QTools

Miha Purg

Group seminar, 8.3.2017

Outline

Intro

- About Qtools
- Motivation

Hands-on exercises

- Installing Qtools
- Converting parameters
- Making a FEP file
- Generating inputs
- Mapping FEP/EVB runs
- Calibrating H_{12} and α_0
- Analysis and Plotting
- Group contributions

Conclusion

About Qtools

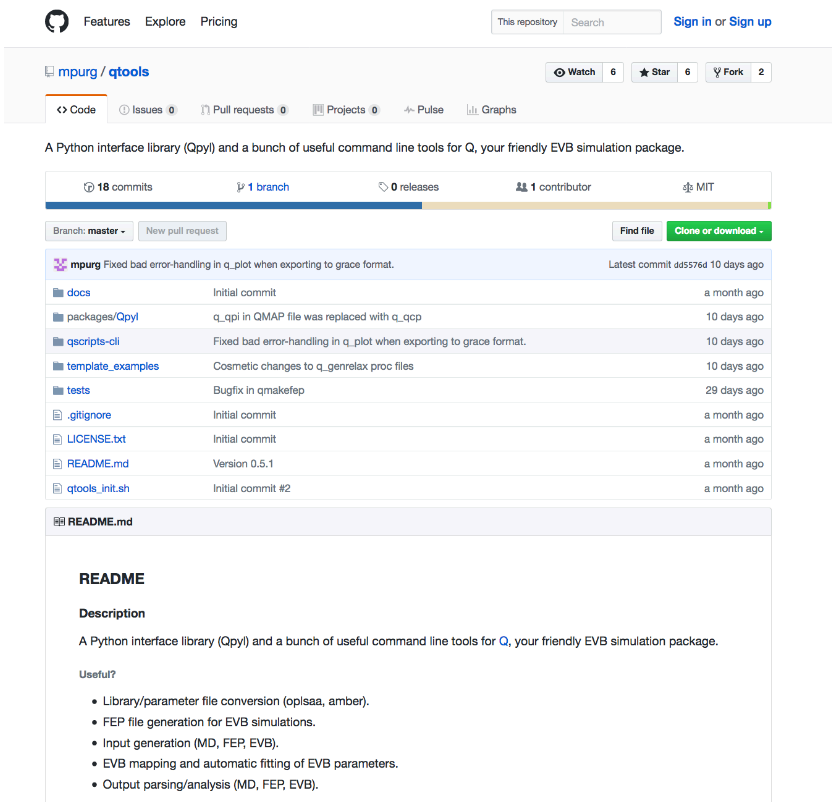
Qtools is a set of tools for Q!

Contains scripts for:

- System preparation
- Parameter conversion
- Input generation
- Mapping
- Calibration
- Analysis, etc.

Code details:

- Python2.7
- Object-oriented, modular
- Open-source (MIT), on GitHub
- Tests (pytest, scripts for CLI)
- No external requirements*
- Probably doesn't work on Windows



A Python interface library (Qpyl) and a bunch of useful command line tools for Q, your friendly EVB simulation package.

mpurg Fixed bad error-handling in q_plot when exporting to grace format. Latest commit dd5576d 10 days ago

docs	Initial commit	a month ago
packages/Qpyl	q_qpi in QMAP file was replaced with q_qcp	10 days ago
qscripts-cli	Fixed bad error-handling in q_plot when exporting to grace format.	10 days ago
template_examples	Cosmetic changes to q_genrelax proc files	10 days ago
tests	Bugfix in qmakefep	29 days ago
.gitignore	Initial commit	a month ago
LICENSE.txt	Initial commit	a month ago
README.md	Version 0.5.1	a month ago
qtools_init.sh	Initial commit #2	a month ago

README.md

README

Description

A Python interface library (Qpyl) and a bunch of useful command line tools for Q, your friendly EVB simulation package.

Useful?

- Library/parameter file conversion (oplsaa, amber).
- FEP file generation for EVB simulations.
- Input generation (MD, FEP, EVB).
- EVB mapping and automatic fitting of EVB parameters.
- Output parsing/analysis (MD, FEP, EVB).

https://github.com/mpurg/qtools

*not quite true

About Qtools

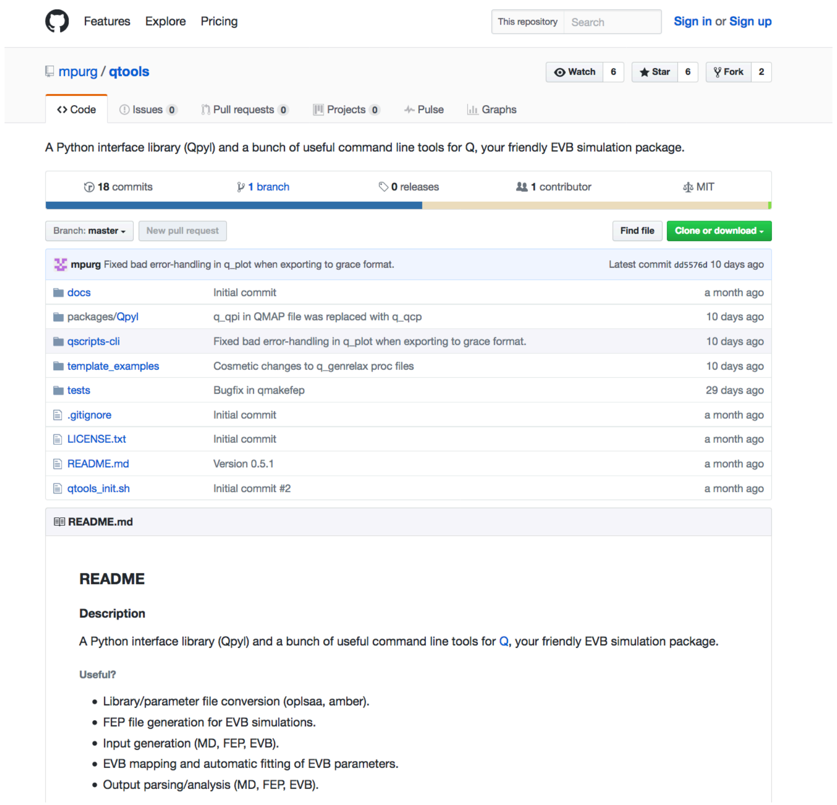
Qtools is a set of tools for Q!

Contains scripts for:

- System preparation
- Parameter conversion
- Input generation
- Mapping
- Calibration
- Analysis, etc.

Code details:

- Python2.7
- Object-oriented, modular
- Open-source (MIT), on GitHub
- Tests (pytest, scripts for CLI)
- No external requirements*
- Probably doesn't work on Windows



mpurg / qtools

A Python interface library (Qpyl) and a bunch of useful command line tools for Q, your friendly EVB simulation package.

18 commits | 1 branch | 0 releases | 1 contributor | MIT

mpurg Fixed bad error-handling in q_plot when exporting to grace format. Latest commit dd5576d 10 days ago

docs	Initial commit	a month ago
packages/Qpyl	q_qpi in QMAP file was replaced with q_qcp	10 days ago
qscripts-cli	Fixed bad error-handling in q_plot when exporting to grace format.	10 days ago
template_examples	Cosmetic changes to q_genrelax proc files	10 days ago
tests	Bugfix in qmakefep	29 days ago
.gitignore	Initial commit	a month ago
LICENSE.txt	Initial commit	a month ago
README.md	Version 0.5.1	a month ago
qtools_init.sh	Initial commit #2	a month ago

README.md

README

Description

A Python interface library (Qpyl) and a bunch of useful command line tools for Q, your friendly EVB simulation package.

Useful?

- Library/parameter file conversion (oplsaa, amber).
- FEP file generation for EVB simulations.
- Input generation (MD, FEP, EVB).
- EVB mapping and automatic fitting of EVB parameters.
- Output parsing/analysis (MD, FEP, EVB).

https://github.com/mpurg/qtools

*not quite true

About Qtools

Code structure: abstraction layers…?

Qpyl

- Python interface library for Q
- Objects and functions for:
  - Reading/writing inputs, libraries
  - Calling qfep5, qcalc5
- Can be implemented in CLI, web, GUI…
- For advanced users/developers

qscripts-cli

- Collection of command-line scripts for Q
- Implements Qpyl functionality
- **User-friendly** CLI
- (Good?) compromise between ease of use and level of control
- For Q-users.

About Qtools

Code structure: abstraction layers…?

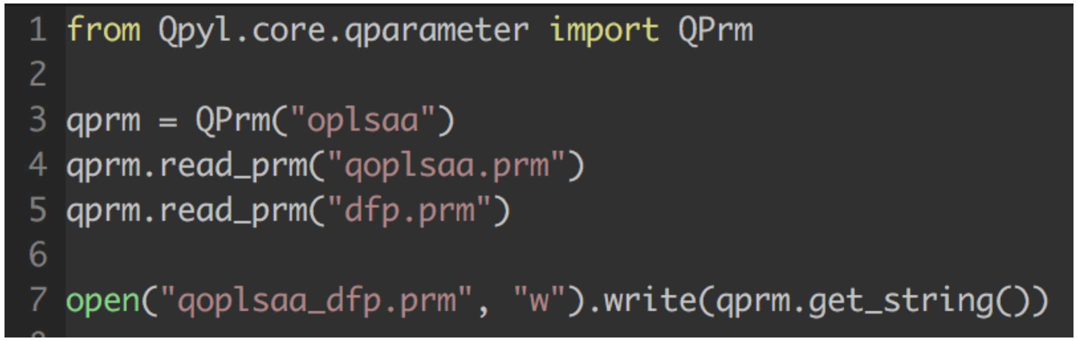
Qpyl

- Python interface library for Q
- Objects and functions for:
  - Reading/writing inputs, libraries
  - Calling qfep5, qcalc5
- Can be implemented in CLI, web, GUI…
- For advanced users/developers

qscripts-cli

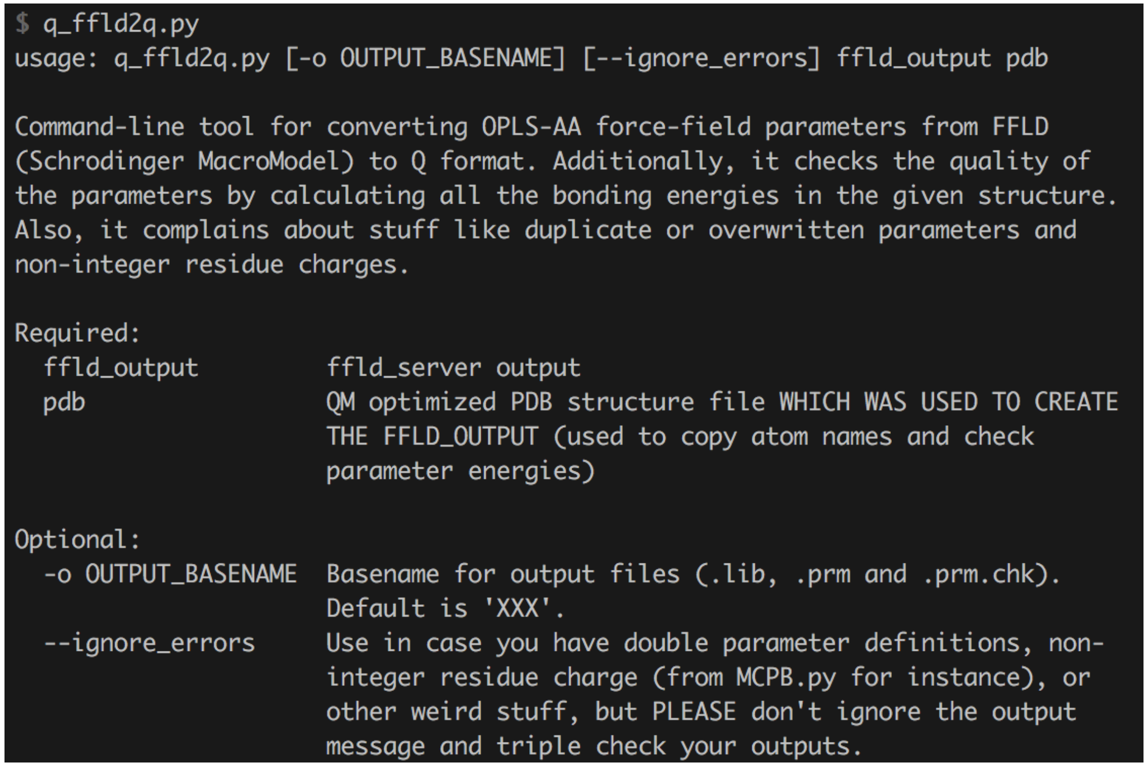
- Collection of command-line scripts for Q
- Implements Qpyl functionality
- **User-friendly** CLI
- (Good?) compromise between ease of use and level of control
- For Q-users.

Qpyl



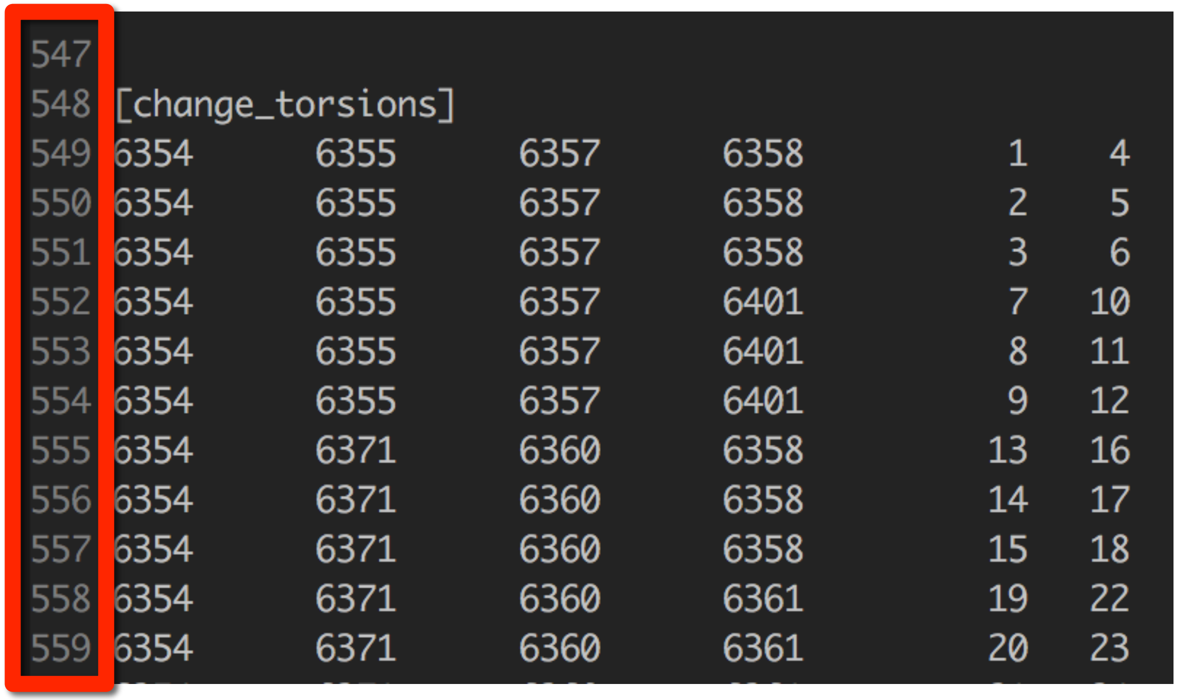
```python
from Qpyl.core.qparameter import QPrm

qprm = QPrm("oplsaa")
qprm.read_prm("qoplsaa.prm")
qprm.read_prm("dfp.prm")

open("qoplsaa_dfp.prm", "w").write(qprm.get_string())
```

qscripts-cli



```
$ q_ffld2q.py
usage: q_ffld2q.py [-o OUTPUT_BASENAME] [--ignore_errors] ffld_output pdb

Command-line tool for converting OPLS-AA force-field parameters from FFLD
(Schrodinger MacroModel) to Q format. Additionally, it checks the quality of
the parameters by calculating all the bonding energies in the given structure.
Also, it complains about stuff like duplicate or overwritten parameters and
non-integer residue charges.

Required:
  ffld_output         ffld_server output
  pdb                 QM optimized PDB structure file WHICH WAS USED TO CREATE
                      THE FFLD_OUTPUT (used to copy atom names and check
                      parameter energies)

Optional:
  -o OUTPUT_BASENAME  Basename for output files (.lib, .prm and .prm.chk).
                      Default is 'XXX'.
  --ignore_errors     Use in case you have double parameter definitions, non-
                      integer residue charge (from MCPB.py for instance), or
                      other weird stuff, but PLEASE don't ignore the output
                      message and triple check your outputs.
```

Motivation



```
547
548 [change_torsions]
549 6354        6355        6357        6358            1    4
550 6354        6355        6357        6358            2    5
551 6354        6355        6357        6358            3    6
552 6354        6355        6357        6401            7   10
553 6354        6355        6357        6401            8   11
554 6354        6355        6357        6401            9   12
555 6354        6371        6360        6358           13   16
556 6354        6371        6360        6358           14   17
557 6354        6371        6360        6358           15   18
558 6354        6371        6360        6361           19   22
559 6354        6371        6360        6361           20   23
```

FEP file for my first project, MAO-A…

More than 600 lines in FEP file

Motivation #2

Calibrating EVB parameters the <u>smart</u> way:

1) <u>manually</u> set H_{12} and α_0
2) run qfep5 to get $dG^\#$ and dG_0
3) if $dG^\#$ and dG_0 not equal to experiment, go back to step 1
4) realize you have just wasted an hour of your life

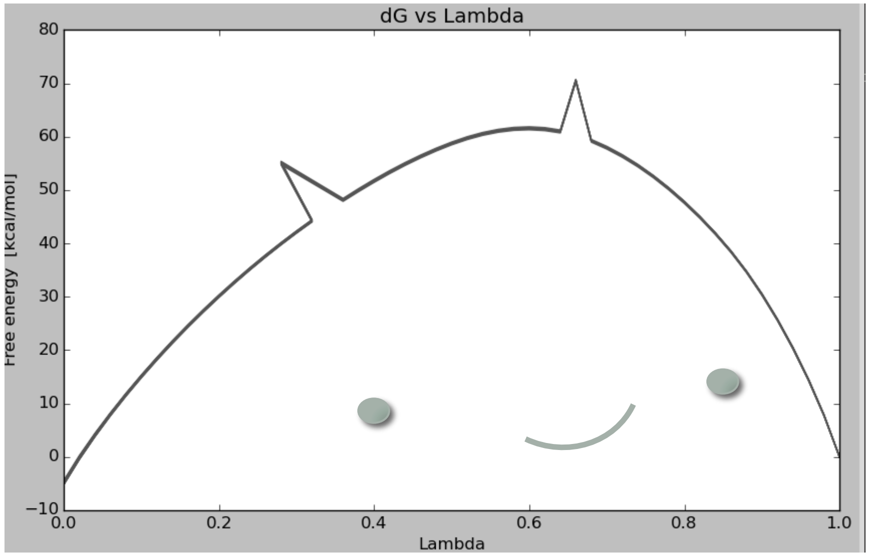
Motivation #3

Available tools/scripts for Q had some drawbacks…

- System specific
- A mess of Bash, Perl, Python, Fortran(??)
- No maintenance
- No tests

Motivation #4

Humans excel at producing typos/random mistakes.



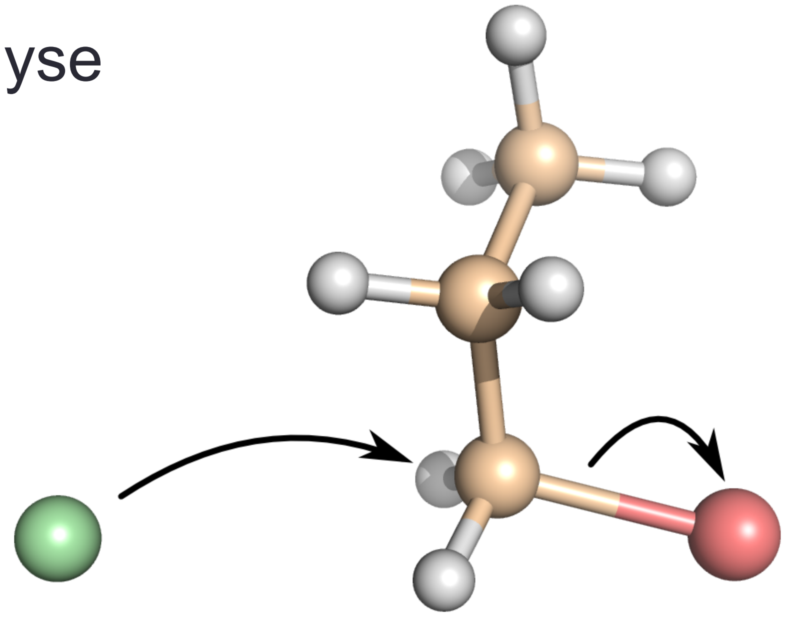
Manual input manipulation produces dG_{cat}

Not convinced? Let's try it out…

Let's prepare, run, map and analyse a simple model S_N2 reaction:

$Cl^- + Pr\text{-}Br \;\; \text{->} \;\; Br^- + Pr\text{-}Cl$

With the help of Qtools.



Tutorial preparation

1) Connect to the rackham (*ssh –X USER@rachkam.uppmax.uu.se*)
2) *$ cp --r /…/qtools_tutorial/ ~/*
3) *$ cd ~/qtools_tutorial/*
4) *$ source source_me.sh*
   (adds Q binaries to PATH and loads a Python module)
5) *$ tree -d*

Exercise: Qtools Installation

1) Open qtools GitHub page: https://github.com/mpurg/qtools
2) Follow instructions in **Installation** section of **README**.
3) Be sure to source your .bashrc after editing it.
4) Run *“q_automapper.py”*

Exercise: Converting parameters

q_ffld2q.py

- Converts FFLD parameters to Q format
- Checks quality of parameters

Creates:

- Q Library file
- Q Parameter file
- Parameter-check file

Requires:

- FFLD output
- Structure file (PDB)

Instructions:

1) cd into **0-topol/0-ff/0-prep/**
2) Run *"q_ffld2q.py"* and read the help
3) Use FFLD (**prb.ffld**) & PDB (**prb.pdb**)
4) Look at the outputs, are all parameters ok?

Exercise: Converting parameters

q_ffld2q.py

- Converts FFLD parameters to Q format
- Checks quality of parameters

Creates:

- Q Library file
- Q Parameter file
- Parameter-check file

Requires:

- FFLD output
- Structure file (PDB)

Instructions:

1) cd into **0-topol/0-ff/0-prep/**
2) Run *"q_ffld2q.py"* and read the help
3) Use FFLD (**prb.ffld**) & PDB (**prb.pdb**)
4) Look at the outputs, are all parameters ok?

Exercise: Making a FEP file

q_makefep.py

- Finds changes between states
- Generates a FEP file
- (also checks for lib/parm issues)

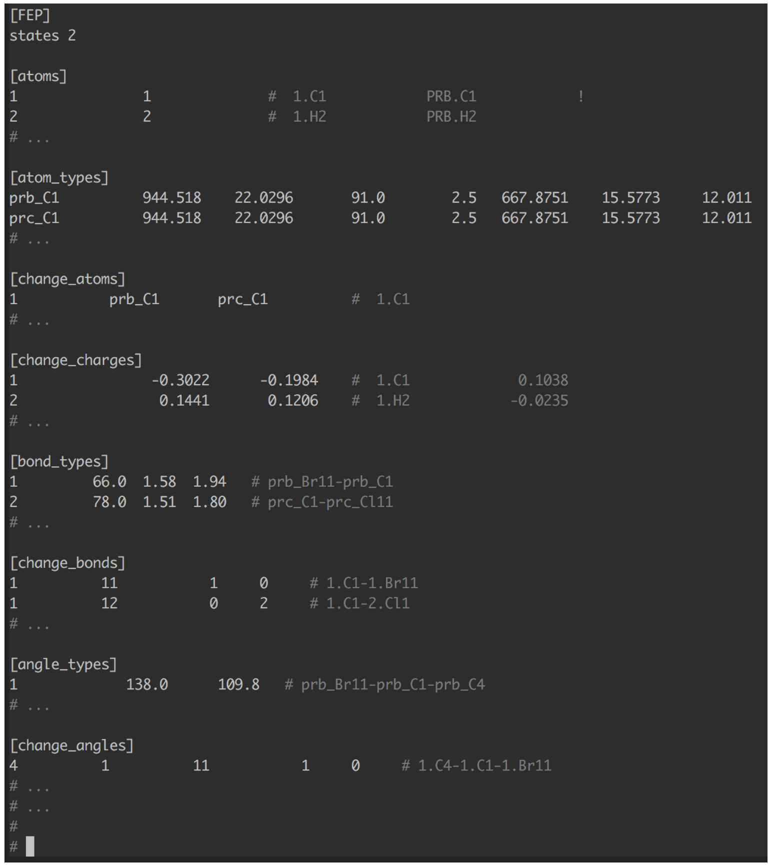
Creates:

- FEP file (template)

Requires:

- Library files (all states)
- Parameter files (all states)
- Structure file (state 1)
- Force-field type (oplsaa/amber)
- QMAP file

Pro-Br + Cl: FEP file (trimmed)



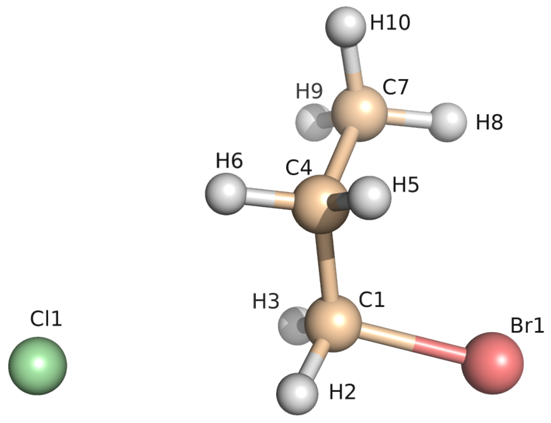
```
[FEP]
states 2

[atoms]
1               1              #  1.C1           PRB.C1           !
2               2              #  1.H2           PRB.H2
# ...

[atom_types]
prb_C1          944.518    22.0296       91.0        2.5    667.8751    15.5773     12.011
prc_C1          944.518    22.0296       91.0        2.5    667.8751    15.5773     12.011
# ...

[change_atoms]
1           prb_C1       prc_C1          #  1.C1
# ...

[change_charges]
1                -0.3022      -0.1984    #  1.C1             0.1038
2                 0.1441       0.1206    #  1.H2            -0.0235
# ...

[bond_types]
1         66.0  1.58  1.94   # prb_Br11-prb_C1
2         78.0  1.51  1.80   # prc_C1-prc_Cl11
# ...

[change_bonds]
1           11          1     0     # 1.C1-1.Br11
1           12          0     2     # 1.C1-2.Cl1
# ...

[angle_types]
1             138.0      109.8    # prb_Br11-prb_C1-prb_C4
# ...

[change_angles]
4           1          11           1     0     # 1.C4-1.C1-1.Br11
# ...
# ...
#
#
```

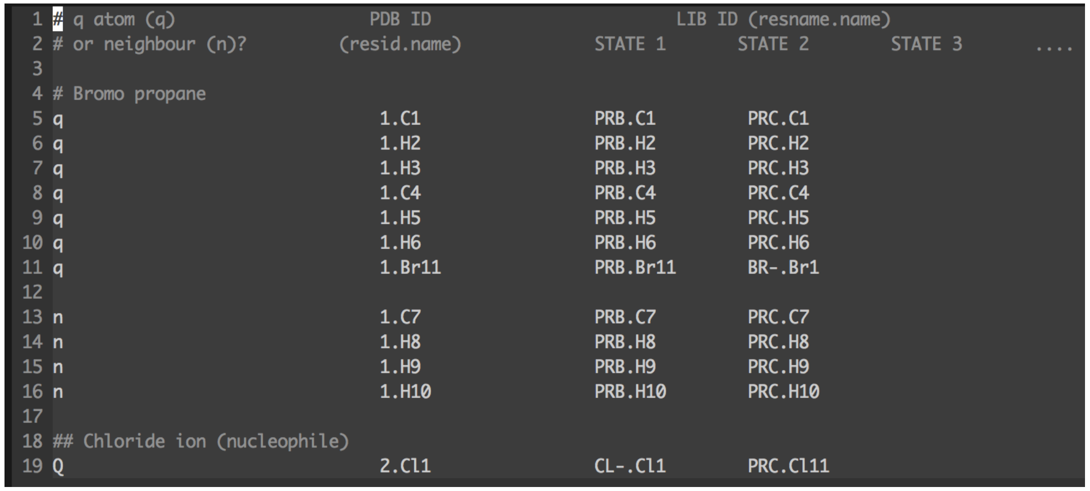
Exercise: Making a FEP file

q_makefep.py

1) cd into **0-topol/1-fep**
2) Have a look at QMAP file (**probr_cl.qmap)**
3) Run script without arguments
4) Generate FEP file using:

   - qmap file (**probr_cl.qmap**)
   - libs and prms in **../0-ff/**
   - PDB file (**../probr_cl_start.pdb**)
   - **oplsaa** forcefield
5) Dafuq Q, bad parms?





```
# q atom (q)                PDB ID                   LIB ID (resname.name)
# or neighbour (n)?       (resid.name)          STATE 1      STATE 2        STATE 3       ....

# Bromo propane
q                            1.C1               PRB.C1       PRC.C1
q                            1.H2               PRB.H2       PRC.H2
q                            1.H3               PRB.H3       PRC.H3
q                            1.C4               PRB.C4       PRC.C4
q                            1.H5               PRB.H5       PRC.H5
q                            1.H6               PRB.H6       PRC.H6
q                            1.Br11             PRB.Br11     BR-.Br1

n                            1.C7               PRB.C7       PRC.C7
n                            1.H8               PRB.H8       PRC.H8
n                            1.H9               PRB.H9       PRC.H9
n                            1.H10              PRB.H10      PRC.H10

## Chloride ion (nucleophile)
Q                            2.Cl1              CL-.Cl1      PRC.Cl11
```

Exercise: Generating MD inputs

q_genrelax.py

- Generates MD inputs

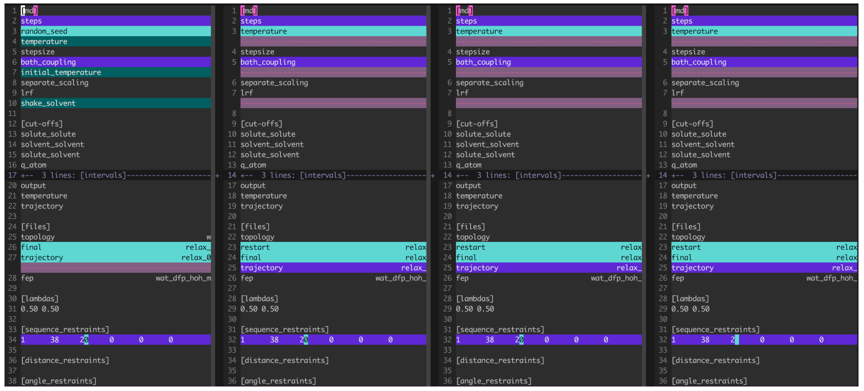
- Supports:
  - internal variables
  - atom placeholders
  - continuations

Creates:

- qdyn5 input files

Requires:

- Genrelax procedure file
- Topology
- FEP file (optional)
- PDB file (if placeholders are used)
- Run-script (optional)
- qdyn5 input from previous run (optional)



```
[md]
steps
random_seed
temperature
stepsize
bath_coupling
initial_temperature
separate_scaling
lrf
shake_solvent

[cut-offs]
solute_solute
solvent_solvent
solute_solvent
q_atom
+--  3 lines: [intervals]-------------------
output
temperature
trajectory

[files]
topology                                    w
final                                  relax_
trajectory                            relax_0
fep                               wat_dfp_hoh_m

[lambdas]
0.50 0.50

[sequence_restraints]
1       38      20      0       0       0

[distance_restraints]

[angle_restraints]
```

```
[md]
steps
temperature
stepsize
bath_coupling
separate_scaling
lrf

[cut-offs]
solute_solute
solvent_solvent
solute_solvent
q_atom
+--  3 lines: [intervals]------------------
output
temperature
trajectory

[files]
topology
restart                                 relax
final                                   relax
trajectory                             relax_
fep                               wat_dfp_hoh_

[lambdas]
0.50 0.50

[sequence_restraints]
1       38      20      0       0       0

[distance_restraints]

[angle_restraints]
```

```
[md]
steps
temperature
stepsize
bath_coupling
separate_scaling
lrf

[cut-offs]
solute_solute
solvent_solvent
solute_solvent
q_atom
+--  3 lines: [intervals]------------------
output
temperature
trajectory

[files]
topology
restart                                 relax
final                                   relax
trajectory                             relax_
fep                               wat_dfp_hoh_

[lambdas]
0.50 0.50

[sequence_restraints]
1       38      20      0       0       0

[distance_restraints]

[angle_restraints]
```

```
[md]
steps
temperature
stepsize
bath_coupling
separate_scaling
lrf

[cut-offs]
solute_solute
solvent_solvent
solute_solvent
q_atom
+--  3 lines: [intervals]------------------
output
temperature
trajectory

[files]
topology
restart                                 relax
final                                   relax
trajectory                             relax_
fep                               wat_dfp_hoh_

[lambdas]
0.50 0.50

[sequence_restraints]
1       38      2       0       0       0

[distance_restraints]

[angle_restraints]
```

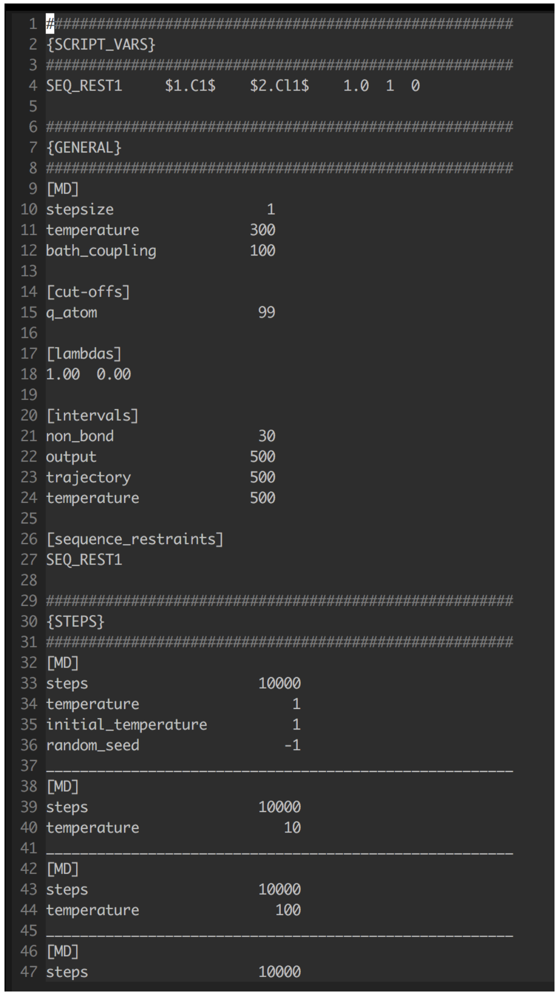
Manually comparing/modifying inputs in Vimdiff

Exercise: Generating MD inputs

q_genrelax.py

1) cd into **1-relax/**
2) Use procedure (**genrelax.proc**), topology and FEP file in the folder to create the inputs. (NOTE: bug hunting time)
3) Note the output from the script.
4) Check the inputs (comments), and run with:

   *“for i in relax*inp; do qdyn5_r8 $i > $i.log; done”*



```
#######################################################
{SCRIPT_VARS}
#######################################################
SEQ_REST1     $1.C1$    $2.Cl1$    1.0  1  0

#######################################################
{GENERAL}
#######################################################
[MD]
stepsize                  1
temperature             300
bath_coupling           100

[cut-offs]
q_atom                   99

[lambdas]
1.00  0.00

[intervals]
non_bond                 30
output                  500
trajectory              500
temperature             500

[sequence_restraints]
SEQ_REST1

#######################################################
{STEPS}
#######################################################
[MD]
steps                    10000
temperature                  1
initial_temperature          1
random_seed                 -1
_______________________________________________________
[MD]
steps                    10000
temperature                 10
_______________________________________________________
[MD]
steps                    10000
temperature                100
_______________________________________________________
[MD]
steps                    10000
```

Exercise: Generating MD/FEP inputs

q_genfeps.py

- Generates FEP inputs
- Automatically gets starting lambda and required files (fep, top, …)
- Frames and reps are easily defined (arguments)
- Can start from any lambda value (0.48)

Creates:

- qdyn5 input files

Requires:

- Genfeps procedure file
- Last qdyn5 input from relaxation
- PDB file (if placeholders are used)
- Run-script (optional)

Exercise: Generating MD/FEP inputs

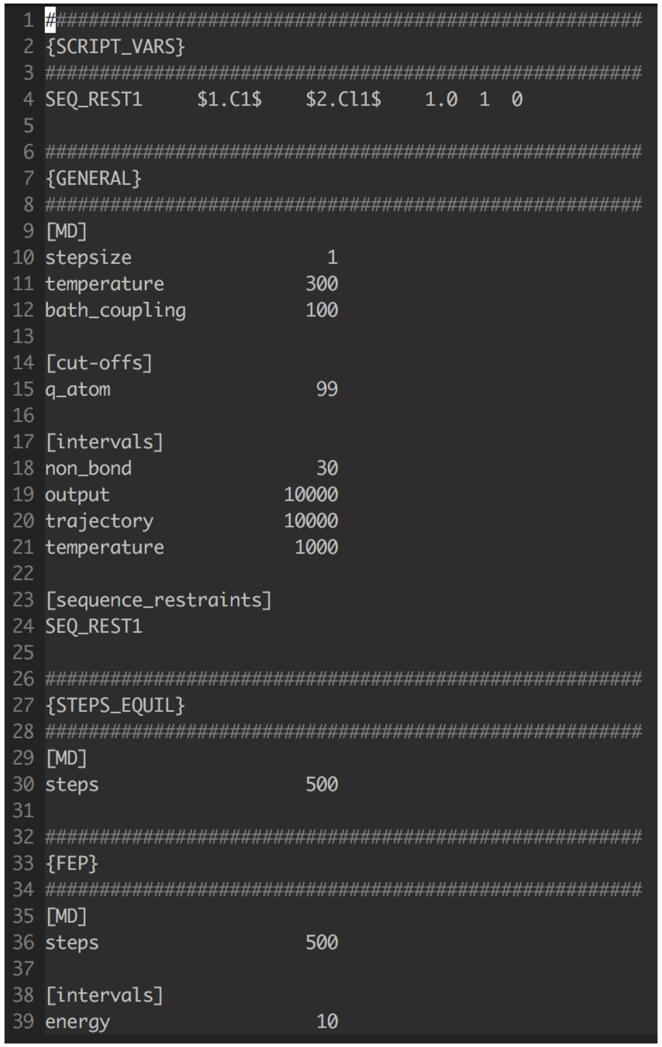
q_genfeps.py

1) cd into **2-fep/**

2) To make inputs, use:
   - procedure file (**genfeps.proc**)
   - last input from relaxation stage
   - **“relax”** as the restraint keyword
   - PDB-file in 0-topol/
   - run-script (**run_q_local.sh**)
   - **3** replicas

3) Note the output from the script.

4) Run: *“for i in rep*; do cd $i; ./run_q_local.sh; cd ../; done”*

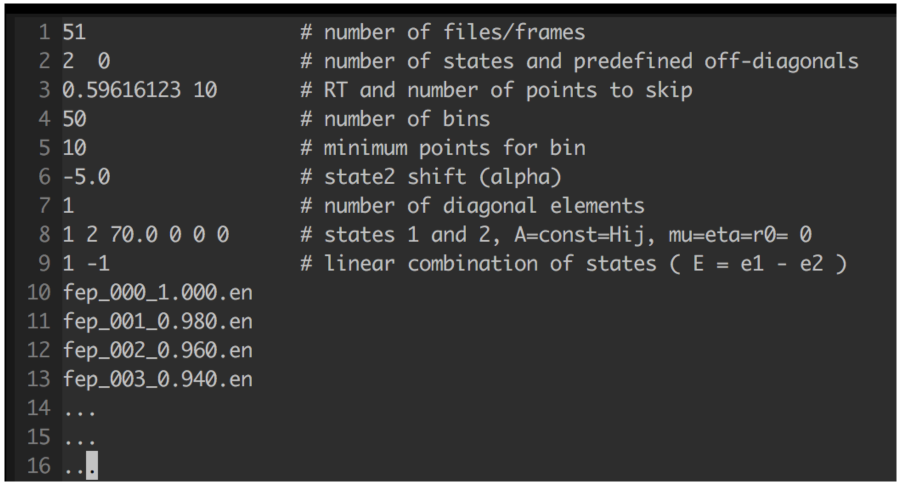


```
#############################################################
{SCRIPT_VARS}
#############################################################
SEQ_REST1     $1.C1$    $2.Cl1$    1.0  1  0

#############################################################
{GENERAL}
#############################################################
[MD]
stepsize                     1
temperature                300
bath_coupling              100

[cut-offs]
q_atom                      99

[intervals]
non_bond                    30
output                   10000
trajectory               10000
temperature               1000

[sequence_restraints]
SEQ_REST1

#############################################################
{STEPS_EQUIL}
#############################################################
[MD]
steps                      500

#############################################################
{FEP}
#############################################################
[MD]
steps                      500

[intervals]
energy                      10
```

Exercise: Mapping FEP/EVB runs

q_manual_labour.py *(not an actual script)*

1) Make **qfep.inp**
2) Run *“qfep5 < qfep.inp > qfep.out”*
3) Get dG profile from **qfep.out**
4) Calculate dG$^{\#}$ and dG$_0$



```
51                   # number of files/frames
2  0                 # number of states and predefined off-diagonals
0.59616123 10        # RT and number of points to skip
50                   # number of bins
10                   # minimum points for bin
-5.0                 # state2 shift (alpha)
1                    # number of diagonal elements
1 2 70.0 0 0 0       # states 1 and 2, A=const=Hij, mu=eta=r0= 0
1 -1                 # linear combination of states ( E = e1 - e2 )
fep_000_1.000.en
fep_001_0.980.en
fep_002_0.960.en
fep_003_0.940.en
...
...
...
```

Example qfep.inp

Exercise: Mapping FEP/EVB runs

q_mapper.py

- Automates free-energy calculations (with qfep5)
- Works on multiple runs (even in parallel)
- Mapping parameters are easily defined through arguments
- Provides stats on **$dG^{\#}$**, **dG_0** and **$dG(\lambda)$**
- Reports failures

Creates:

- q_mapper.log (statistics, details)
- qfep5 input file(s)
- qfep5 output file(s)

Requires:

- EVB/FEP simulation (with **q_enfiles.list**)
- H_{ij} and α
- qfep5 binary (defined in config file)

Exercise: Mapping FEP/EVB runs

q_mapper.py

1) cd into **2-fep/**
2) Use q_mapper.py to obtain dG#, dG0, with **Hij = 70.0** and **α = -5.0**
3) Note the output from the script.
4) Try to fix the issue by changing one of the mapping parameters (bins, skip, min, temp)

Exercise: Calibrating H_{ij} and α_0

q_automapper.py

- Finds H_{ij} and α_0 that reproduce reference $dG^{\#}$ & dG_0
- Works on multiple runs (means)
- Mapping parameters are easily defined through arguments
- Reports failures

Creates:

- q_automapper.log (statistics, details)
- qfep5 input file(s)
- qfep5 output file(s)

Requires:

- EVB/FEP simulation (with **q_enfiles.list**)
- Initial guess for H_{ij} and α
- Reference $dG^{\#}$ & dG_0
- qfep5 binary (defined in config file)

Exercise: Calibrating H_{ij} and α_0

q_automapper.py

1) cd into **2-fep/**

2) Fit the parameters to **$dG^{\#}$ = 12.345** and **dG_0 = -4.321**
(use your imagination for initial guess for Hij and alpha)

3) Note the output from the script

Exercise: Analysis

q_analysefeps.py

- Extracts data from qfep outputs
- Calculates:
  - $dG^{\#}$, dG_0, $dG(\lambda)$
  - LRA & reorg energies
- Produces statistics over all outputs
- Supports group-exclusions and QCP

Creates:

- q_analysefeps.log (statistics, details)
- qaf.PlotData.json (use with q_plot.py)

Requires:

- Mapped EVB/FEP simulation(s)

Exercise: Analysis

q_analysefeps.py

1) cd into **2-fep/**
2) Run *“q_analysefeps.py rep_*”*
3) Note the output

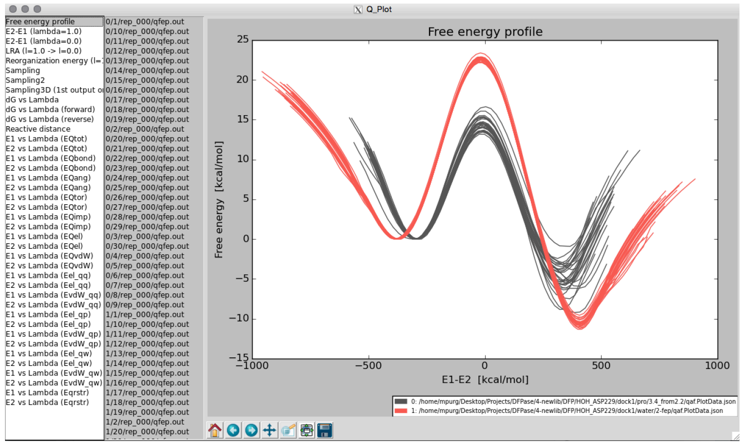
Exercise: Analysis

q_plot.py

- Graphical interface for visualizing data from:
  - q_analysefeps.py
  - q_analysedyns.py
  - q_calc.py
- Useful for debugging
- Supports multiple inputs (water, wt, mutant)
- Supports exporting to ASCII Grace format

Requires (either):

- qaf.PlotData.json
- qad.PlotData.json
- qgc.PlotData.json



Comparing Water and Enzyme profiles

Exercise: Analysis

q_plot.py

1) cd into **2-fep/**

2) Run *“q_plot.py qaf.PlotData.json”*

3) Are your numbers reliable?

Note: If it complains about matplotlib, try to load a different python module or install matplotlib (try with anaconda - https://www.continuum.io/downloads).

Exercise: Group Contributions

q_calc.py

- Automates calculations with qcalc5
- At the moment supports only Group Contributions (dG(LRA) of nonbonding interactions between the reactive system and each residue going from state λ=1.0 to λ=0.5)
- GCs are calculated with Q-atoms as the mask (as defined in fep file)
- Bond/Angle/Torsion/RMSD implementation in progress

Creates:

- q_calcs.log (statistics, details)
- qgc.PlotData.json (for GC)
- qcalc5 input(s) (optional)
- Qcalc5 output(s) (optional)

Requires:

- EVB/FEP simulation(s)

Exercise: Group Contributions

q_calc.py

$$\Delta G(A \rightarrow B) = \frac{1}{2}\Big(\langle E_B - E_A \rangle_A + \langle E_B - E_A \rangle_B \Big)$$

Group contributions are calculated using the LRA approach.

Sham, Y. Y., Chu, Z. T., Tao, H. and Warshel, A. (2000), Proteins, 39: 393–407.

1) cd into **3-gcs/**
2) Run *"q_calc.py"*
3) Now run *"q_calc.py gc"*
4) Now calculate GCs by using:
   - PDB file (**dfpase_dfp_start.pdb**)
   - residue indexes **15** to **35**
5) Note the output
6) Use q_plot.py to visualize the results (**qgc.PlotData.json**)

Other scripts and stuff

q_pdbindex.py
- Converts atom placeholders ($395.P1$, $396.O1$, …) to PDB indexes

q_amber2q.py
- Converts amber parameters to Q format (q_ffld2q.py analog)

q_setprot.py
- Sets the protonation states in the enzyme PDB (ASP <> ASH, LYS <> LYN, etc…)

q_analysedyns.py
- Extracts data from qdyn logfiles (q_analysefeps.py analog)

qscripts_config.py
- Creates a config file with some default values for the scripts (mapping parameters, filenames, …)

Conclusion

Qtools is a set of tools for Q:
- a Python programming interface (Qpyl)
- command-line scripts (qscripts-cli)

Qtools will help you do science:
- more efficiently
- with less mistakes

Qtools should not be used:
- without learning how to do things manually first!

Acknowledgments

Kamerlin group (especially Paul Bauer)
ICM, BMC, Uppsala University

Janez Mavri's group
National Institute of Chemistry, Slovenia

All Qtools users - you are beautiful human beings!

Extra

If you like Qtools go to https://github.com/mpurg/qtools/ and click on Star.

If you find bugs or **want to request features**:

- You are awesome.
- Use the Github **issues** page or
- Send an email to miha.purg@gmail.com