,
            'rmsd_lb' : pose.rmsd_lb,
            'pvr_resis' : pose.pvr_resis,
            'pvr_resis_atoms' : pose.pvr_resis_atoms,
            'pvr_resis_objs' : pose.pvr_resis_objs,
            'torsdof' : pose.torsdof,
            'pvr_n_contacts' : pose.macro_close_ats,
            'pvr_model' : pose.pvr_model,
            'pvr_effic' : pose.pvr_effic,
            'coords' : pose.coords,
            'lig' : lig,
            'model' : m,
            'pvr_obj' : pose,
            'pdb_address' : re.sub('pdbqt', 'pdb', processed_pdbqt)
          }
        except AttributeError:
          print("! ! ! pose {} failed, check for the processed PDBQT".format(
    key))
        self.data_dic[key] = pose_dic
        self.keys.append(key)
    self.is_assembled = True
    print("---> Data dictionary created with data from process_VinaResult.py")

  # Create a list of the binding sites previously prepared for the protein
  def get_binding_sites_list(self):
    if not self.is_assembled: self.assemble_dic()

    binding_sites_dir = "{b_d}/binding_sites/{p_f}".format(
      b_d=base_dir, p_f=self.prot_file)
    self.binding_sites_list = []
    self.binding_sites_objs = {}
    for root, dirs, files in os.walk(binding_sites_dir):
      for file in files:
        self.binding_sites_list.append(re.sub('.pdb', '', file))
        self.binding_sites_objs[re.sub('.pdb', '', file)] = (Pdb("{}{}".format
    (root, file)))

    self.bs_resis_lists = {}
    self.bs_resis_atoms_lists = {}

    for bs, bso in self.binding_sites_objs.items():
      bs_resis = []
      bs_resis_atoms = []
      for atom in bso.coords:
        bs_resis.append("{}{}".format(atom['resn'], atom['resi']))
```

```
201         bs_resis_atoms.append("{}{}_{}".format(atom['resn'], atom['resi'],
    atom['atomn']))
202       self.bs_resis_lists[bs] = list(set(bs_resis))
203       self.bs_resis_atoms_lists[bs] = list(set(bs_resis_atoms))
204
205     self.is_bs_listed = True
206     print("---> Retrieved binding sites")
207
208   # Score the binding sites in terms of the residues they contact
209   # relative to those contained in the reference PDB
210   def score_binding_sites(self):
211     if not self.is_assembled: self.assemble_dic()
212     if not self.is_bs_listed: self.get_binding_sites_list()
213
214     for pose in self.data_dic:
215       for bs in self.binding_sites_list:
216         resis_union = ( set(self.bs_resis_lists[bs]) & set(self.data_dic[pose
    ]['pvr_resis']) )
217         self.data_dic[pose]["{}_fraction".format(bs)] = float(len(resis_union)
    ) / float(len(self.bs_resis_lists[bs]))
218         resis_atoms_union = ( set(self.bs_resis_atoms_lists[bs]) & set(self.
    data_dic[pose]['pvr_resis_atoms']) )
219         self.data_dic[pose]["{}_atoms_fraction".format(bs)] = float(len(
    resis_atoms_union)) / float(len(self.bs_resis_atoms_lists[bs]))
220
221     self.is_bs_scored = True
222     print("---> Scored binding sites")
223
224   # Score binding sites by average inter-atomic distance
225   # and inter-centerpoint difference
226   # Acting via aiad_icpd.py
227   def aiad_icpd_binding_sites(self):
228     if not self.is_assembled: self.assemble_dic()
229
230     for pose in self.data_dic:
231       for bs, bso in self.binding_sites_objs.items():
232         aiad = caclulate_aiad(self.data_dic[pose]['pvr_obj'], bso)
233         self.data_dic[pose]["{}_aiad".format(bs)] = aiad
234         icpd = calculate_icpd(self.data_dic[pose]['pvr_obj'], bso)
235         self.data_dic[pose]["{}_icpd".format(bs)] = icpd
236
237     self.are_aiad_icpd_calcd = True
238     print("---> Calculated AIAD and ICPD")
239
240   # Add attributes for whether the ligand contacts each residue of the protein
```

```
241   def assess_all_resis(self):
242     if not self.is_assembled: self.assemble_dic()
243
244     self.prot_pdbqt = "{p_d}/{p_f}.pdbqt".format(p_d=self.prot_dir, p_f=self.
      prot_file)
245     self.prot_obj = Pdb(self.prot_pdbqt)
246     self.prot_resis_list = []
247 #     self.prot_resis_atoms_list = []
248     for atom in self.prot_obj.coords:
249       self.prot_resis_list.append("{}{}".format(atom['resn'], atom['resi']))
250 #       self.prot_resis_atoms_list.append("{}{}_{}".format(atom['resn'], atom
      ['resi'], atom['atomn']))
251     self.prot_resis_list = list(set(self.prot_resis_list))
252 #     self.prot_resis_atoms_list = list(set(self.prot_resis_atoms_list))
253
254     for pose in self.data_dic:
255       for res in self.prot_resis_list:
256         if res in self.data_dic[pose]['pvr_resis']:
257           self.data_dic[pose][res] = 1
258         else:
259           self.data_dic[pose][res] = 0
260 #       for atom in self.prot_resis_atoms_list:
261 #         if atom in self.data_dic[pose]['pvr_resis_atoms']:
262 #           self.data_dic[pose][atom] = 1
263 #         else:
264 #           self.data_dic[pose][atom] = 0
265
266     self.are_all_resis_assessed = True
267     print("---> Residues contacts added to data dictionary")
268
269   # Output all the data mined and analyzed into a CSV file
270   def write_alldata_csv(self):
271     self.alldata_csv = "{d_d}/{d}_alldata.csv".format(d_d=self.dock_dir, d=
      dock)
272
273     # If the CSV already exists, read it in as a dictionary
274     if os.path.isfile(self.alldata_csv):
275       with open(self.alldata_csv) as f:
276         reader = csv.DictReader(f)
277         self.data_dic = {}
278         for row in reader:
279           self.data_dic[row['key']] = row
280     # If it doesn't, write it
281     else:
282       if not self.is_assembled: self.assemble_dic()
```

```
283       if not self.is_bs_listed: self.get_binding_sites_list()
284       if not self.is_bs_scored: self.score_binding_sites()
285       if not self.are_aiad_icpd_calcd: self.aiad_icpd_binding_sites()
286       if not self.are_all_resis_assessed: self.assess_all_resis()
287
288       fieldnames = ['key', 'lig', 'model', 'E', 'rmsd_lb', 'rmsd_ub',
289         'pvr_effic', 'pvr_n_contacts', 'torsdof', 'pdb_address']
290       for bs in self.binding_sites_list:
291         fieldnames.append("{}_fraction".format(bs))
292         fieldnames.append("{}_atoms_fraction".format(bs))
293         fieldnames.append("{}_aiad".format(bs))
294         fieldnames.append("{}_icpd".format(bs))
295       for res in self.prot_resis_list: fieldnames.append(res)
296       # for atom in self.prot_resis_atoms_list: fieldnames.append(atom)
297
298       with open(self.alldata_csv, 'w') as csvfile:
299         writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
300         writer.writeheader()
301         for pose in self.data_dic:
302           row = {}
303           for f in fieldnames:
304             row[f] = self.data_dic[pose][f]
305           writer.writerow(row)
306
307     self.is_csv_written = True
308     print("---> Completed alldata.csv is located at:\n\t{}".format(self.
        alldata_csv))
309
310   # Write another CSV that shows the AIAD values between all ligands
311   def cluster_poses(self):
312     if not self.is_assembled: self.assemble_dic()
313
314     self.clustering_csv = "{d_d}/{d}_clustering.csv".format(d_d=self.dock_dir,
        d=dock)
315     self.clustering_dic = {}
316
317     # If the CSV already exists, read it in as a dictionary
318     if os.path.isfile(self.clustering_csv):
319       with open(self.clustering_csv) as f:
320         reader = csv.DictReader(f)
321         for row in reader:
322           key = row['compared']
323           del row['compared']
324           self.clustering_dic[key] = row
325     # If it doesn't, write it
```

```
    else:
      c = 0
      print("---> Calculating AIAD between poses (for clustering)")
      for key1 in self.data_dic:
        self.clustering_dic[key1] = {}
        c += 1
        print("\t- calculated for {:25}{:<9}of{:>9}".format(key1,c,len(self.
    keys)))
        for key2 in self.data_dic:
          aiad12 = caclulate_aiad(self.data_dic[key1]['pvr_obj'], self.
    data_dic[key2]['pvr_obj'])
          self.clustering_dic[key1][key2] = aiad12

      fieldnames = ['compared'] + self.keys
      with open(self.clustering_csv, 'w') as csvfile:
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for key in self.keys:
          row = self.clustering_dic[key]
          row['compared'] = key
          writer.writerow(row)

    self.are_poses_clustered = True
    print("   > Completed clustering.csv is located at:\n\t{}".format(self.
    clustering_csv))

  # Save the data dictionary as a pickled file (i.e. in native python format)
  def save_pickled_docking_obj(self):
    self.pickled_docking_obj = "{d_d}/{d}.p".format(d_d=self.dock_dir, d=dock)
    pickle.dump(self, open(self.pickled_docking_obj, 'wb'))

    self.is_pickled = True
    print("---> Pickled docking object located at:\n\t{}".format(self.
    pickled_docking_obj))

  def __init__(self, d):
    global dock
    dock = d

    self.dock = dock
    self.load_parameters()
    self.get_ligset_list()

    self.is_assembled = False
    self.is_bs_listed = False
```

```
367     self.is_bs_scored = False
368     self.are_aiad_icpd_calcd = False
369     self.are_all_resis_assessed = False
370     self.is_csv_written = False
371     self.are_poses_clustered = False
372     self.is_pickled = False
373
374
375
376
377
378     # WOOOT!
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/docking_data_assembly.py

## 2.9 `pre_and_post_control.py`

### 2.9.1 Function

Transcompiler picojava pseudolanguage java ee regular expression syntactic sugar eclipse rust bug. Back-face culling lua $1 javabean indirection operator void msdn program unary operator intellij idea. Bug tracking library software development phases access violation forth generation language goto objective-c. Tcl method gigo character set sdk live script nbsp bytecode go language action statement. Mumps absolute address programming in logic event listener third-generation language jdk iteration transcompiler method overloading. Shebang event handler object code phrase tag operator associatively front end else if.

```
#!/usr/bin/env python

### One script to control them AAAAAALLLLL!!!!
# (c) Zarek Siegel
# v1 3/6/16

import argparse, subprocess
import new_grid_or_dock_entry
from constants import *

def main():
  print("")
  print("->-> hi")

  parser = argparse.ArgumentParser(description='Pre- and post-Vina file fun
    times')
  parser.add_argument('-d', '--dock', metavar='DOCK', type=str, nargs='?',
    help='the ID for this docking')

  parser.add_argument('-nd', '--new_docking', action='store_true', default=
    False,
    help='execute the R script to write the parameters.csv')
  parser.add_argument('-ng', '--new_gridbox', action='store_true', default=
    False,
    help='execute the R script to write the parameters.csv')
  parser.add_argument('-w', '--write_params', action='store_true', default=
    False,
    help='execute the R script to write the parameters.csv')
  parser.add_argument('-v', '--vina', action='store_true', default=False,
    help='write the Vina job submission script')
```

```
27
28    parser.add_argument('-s', '--separate', action='store_true', default=False,
29      help='execute the bash script to separate row Vina results')
30    parser.add_argument('-n', '--clean', action='store_true', default=False,
31      help='execute the bash script to clean up processed Vina results')
32
33    parser.add_argument('-c', '--csv', action='store_true', default=False,
34      help='generate and save the alldata CSV file')
35    parser.add_argument('-l', '--cluster', action='store_true', default=False,
36      help='create cluster CSV files (all lig x lig AIADs)')
37    parser.add_argument('-i', '--pickle', action='store_true', default=False,
38      help='save the docking as a pickled object')
39
40
41    parser.add_argument('-o', '--post_proc', action='store_true', default=False,
42      help='Perform all post-processing steps (separate, clean, csv, cluster,
      pickle)')
43
44    args = vars(parser.parse_args())
45    #
46    new_docking = args['new_docking']
47    new_gridbox = args['new_gridbox']
48    write_params = args['write_params']
49    vina = args['vina']
50    #
51    separate = args['separate']
52    clean = args['clean']
53    #
54    csv = args['csv']
55    cluster = args['cluster']
56    pickle = args['pickle']
57
58    post_proc = args['post_proc']
59
60  #   if write_params:
61  #     print("---> Parameters CSV for docking h11 has been created. It can be
      found at:")
62  #     subprocess.call(["./write_params_csv.R", dock, base_dir])
63
64    if new_docking:
65      print("---> Write new set of docking parameters to Dockings.csv...")
66      new_grid_or_dock_entry.new_docking_entry()
67    if new_gridbox:
68      print("---> Write new set of grid box parameters to Gridboxes.csv...")
69      new_grid_or_dock_entry.new_gridbox_entry()
```

```
70
71    if separate or clean or vina or csv or cluster or pickle or post_proc:
72      dock = str(args['dock'])
73
74    if separate:
75      print("---> Processing raw Vina output PDBQTs")
76      subprocess.call(["{b_d}/scripts/separate_vina_results.sh".format(b_d=
        base_dir), dock])
77    if clean:
78      print("---> Cleaning up processed PDBQTs and converting to PDBs")
79      subprocess.call(["{b_d}/scripts/cleanup_processed_vina_results.sh".format(
        b_d=base_dir), dock])
80
81    if vina or csv or cluster or pickle or post_proc:
82      from docking_data_assembly import Docking
83      d = Docking(dock)
84
85    if vina:
86      print("---> Writing Vina submission script")
87      d.write_vina_submit()
88
89    if csv: d.write_alldata_csv()
90    if cluster: d.cluster_poses()
91    if pickle: d.save_pickled_docking_obj()
92
93    if post_proc:
94      print("---> Processing raw Vina output PDBQTs")
95      subprocess.call(["{b_d}/scripts/separate_vina_results.sh".format(b_d=
        base_dir),
96        dock, base_dir, ADT_dir, MGL_py_bin])
97      print("---> Cleaning up processed PDBQTs and converting to PDBs")
98      subprocess.call(["{b_d}/scripts/cleanup_processed_vina_results.sh".format(
        b_d=base_dir),
99        dock, base_dir, ADT_dir, MGL_py_bin])
100     d.write_alldata_csv()
101     d.cluster_poses()
102     d.save_pickled_docking_obj()
103
104
105   print("->-> All done!!!!!!!!!!!!!!!")
106   print("")
107
108 if __name__ == "__main__": main()
```

/Users/zarek/GitHub/TaylorLab/zvina/scripts/pre_and_post_control.py