In Silico High Throughput Enzyme Reactivity Screening Tutorial

MRH - Copenhagen University, v. 1.0
-For IRENE Internal use only-
martin@chem.ku.dk

March 5, 2012

1 Introduction

In this tutorial, enzyme reactivity screening of the reaction shown in Fig. 1 is illustrated. If the background is not relevant, one can start the tutorial with section 3.

2 Setup

2.1 Requirements

This tutorial assumes that version 11.038 or higher of MOPAC is installed, availability of the keywords `NORES` is required.
It also requires PyMOL and Python (version 2.5 or 2.6) to be installed. Instructions for the command line are in verbatim font (`>>> instruction`).
In order to analyse the output, GNUPlot is used. This is not critical and the user can employ other data analysis software as well.
The scripts for file generation are written and tested on Bash and require the `vim` editor to be installed.
The tutorial is based on the `1LBS` crystal structure (CS) from the protein databank.
It is assumed that the user has a working knowledge of basic computational chemistry principles and has a fair knowledge of how to operate at the command line.

2.2 Preparation

We start by inspecting the file `3-wt-opt.pdb`. This file is the starting point for the whole simulation. It is obtained from minimizing a representive part of the active site of the enzyme with the substrate in the covalently bound state. It was obtained from a MOPAC optimzation, but it can also origin from other simulations (eg. QM/MM or MD).
Two facts are of major importance for the variant generation procedure to succeed:

- For a given variant (or WT), all atoms of the system need to remain on the same line number of the structure file, otherwise the interpolation will fail (this has to be checked manually, especially in the cases where an atom changes chemically by forming or breaking bonds).
- The substrate and the water molecules need to be labeled correctly.

The residues, substrate and water molecules in the starting structure file need to be properly labeled, see the end of the file `3-wt-opt.pdb`. The substrate is characterized by the `LIG` identifier and was set to the arbitrary sequence ID number `550`. The water molecules all carry the same (arbitrary) sequence ID number 500, this is valid since in all variant structure files all water molecules are included and we do not need to address them individually (to paraphrase it in a more quantum chemical way, in this illustrative application they are *indistinguishable*).
Special care needs to be taken considering the reaction type (Fig. 1). In this case, a hydrogen atom (`H01`) is transfered from S105 to H224. In the file representing the tetrahedral intermediate, the transfered hydrogen (H^{γ}) needs to remain in the block of atom coordinates representing S105.
Below is an extract of the MOPAC input file of the TI (H01 is located on $N^{\epsilon 2}$ of H224, the first numbers

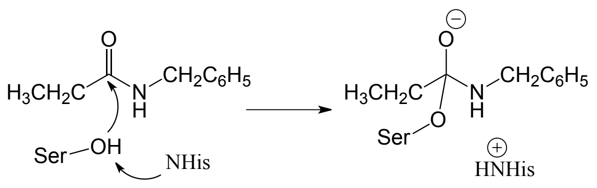

Figure 1: Nucleophilic attack of O^γ S105 on substrate carbonyl C and concerted hydrogen transfer to H224.

are the line numbers of the file and so by chemical terms is not part of the serine).

```
242 ATOM    240  N   SER A 105      57.5..
243 ATOM    241  CA  SER A 105      57.8..
244 ATOM    242  C   SER A 105      58.0..
245 ATOM    243  O   SER A 105      59.0..
246 ATOM    244  CB  SER A 105      56.8..
247 ATOM    245  OG  SER A 105      57.4..
248 ATOM    246  H01 SER A 105      57.0..
249 ATOM    247  H02 SER A 105      58.1..
250 ATOM    248  H03 SER A 105      58.8..
251 ATOM    249  H04 SER A 105      55.9..
```

Later in the tutorial, a technique for verifying the correct ordering of atoms is discussed.

2.3 File Name Conventions

Jobtypes:

- `cha`: MOPAC overall charge estimation.
- `scf`: MOPAC low-low level self-consistent field calculation used to transform from PDB to MOPAC input format providing `+1/+0` optimization flags (`MOZYME CHARGE=xy CUTOFF=3 1SCF`)
- `opt`: Geometry optimization
- `spe`: MOPAC PM6 single point energy calculation (not using `MOZYME`)

Data:

- `heat-X100Y.dat`: Heat in MOPAC format.
- `0000-X100Y-K200L.dat`: Normalized heat data for a double mutant.

3 Overview

3.1 Ultra-short summary

- Generate model for TI
- `$ mopac 3-wt-opt.mop`: Optimize WT TI
- Generate WT ES (i.e. `1-wt-opt.mop`) from `3-wt-opt.pdb`
- `$ mopac 1-wt-opt.mop`: Optimize WT ES
- `PyMOL> load 1-wt-opt.pdb`
- `PyMOL> run seq.py`
- `PyMOL> seq 1`
- `PyMOL> delete all`
- `PyMOL> load 3-wt-opt.pdb`
- `PyMOL> run seq.py`
- `PyMOL> seq 3`
- In `vsc.py`, set `obj = '1-wt-opt.pdb'` and adjust `mutations` dictionary
- `$ python vsc.py 1`
- In `vsc.py`, set `obj = '3-wt-opt.pdb'` and adjust `mutations` dictionary
- `$ python vsc.py 3`
- Fix side chain fragments (this step takes longest)
- Adjust `avf.py`: Enter correct back-bone sequence in `chain` list; confirm that water and substrate are correctly entered in the list
- Enter the variants in the `vars` of the `avf.py` script - `$ python avf.py 1`: Generate the reactant state variant assembly script (`seq.sh`) - `$ bash seq.sh`: Assemble the variant reactant state structure files - `$ python avf.py 3`: Generate the TI state variant assembly script (`seq.sh`) - `$ bash seq.sh`: Assemble the variant TI state structure files
- Run code in section 4.6

4 Step by Step Instructions

4.1 Optimization of wild-type (WT)

`3-wt-opt.mop`: The initial structure is from the crystal structure of the protein. The substrate was placed manually. The size of the model depends on the residues to be mutated.

4.2 Generation of WT enyzme substrate complex

1-wt-opt.mop: Copy the output PDB file from the optimization of 3-wt-opt.mop (above) to a new file called 1-wt-ini.pdb. Remove the substrate atoms and replace by atoms corresponding to planar (non-covalently bound) substrate at the end of the file. The planar substrate has to be modeled manually, e.g. using the Avogadro program. In the present example, the carbonyl carbon of the substrate is around 3.4Å away from the O$^{\gamma}$ of S105. Open the new file in PyMOL and translate the hydrogen H01 from the N$^{\epsilon2}$ of H224 towards O$^{\gamma}$ of S105 (without deleting and readding, since this changes the atom ordering). Once the distane of the oxygen to hydrogen is decreased down to around 1.0Å, select only the hydrogen and save it to a separate file (h01.pdb). Append the ATOM line of the file h01.pdb to the file 1-wt-ini.pdb. Open the file 1-wt-ini.pdb with an editor and find the appended hydrogen on the last line of the file. Cut the line and find the line H01 SER A 105 (this is the hydrogen still placed on the nitrogen). Replace the old hydrogen by the new one, save and exit the file. Copy the 1-wt-ini.pdb file to a file called 1-wt-opt.mop and add the appropriate keywords. This is the input for the optimization of the ES complex.
To check that the ordering of atoms did not change, run the following command:

```
python comframes.py 1-wt-ini.pdb 3-wt-opt.pdb wt
```

This will produce a file called polation-wt.pdb. The most common reason for error messages is that TER, END, CONECT or HEADER statements are still present in any of the PDB files, these need to be removed. If error messages continue to appear, make sure that both files contain exactly the same number of atoms (i.e line numbers). The script writes a file with 10 models, this can be viewed in PyMOL. With the arrow-keys it is possible to step forward and backward between the interpolation frames. Enter show as sticks on the PyMOL command line and confirm that all atoms are interpolating correctly.
When it is confirmed that the interpolation is correct, start the optimization with mopac 1-wt-opt.mop.

4.3 Generation of WT back-bone sequence files

The optimized structures of the end points of the reaction are sequenced into seq-*.pdb files which are used together with frag-*.pdb files to assemble structures of variants.
Load the file 3-wt-opt.pdb in PyMOL. Run the script seq_files.py in PyMOL by entering run seq_files.py on the PyMOL command line followed by seq 3, where '3' corresponds to the state of the reaction (1: ES complex, 3: TI).

```
PyMOL> run seq_files.py
PyMOL> seq {1,3}
```

In the above, again, only either '1' or '3' is entered. Every amino acid of the protein has been written to a separate file. Also the waters and the substrate are written to separate files (seq-x500-3.pdb or seq-x550-3.pdb, respectively). The same is done for the ES state: delete all loaded objects from PyMOL, load the file 1-wt-opt.pdb, enter run seq_files.py and enter seq 1. After these steps, in total 2·57 seq-*-{1,3}.pdb files have been created.
In PyMOL, the generated files can be loaded and it can be confirmed that the complete protein is represented by the sequence files.

4.4 Generation of variant side-chain fragment files

Generation of frag-*-{1,3}.pdb: To generate the variant side-chain files, open PyMOL and change to the directory where the vsc.py script is located. In order to run the script, a variable needs to be set in the script. Set the 'obj' variable to the name of the file for which the variant side-chain files should be generated. Note, the 'obj'-file does not need to be loaded into PyMOL.

```
[...]
# PARAMETER:
# - mutations dictionary below
obj = '3-wt-opt.pdb'
[...]
```

Then in the script the mutations can be defined within the following part of the code. This is done with the dictionary called mutations in the code.

```
[...]
 50     mutations = {
 51       '140': ['ARG', 'LYS', 'ASN'],
 52       '141': ['ASN', 'GLN'],
 53       '189': ['ALA', 'LYS', 'TYR'],
 54        '38': ['HIS', 'ASN'],
 55        '39': ['ALA'],
 56        '40': ['GLY'],
 57        '41': ['SER']
 58        }
[...]
```

The residue to mutate is selected by the residue number (in quotes), followed by a Python list of amino acid identifiers (in quotes). A comma is added at the end of every mutation definition, except for the last one (line 57).

Then enter at the PyMOL prompt:

```
PyMOL> run vsc.py
```

followed by `PyMOL> frag 3` (when operating on the TI structure file). The variant side-chain fragments are generated in the local directory.
The writing of the files may take a couple of seconds. When the script is finished, a number of `frag-*-3.pdb` files are written. This is done for both the ES and TI state of the enzyme.
Notice, after changing a variable in the code of the `vsc.py` script, the command `run vsc.py` has to be entered again in PyMOL before the `frag 1` command is entered.
We then need to check if the files have the correct atom numbering. In order to do so, we interpolate all fragment files. But first, enter the following commands on the command line (this removes any `END` strings in the files).

```
for i in frag-*pdb
do
    vi -c "g/END/d -c "wq" $i
done
for i in seq-*pdb
do
    vi -c "g/END/d -c "wq" $i
done
```

We use then again the `comframes.py` script to do so, just within a `for`-loop. Enter the following command on the Bash command line:

```
for i in frag-*-1pdb
do
    name=${i/frag-/}
    name=${name/-1.pdb/}
    python comframes.py $i frag-$name-3.pdb $name
done
```

Notice, the `comframes.py` script can obviously also be applied to only two files (without any `for`-loop). This is very useful to check specific files and the user is very encouraged to check all files whenever they have changed.
This will produce a number of `polation-*.pdb` files, each containing an interpolation between the variant fragment of the initial and final state. The interpolation files should be loaded into PyMOL and inspected for correct interpolation using the arrow keys. Two problems can occur now. 1) The fragments are not the same rotamer, 2) A fragment side chain is unphysically close to another atom of the enzyme. In these cases, the fragment of one of the states has to be adjusted to be more in geometrical agreement with the other (the linear interpolation will then not have such drastic effects on the geometry of the intermediate frames), and it has to be made sure that none of the atoms are too close to other protein atoms.
In any of the above cases, the user needs to manually fix the fragments and save them. This takes effort, but it has to be done only once. Also, since the files are saved in PyMOL, it has to be verified that the atom numbering does not change or else it has to be synchronized again. Depending on which PyMOL version is used, the saving behaviour is somewhat different.
With both the corrected `frag-*.pdb` and the `seq-*pdb` files, the structure files of the variants can now be assembled.

4.5 Assembly of variant files

The assembly is carried out with two scripts: `avf.py` and `seq.sh`, where the second script is written by the first one.
In the `asf.py` script, the variants to be generated are entered. In the script, the back-bone sequence of the protein has to be entered in terms of the sequence files. It is important to make sure that also the water- and substrate-sequence file are in the list (preferably at the end).

```
[...]
# Available backbone sequence fragements.
chain =\
['seq-v37',\
 'seq-p38',\
 'seq-g39',\
 'seq-t40',\
 'seq-g41',\
 'seq-t42',\
 'seq-t43',\
[...]
 'seq-x500',
 'seq-x550']
[...]
```

When another protein is used (or a different size for the model is chosen) this list has to be adjusted properly. In the script, the variants are entered in a second list (see below).

```
[...]
vars=[
'G39A',
'T103G',
'G39A+T103G+L278A',
'G39A+W104F+L278A',
'G39A+T42G+T103G+W104F+D223G+A225K+I189Y',
'G39A+T42G+T103G+W104F+Q191R+D223G+A225K+I189Y'
       ]
[...]
```

Again, the variants are entered between quotes and separated by commata. Multi-variants are entered with +-signs between them. Any variant can be entered, for which a corresponding fragment file has been generated in the previous step. Other variants can be entered once their fragment files have been generated at any point later as well.
When the desired variants are entered, the script is executed with the command

```
python avf.py 1
```

where again the '1' stands for the respective reaction state. The scripts generates a Bash script called `seq.sh`. This file contains the information about which files to assemble in what order. It is not supposed to be edited by the user. The script is executed with the command

```
bash seq.sh
```

The previous two commands are obviously also entered but with '3' instead of '1' to generate the files of the TI state. This produces a number of `1-res-*pdb` and `3-res-*pdb` files which we use for the interpolation.
Now the structure file generation part is done and we can start to prepare the optimization jobs.

4.6 Optimization Setup

The constraint optimization requires two preparation steps: 1) Calculation of the overall charge of the molecule and 2) Setting of the constraint flags. Both are automated.
First, run the `intcha.py` script as below.

```
for i in 1-res-*pdb
do
    name=${i/1-res-/}
    name=${name/-ste-ini.pdb/}
    python intcha.py $i 3-res-$name-ste-ini.pdb $name
done
```

This produces for every variant 10 interpolation frames with a single keyword, `charges`. The calculation of the overall charge takes only very short time, the output of these jobs is used to automatically generate the MOPAC formatted input for the optimization. The calculation can be run by entering

```
for i in 1-3-cha-*mop
do
    mopac $i
done
```

The output files can be used by the next script, `cha2scf.sh`. Enter the following on the command line.

```
for i in 1-3-cha*out
do
    cha2scf.sh $i
done
```

For every `*cha*` file, a `scf` file has been generated where the appropriate charge has been automatically written into the keyword line. The `*scf*` files are executed by MOPAC and the output is used as input for the optimization. The script `scf2opt.sh` is used to transfer the MOPAC `1-3-scf*.arc` files to input files for the optimization.
In the next step the input for the optimization is generated. The `scf2opt.sh` copies the output of the `*scf*arc` files to `*opt*mop` files and sets the constraints for fixing the substrate carbonyl C and serine 105 oxygen along the reaction coordinate. This requires the script to identify the respective locations of these atoms in the file. The script therefore requires all `*scf*mop` and `scf*arc*` files to be in the working directory (a common error that might occur is a `File not found` message).

In the present case study, the S105 O^{γ} is located by a `grep` call within the script (it assumes the oxygen to be the O^{γ} of the last serine occuring in the protein model, in a larger model, this could be different. The substrate carbonyl C is located by direct substitution of the appropriate line number in the script (24 above the last line in this case). The parameter is entered at the start of the script and needs to be adjusted if other substrates are studied. Also, it is possible to change the optimization parameters for MOPAC. The parameters are defined in the last line of the `scf2opt.sh` script.

```
for i in 1-3-scf-*arc
do
    scf2opt.sh $i
done
```

Now, before submitting the calculations, we set the optimization flags of a number of side chains to '+0' to avoid large rearrangement in the optimization. This is done with the `fix.py` script. In the `fix.py` script, the residues to fix can be entered in the `residues_to_fix` list, the numbers correspond to their sequence ID from the crystal structure. Note, the residue numbers need to be entered inbetween quotes.

```
[...]
 # PARAMETERS:
 # Choose the side chains in the list below:
 residues_to_fix = [
                     '50',
                    '133',
                    '156',
                    '277',
                    '280'
                    ]
[...]
```

After choosing the appropriate side chains to constrain, the script is operated on all `1-3-opt-*.mop` files.

```
for i in 1-3-opt-*.mop
do
    python fix.py $i
done
```

If everything went correctly, we are now ready to submit the optimization jobs. Submit the jobs to your queing system.
Once the jobs are finished, the script `opt2spe.sh` is used to generate input files for PM6 single point energy calculations.

```
for i in 1-3-opt*arc
do
    opt2spe.sh $i
done
```

This generates single point energy input files which are submitted again. From these files, the energy profiles are obtained with the `profiles.py` script
This concludes this tutorial.

5 Index

5.1 Notations

- '`*`' means any number of letters or numbers. E.g. `seq-*-{1,3}.pdb`, any file starting with `seq-`, then any kind of string, e.g. `w104`, followed by *either* `1` or `3` and file extension `.pdb`.

- `PyMOL> seq {1,3}`: Choose between '`1`' or '`3`'.

5.2 Scripts

- `seq_files.py`: Save every amino acid from the back-bone in a `seq-*.pdb` file. Takes `{1,3}` as an argument. Run `frag` method with PyMOL
- `vsc.py`: Variant side chain generation. Writes `frag-*-{1,3}.pdb` files, run with Python
- `avf.py`: Assemble variant files. Writes `seq.sh` script. Run with Python
- `seq.sh`: Variant structure file sequence. Generated from `avf.py`. Run with Bash
- `comframes.py`: Takes `1-res-*.pdb`, `3-res-*.pdb` and `name` as arguments. Interpolates and combines. Run with Python
- `intcha.py`: Interpolates and generates separate files to calculate overall charge. Takes initial state, final state and name as arguments. Run with Python
- `cha2scf.sh`: Transfers output from `1-3-cha-*out` files to `1-3-1scf-*.mop` files. Run with Bash
- `scf2opt.sh`: Transfer output from `1-3-1scf-*arc` files to `1-3-opt-*.mop` files. Set parameters for which atoms are on the reaction coordinate. Run with Bash
- `fix.py`: Sets constraints for side chains that are not allowed to rearrange. Run with Python

- `opt2spe.sh`: Transfers output from `1-3-opt-*.arc` files to `1-3-spe-*.mop` files. Run with Bash
- `profiles.py`: Generates energy profiles using GNUPlot. Requires a `data.txt` file containing the `HEAT` from the `1-3-spe-*.arc` files. Run with Python