ForceBalance Developer API Guide version 1.1

Generated by Doxygen 1.7.6.1

Contents

1 Project Roadmap **1**

1.1 Current Development Goals: . . . 1

1.2 Longterm Development Ideas . . . 1

2 Todo List **1**

3 Namespace Index **3**

3.1 Packages . . . 3

4 Hierarchical Index **4**

4.1 Class Hierarchy . . . 4

5 Class Index **6**

5.1 Class List . . . 6

6 File Index **9**

6.1 File List . . . 10

7 Namespace Documentation **11**

7.1 forcebalance Namespace Reference . . . 11

7.1.1 Variable Documentation . . . 12

7.2 forcebalance.abinitio Namespace Reference . . . 13

7.2.1 Detailed Description . . . 13

7.2.2 Function Documentation . . . 13

7.2.3 Variable Documentation . . . 14

7.3 forcebalance.abinitio_internal Namespace Reference . . . 14

7.3.1 Detailed Description . . . 14

7.4 forcebalance.amberio Namespace Reference . . . 15

7.4.1 Detailed Description . . . 15

7.4.2 Function Documentation . . . 15

7.4.3 Variable Documentation . . . 16

7.5 forcebalance.binding Namespace Reference . . . 16

7.5.1 Detailed Description . . . 17

7.5.2 Function Documentation . . . 17

7.5.3 Variable Documentation . . . 17

7.6 forcebalance.chemistry Namespace Reference . . . 17

7.6.1 Function Documentation . . . 18

7.6.2 Variable Documentation . . . 18

7.7 forcebalance.contact Namespace Reference . . . 20
7.7.1 Function Documentation . . . 20
7.8 forcebalance.counterpoise Namespace Reference . . . 21
7.8.1 Detailed Description . . . 21
7.8.2 Variable Documentation . . . 22
7.9 forcebalance.custom_io Namespace Reference . . . 22
7.9.1 Detailed Description . . . 22
7.9.2 Variable Documentation . . . 22
7.10 forcebalance.finite_difference Namespace Reference . . . 23
7.10.1 Function Documentation . . . 24
7.10.2 Variable Documentation . . . 27
7.11 forcebalance.forcefield Namespace Reference . . . 28
7.11.1 Detailed Description . . . 28
7.11.2 Function Documentation . . . 29
7.11.3 Variable Documentation . . . 30
7.12 forcebalance.gmxio Namespace Reference . . . 30
7.12.1 Detailed Description . . . 31
7.12.2 Function Documentation . . . 32
7.12.3 Variable Documentation . . . 33
7.13 forcebalance.gmxqpio Namespace Reference . . . 34
7.13.1 Function Documentation . . . 35
7.13.2 Variable Documentation . . . 35
7.14 forcebalance.interaction Namespace Reference . . . 35
7.14.1 Detailed Description . . . 35
7.14.2 Variable Documentation . . . 36
7.15 forcebalance.leastsq Namespace Reference . . . 36
7.15.1 Function Documentation . . . 36
7.15.2 Variable Documentation . . . 36
7.16 forcebalance.liquid Namespace Reference . . . 37
7.16.1 Detailed Description . . . 37
7.16.2 Function Documentation . . . 37
7.16.3 Variable Documentation . . . 37
7.17 forcebalance.Mol2 Namespace Reference . . . 38
7.17.1 Variable Documentation . . . 38
7.18 forcebalance.mol2io Namespace Reference . . . 38
7.18.1 Detailed Description . . . 38
7.18.2 Variable Documentation . . . 39

7.19 forcebalance.molecule Namespace Reference . . . 39
7.19.1 Function Documentation . . . 40
7.19.2 Variable Documentation . . . 47
7.20 forcebalance.moments Namespace Reference . . . 49
7.20.1 Detailed Description . . . 49
7.20.2 Variable Documentation . . . 49
7.21 forcebalance.nifty Namespace Reference . . . 49
7.21.1 Detailed Description . . . 52
7.21.2 Function Documentation . . . 52
7.21.3 Variable Documentation . . . 66
7.22 forcebalance.objective Namespace Reference . . . 67
7.22.1 Detailed Description . . . 67
7.22.2 Variable Documentation . . . 67
7.23 forcebalance.openmmio Namespace Reference . . . 68
7.23.1 Detailed Description . . . 69
7.23.2 Function Documentation . . . 69
7.23.3 Variable Documentation . . . 73
7.24 forcebalance.optimizer Namespace Reference . . . 74
7.24.1 Detailed Description . . . 74
7.24.2 Function Documentation . . . 74
7.24.3 Variable Documentation . . . 75
7.25 forcebalance.output Namespace Reference . . . 75
7.26 forcebalance.parser Namespace Reference . . . 75
7.26.1 Detailed Description . . . 76
7.26.2 Function Documentation . . . 77
7.26.3 Variable Documentation . . . 79
7.27 forcebalance.psi4io Namespace Reference . . . 80
7.27.1 Detailed Description . . . 81
7.27.2 Variable Documentation . . . 81
7.28 forcebalance.PT Namespace Reference . . . 81
7.28.1 Variable Documentation . . . 81
7.29 forcebalance.qchemio Namespace Reference . . . 82
7.29.1 Detailed Description . . . 82
7.29.2 Function Documentation . . . 82
7.29.3 Variable Documentation . . . 83
7.30 forcebalance.target Namespace Reference . . . 83
7.30.1 Variable Documentation . . . 83

7.31 forcebalance.tinkerio Namespace Reference . . . 84
7.31.1 Detailed Description . . . 84
7.31.2 Function Documentation . . . 85
7.31.3 Variable Documentation . . . 85
7.32 forcebalance.vibration Namespace Reference . . . 86
7.32.1 Detailed Description . . . 86
7.32.2 Variable Documentation . . . 86

8 Class Documentation **87**
8.1 forcebalance.abinitio.AbInitio Class Reference . . . 87
8.1.1 Detailed Description . . . 90
8.1.2 Constructor & Destructor Documentation . . . 91
8.1.3 Member Function Documentation . . . 91
8.1.4 Member Data Documentation . . . 101
8.2 forcebalance.amberio.AbInitio_AMBER Class Reference . . . 106
8.2.1 Detailed Description . . . 110
8.2.2 Constructor & Destructor Documentation . . . 110
8.2.3 Member Function Documentation . . . 110
8.2.4 Member Data Documentation . . . 121
8.3 forcebalance.gmxio.AbInitio_GMX Class Reference . . . 126
8.3.1 Detailed Description . . . 130
8.3.2 Constructor & Destructor Documentation . . . 130
8.3.3 Member Function Documentation . . . 130
8.3.4 Member Data Documentation . . . 141
8.4 forcebalance.abinitio_internal.AbInitio_Internal Class Reference . . . 146
8.4.1 Detailed Description . . . 151
8.4.2 Constructor & Destructor Documentation . . . 151
8.4.3 Member Function Documentation . . . 151
8.4.4 Member Data Documentation . . . 161
8.5 forcebalance.openmmio.AbInitio_OpenMM Class Reference . . . 166
8.5.1 Detailed Description . . . 170
8.5.2 Constructor & Destructor Documentation . . . 170
8.5.3 Member Function Documentation . . . 170
8.5.4 Member Data Documentation . . . 181
8.6 forcebalance.tinkerio.AbInitio_TINKER Class Reference . . . 186
8.6.1 Detailed Description . . . 191
8.6.2 Constructor & Destructor Documentation . . . 191

8.6.3 Member Function Documentation . . . 191
8.6.4 Member Data Documentation . . . 202
8.7 forcebalance.forcefield.BackedUpDict Class Reference . . . 207
8.7.1 Detailed Description . . . 208
8.7.2 Constructor & Destructor Documentation . . . 208
8.7.3 Member Function Documentation . . . 209
8.7.4 Member Data Documentation . . . 209
8.8 forcebalance.BaseClass Class Reference . . . 209
8.8.1 Detailed Description . . . 211
8.8.2 Constructor & Destructor Documentation . . . 211
8.8.3 Member Function Documentation . . . 211
8.8.4 Member Data Documentation . . . 211
8.9 forcebalance.BaseReader Class Reference . . . 211
8.9.1 Detailed Description . . . 213
8.9.2 Constructor & Destructor Documentation . . . 214
8.9.3 Member Function Documentation . . . 214
8.9.4 Member Data Documentation . . . 215
8.10 forcebalance.binding.BindingEnergy Class Reference . . . 216
8.10.1 Detailed Description . . . 218
8.10.2 Constructor & Destructor Documentation . . . 218
8.10.3 Member Function Documentation . . . 218
8.10.4 Member Data Documentation . . . 224
8.11 forcebalance.tinkerio.BindingEnergy_TINKER Class Reference . . . 226
8.11.1 Detailed Description . . . 228
8.11.2 Constructor & Destructor Documentation . . . 228
8.11.3 Member Function Documentation . . . 229
8.11.4 Member Data Documentation . . . 234
8.12 forcebalance.output.CleanFileHandler Class Reference . . . 236
8.12.1 Detailed Description . . . 237
8.12.2 Member Function Documentation . . . 237
8.13 forcebalance.output.CleanStreamHandler Class Reference . . . 237
8.13.1 Detailed Description . . . 238
8.13.2 Constructor & Destructor Documentation . . . 238
8.13.3 Member Function Documentation . . . 239
8.14 forcebalance.counterpoise.Counterpoise Class Reference . . . 239
8.14.1 Detailed Description . . . 241
8.14.2 Constructor & Destructor Documentation . . . 241

8.14.3 Member Function Documentation . . . 242
8.14.4 Member Data Documentation . . . 249
8.15 forcebalance.forcefield.FF Class Reference . . . 250
8.15.1 Detailed Description . . . 253
8.15.2 Constructor & Destructor Documentation . . . 253
8.15.3 Member Function Documentation . . . 254
8.15.4 Member Data Documentation . . . 261
8.16 forcebalance.output.ForceBalanceLogger Class Reference . . . 264
8.16.1 Detailed Description . . . 265
8.16.2 Constructor & Destructor Documentation . . . 266
8.16.3 Member Function Documentation . . . 266
8.16.4 Member Data Documentation . . . 266
8.17 forcebalance.amberio.FrcMod_Reader Class Reference . . . 267
8.17.1 Detailed Description . . . 268
8.17.2 Constructor & Destructor Documentation . . . 268
8.17.3 Member Function Documentation . . . 268
8.17.4 Member Data Documentation . . . 269
8.18 forcebalance.psi4io.GBS_Reader Class Reference . . . 270
8.18.1 Detailed Description . . . 272
8.18.2 Constructor & Destructor Documentation . . . 272
8.18.3 Member Function Documentation . . . 272
8.18.4 Member Data Documentation . . . 274
8.19 forcebalance.custom_io.Gen_Reader Class Reference . . . 275
8.19.1 Detailed Description . . . 277
8.19.2 Constructor & Destructor Documentation . . . 277
8.19.3 Member Function Documentation . . . 277
8.19.4 Member Data Documentation . . . 278
8.20 forcebalance.psi4io.Grid_Reader Class Reference . . . 279
8.20.1 Detailed Description . . . 281
8.20.2 Constructor & Destructor Documentation . . . 281
8.20.3 Member Function Documentation . . . 281
8.20.4 Member Data Documentation . . . 282
8.21 forcebalance.interaction.Interaction Class Reference . . . 283
8.21.1 Detailed Description . . . 286
8.21.2 Constructor & Destructor Documentation . . . 286
8.21.3 Member Function Documentation . . . 286
8.21.4 Member Data Documentation . . . 293

8.22 forcebalance.gmxio.Interaction_GMX Class Reference . . . 295
8.22.1 Detailed Description . . . 299
8.22.2 Constructor & Destructor Documentation . . . 299
8.22.3 Member Function Documentation . . . 299
8.22.4 Member Data Documentation . . . 307
8.23 forcebalance.openmmio.Interaction_OpenMM Class Reference . . . 309
8.23.1 Detailed Description . . . 313
8.23.2 Constructor & Destructor Documentation . . . 313
8.23.3 Member Function Documentation . . . 313
8.23.4 Member Data Documentation . . . 320
8.24 forcebalance.tinkerio.Interaction_TINKER Class Reference . . . 322
8.24.1 Detailed Description . . . 326
8.24.2 Constructor & Destructor Documentation . . . 326
8.24.3 Member Function Documentation . . . 326
8.24.4 Member Data Documentation . . . 333
8.25 forcebalance.gmxio.ITP_Reader Class Reference . . . 335
8.25.1 Detailed Description . . . 337
8.25.2 Constructor & Destructor Documentation . . . 338
8.25.3 Member Function Documentation . . . 338
8.25.4 Member Data Documentation . . . 339
8.26 forcebalance.leastsq.LeastSquares Class Reference . . . 341
8.26.1 Detailed Description . . . 343
8.26.2 Constructor & Destructor Documentation . . . 343
8.26.3 Member Function Documentation . . . 344
8.26.4 Member Data Documentation . . . 350
8.27 forcebalance.liquid.Liquid Class Reference . . . 351
8.27.1 Detailed Description . . . 354
8.27.2 Constructor & Destructor Documentation . . . 354
8.27.3 Member Function Documentation . . . 354
8.27.4 Member Data Documentation . . . 362
8.28 forcebalance.gmxio.Liquid_GMX Class Reference . . . 365
8.28.1 Detailed Description . . . 368
8.28.2 Constructor & Destructor Documentation . . . 368
8.28.3 Member Function Documentation . . . 368
8.28.4 Member Data Documentation . . . 376
8.29 forcebalance.openmmio.Liquid_OpenMM Class Reference . . . 380
8.29.1 Detailed Description . . . 383

8.29.2 Constructor & Destructor Documentation . . . 383
8.29.3 Member Function Documentation . . . 383
8.29.4 Member Data Documentation . . . 391
8.30 forcebalance.tinkerio.Liquid_TINKER Class Reference . . . 395
8.30.1 Detailed Description . . . 398
8.30.2 Constructor & Destructor Documentation . . . 398
8.30.3 Member Function Documentation . . . 398
8.30.4 Member Data Documentation . . . 406
8.31 forcebalance.Mol2.mol2 Class Reference . . . 409
8.31.1 Detailed Description . . . 410
8.31.2 Constructor & Destructor Documentation . . . 410
8.31.3 Member Function Documentation . . . 410
8.31.4 Member Data Documentation . . . 414
8.32 forcebalance.Mol2.mol2_atom Class Reference . . . 415
8.32.1 Detailed Description . . . 416
8.32.2 Constructor & Destructor Documentation . . . 416
8.32.3 Member Function Documentation . . . 416
8.32.4 Member Data Documentation . . . 419
8.33 forcebalance.Mol2.mol2_bond Class Reference . . . 420
8.33.1 Detailed Description . . . 421
8.33.2 Constructor & Destructor Documentation . . . 421
8.33.3 Member Function Documentation . . . 421
8.33.4 Member Data Documentation . . . 423
8.34 forcebalance.amberio.Mol2_Reader Class Reference . . . 424
8.34.1 Detailed Description . . . 425
8.34.2 Constructor & Destructor Documentation . . . 425
8.34.3 Member Function Documentation . . . 425
8.34.4 Member Data Documentation . . . 426
8.35 forcebalance.mol2io.Mol2_Reader Class Reference . . . 427
8.35.1 Detailed Description . . . 429
8.35.2 Constructor & Destructor Documentation . . . 429
8.35.3 Member Function Documentation . . . 429
8.35.4 Member Data Documentation . . . 430
8.36 forcebalance.Mol2.mol2_set Class Reference . . . 431
8.36.1 Detailed Description . . . 431
8.36.2 Constructor & Destructor Documentation . . . 431
8.36.3 Member Function Documentation . . . 432

8.36.4 Member Data Documentation . . . 432
8.37 forcebalance.molecule.Molecule Class Reference . . . 432
8.37.1 Detailed Description . . . 435
8.37.2 Constructor & Destructor Documentation . . . 436
8.37.3 Member Function Documentation . . . 436
8.37.4 Member Data Documentation . . . 454
8.38 forcebalance.molecule.MolfileTimestep Class Reference . . . 455
8.38.1 Detailed Description . . . 456
8.39 forcebalance.moments.Moments Class Reference . . . 456
8.39.1 Detailed Description . . . 459
8.39.2 Constructor & Destructor Documentation . . . 460
8.39.3 Member Function Documentation . . . 460
8.39.4 Member Data Documentation . . . 467
8.40 forcebalance.tinkerio.Moments_TINKER Class Reference . . . 469
8.40.1 Detailed Description . . . 471
8.40.2 Constructor & Destructor Documentation . . . 472
8.40.3 Member Function Documentation . . . 472
8.40.4 Member Data Documentation . . . 479
8.41 forcebalance.gmxqpio.Monomer_QTPIE Class Reference . . . 481
8.41.1 Detailed Description . . . 484
8.41.2 Constructor & Destructor Documentation . . . 484
8.41.3 Member Function Documentation . . . 485
8.41.4 Member Data Documentation . . . 492
8.42 forcebalance.objective.Objective Class Reference . . . 493
8.42.1 Detailed Description . . . 495
8.42.2 Constructor & Destructor Documentation . . . 495
8.42.3 Member Function Documentation . . . 496
8.42.4 Member Data Documentation . . . 496
8.43 forcebalance.openmmio.OpenMM_Reader Class Reference . . . 497
8.43.1 Detailed Description . . . 499
8.43.2 Constructor & Destructor Documentation . . . 499
8.43.3 Member Function Documentation . . . 499
8.43.4 Member Data Documentation . . . 501
8.44 forcebalance.optimizer.Optimizer Class Reference . . . 502
8.44.1 Detailed Description . . . 504
8.44.2 Constructor & Destructor Documentation . . . 504
8.44.3 Member Function Documentation . . . 505

8.44.4 Member Data Documentation . . . 511
8.45 forcebalance.objective.Penalty Class Reference . . . 513
8.45.1 Detailed Description . . . 514
8.45.2 Constructor & Destructor Documentation . . . 514
8.45.3 Member Function Documentation . . . 514
8.45.4 Member Data Documentation . . . 516
8.46 forcebalance.nifty.Pickler_LP Class Reference . . . 517
8.46.1 Detailed Description . . . 518
8.46.2 Constructor & Destructor Documentation . . . 518
8.47 forcebalance.qchemio.QCIn_Reader Class Reference . . . 519
8.47.1 Detailed Description . . . 520
8.47.2 Constructor & Destructor Documentation . . . 520
8.47.3 Member Function Documentation . . . 520
8.47.4 Member Data Documentation . . . 521
8.48 forcebalance.output.RawFileHandler Class Reference . . . 522
8.48.1 Detailed Description . . . 523
8.48.2 Member Function Documentation . . . 523
8.49 forcebalance.output.RawStreamHandler Class Reference . . . 524
8.49.1 Detailed Description . . . 525
8.49.2 Constructor & Destructor Documentation . . . 525
8.49.3 Member Function Documentation . . . 525
8.50 forcebalance.psi4io.RDVR3_Psi4 Class Reference . . . 525
8.50.1 Detailed Description . . . 528
8.50.2 Constructor & Destructor Documentation . . . 528
8.50.3 Member Function Documentation . . . 528
8.50.4 Member Data Documentation . . . 535
8.51 forcebalance.target.RemoteTarget Class Reference . . . 537
8.51.1 Detailed Description . . . 539
8.51.2 Constructor & Destructor Documentation . . . 539
8.51.3 Member Function Documentation . . . 539
8.51.4 Member Data Documentation . . . 545
8.52 forcebalance.target.Target Class Reference . . . 546
8.52.1 Detailed Description . . . 549
8.52.2 Constructor & Destructor Documentation . . . 549
8.52.3 Member Function Documentation . . . 550
8.52.4 Member Data Documentation . . . 556
8.53 forcebalance.psi4io.THCDF_Psi4 Class Reference . . . 557

8.53.1 Detailed Description . . . 559
8.53.2 Constructor & Destructor Documentation . . . 560
8.53.3 Member Function Documentation . . . 560
8.53.4 Member Data Documentation . . . 567
8.54 forcebalance.tinkerio.Tinker_Reader Class Reference . . . 569
8.54.1 Detailed Description . . . 571
8.54.2 Constructor & Destructor Documentation . . . 571
8.54.3 Member Function Documentation . . . 571
8.54.4 Member Data Documentation . . . 573
8.55 forcebalance.nifty.Unpickler_LP Class Reference . . . 573
8.55.1 Detailed Description . . . 574
8.55.2 Constructor & Destructor Documentation . . . 574
8.56 forcebalance.vibration.Vibration Class Reference . . . 575
8.56.1 Detailed Description . . . 577
8.56.2 Constructor & Destructor Documentation . . . 577
8.56.3 Member Function Documentation . . . 578
8.56.4 Member Data Documentation . . . 585
8.57 forcebalance.tinkerio.Vibration_TINKER Class Reference . . . 586
8.57.1 Detailed Description . . . 587
8.57.2 Constructor & Destructor Documentation . . . 587
8.57.3 Member Function Documentation . . . 588

9 File Documentation **588**
9.1 __init__.py File Reference . . . 589
9.2 abinitio.py File Reference . . . 589
9.3 abinitio_internal.py File Reference . . . 590
9.4 amberio.py File Reference . . . 590
9.5 api.dox File Reference . . . 590
9.6 binding.py File Reference . . . 590
9.7 chemistry.py File Reference . . . 591
9.8 contact.py File Reference . . . 591
9.9 counterpoise.py File Reference . . . 592
9.10 custom_io.py File Reference . . . 592
9.11 finite_difference.py File Reference . . . 593
9.12 forcefield.py File Reference . . . 593
9.13 gmxio.py File Reference . . . 594
9.14 gmxqpio.py File Reference . . . 595

9.15 interaction.py File Reference . . . 595
9.16 leastsq.py File Reference . . . 596
9.17 liquid.py File Reference . . . 596
9.18 Mol2.py File Reference . . . 597
9.19 mol2io.py File Reference . . . 597
9.20 molecule.py File Reference . . . 597
9.21 moments.py File Reference . . . 599
9.22 nifty.py File Reference . . . 599
9.23 objective.py File Reference . . . 601
9.24 openmmio.py File Reference . . . 602
9.25 optimizer.py File Reference . . . 603
9.26 output.py File Reference . . . 603
9.27 parser.py File Reference . . . 604
9.28 psi4io.py File Reference . . . 605
9.29 PT.py File Reference . . . 605
9.30 qchemio.py File Reference . . . 605
9.31 target.py File Reference . . . 606
9.32 tinkerio.py File Reference . . . 606
9.33 vibration.py File Reference . . . 607

Index **607**

1 Project Roadmap

ForceBalance is a work in progress and is continually being improved and expanded! Here are some current and future project development ideas.

1.1 Current Development Goals:

- Implementing and expanding support for running target calculations in parallel across multiple nodes
- Expand unit test cases to provide more thorough coverage of forcebalance modules

1.2 Longterm Development Ideas

- Development of a ForceBalance GUI interface to complement the current command line interface
- Visualization of running calculations
- More comprehensive tutorial to walk users through the initial process of setting up targets and preparing for a successful ForceBalance run

2 Todo List

Member forcebalance.abinitio.AbInitio.__init__

Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Member forcebalance.abinitio.AbInitio.get_energy_force_

Parallelization over snapshots is not implemented yet

Member forcebalance.abinitio.AbInitio.read_reference_data

Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

Member forcebalance.counterpoise.Counterpoise.loadxyz

I should probably put this into a more general library for reading coordinates.

Member forcebalance.forcefield.FF.mktransmat

Only project out changes in total charge of a molecule, and perhaps generalize to fragments of molecules or other types of parameters.

The AMOEBA selection of charge depends not only on the atom type, but what that atom is bonded to.

Member forcebalance.forcefield.FF.rsmake

Pass in rsfactors through the input file

Namespace forcebalance.gmxio

Even more stuff from forcefield.py needs to go into here.

Even more stuff from forcefield.py needs to go into here.

Class forcebalance.gmxio.ITP_Reader

Note that I can also create the opposite virtual site position by changing the atom labeling, woo!

Member forcebalance.openmmio.OpenMM_Reader.build_pid

Add a link here

Member forcebalance.optimizer.Optimizer.GeneticAlgorithm

Massive parallelization hasn't been implemented yet

Member forcebalance.optimizer.Optimizer.Scan_Values

Maybe a multidimensional grid can be done.

Member forcebalance.tinkerio.Tinker_Reader.feed

Put the rescaling factors for TINKER parameters in here. Currently we're using the initial value to determine the rescaling factor which is not very good.

Member forcebalance::gmxio.pdict

This needs to become more flexible because the parameter isn't always in the same field. Still need to figure out how to do this.

How about making the PDIHS less ugly?

Member forcebalance::nifty.floatornan

I could use suggestions for making this better.

Member forcebalance::parser.parse_inputs

Implement internal coordinates.

Implement sampling correction.

Implement charge groups.

3 Namespace Index

3.1 Packages

Here are the packages with brief descriptions (if available):

forcebalance 11

forcebalance.abinitio
Ab-initio fitting module (energies, forces, resp) 13

forcebalance.abinitio_internal
Internal implementation of energy matching (for TIP3P water only) 14

forcebalance.amberio
AMBER force field input/output 15

forcebalance.binding
Binding energy fitting module 16

forcebalance.chemistry 17

forcebalance.contact 20

forcebalance.counterpoise
Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE) 21

forcebalance.custom_io
Custom force field parser 22

forcebalance.finite_difference 23

forcebalance.forcefield
Force field module 28

forcebalance.gmxio
GROMACS input/output 30

forcebalance.gmxqpio 34

forcebalance.interaction
Interaction energy fitting module 35

forcebalance.leastsq 36

forcebalance.liquid
Matching of liquid bulk properties 37

forcebalance.Mol2 38

forcebalance.mol2io
Mol2 I/O 38

forcebalance.molecule 39

forcebalance.moments
Multipole moment fitting module 49

forcebalance.nifty
Nifty functions, intended to be imported by any module within ForceBalance 49

forcebalance.objective
ForceBalance objective function 67

forcebalance.openmmio
OpenMM input/output 68

forcebalance.optimizer
Optimization algorithms 74

forcebalance.output 75

forcebalance.parser
Input file parser for ForceBalance jobs 75

forcebalance.psi4io
PSI4 force field input/output 80

forcebalance.PT 81

forcebalance.qchemio
Q-Chem input file parser 82

forcebalance.target 83

forcebalance.tinkerio
TINKER input/output 84

forcebalance.vibration
Vibrational mode fitting module 86

4 Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

dict

forcebalance.forcefield.BackedUpDict 207

FileHandler

forcebalance.output.CleanFileHandler 236

forcebalance.output.RawFileHandler 522

Logger

forcebalance.output.ForceBalanceLogger 264

forcebalance.Mol2.mol2 409

forcebalance.Mol2.mol2_atom 415

forcebalance.Mol2.mol2_bond 420

forcebalance.Mol2.mol2_set 431

object

forcebalance.BaseClass 209

forcebalance.forcefield.FF 250

forcebalance.objective.Objective 493

forcebalance.optimizer.Optimizer 502

forcebalance.target.Target 546

forcebalance.abinitio.AbInitio 87

forcebalance.abinitio_internal.AbInitio_Internal 146

forcebalance.amberio.AbInitio_AMBER 106

forcebalance.gmxio.AbInitio_GMX 126

forcebalance.openmmio.AbInitio_OpenMM 166

forcebalance.tinkerio.AbInitio_TINKER 186

forcebalance.binding.BindingEnergy 216

forcebalance.tinkerio.BindingEnergy_TINKER 226

forcebalance.counterpoise.Counterpoise 239

forcebalance.gmxqpio.Monomer_QTPIE 481

forcebalance.interaction.Interaction 283

forcebalance.gmxio.Interaction_GMX 295

forcebalance.openmmio.Interaction_OpenMM 309

forcebalance.tinkerio.Interaction_TINKER 322

forcebalance.leastsq.LeastSquares 341

forcebalance.psi4io.THCDF_Psi4 557

forcebalance.liquid.Liquid 351

forcebalance.gmxio.Liquid_GMX 365

forcebalance.openmmio.Liquid_OpenMM 380

forcebalance.tinkerio.Liquid_TINKER 395

forcebalance.moments.Moments 456

forcebalance.tinkerio.Moments_TINKER 469

forcebalance.psi4io.RDVR3_Psi4 525

forcebalance.target.RemoteTarget 537

forcebalance.vibration.Vibration 575

forcebalance.BaseReader 211

forcebalance.amberio.FrcMod_Reader 267

forcebalance.amberio.Mol2_Reader 424

forcebalance.custom_io.Gen_Reader 275

forcebalance.gmxio.ITP_Reader 335

forcebalance.mol2io.Mol2_Reader 427

forcebalance.openmmio.OpenMM_Reader 497

forcebalance.psi4io.GBS_Reader 270

forcebalance.psi4io.Grid_Reader 279

forcebalance.qchemio.QCIn_Reader 519

forcebalance.tinkerio.Tinker_Reader 569

forcebalance.molecule.Molecule 432

forcebalance.objective.Penalty 513
Pickler

forcebalance.nifty.Pickler_LP 517
StreamHandler

forcebalance.output.CleanStreamHandler 237

forcebalance.output.RawStreamHandler 524
Structure

forcebalance.molecule.MolfileTimestep 455
Unpickler

forcebalance.nifty.Unpickler_LP 573
Vibration

forcebalance.tinkerio.Vibration_TINKER 586

5 Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

forcebalance.abinitio.AbInitio
Subclass of Target for fitting force fields to ab initio data 87

forcebalance.amberio.AbInitio_AMBER
Subclass of Target for force and energy matching using AMBER 106

forcebalance.gmxio.AbInitio_GMX
Subclass of AbInitio for force and energy matching using normal GROMACS 126

forcebalance.abinitio_internal.AbInitio_Internal
Subclass of Target for force and energy matching using an internal implementation 146

forcebalance.openmmio.AbInitio_OpenMM
Subclass of AbInitio for force and energy matching using OpenMM 166

forcebalance.tinkerio.AbInitio_TINKER
Subclass of Target for force and energy matching using TINKER 186

forcebalance.forcefield.BackedUpDict 207

forcebalance.BaseClass
Provides some nifty functions that are common to all ForceBalance classes 209

forcebalance.BaseReader
The 'reader' class 211

forcebalance.binding.BindingEnergy
Improved subclass of Target for fitting force fields to binding energies 216

forcebalance.tinkerio.BindingEnergy_TINKER
Subclass of BindingEnergy for binding energy matching using TINKER 226

forcebalance.output.CleanFileHandler
File handler that does not write terminal escape codes and carriage returns to files 236

forcebalance.output.CleanStreamHandler
Similar to RawStreamHandler except it does not write terminal escape codes 237

forcebalance.counterpoise.Counterpoise
Target subclass for matching the counterpoise correction 239

forcebalance.forcefield.FF
Force field class 250

forcebalance.output.ForceBalanceLogger
This logger starts out with a default handler that writes to stdout addHandler removes this default the first time another handler is added 264

forcebalance.amberio.FrcMod_Reader
Finite state machine for parsing FrcMod force field file 267

forcebalance.psi4io.GBS_Reader
Interaction type -> Parameter Dictionary 270

forcebalance.custom_io.Gen_Reader
Finite state machine for parsing custom GROMACS force field files 275

forcebalance.psi4io.Grid_Reader
Finite state machine for parsing DVR grid files 279

forcebalance.interaction.Interaction
Subclass of Target for fitting force fields to interaction energies 283

forcebalance.gmxio.Interaction_GMX
Subclass of Interaction for interaction energy matching using GROMACS 295

forcebalance.openmmio.Interaction_OpenMM
Subclass of Target for interaction matching using OpenMM 309

forcebalance.tinkerio.Interaction_TINKER
Subclass of Target for interaction matching using TINKER 322

forcebalance.gmxio.ITP_Reader
Finite state machine for parsing GROMACS force field files 335

forcebalance.leastsq.LeastSquares
Subclass of Target for general least squares fitting 341

forcebalance.liquid.Liquid
Subclass of Target for liquid property matching 351

forcebalance.gmxio.Liquid_GMX 365

forcebalance.openmmio.Liquid_OpenMM 380

forcebalance.tinkerio.Liquid_TINKER 395

forcebalance.Mol2.mol2
This is to manage one mol2 series of lines on the form: 409

forcebalance.Mol2.mol2_atom
This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424 415

forcebalance.Mol2.mol2_bond
This is to manage mol2 bond lines on the form: 1 1 2 ar 420

forcebalance.amberio.Mol2_Reader
Finite state machine for parsing Mol2 force field file 424

forcebalance.mol2io.Mol2_Reader
Finite state machine for parsing Mol2 force field file 427

forcebalance.Mol2.mol2_set 431

forcebalance.molecule.Molecule
Lee-Ping's general file format conversion class 432

forcebalance.molecule.MolfileTimestep
Wrapper for the timestep C structure used in molfile plugins 455

forcebalance.moments.Moments
Subclass of Target for fitting force fields to multipole moments (from experiment or theory) 456

forcebalance.tinkerio.Moments_TINKER
Subclass of Target for multipole moment matching using TINKER 469

forcebalance.gmxqpio.Monomer_QTPIE
Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch) 481

forcebalance.objective.Objective
Objective function 493

forcebalance.openmmio.OpenMM_Reader
Class for parsing OpenMM force field files 497

forcebalance.optimizer.Optimizer
Optimizer class 502

forcebalance.objective.Penalty
Penalty functions for regularizing the force field optimizer 513

forcebalance.nifty.Pickler_LP
A subclass of the python Pickler that implements pickling of _ElementTree types 517

forcebalance.qchemio.QCIn_Reader
Finite state machine for parsing Q-Chem input files 519

forcebalance.output.RawFileHandler
Exactly like output.FileHandler except it does no extra formatting before sending logging messages to the file 522

forcebalance.output.RawStreamHandler
Exactly like output.StreamHandler except it does no extra formatting before sending logging messages to the stream 524

forcebalance.psi4io.RDVR3_Psi4
Subclass of Target for R-DVR3 grid fitting 525

forcebalance.target.RemoteTarget 537

forcebalance.target.Target
Base class for all fitting targets 546

forcebalance.psi4io.THCDF_Psi4 557

forcebalance.tinkerio.Tinker_Reader
Finite state machine for parsing TINKER force field files 569

forcebalance.nifty.Unpickler_LP
A subclass of the python Unpickler that implements unpickling of _ElementTree types 573

forcebalance.vibration.Vibration
Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory) 575

forcebalance.tinkerio.Vibration_TINKER
Subclass of Target for vibrational frequency matching using TINKER 586

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

__init__.py 589

abinitio.py 589

abinitio_internal.py 590

amberio.py 590

binding.py 590

chemistry.py 591

contact.py 591

counterpoise.py 592

custom_io.py 592

finite_difference.py 593

forcefield.py 593

gmxio.py 594

gmxqpio.py 595

interaction.py 595

leastsq.py 596

liquid.py 596

Mol2.py 597

mol2io.py 597

molecule.py 597

moments.py 599

nifty.py 599

objective.py 601

openmmio.py 602

optimizer.py 603

output.py 603

parser.py 604

psi4io.py 605

PT.py 605

qchemio.py 605

target.py 606

tinkerio.py 606

vibration.py 607

7 Namespace Documentation

7.1 forcebalance Namespace Reference

Namespaces

- namespace abinitio

  Ab-initio fitting module (energies, forces, resp).
- namespace abinitio_internal

  Internal implementation of energy matching (for TIP3P water only)
- namespace amberio

  AMBER force field input/output.
- namespace binding

  Binding energy fitting module.
- namespace chemistry
- namespace contact
- namespace counterpoise

Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE).

- namespace custom_io

  Custom force field parser.
- namespace finite_difference
- namespace forcefield

  Force field module.
- namespace gmxio

  GROMACS input/output.
- namespace gmxqpio
- namespace interaction

  Interaction energy fitting module.
- namespace leastsq
- namespace liquid

  Matching of liquid bulk properties.
- namespace Mol2
- namespace mol2io

  Mol2 I/O.
- namespace molecule
- namespace moments

  Multipole moment fitting module.
- namespace nifty

  Nifty functions, intended to be imported by any module within ForceBalance.
- namespace objective

  ForceBalance objective function.
- namespace openmmio

  OpenMM input/output.
- namespace optimizer

  Optimization algorithms.
- namespace output
- namespace parser

  Input file parser for ForceBalance jobs.
- namespace psi4io

  PSI4 force field input/output.
- namespace PT
- namespace qchemio

  Q-Chem input file parser.
- namespace target
- namespace tinkerio

  TINKER input/output.
- namespace vibration

  Vibrational mode fitting module.

Classes

- class BaseClass

  Provides some nifty functions that are common to all ForceBalance classes.
- class BaseReader

  The 'reader' class.

Variables

- tuple __version__ = pkg_resources.get_distribution("forcebalance")
- WORK_QUEUE = None
- tuple WQIDS = defaultdict(list)

7.1.1 Variable Documentation

7.1.1.1 tuple forcebalance.__version__ = pkg_resources.get_distribution("forcebalance")

Definition at line 16 of file __init__.py.

7.1.1.2 forcebalance.WORK_QUEUE = None

Definition at line 19 of file __init__.py.

7.1.1.3 tuple forcebalance.WQIDS = defaultdict(list)

Definition at line 22 of file __init__.py.

7.2 forcebalance.abinitio Namespace Reference

Ab-initio fitting module (energies, forces, resp).

Classes

- class AbInitio

  Subclass of Target for fitting force fields to ab initio data.

Functions

- def weighted_variance

  A more generalized version of build_objective which is callable for derivatives, but the covariance is not there anymore.
- def weighted_variance2

  A bit of a hack, since we have to subtract out two mean quantities to get Hessian elements.
- def build_objective

  This function builds an objective function (number) from the complicated polytensor and covariance matrices.

Variables

- tuple logger = getLogger(__name__)

7.2.1 Detailed Description

Ab-initio fitting module (energies, forces, resp).

Author

Lee-Ping Wang

Date

05/2012

7.2.2 Function Documentation

7.2.2.1 def forcebalance.abinitio.build_objective (*SPiXi, WCiW, Z, Q0, M0, NCP1, subtract_mean* = `True`)

This function builds an objective function (number) from the complicated polytensor and covariance matrices.

Definition at line 1154 of file abinitio.py.

7.2.2.2 def forcebalance.abinitio.weighted_variance (*SPiXi, WCiW, Z, L, R, NCP1, subtract_mean* = `True`)

A more generalized version of build_objective which is callable for derivatives, but the covariance is not there anymore.

Definition at line 1124 of file abinitio.py.

Here is the call graph for this function:

7.2.2.3 def forcebalance.abinitio.weighted_variance2 (*SPiXi, WCiW, Z, L, R, L2, R2, NCP1, subtract_mean* = `True`)

A bit of a hack, since we have to subtract out two mean quantities to get Hessian elements.

Definition at line 1138 of file abinitio.py.

Here is the call graph for this function:

7.2.3 Variable Documentation

7.2.3.1 tuple forcebalance.abinitio.logger = getLogger(__name__)

Definition at line 24 of file abinitio.py.

7.3 forcebalance.abinitio_internal Namespace Reference

Internal implementation of energy matching (for TIP3P water only)

Classes

- class AbInitio_Internal

  Subclass of Target for force and energy matching using an internal implementation.

7.3.1 Detailed Description

Internal implementation of energy matching (for TIP3P water only)

Author

Lee-Ping Wang

Date

04/2012

7.4 forcebalance.amberio Namespace Reference

AMBER force field input/output.

Classes

- class Mol2_Reader

  Finite state machine for parsing Mol2 force field file.
- class FrcMod_Reader

  Finite state machine for parsing FrcMod force field file.
- class AbInitio_AMBER

  Subclass of Target for force and energy matching using AMBER.

Functions

- def is_mol2_atom

Variables

- tuple logger = getLogger(__name__)
- dictionary mol2_pdict = {'COUL':{'Atom':[1], 8:''}}
- dictionary frcmod_pdict

7.4.1 Detailed Description

AMBER force field input/output. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

Author

Lee-Ping Wang

Date

01/2012

7.4.2 Function Documentation

7.4.2.1 def forcebalance.amberio.is_mol2_atom (*line*)

Definition at line 35 of file amberio.py.

Here is the call graph for this function:

7.4.3 Variable Documentation

7.4.3.1 dictionary forcebalance.amberio.frcmod_pdict

Initial value:

```
= {'BONDS': {'Atom':[0], 1:'K', 2:'B'},
                 'ANGLES':{'Atom':[0], 1:'K', 2:'B'},
                 'PDIHS1':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS2':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS3':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS4':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS5':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS6':{'Atom':[0], 2:'K', 3:'B'},
                 'IDIHS' :{'Atom':[0], 1:'K', 3:'B'},
                  'VDW':{'Atom':[0], 1:'S', 2:'T'}
                  }
```

Definition at line 23 of file amberio.py.

7.4.3.2 tuple forcebalance.amberio.logger = getLogger(__name__)

Definition at line 19 of file amberio.py.

7.4.3.3 dictionary forcebalance.amberio.mol2_pdict = {'COUL':{'Atom':[1], 8:''}}

Definition at line 21 of file amberio.py.

7.5 forcebalance.binding Namespace Reference

Binding energy fitting module.

Classes

- class BindingEnergy

  Improved subclass of Target for fitting force fields to binding energies.

Functions

- def parse_interactions

  Parse through the interactions input file.

Variables

- tuple logger = getLogger(__name__)

7.5.1 Detailed Description

Binding energy fitting module.

Author

Lee-Ping Wang

Date

05/2012

7.5.2 Function Documentation

7.5.2.1 def forcebalance.binding.parse_interactions (*input_file*)

Parse through the interactions input file.

Parameters

in	*input_file*	The name of the input file.

Definition at line 33 of file binding.py.

Here is the call graph for this function:

7.5.3 Variable Documentation

7.5.3.1 tuple forcebalance.binding.logger = getLogger(__name__)

Definition at line 22 of file binding.py.

7.6 forcebalance.chemistry Namespace Reference

Functions

- def LookupByMass
- def BondStrengthByLength

Variables

- tuple BondEnergies = defaultdict(lambda:defaultdict(dict))
- list Radii

  Covalent radii from Cordero et al.
- dictionary PeriodicTable
- list Elements
- list BondChars = ['-','=','3']
- string data_from_web
- tuple line = line.expandtabs()
- tuple BE = float(line.split()[1])
- tuple L = float(line.split()[2])
- tuple atoms = re.split('[-=3]', line.split()[0])
- list A = atoms[0]
- list B = atoms[1]
- tuple bo = BondChars.index(re.findall('[-=3]', line.split()[0])[0])

7.6.1 Function Documentation

7.6.1.1 def forcebalance.chemistry.BondStrengthByLength (*A, B, length, artol* = 0.33, *bias* = 0.0)

Definition at line 164 of file chemistry.py.

7.6.1.2 def forcebalance.chemistry.LookupByMass (*mass*)

Definition at line 155 of file chemistry.py.

7.6.2 Variable Documentation

7.6.2.1 list forcebalance.chemistry.A = atoms[0]

Definition at line 149 of file chemistry.py.

7.6.2.2 tuple forcebalance.chemistry.atoms = re.split('[-=3]', line.split()[0])

Definition at line 148 of file chemistry.py.

7.6.2.3 list forcebalance.chemistry.B = atoms[1]

Definition at line 150 of file chemistry.py.

7.6.2.4 tuple forcebalance.chemistry.BE = float(line.split()[1])

Definition at line 146 of file chemistry.py.

7.6.2.5 tuple forcebalance.chemistry.bo = BondChars.index(re.findall('[-=3]', line.split()[0])[0])

Definition at line 151 of file chemistry.py.

7.6.2.6 list forcebalance.chemistry.BondChars = ['-','=','3']

Definition at line 49 of file chemistry.py.

7.6.2.7 tuple forcebalance.chemistry.BondEnergies = defaultdict(lambda:defaultdict(dict))

Definition at line 7 of file chemistry.py.

7.6.2.8 string forcebalance.chemistry.data_from_web

Definition at line 51 of file chemistry.py.

7.6.2.9 list forcebalance.chemistry.Elements

Initial value:

```
= ["None",'H','He',
            'Li','Be','B','C','N','O','F','Ne',
            'Na','Mg','Al','Si','P','S','Cl','Ar',
            'K','Ca','Sc','Ti','V','Cr','Mn','Fe','Co','Ni','Cu','Zn','Ga','Ge','As','Se','Br','Kr',
            'Rb','Sr','Y','Zr','Nb','Mo','Tc','Ru','Rh','Pd','Ag','Cd','In','Sn','Sb','Te','I','Xe',
            'Cs','Ba','La','Ce','Pr','Nd','Pm','Sm','Eu','Gd','Tb','Dy','Ho','Er','Tm','Yb',
            'Lu','Hf','Ta','W','Re','Os','Ir','Pt','Au','Hg','Tl','Pb','Bi','Po','At','Rn',
            'Fr','Ra','Ac','Th','Pa','U','Np','Pu','Am','Cm','Bk','Cf','Es','Fm','Md','No','Lr','Rf','Db','Sg','Bh','Hs','Mt']
```

Definition at line 40 of file chemistry.py.

7.6.2.10 tuple forcebalance.chemistry.L = float(line.split()[2])

Definition at line 147 of file chemistry.py.

7.6.2.11 tuple forcebalance.chemistry.line = line.expandtabs()

Definition at line 145 of file chemistry.py.

7.6.2.12 dictionary forcebalance.chemistry.PeriodicTable

Initial value:

```
= {'H' : 1.0079, 'He' : 4.0026,
                'Li' : 6.941, 'Be' : 9.0122, 'B' : 10.811, 'C' : 12.0107, 'N' : 14.0067, 'O' : 15.9994, 'F' : 18.9984, 'Ne' : 20.1797,
                'Na' : 22.9897, 'Mg' : 24.305, 'Al' : 26.9815, 'Si' : 28.0855, 'P' : 30.9738, 'S' : 32.065, 'Cl' : 35.453, 'Ar' : 39.948,
                'K' : 39.0983, 'Ca' : 40.078, 'Sc' : 44.9559, 'Ti' : 47.867, 'V' : 50.9415, 'Cr' : 51.9961, 'Mn' : 54.938, 'Fe' : 55.845, 'Co' : 58.9332,
                'Ni' : 58.6934, 'Cu' : 63.546, 'Zn' : 65.39, 'Ga' : 69.723, 'Ge' : 72.64, 'As' : 74.9216, 'Se' : 78.96, 'Br' : 79.904, 'Kr' : 83.8,
                'Rb' : 85.4678, 'Sr' : 87.62, 'Y' : 88.9059, 'Zr' : 91.224, 'Nb' : 92.9064, 'Mo' : 95.94, 'Tc' : 98, 'Ru' : 101.07, 'Rh' : 102.9055,
                'Pd' : 106.42, 'Ag' : 107.8682, 'Cd' : 112.411, 'In' : 114.818, 'Sn' : 118.71, 'Sb' : 121.76, 'Te' : 127.6, 'I' : 126.9045, 'Xe' : 131.293,
                'Cs' : 132.9055, 'Ba' : 137.327, 'La' : 138.9055, 'Ce' : 140.116, 'Pr' : 140.9077, 'Nd' : 144.24, 'Pm' : 145, 'Sm' : 150.36,
                'Eu' : 151.964, 'Gd' : 157.25, 'Tb' : 158.9253, 'Dy' : 162.5, 'Ho' : 164.9303, 'Er' : 167.259, 'Tm' : 168.9342, 'Yb' : 173.04,
                'Lu' : 174.967, 'Hf' : 178.49, 'Ta' : 180.9479, 'W' : 183.84, 'Re' : 186.207, 'Os' : 190.23, 'Ir' : 192.217, 'Pt' : 195.078,
                'Au' : 196.9665, 'Hg' : 200.59, 'Tl' : 204.3833, 'Pb' : 207.2, 'Bi' : 208.9804, 'Po' : 209, 'At' : 210, 'Rn' : 222,
                'Fr' : 223, 'Ra' : 226, 'Ac' : 227, 'Th' : 232.0381, 'Pa' : 231.0359, 'U' : 238.0289, 'Np' : 237, 'Pu' : 244,
                'Am' : 243, 'Cm' : 247, 'Bk' : 247, 'Cf' : 251, 'Es' : 252, 'Fm' : 257, 'Md' : 258, 'No' : 259,
                'Lr' : 262, 'Rf' : 261, 'Db' : 262, 'Sg' : 266, 'Bh' : 264, 'Hs' : 277, 'Mt' : 268}
```

Definition at line 25 of file chemistry.py.

7.6.2.13 list forcebalance.chemistry.Radii

Initial value:

```
1 = [0.31, 0.28, # H and He
2          1.28, 0.96, 0.84, 0.76, 0.71, 0.66, 0.57, 0.58, # First row elements
3          1.66, 1.41, 1.21, 1.11, 1.07, 1.05, 1.02, 1.06, # Second row elements
4          2.03, 1.76, 1.70, 1.60, 1.53, 1.39, 1.61, 1.52, 1.50,
5          1.24, 1.32, 1.22, 1.22, 1.20, 1.19, 1.20, 1.20, 1.16, # Third row elements, K through Kr
6          2.20, 1.95, 1.90, 1.75, 1.64, 1.54, 1.47, 1.46, 1.42,
7          1.39, 1.45, 1.44, 1.42, 1.39, 1.39, 1.38, 1.39, 1.40, # Fourth row elements, Rb through Xe
8          2.44, 2.15, 2.07, 2.04, 2.03, 2.01, 1.99, 1.98,
9          1.98, 1.96, 1.94, 1.92, 1.92, 1.89, 1.90, 1.87, # Fifth row elements, s and f blocks
10          1.87, 1.75, 1.70, 1.62, 1.51, 1.44, 1.41, 1.36,
11          1.36, 1.32, 1.45, 1.46, 1.48, 1.40, 1.50, 1.50, # Fifth row elements, d and p blocks
12          2.60, 2.21, 2.15, 2.06, 2.00, 1.96, 1.90, 1.87, 1.80, 1.69]
```

Covalent radii from Cordero et al.

'Covalent radii revisited' Dalton Transactions 2008, 2832-2838.

Definition at line 10 of file chemistry.py.

7.7 forcebalance.contact Namespace Reference

Functions

- def atom_distances

  For each frame in xyzlist, compute the (euclidean) distance between pairs of atoms whos indices are given in contacts.
- def residue_distances

  For each frame in xyzlist, and for each pair of residues in the array contact, compute the distance between the closest pair of atoms such that one of them belongs to each residue.

7.7.1 Function Documentation

7.7.1.1 def forcebalance.contact.atom_distances (*xyzlist, atom_contacts*)

For each frame in xyzlist, compute the (euclidean) distance between pairs of atoms whos indices are given in contacts.

xyzlist should be a traj_length x num_atoms x num_dims array of type float32

contacts should be a num_contacts x 2 array where each row gives the indices of 2 atoms whos distance you care to monitor.

Returns: traj_length x num_contacts array of euclidean distances

Note: For nice wrappers around this, see the prepare_trajectory method of various metrics in metrics.py

Definition at line 26 of file contact.py.

Here is the call graph for this function:

7.7.1.2 def forcebalance.contact.residue_distances (*xyzlist, residue_membership, residue_contacts*)

For each frame in xyzlist, and for each pair of residues in the array contact, compute the distance between the closest pair of atoms such that one of them belongs to each residue.

xyzlist should be a traj_length x num_atoms x num_dims array of type float32

residue_membership should be a list of lists where residue_membership[i] gives the list of atomindices that belong to residue i. residue_membership should NOT be a numpy 2D array unless you really mean that all of the residues have the same number of atoms

residue_contacts should be a 2D numpy array of shape num_contacts x 2 where each row gives the indices of the two RESIDUES who you are interested in monitoring for a contact.

Returns: a 2D array of traj_lenth x num_contacts where out[i,j] contains the distance between the pair of atoms, one from residue_membership[residue_contacts[j,0]] and one from residue_membership[residue_contacts[j,1]] that are closest.

Definition at line 85 of file contact.py.

7.8 forcebalance.counterpoise Namespace Reference

Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE).

Classes

- class Counterpoise

  Target subclass for matching the counterpoise correction.

Variables

- tuple logger = getLogger(__name__)

7.8.1 Detailed Description

Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE). Here we test two different functional forms: a three-parameter Gaussian repulsive potential and a four-parameter Gaussian which goes smoothly to an exponential. The latter can be written in two different ways - one which gives us control over the exponential, the switching distance and the Gaussian decay constant, and another which gives us control over the Gaussian and the switching distance. They are called 'CPGAUSS', 'CPEXPG', and 'CPGEXP'. I think the third option is the best although our early tests have indicated that none of the force fields perform particularly well for the water dimer.

This subclass of Target implements the 'get' method.

Author

Lee-Ping Wang

Date

12/2011

7.8.2 Variable Documentation

7.8.2.1 tuple forcebalance.counterpoise.logger = getLogger(__name__)

Definition at line 29 of file counterpoise.py.

7.9 forcebalance.custom_io Namespace Reference

Custom force field parser.

Classes

- class Gen_Reader

  Finite state machine for parsing custom GROMACS force field files.

Variables

- list cptypes = [None, 'CPGAUSS', 'CPEXPG', 'CPGEXP']

  Types of counterpoise correction.
- list ndtypes = [None]

  Types of NDDO correction.
- dictionary fdict

  Section -> Interaction type dictionary.
- dictionary pdict

  Interaction type -> Parameter Dictionary.

7.9.1 Detailed Description

Custom force field parser. We take advantage of the sections in GROMACS and the 'interaction type' concept, but these interactions are not supported in GROMACS; rather, they are computed within our program.

Author

Lee-Ping Wang

Date

12/2011

7.9.2 Variable Documentation

7.9.2.1 list forcebalance.custom_io.cptypes = [None, 'CPGAUSS', 'CPEXPG', 'CPGEXP']

Types of counterpoise correction.

Definition at line 16 of file custom_io.py.

7.9.2.2 dictionary forcebalance.custom_io.fdict

Initial value:

```
= {
    'counterpoise'  : cptypes    }
```

Section -> Interaction type dictionary.

Definition at line 21 of file custom_io.py.

7.9.2.3 list forcebalance.custom_io.ndtypes = [None]

Types of NDDO correction.

Definition at line 18 of file custom_io.py.

7.9.2.4 dictionary forcebalance.custom_io.pdict

Initial value:

```
= {'CPGAUSS':{3:'A', 4:'B', 5:'C'},
         'CPGEXP' :{3:'A', 4:'B', 5:'G', 6:'X'},
         'CPEXPG' :{3:'A1', 4:'B', 5:'X0', 6:'A2'}
         }
```

Interaction type -> Parameter Dictionary.

Definition at line 25 of file custom_io.py.

7.10 forcebalance.finite_difference Namespace Reference

Functions

- def f1d2p

  A two-point finite difference stencil.
- def f1d5p

  A highly accurate five-point finite difference stencil for computing derivatives of a function.
- def f1d7p

  A highly accurate seven-point finite difference stencil for computing derivatives of a function.
- def f12d7p
- def f12d3p

  A three-point finite difference stencil.
- def in_fd

  Invoking this function from anywhere will tell us whether we're being called by a finite-difference function.
- def fdwrap

  A function wrapper for finite difference designed for differentiating 'get'-type functions.
- def fdwrap_G

  A driver to fdwrap for gradients (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.
- def fdwrap_H

  A driver to fdwrap for Hessians (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.

Variables

- tuple logger = getLogger(__name__)

7.10.1 Function Documentation

7.10.1.1 def forcebalance.finite_difference.f12d3p (*f, h, f0* = None)

A three-point finite difference stencil.

This function does either two computations or three, depending on whether the 'center' value is supplied. This is done in order to avoid recomputing the center value many times.

The first derivative is evaluated using central difference. One advantage of using central difference (as opposed to forward difference) is that we get zero at the bottom of a parabola.

Using this formula we also get an approximate second derivative, which can then be inserted into the diagonal of the Hessian. This is very useful for optimizations like BFGS where the diagonal determines how far we step in the parameter space.

How to use: use fdwrap or something similar to generate a one-variable function from the (usually) much more complicated function that we wish to differentate. Then pass it to this function.

Inputs: f = The one-variable function f(x) that we're differentiating h = The finite difference step size, usually a small number

Outputs: fp = The finite difference derivative of the function f(x) around x=0.

Definition at line 109 of file finite_difference.py.

Here is the call graph for this function:

7.10.1.2 def forcebalance.finite_difference.f12d7p (*f, h*)

Definition at line 75 of file finite_difference.py.

Here is the call graph for this function:

7.10.1.3 def forcebalance.finite_difference.f1d2p (*f, h, f0* = `None`)

A two-point finite difference stencil.

This function does either two computations or one, depending on whether the 'center' value is supplied. This is done in order to avoid recomputing the center value many times when we repeat this function for each index of the gradient.

How to use: use fdwrap or something similar to generate a one-variable function from the (usually) much more complicated function that we wish to differentate. Then pass it to this function.

Inputs: f = The one-variable function f(x) that we're differentiating h = The finite difference step size, usually a small number

Outputs: fp = The finite difference derivative of the function f(x) around x=0.

Definition at line 29 of file finite_difference.py.

Here is the call graph for this function:

7.10.1.4 def forcebalance.finite_difference.f1d5p (*f, h*)

A highly accurate five-point finite difference stencil for computing derivatives of a function.

It works on both scalar and vector functions (i.e. functions that return arrays). Since the function does four computations, it's costly but recommended if we really need an accurate reference value.

The function is evaluated at points -2h, -h, +h and +2h and these values are combined to make the derivative according to: `http://www.holoborodko.com/pavel/numerical-methods/numerical-derivative/central-differen`

How to use: use fdwrap or something similar to generate a one-variable function from the (usually) much more complicated function that we wish to differentate. Then pass it to this function.

Inputs: f = The one-variable function f(x) that we're differentiating h = The finite difference step size, usually a small number

Outputs: fp = The finite difference derivative of the function f(x) around x=0.

Definition at line 60 of file finite_difference.py.

Here is the call graph for this function:

7.10.1.5 def forcebalance.finite_difference.f1d7p (*f, h*)

A highly accurate seven-point finite difference stencil for computing derivatives of a function.

Definition at line 70 of file finite_difference.py.

Here is the call graph for this function:

7.10.1.6 def forcebalance.finite_difference.fdwrap (*func, mvals0, pidx, key* = `None`, *kwargs*)

A function wrapper for finite difference designed for differentiating 'get'-type functions.

Since our finite difference stencils take single-variable functions and differentiate them around zero, and our objective function is quite a complicated function, we need a wrapper to serve as a middleman. The alternative would be to copy the finite difference formula to wherever we're taking the derivative, and that is prone to mistakes.

Inputs: func = Either get_X or get_G; these functions return dictionaries. ['X'] = 1.23, ['G'] = [0.12, 3,45, ...] mvals0 = The 'central' values of the mathematical parameters - i.e. the wrapped function's origin is here. pidx = The index of the parameter that we're differentiating key = either 'G' or 'X', the value we wish to take out of the dictionary kwargs = Anything else we want to pass to the objective function (for instance, Project.Objective takes Order as an argument)

Outputs: func1 = Wrapped version of func, which takes a single float argument.

Definition at line 147 of file finite_difference.py.

Here is the call graph for this function:

7.10.1.7 def forcebalance.finite_difference.fdwrap_G (*tgt, mvals0, pidx*)

A driver to fdwrap for gradients (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.

the wrapped function's origin is here. pidx = The index of the parameter that we're differentiating

Definition at line 166 of file finite_difference.py.

Here is the call graph for this function:

7.10.1.8 def forcebalance.finite_difference.fdwrap_H (*tgt, mvals0, pidx*)

A driver to fdwrap for Hessians (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.

the wrapped function's origin is here. pidx = The index of the parameter that we're differentiating

Definition at line 177 of file finite_difference.py.

7.10.1.9 def forcebalance.finite_difference.in_fd ( )

Invoking this function from anywhere will tell us whether we're being called by a finite-difference function.

This is mainly useful for deciding when to update the 'qualitative indicators' and when not to.

Definition at line 121 of file finite_difference.py.

7.10.2 Variable Documentation

7.10.2.1 tuple forcebalance.finite_difference.logger = getLogger(__name__)

Definition at line 7 of file finite_difference.py.

7.11 forcebalance.forcefield Namespace Reference

Force field module.

Classes

- class BackedUpDict
- class FF

  Force field class.

Functions

- def determine_fftype

  Determine the type of a force field file.
- def rs_override

  This function takes in a dictionary (rsfactors) and a string (termtype).

Variables

- tuple logger = getLogger(__name__)
- dictionary FF_Extensions
- dictionary FF_IOModules

7.11.1 Detailed Description

Force field module. In ForceBalance a 'force field' is built from a set of files containing physical parameters. These files can be anything that enter into any computation - our original program was quite dependent on the GROMACS force field format, but this program is set up to allow very general input formats.

We introduce several important concepts:

1) Adjustable parameters are allocated into a vector.

To cast the force field optimization as a math problem, we treat all of the parameters on equal footing and write them as indices in a parameter vector.

2) A mapping from interaction type to parameter number.

Each element in the parameter vector corresponds to one or more interaction types. Whenever we change the parameter vector and recompute the objective function, this amounts to changing the physical parameters in the simulations, so we print out new force field files for external programs. In addition, when these programs are computing the objective function we are often in low-level subroutines that compute terms in the energy and force. If we need an analytic derivative of the objective function, then these subroutines need to know which index of the parameter vector needs to be modified.

This is done by way of a hash table: for example, when we are computing a Coulomb interaction between atom 4 and atom 5, we can build the words 'COUL4' and 'COUL5' and look it up in the parameter map; this gives us two numbers (say, 10 and 11) corresponding to the eleventh and twelfth element of the parameter vector. Then we can compute the

derivatives of the energy w/r.t. these parameters (in this case, COUL5/rij and COUL4/rij) and increment these values in the objective function gradient.

In custom-implemented force fields (see counterpoisematch.py) the hash table can also be used to look up parameter values for computation of interactions. This is probably not the fastest way to do things, however.

3) Distinction between physical and mathematical parameters.

The optimization algorithm works in a space that is related to, but not exactly the same as the physical parameter space. The reasons for why we do this are:

a) Each parameter has its own physical units. On the one hand it's not right to treat different physical units all on the same footing, so nondimensionalization is desirable. To make matters worse, the force field parameters can be small as 1e-8 or as large as 1e+6 depending on the parameter type. This means the elements of the objective function gradient / Hessian have elements that differ from each other in size by 10+ orders of magnitude, leading to mathematical instabilities in the optimizer.

b) The parameter space can be constrained, most notably for atomic partial charges where we don't want to change the overall charge on a molecule. Thus we wish to project out certain movements in the mathematical parameters such that they don't change the physical parameters.

c) We wish to regularize our optimization so as to avoid changing our parameters in very insensitive directions (linear dependencies). However, the sensitivity of the objective function to changes in the force field depends on the physical units!

For all of these reasons, we introduce a 'transformation matrix' which maps mathematical parameters onto physical parameters. The diagonal elements in this matrix are rescaling factors; they take the mathematical parameter and magnify it by this constant factor. The off-diagonal elements correspond to rotations and other linear transformations, and currently I just use them to project out the 'increase the net charge' direction in the physical parameter space.

Note that with regularization, these rescaling factors are equivalent to the widths of prior distributions in a maximum likelihood framework. Because there is such a correspondence between rescaling factors and choosing a prior, they need to be chosen carefully. This is work in progress. Another possibility is to sample the width of the priors from a noninformative distribution – the hyperprior (we can choose the Jeffreys prior or something). This is work in progress.

Right now only GROMACS parameters are supported, but this class is extensible, we need more modules!

Author

Lee-Ping Wang

Date

04/2012

7.11.2 Function Documentation

7.11.2.1 def forcebalance.forcefield.determine_fftype (*ffname, verbose* = `False`)

Determine the type of a force field file.

It is possible to specify the file type explicitly in the input file using the syntax 'force_field.ext:type'. Otherwise this function will try to determine the force field type by extension.

Definition at line 145 of file forcefield.py.

7.11.2.2 def forcebalance.forcefield.rs_override (*rsfactors, termtype, Temperature* = `298.15`)

This function takes in a dictionary (rsfactors) and a string (termtype).

```
If termtype matches any of the strings below, rsfactors[termtype] is assigned
to one of the numbers below.

This is LPW's attempt to simplify the rescaling factors.
```

Parameters

out	*rsfactors*	The computed rescaling factor.
in	*termtype*	The interaction type (corresponding to a physical unit)
in	*Temperature*	The temperature for computing the kT energy scale

Definition at line 1172 of file forcefield.py.

7.11.3 Variable Documentation

7.11.3.1 dictionary forcebalance.forcefield.FF_Extensions

Initial value:

```
= {"itp" : "gmx",
                  "in"  : "qchem",
                  "prm" : "tinker",
                  "gen" : "custom",
                  "xml" : "openmm",
                  "frcmod" : "frcmod",
                  "mol2" : "mol2",
                  "gbs"  : "gbs",
                  "grid" : "grid"
                   }
```

Definition at line 117 of file forcefield.py.

7.11.3.2 dictionary forcebalance.forcefield.FF_IOModules

Initial value:

```
= {"gmx": gmxio.ITP_Reader ,
                 "qchem": qchemio.QCIn_Reader ,
                 "tinker": tinkerio.Tinker_Reader ,
                 "custom": custom_io.Gen_Reader ,
                 "openmm" : openmmio.OpenMM_Reader,
                 "frcmod" : amberio.FrcMod_Reader,
                 "mol2" : amberio.Mol2_Reader,
                 "gbs" : psi4io.GBS_Reader,
                 "grid" : psi4io.Grid_Reader
                  }
```

Definition at line 129 of file forcefield.py.

7.11.3.3 tuple forcebalance.forcefield.logger = getLogger(__name__)

Definition at line 115 of file forcefield.py.

7.12 forcebalance.gmxio Namespace Reference

GROMACS input/output.

Classes

- class ITP_Reader

  Finite state machine for parsing GROMACS force field files.
- class AbInitio_GMX

  Subclass of AbInitio for force and energy matching using normal GROMACS.
- class Liquid_GMX
- class Interaction_GMX

  Subclass of Interaction for interaction energy matching using GROMACS.

Functions

- def edit_mdp

  Create or edit a Gromacs MDP file.
- def parse_atomtype_line

  Parses the 'atomtype' line.
- def rm_gmx_baks

Variables

- tuple logger = getLogger(__name__)
- list nftypes = [None, 'VDW', 'VDW_BHAM']

  VdW interaction function types.
- list pftypes = [None, 'VPAIR', 'VPAIR_BHAM']

  Pairwise interaction function types.
- list bftypes = [None, 'BONDS', 'G96BONDS', 'MORSE']

  Bonded interaction function types.
- list aftypes

  Angle interaction function types.
- list dftypes = [None, 'PDIHS', 'IDIHS', 'RBDIHS', 'PIMPDIHS', 'FOURDIHS', None, None, 'TABDIHS', 'PDIHMULS']

  Dihedral interaction function types.
- dictionary fdict

  Section -> Interaction type dictionary.
- dictionary pdict

  Interaction type -> Parameter Dictionary.

7.12.1 Detailed Description

GROMACS input/output.

Todo Even more stuff from forcefield.py needs to go into here.

Author

Lee-Ping Wang

Date

12/2011

Todo Even more stuff from forcefield.py needs to go into here.

Author

Lee-Ping Wang

Date

12/2011

7.12.2 Function Documentation

7.12.2.1 def forcebalance.gmxio.edit_mdp (*fin, fout, options, verbose* = `False`)

Create or edit a Gromacs MDP file.

Parameters

in	*fin*	Input file name.
in	*fout*	Output file name, can be the same as input file name.
in	*options*	Dictionary containing mdp options. Existing options are replaced, new options are added at the end.

Definition at line 35 of file gmxio.py.

Here is the call graph for this function:

7.12.2.2 def forcebalance.gmxio.parse_atomtype_line (*line*)

Parses the 'atomtype' line.

```
Parses lines like this:\n
<tt> opls_135     CT    6   12.0107    0.0000    A    3.5000e-01    2.7614e-01\n
C       12.0107    0.0000    A    3.7500e-01    4.3932e-01\n
Na  11    22.9897    0.0000    A    6.068128070229e+03  2.662662556402e+01  0.0000e+00 ; PARM 5 6\n </tt>
Look at all the variety!
```

Parameters

in	*line*	Input line.

Returns

answer Dictionary containing:
atom type
bonded atom type (if any)
atomic number (if any)
atomic mass
charge
particle type
force field parameters
number of optional fields

Definition at line 181 of file gmxio.py.

Here is the call graph for this function:

7.12.2.3 def forcebalance.gmxio.rm_gmx_baks (*dir*)

Definition at line 428 of file gmxio.py.

7.12.3 Variable Documentation

7.12.3.1 list forcebalance.gmxio.aftypes

Initial value:

```
1 = [None, 'ANGLES', 'G96ANGLES', 'CROSS_BOND_BOND',
2            'CROSS_BOND_ANGLE', 'UREY_BRADLEY', 'QANGLES']
```

Angle interaction function types.

Definition at line 91 of file gmxio.py.

7.12.3.2 list forcebalance.gmxio.bftypes = [None, 'BONDS', 'G96BONDS', 'MORSE']

Bonded interaction function types.

Definition at line 89 of file gmxio.py.

7.12.3.3 list forcebalance.gmxio.dftypes = [None, 'PDIHS', 'IDIHS', 'RBDIHS', 'PIMPDIHS', 'FOURDIHS', None, None, 'TABDIHS', 'PDIHMULS']

Dihedral interaction function types.

Definition at line 94 of file gmxio.py.

7.12.3.4 dictionary forcebalance.gmxio.fdict

Initial value:

```
1 = {
2     'atomtypes'     : nftypes,
3     'nonbond_params': pftypes,
4     'bonds'         : bftypes,
5     'bondtypes'     : bftypes,
6     'angles'        : aftypes,
7     'angletypes'    : aftypes,
8     'dihedrals'     : dftypes,
9     'dihedraltypes' : dftypes,
10     'virtual_sites2': ['NONE','VSITE2'],
11     'virtual_sites3': ['NONE','VSITE3','VSITE3FD','VSITE3FAD','VSITE3OUT'],
12     'virtual_sites4': ['NONE','VSITE4FD','VSITE4FDN']
13     }
```

Section -> Interaction type dictionary.

Based on the section you're in and the integer given on the current line, this looks up the 'interaction type' - for example, within bonded interactions there are four interaction types: harmonic, G96, Morse, and quartic interactions.

Definition at line 102 of file gmxio.py.

7.12.3.5 tuple forcebalance.gmxio.logger = getLogger(__name__)

Definition at line 26 of file gmxio.py.

7.12.3.6 list forcebalance.gmxio.nftypes = [None, 'VDW', 'VDW_BHAM']

VdW interaction function types.

Definition at line 85 of file gmxio.py.

7.12.3.7 dictionary forcebalance.gmxio.pdict

Interaction type -> Parameter Dictionary.

A list of supported GROMACS interaction types in force matching. The keys in this dictionary (e.g. 'BONDS','ANGLES') are values in the interaction type dictionary. As the program loops through the force field file, it first looks up the interaction types in 'fdict' and then goes here to do the parameter lookup by field.

Todo This needs to become more flexible because the parameter isn't always in the same field. Still need to figure out how to do this.

How about making the PDIHS less ugly?

Definition at line 125 of file gmxio.py.

7.12.3.8 list forcebalance.gmxio.pftypes = [None, 'VPAIR', 'VPAIR_BHAM']

Pairwise interaction function types.

Definition at line 87 of file gmxio.py.

7.13 forcebalance.gmxqpio Namespace Reference

Classes

- class Monomer_QTPIE

  Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).

Functions

- def get_monomer_properties

Variables

- tuple logger = getLogger(__name__)

7.13.1 Function Documentation

7.13.1.1 def forcebalance.gmxqpio.get_monomer_properties (*print_stuff* = 0)

Definition at line 28 of file gmxqpio.py.

Here is the call graph for this function:

7.13.2 Variable Documentation

7.13.2.1 tuple forcebalance.gmxqpio.logger = getLogger(__name__)

Definition at line 22 of file gmxqpio.py.

7.14 forcebalance.interaction Namespace Reference

Interaction energy fitting module.

Classes

- class Interaction

  Subclass of Target for fitting force fields to interaction energies.

Variables

- tuple logger = getLogger(__name__)

7.14.1 Detailed Description

Interaction energy fitting module.

Author

Lee-Ping Wang

Date

05/2012

7.14.2 Variable Documentation

7.14.2.1 tuple forcebalance.interaction.logger = getLogger(__name__)

Definition at line 21 of file interaction.py.

7.15 forcebalance.leastsq Namespace Reference

Classes

- class LeastSquares

  Subclass of Target for general least squares fitting.

Functions

- def CheckBasis
- def LastMvals

Variables

- tuple logger = getLogger(__name__)
- CHECK_BASIS = False
- LAST_MVALS = None

7.15.1 Function Documentation

7.15.1.1 def forcebalance.leastsq.CheckBasis ( )

Definition at line 24 of file leastsq.py.

7.15.1.2 def forcebalance.leastsq.LastMvals ( )

Definition at line 29 of file leastsq.py.

7.15.2 Variable Documentation

7.15.2.1 forcebalance.leastsq.CHECK_BASIS = False

Definition at line 23 of file leastsq.py.

7.15.2.2 forcebalance.leastsq.LAST_MVALS = None

Definition at line 28 of file leastsq.py.

7.15.2.3 tuple forcebalance.leastsq.logger = getLogger(__name__)

Definition at line 21 of file leastsq.py.

7.16 forcebalance.liquid Namespace Reference

Matching of liquid bulk properties.

Classes

- class Liquid

  Subclass of Target for liquid property matching.

Functions

- def weight_info

Variables

- tuple logger = getLogger(__name__)

7.16.1 Detailed Description

Matching of liquid bulk properties. Under development.

Author

Lee-Ping Wang

Date

04/2012

7.16.2 Function Documentation

7.16.2.1 def forcebalance.liquid.weight_info (*W, PT, N_k, verbose* = `True`)

Definition at line 32 of file liquid.py.

7.16.3 Variable Documentation

7.16.3.1 tuple forcebalance.liquid.logger = getLogger(__name__)

Definition at line 30 of file liquid.py.

7.17 forcebalance.Mol2 Namespace Reference

Classes

- class mol2_atom

  This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.
- class mol2_bond

  This is to manage mol2 bond lines on the form: 1 1 2 ar.
- class mol2

  This is to manage one mol2 series of lines on the form:
- class mol2_set

Variables

- tuple data = mol2_set(sys.argv[1], subset=["RNAse.xray.inh8.1QHC"])

7.17.1 Variable Documentation

7.17.1.1 tuple forcebalance.Mol2.data = mol2_set(sys.argv[1], subset=["RNAse.xray.inh8.1QHC"])

Definition at line 651 of file Mol2.py.

7.18 forcebalance.mol2io Namespace Reference

Mol2 I/O.

Classes

- class Mol2_Reader

  Finite state machine for parsing Mol2 force field file.

Variables

- dictionary mol2_pdict = {'COUL':{'Atom':[1], 6:''}}

7.18.1 Detailed Description

Mol2 I/O. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

Author

Lee-Ping Wang

Date

05/2012

7.18.2 Variable Documentation

7.18.2.1 dictionary forcebalance.mol2io.mol2_pdict = {'COUL':{'Atom':[1], 6:''}}

Definition at line 18 of file mol2io.py.

7.19 forcebalance.molecule Namespace Reference

Classes

- class MolfileTimestep

  Wrapper for the timestep C structure used in molfile plugins.
- class Molecule

  Lee-Ping's general file format conversion class.

Functions

- def getElement
- def nodematch
- def isint

  ONLY matches integers! If you have a decimal point? None shall pass!
- def isfloat

  Matches ANY number; it can be a decimal, scientific notation, integer, or what have you.
- def BuildLatticeFromLengthsAngles

  This function takes in three lattice lengths and three lattice angles, and tries to return a complete box specification.
- def BuildLatticeFromVectors

  This function takes in three lattice vectors and tries to return a complete box specification.
- def format_xyz_coord

  Print a line consisting of (element, x, y, z) in accordance with .xyz file format.
- def format_gro_coord

  Print a line in accordance with .gro file format, with six decimal points of precision.
- def format_xyzgen_coord

  Print a line consisting of (element, p, q, r, s, t, ...) where (p, q, r) are arbitrary atom-wise data (this might happen, for instance, with atomic charges)
- def format_gro_box

  Print a line corresponding to the box vector in accordance with .gro file format.
- def is_gro_coord

  Determines whether a line contains GROMACS data or not.
- def is_charmm_coord

  Determines whether a line contains CHARMM data or not.
- def is_gro_box

  Determines whether a line contains a GROMACS box vector or not.
- def add_strip_to_mat
- def pvec
- def grouper

  Groups a big long iterable into groups of ten or what have you.
- def even_list

  Creates a list of number sequences divided as evenly as possible.

- def both
- def diff
- def either
- def EulerMatrix

  Constructs an Euler matrix from three Euler angles.
- def ComputeOverlap

  Computes an 'overlap' between two molecules based on some fictitious density.
- def AlignToDensity

  Computes a "overlap density" from two frames.
- def AlignToMoments

  Pre-aligns molecules to 'moment of inertia'.
- def get_rotate_translate
- def main

Variables

- tuple FrameVariableNames
- tuple AtomVariableNames = set(['elem', 'partial_charge', 'atomname', 'atomtype', 'tinkersuf', 'resid', 'resname', 'qcsuf', 'qm_ghost', 'chain', 'altloc', 'icode'])
- tuple MetaVariableNames = set(['fnm', 'ftype', 'qcrems', 'qctemplate', 'charge', 'mult', 'bonds'])
- tuple QuantumVariableNames = set(['qcrems', 'qctemplate', 'charge', 'mult', 'qcsuf', 'qm_ghost'])
- AllVariableNames = QuantumVariableNames|AtomVariableNames|MetaVariableNames|FrameVariableNames
- list Radii
- list Elements
- tuple PeriodicTable
- float bohrang = 0.529177249

  One bohr equals this many angstroms.
- tuple splitter = re.compile(r'(\s+|\S+)')
- tuple Box = namedtuple('Box',['a','b','c','alpha','beta','gamma','A','B','C','V'])
- int radian = 180
- Alive

7.19.1 Function Documentation

7.19.1.1 def forcebalance.molecule.add_strip_to_mat (*mat, strip*)

Definition at line 436 of file molecule.py.

Here is the call graph for this function:

7.19.1.2 def forcebalance.molecule.AlignToDensity (*elem, xyz1, xyz2, binary* = `False`)

Computes a "overlap density" from two frames.

This function can be called by AlignToMoments to get rid of inversion problems

Definition at line 539 of file molecule.py.

Here is the call graph for this function:

7.19.1.3 def forcebalance.molecule.AlignToMoments (*elem, xyz1, xyz2* = `None`)

Pre-aligns molecules to 'moment of inertia'.

If xyz2 is passed in, it will assume that xyz1 is already aligned to the moment of inertia, and it simply does 180-degree rotations to make sure nothing is inverted.

Definition at line 551 of file molecule.py.

7.19.1.4 def forcebalance.molecule.both (*A, B, key*)

Definition at line 474 of file molecule.py.

7.19.1.5 def forcebalance.molecule.BuildLatticeFromLengthsAngles (*a, b, c, alpha, beta, gamma*)

This function takes in three lattice lengths and three lattice angles, and tries to return a complete box specification.

Definition at line 256 of file molecule.py.

Here is the call graph for this function:

7.19.1.6 def forcebalance.molecule.BuildLatticeFromVectors (*v1, v2, v3*)

This function takes in three lattice vectors and tries to return a complete box specification.

The hash function is something we can use to discard two things that are obviously not equal. Here we neglect the hash. Return a list of the sorted atom numbers in this graph. Return a string of atoms, which serves as a rudimentary

'fingerprint' : '99,100,103,151' . Return an array of the elements. For instance ['H' 'C' 'C' 'H']. Create an Empirical Formula Get a list of the coordinates.

Definition at line 271 of file molecule.py.

7.19.1.7 def forcebalance.molecule.ComputeOverlap (*theta, elem, xyz1, xyz2*)

Computes an 'overlap' between two molecules based on some fictitious density.

Good for fine-tuning alignment but gets stuck in local minima.

Definition at line 522 of file molecule.py.

Here is the call graph for this function:

7.19.1.8 def forcebalance.molecule.diff (*A, B, key*)

Definition at line 477 of file molecule.py.

7.19.1.9 def forcebalance.molecule.either (*A, B, key*)

Definition at line 485 of file molecule.py.

7.19.1.10 def forcebalance.molecule.EulerMatrix (*T1, T2, T3*)

Constructs an Euler matrix from three Euler angles.

Definition at line 494 of file molecule.py.

Here is the call graph for this function:

7.19.1.11 def forcebalance.molecule.even_list (*totlen, splitsize*)

Creates a list of number sequences divided as evenly as possible.

Definition at line 456 of file molecule.py.

Here is the call graph for this function:

7.19.1.12 def forcebalance.molecule.format_gro_box (*box*)

Print a line corresponding to the box vector in accordance with .gro file format.

Parameters

in	*box*	Box NamedTuple

Definition at line 387 of file molecule.py.

Here is the call graph for this function:

7.19.1.13 def forcebalance.molecule.format_gro_coord (*resid, resname, aname, seqno, xyz*)

Print a line in accordance with .gro file format, with six decimal points of precision.

Parameters

in	*resid*	The number of the residue that the atom belongs to
in	*resname*	The name of the residue that the atom belongs to
in	*aname*	The name of the atom
in	*seqno*	The sequential number of the atom
in	*xyz*	A 3-element array containing x, y, z coordinates of that atom

Definition at line 366 of file molecule.py.

Here is the call graph for this function:

7.19.1.14 def forcebalance.molecule.format_xyz_coord (*element, xyz, tinker* = `False`)

Print a line consisting of (element, x, y, z) in accordance with .xyz file format.

Parameters

in	*element*	A chemical element of a single atom
in	*xyz*	A 3-element array containing x, y, z coordinates of that atom

Definition at line 350 of file molecule.py.

Here is the call graph for this function:

7.19.1.15 def forcebalance.molecule.format_xyzgen_coord (*element, xyzgen*)

Print a line consisting of (element, p, q, r, s, t, ...) where (p, q, r) are arbitrary atom-wise data (this might happen, for instance, with atomic charges)

Parameters

in	*element*	A chemical element of a single atom
in	*xyzgen*	A N-element array containing data for that atom

Definition at line 378 of file molecule.py.

Here is the call graph for this function:

7.19.1.16 def forcebalance.molecule.get_rotate_translate (*matrix1, matrix2*)

Definition at line 574 of file molecule.py.

7.19.1.17 def forcebalance.molecule.getElement (*mass*)

Definition at line 189 of file molecule.py.

7.19.1.18 def forcebalance.molecule.grouper (*n, iterable*)

Groups a big long iterable into groups of ten or what have you.

Definition at line 450 of file molecule.py.

Here is the call graph for this function:

7.19.1.19 def forcebalance.molecule.is_charmm_coord (*line*)

Determines whether a line contains CHARMM data or not.

Parameters

in	*line*	The line to be tested

Definition at line 414 of file molecule.py.

Here is the call graph for this function:

7.19.1.20 def forcebalance.molecule.is_gro_box (*line*)

Determines whether a line contains a GROMACS box vector or not.

Parameters

in	*line*	The line to be tested

Definition at line 427 of file molecule.py.

Here is the call graph for this function:

7.19.1.21 def forcebalance.molecule.is_gro_coord (*line*)

Determines whether a line contains GROMACS data or not.

Parameters

in	*line*	The line to be tested

Definition at line 399 of file molecule.py.

Here is the call graph for this function:

7.19.1.22 def forcebalance.molecule.isfloat (*word*)

Matches ANY number; it can be a decimal, scientific notation, integer, or what have you.

Definition at line 245 of file molecule.py.

7.19.1.23 def forcebalance.molecule.isint (*word*)

ONLY matches integers! If you have a decimal point? None shall pass!

Definition at line 240 of file molecule.py.

Here is the call graph for this function:

7.19.1.24 def forcebalance.molecule.main ( )

Definition at line 2500 of file molecule.py.

7.19.1.25 def forcebalance.molecule.nodematch (*node1*, *node2*)

Definition at line 234 of file molecule.py.

Here is the call graph for this function:

7.19.1.26 def forcebalance.molecule.pvec (*vec*)

Definition at line 445 of file molecule.py.

7.19.2 Variable Documentation

7.19.2.1 forcebalance.molecule.Alive

Definition at line 300 of file molecule.py.

7.19.2.2 forcebalance.molecule.AllVariableNames = QuantumVariableNames|AtomVariableNames|MetaVariableNames|FrameVariableNames

Definition at line 137 of file molecule.py.

7.19.2.3 tuple forcebalance.molecule.AtomVariableNames = set(['elem', 'partial_charge', 'atomname', 'atomtype', 'tinkersuf', 'resid', 'resname', 'qcsuf', 'qm_ghost', 'chain', 'altloc', 'icode'])

Definition at line 120 of file molecule.py.

7.19.2.4 float forcebalance.molecule.bohrang = 0.529177249

One bohr equals this many angstroms.

Definition at line 232 of file molecule.py.

7.19.2.5 tuple forcebalance.molecule.Box = namedtuple('Box',['a','b','c','alpha','beta','gamma','A','B','C','V'])

Definition at line 252 of file molecule.py.

7.19.2.6 list forcebalance.molecule.Elements

Initial value:

```
= ["None",'H','He',
            'Li','Be','B','C','N','O','F','Ne',
            'Na','Mg','Al','Si','P','S','Cl','Ar',
            'K','Ca','Sc','Ti','V','Cr','Mn','Fe','Co','Ni','Cu','Zn','Ga','Ge','As','Se','Br','Kr',
            'Rb','Sr','Y','Zr','Nb','Mo','Tc','Ru','Rh','Pd','Ag','Cd','In','Sn','Sb','Te','I','Xe',
            'Cs','Ba','La','Ce','Pr','Nd','Pm','Sm','Eu','Gd','Tb','Dy','Ho','Er','Tm','Yb',
            'Lu','Hf','Ta','W','Re','Os','Ir','Pt','Au','Hg','Tl','Pb','Bi','Po','At','Rn',
```

```
8            'Fr','Ra','Ac','Th','Pa','U','Np','Pu','Am','Cm','Bk','Cf','Es','Fm','Md','No','Lr','Rf','Db','
      Sg','Bh','Hs','Mt']
```

Definition at line 164 of file molecule.py.

7.19.2.7 tuple forcebalance.molecule.FrameVariableNames

Initial value:

```
1 = set(['xyzs', 'comms', 'boxes', 'qm_forces', 'qm_energies', 'qm_interaction',
2                          'qm_espxyzs', 'qm_espvals', 'qm_extchgs', 'qm_mulliken_charges', '
      qm_mulliken_spins'])
```

Definition at line 106 of file molecule.py.

7.19.2.8 tuple forcebalance.molecule.MetaVariableNames = set(['fnm', 'ftype', 'qcrems', 'qctemplate', 'charge', 'mult', 'bonds'])

Definition at line 133 of file molecule.py.

7.19.2.9 tuple forcebalance.molecule.PeriodicTable

Initial value:

```
1 = OrderedDict([('H' , 1.0079), ('He' , 4.0026),
2                            ('Li' , 6.941), ('Be' , 9.0122), ('B' , 10.811), ('C' , 12.0107), ('N' , 14.00
      67), ('O' , 15.9994), ('F' , 18.9984), ('Ne' , 20.1797),
3                            ('Na' , 22.9897), ('Mg' , 24.305), ('Al' , 26.9815), ('Si' , 28.0855), ('P' ,
      30.9738), ('S' , 32.065), ('Cl' , 35.453), ('Ar' , 39.948),
4                            ('K' , 39.0983), ('Ca' , 40.078), ('Sc' , 44.9559), ('Ti' , 47.867), ('V' , 50
      .9415), ('Cr' , 51.9961), ('Mn' , 54.938), ('Fe' , 55.845), ('Co' , 58.9332),
5                            ('Ni' , 58.6934), ('Cu' , 63.546), ('Zn' , 65.39), ('Ga' , 69.723), ('Ge' , 72
      .64), ('As' , 74.9216), ('Se' , 78.96), ('Br' , 79.904), ('Kr' , 83.8),
6                            ('Rb' , 85.4678), ('Sr' , 87.62), ('Y' , 88.9059), ('Zr' , 91.224), ('Nb' , 92
      .9064), ('Mo' , 95.94), ('Tc' , 98), ('Ru' , 101.07), ('Rh' , 102.9055),
7                            ('Pd' , 106.42), ('Ag' , 107.8682), ('Cd' , 112.411), ('In' , 114.818), ('Sn'
      , 118.71), ('Sb' , 121.76), ('Te' , 127.6), ('I' , 126.9045), ('Xe' , 131.293),
8                            ('Cs' , 132.9055), ('Ba' , 137.327), ('La' , 138.9055), ('Ce' , 140.116), ('Pr
      ' , 140.9077), ('Nd' , 144.24), ('Pm' , 145), ('Sm' , 150.36),
9                            ('Eu' , 151.964), ('Gd' , 157.25), ('Tb' , 158.9253), ('Dy' , 162.5), ('Ho' ,
      164.9303), ('Er' , 167.259), ('Tm' , 168.9342), ('Yb' , 173.04),
10                            ('Lu' , 174.967), ('Hf' , 178.49), ('Ta' , 180.9479), ('W' , 183.84), ('Re' ,
      186.207), ('Os' , 190.23), ('Ir' , 192.217), ('Pt' , 195.078),
11                            ('Au' , 196.9665), ('Hg' , 200.59), ('Tl' , 204.3833), ('Pb' , 207.2), ('Bi' ,
       208.9804), ('Po' , 209), ('At' , 210), ('Rn' , 222),
12                            ('Fr' , 223), ('Ra' , 226), ('Ac' , 227), ('Th' , 232.0381), ('Pa' , 231.0359)
      , ('U' , 238.0289), ('Np' , 237), ('Pu' , 244),
13                            ('Am' , 243), ('Cm' , 247), ('Bk' , 247), ('Cf' , 251), ('Es' , 252), ('Fm' ,
      257), ('Md' , 258), ('No' , 259),
14                            ('Lr' , 262), ('Rf' , 261), ('Db' , 262), ('Sg' , 266), ('Bh' , 264), ('Hs' ,
      277), ('Mt' , 268)])
```

Definition at line 174 of file molecule.py.

7.19.2.10 tuple forcebalance.molecule.QuantumVariableNames = set(['qcrems', 'qctemplate', 'charge', 'mult', 'qcsuf', 'qm_ghost'])

Definition at line 135 of file molecule.py.

7.19.2.11 int forcebalance.molecule.radian = 180

Definition at line 253 of file molecule.py.

7.19.2.12 list forcebalance.molecule.Radii

Initial value:

```
1 = [0.31, 0.28, # H and He
2          1.28, 0.96, 0.84, 0.76, 0.71, 0.66, 0.57, 0.58, # First row elements
3          1.66, 1.41, 1.21, 1.11, 1.07, 1.05, 1.02, 1.06, # Second row elements
4          2.03, 1.76, 1.70, 1.60, 1.53, 1.39, 1.61, 1.52, 1.50,
5          1.24, 1.32, 1.22, 1.22, 1.20, 1.19, 1.20, 1.20, 1.16, # Third row elements, K through Kr
6          2.20, 1.95, 1.90, 1.75, 1.64, 1.54, 1.47, 1.46, 1.42,
7          1.39, 1.45, 1.44, 1.42, 1.39, 1.39, 1.38, 1.39, 1.40, # Fourth row elements, Rb through Xe
8          2.44, 2.15, 2.07, 2.04, 2.03, 2.01, 1.99, 1.98,
9          1.98, 1.96, 1.94, 1.92, 1.92, 1.89, 1.90, 1.87, # Fifth row elements, s and f blocks
10          1.87, 1.75, 1.70, 1.62, 1.51, 1.44, 1.41, 1.36,
11          1.36, 1.32, 1.45, 1.46, 1.48, 1.40, 1.50, 1.50, # Fifth row elements, d and p blocks
12          2.60, 2.21, 2.15, 2.06, 2.00, 1.96, 1.90, 1.87, 1.80, 1.69]
```

Definition at line 150 of file molecule.py.

7.19.2.13 tuple forcebalance.molecule.splitter = re.compile(r'(\s+|\S+)')

Definition at line 249 of file molecule.py.

7.20 forcebalance.moments Namespace Reference

Multipole moment fitting module.

Classes

- class Moments

  Subclass of Target for fitting force fields to multipole moments (from experiment or theory).

Variables

- tuple logger = getLogger(__name__)

7.20.1 Detailed Description

Multipole moment fitting module.

Author

Lee-Ping Wang

Date

09/2012

7.20.2 Variable Documentation

7.20.2.1 tuple forcebalance.moments.logger = getLogger(__name__)

Definition at line 22 of file moments.py.

7.21 forcebalance.nifty Namespace Reference

Nifty functions, intended to be imported by any module within ForceBalance.

Classes

- class Pickler_LP

  A subclass of the python Pickler that implements pickling of _ElementTree types.
- class Unpickler_LP

  A subclass of the python Unpickler that implements unpickling of _ElementTree types.

Functions

- def pvec1d

  Printout of a 1-D vector.
- def pmat2d

  Printout of a 2-D matrix.
- def encode
- def segments
- def commadash
- def uncommadash
- def printcool

  Cool-looking printout for slick formatting of output.
- def printcool_dictionary

  See documentation for printcool; this is a nice way to print out keys/values in a dictionary.
- def isint

  ONLY matches integers! If you have a decimal point? None shall pass!
- def isfloat

  Matches ANY number; it can be a decimal, scientific notation, what have you CAUTION - this will also match an integer.
- def isdecimal

  Matches things with a decimal only; see isint and isfloat.
- def floatornan

  Returns a big number if we encounter NaN.
- def col

  Given any list, array, or matrix, return a 1-column matrix.
- def row

  Given any list, array, or matrix, return a 1-row matrix.
- def flat

  Given any list, array, or matrix, return a single-index array.
- def orthogonalize

  Given two vectors vec1 and vec2, project out the component of vec1 that is along the vec2-direction.
- def invert_svd

  Invert a matrix using singular value decomposition.
- def get_least_squares
- def statisticalInefficiency

  Compute the (cross) statistical inefficiency of (two) timeseries.
- def lp_dump

  Use this instead of pickle.dump for pickling anything that contains _ElementTree types.
- def lp_load

  Use this instead of pickle.load for unpickling anything that contains _ElementTree types.
- def getWorkQueue

- def getWQIds
- def createWorkQueue
- def queue_up

  Submit a job to the Work Queue.
- def queue_up_src_dest

  Submit a job to the Work Queue.
- def wq_wait1

  This function waits ten seconds to see if a task in the Work Queue has finished.
- def wq_wait

  This function waits until the work queue is completely empty.
- def GoInto
- def allsplit
- def Leave
- def MissingFileInspection
- def LinkFile
- def CopyFile
- def link_dir_contents
- def remove_if_exists

  Remove the file if it exists (doesn't return an error).
- def which
- def warn_press_key
- def warn_once

  Prints a warning but will only do so once in a given run.
- def concurrent_map

  Similar to the bultin function map().
- def multiopen

  This function be given any of several variable types (single file name, file object, or list of lines, or a list of the above) and give a list of files:

Variables

- tuple logger = getLogger(__name__)
- float kb = 0.0083144100163

  Boltzmann constant.
- float eqcgmx = 2625.5002

  Q-Chem to GMX unit conversion for energy.
- float fqcgmx = -49621.9

  Q-Chem to GMX unit conversion for force.
- float bohrang = 0.529177249

  One bohr equals this many angstroms.
- string XMLFILE = 'x'

  Pickle uses 'flags' to pickle and unpickle different variable types.
- list specific_lst
- tuple specific_dct = dict(list(itertools.chain(*[[(j,i[1]) for j in i[0]] for i in specific_lst])))

7.21.1 Detailed Description

Nifty functions, intended to be imported by any module within ForceBalance. Table of Contents:

- I/O formatting
- Math: Variable manipulation, linear algebra, least squares polynomial fitting
- Pickle: Expand Python's own pickle to accommodate writing XML etree objects
- Commands for submitting things to the Work Queue
- Various file and process management functions
- Development stuff (not commonly used)

Named after the mighty Sniffy Handy Nifty (King Sniffy)

Author

Lee-Ping Wang

Date

12/2011

7.21.2 Function Documentation

7.21.2.1 def forcebalance.nifty.allsplit (*Dir*)

Definition at line 706 of file nifty.py.

Here is the call graph for this function:

7.21.2.2 def forcebalance.nifty.col (*vec*)

Given any list, array, or matrix, return a 1-column matrix.

Input: vec = The input vector that is to be made into a column

Output: A column matrix

Definition at line 270 of file nifty.py.

Here is the call graph for this function:

7.21.2.3 def forcebalance.nifty.commadash (*l*)

Definition at line 81 of file nifty.py.

Here is the call graph for this function:

7.21.2.4 def forcebalance.nifty.concurrent_map (*func, data*)

Similar to the bultin function map().

But spawn a thread for each argument and apply `func` concurrently.

Note: unlike map(), we cannot take an iterable argument. `data` should be an indexable sequence.

Definition at line 911 of file nifty.py.

7.21.2.5 def forcebalance.nifty.CopyFile (*src, dest*)

Definition at line 754 of file nifty.py.

Here is the call graph for this function:

7.21.2.6 def forcebalance.nifty.createWorkQueue (*wq_port, debug* = `True`)

Definition at line 572 of file nifty.py.

Here is the call graph for this function:

7.21.2.7 def forcebalance.nifty.encode (*l*)

Definition at line 72 of file nifty.py.

Here is the call graph for this function:

7.21.2.8 def forcebalance.nifty.flat (*vec*)

Given any list, array, or matrix, return a single-index array.

Parameters

in	*vec*	The data to be flattened

Returns

answer The flattened data

Definition at line 289 of file nifty.py.

Here is the call graph for this function:

7.21.2.9 def forcebalance.nifty.floatornan (*word*)

Returns a big number if we encounter NaN.

Parameters

in	*word*	The string to be converted

Returns

answer The string converted to a float; if not a float, return 1e10

Todo I could use suggestions for making this better.

Definition at line 252 of file nifty.py.

Here is the call graph for this function:

7.21.2.10 def forcebalance.nifty.get_least_squares (*x*, *y*, *w* = None, *thresh* = 1e-12)

```
 __                    __
|                        |
| 1 (x0)  (x0)^2  (x0)^3 |
| 1 (x1)  (x1)^2  (x1)^3 |
| 1 (x2)  (x2)^2  (x2)^3 |
| 1 (x3)  (x3)^2  (x3)^3 |
| 1 (x4)  (x4)^2  (x4)^3 |
|__                    __|
```

Parameters

in	*X*	(2-D array) An array of X-values (see above)
in	*Y*	(array) An array of Y-values (only used in getting the least squares coefficients)
in	*w*	(array) An array of weights, hopefully normalized to one.
out	*Beta*	The least-squares coefficients
out	*Hat*	The hat matrix that takes linear combinations of data y-values to give fitted y-values (weights)
out	*yfit*	The fitted y-values
out	*MPPI*	The Moore-Penrose pseudoinverse (multiply by Y to get least-squares coefficients, multiply by dY/dk to get derivatives of least-squares coefficients)

Definition at line 357 of file nifty.py.

Here is the call graph for this function:

7.21.2.11 def forcebalance.nifty.getWorkQueue ( )

Definition at line 566 of file nifty.py.

7.21.2.12 def forcebalance.nifty.getWQIds ( )

Definition at line 569 of file nifty.py.

7.21.2.13 def forcebalance.nifty.GoInto (*Dir*)

Definition at line 698 of file nifty.py.

7.21.2.14 def forcebalance.nifty.invert_svd (*X*, *thresh* = `1e-12`)

Invert a matrix using singular value decomposition.

Parameters

in	*X*	The matrix to be inverted
in	*thresh*	The SVD threshold; eigenvalues below this are not inverted but set to zero

Returns

Xt The inverted matrix

Definition at line 316 of file nifty.py.

Here is the call graph for this function:

7.21.2.15 def forcebalance.nifty.isdecimal (*word*)

Matches things with a decimal only; see isint and isfloat.

Parameters

in	*word*	String (for instance, '123', '153.0', '2.', '-354')

Returns

answer Boolean which specifies whether the string is a number with a decimal point

Definition at line 242 of file nifty.py.

Here is the call graph for this function:

7.21.2.16 def forcebalance.nifty.isfloat (*word*)

Matches ANY number; it can be a decimal, scientific notation, what have you CAUTION - this will also match an integer.

Parameters

in	*word*	String (for instance, '123', '153.0', '2.', '-354')

Returns

answer Boolean which specifies whether the string is any number

Definition at line 232 of file nifty.py.

Here is the call graph for this function:

7.21.2.17 def forcebalance.nifty.isint (*word*)

ONLY matches integers! If you have a decimal point? None shall pass!

Parameters

in	*word*	String (for instance, '123', '153.0', '2.', '-354')

Returns

answer Boolean which specifies whether the string is an integer (only +/- sign followed by digits)

Definition at line 221 of file nifty.py.

Here is the call graph for this function:

7.21.2.18 def forcebalance.nifty.Leave (*Dir*)

Definition at line 712 of file nifty.py.

7.21.2.19 def forcebalance.nifty.link_dir_contents (*abssrcdir, absdestdir*)

Definition at line 764 of file nifty.py.

7.21.2.20 def forcebalance.nifty.LinkFile (*src, dest*)

Definition at line 743 of file nifty.py.

Here is the call graph for this function:

7.21.2.21 def forcebalance.nifty.lp_dump (*obj, file, protocol* = `None`)

Use this instead of pickle.dump for pickling anything that contains _ElementTree types.

Definition at line 549 of file nifty.py.

Here is the call graph for this function:

7.21.2.22 def forcebalance.nifty.lp_load (*file*)

Use this instead of pickle.load for unpickling anything that contains _ElementTree types.

Definition at line 554 of file nifty.py.

Here is the call graph for this function:

7.21.2.23 def forcebalance.nifty.MissingFileInspection (*fnm*)

Definition at line 733 of file nifty.py.

Here is the call graph for this function:

7.21.2.24 def forcebalance.nifty.multiopen (*arg*)

This function be given any of several variable types (single file name, file object, or list of lines, or a list of the above) and give a list of files:

[file1, file2, file3 ...]

each of which can then be iterated over:

[[file1_line1, file1_line2 ...], [file2_line1, file2_line2 ...]]

Definition at line 941 of file nifty.py.

7.21.2.25 def forcebalance.nifty.orthogonalize (*vec1, vec2*)

Given two vectors vec1 and vec2, project out the component of vec1 that is along the vec2-direction.

Parameters

in	*vec1*	The projectee (i.e. output is some modified version of vec1)
in	*vec2*	The projector (component subtracted out from vec1 is parallel to this)

Returns

answer A copy of vec1 but with the vec2-component projected out.

Definition at line 303 of file nifty.py.

7.21.2.26 def forcebalance.nifty.pmat2d (*mat2d*, *precision* = `1`, *loglevel* = `forcebalance.output.INFO`)

Printout of a 2-D matrix.

Parameters

in	*mat2d*	a 2-D matrix

Definition at line 65 of file nifty.py.

Here is the call graph for this function:

7.21.2.27 def forcebalance.nifty.printcool (*text*, *sym* = `"#"`, *bold* = `False`, *color* = `2`, *ansi* = `None`, *bottom* = `'-'`, *minwidth* = `50`, *center* = `True`)

Cool-looking printout for slick formatting of output.

Parameters

in	*text*	The string that the printout is based upon. This function will print out the string, ANSI-colored and enclosed in the symbol for example:
		`#################`
		`### I am cool ###`
		`#################`

in	*sym*	The surrounding symbol
in	*bold*	Whether to use bold print
in	*color*	The ANSI color: 1 red 2 green 3 yellow 4 blue 5 magenta 6 cyan 7 white
in	*bottom*	The symbol for the bottom bar
in	*minwidth*	The minimum width for the box, if the text is very short then we insert the appropriate number of padding spaces

Returns

bar The bottom bar is returned for the user to print later, e.g. to mark off a 'section'

Definition at line 154 of file nifty.py.

Here is the call graph for this function:

7.21.2.28 def forcebalance.nifty.printcool_dictionary (*Dict*, *title* = "General options", *bold* = False, *color* = 2, *keywidth* = 25, *topwidth* = 50, *center* = True, *leftpad* = 0)

See documentation for printcool; this is a nice way to print out keys/values in a dictionary.

```
The keys in the dictionary are sorted before printing out.
```

Parameters

in	*dict*	The dictionary to be printed
in	*title*	The title of the printout

Definition at line 197 of file nifty.py.

Here is the call graph for this function:

7.21.2.29 def forcebalance.nifty.pvec1d (*vec1d*, *precision* = 1, *loglevel* = forcebalance.output.INFO)

Printout of a 1-D vector.

Parameters

in	*vec1d*	a 1-D vector

Definition at line 54 of file nifty.py.

Here is the call graph for this function:

7.21.2.30 def forcebalance.nifty.queue_up (*wq*, *command*, *input_files*, *output_files*, *tgt* = None, *verbose* = True)

Submit a job to the Work Queue.

Parameters

in	*wq*	(Work Queue Object)
in	*command*	(string) The command to run on the remote worker.
in	*input_files*	(list of files) A list of locations of the input files.
in	*output_files*	(list of files) A list of locations of the output files.

Definition at line 589 of file nifty.py.

Here is the call graph for this function:

7.21.2.31 def forcebalance.nifty.queue_up_src_dest (*wq, command, input_files, output_files, tgt =* `None`, *verbose =* `True`)

Submit a job to the Work Queue.

This function is a bit fancier in that we can explicitly specify where the input files come from, and where the output files go to.

Parameters

in	*wq*	(Work Queue Object)
in	*command*	(string) The command to run on the remote worker.
in	*input_files*	(list of 2-tuples) A list of local and remote locations of the input files.
in	*output_files*	(list of 2-tuples) A list of local and remote locations of the output files.

Definition at line 620 of file nifty.py.

7.21.2.32 def forcebalance.nifty.remove_if_exists (*fnm*)

Remove the file if it exists (doesn't return an error).

Definition at line 775 of file nifty.py.

7.21.2.33 def forcebalance.nifty.row (*vec*)

Given any list, array, or matrix, return a 1-row matrix.

Parameters

in	*vec*	The input vector that is to be made into a row

Returns

answer A row matrix

Definition at line 280 of file nifty.py.

Here is the call graph for this function:

7.21.2.34 def forcebalance.nifty.segments (*e*)

Definition at line 75 of file nifty.py.

Here is the call graph for this function:

7.21.2.35 def forcebalance.nifty.statisticalInefficiency (*A_n*, *B_n* = None, *fast* = False, *mintime* = 3, *warn* = True)

Compute the (cross) statistical inefficiency of (two) timeseries.

```
Notes
  The same timeseries can be used for both A_n and B_n to get the autocorrelation statistical inefficiency.
  The fast method described in Ref [1] is used to compute g.

References
  [1] J. D. Chodera, W. C. Swope, J. W. Pitera, C. Seok, and K. A. Dill. Use of the weighted
  histogram analysis method for the analysis of simulated and parallel tempering simulations.
  JCTC 3(1):26-41, 2007.

Examples

Compute statistical inefficiency of timeseries data with known correlation time.

>>> import timeseries
>>> A_n = timeseries.generateCorrelatedTimeseries(N=100000, tau=5.0)
>>> g = statisticalInefficiency(A_n, fast=True)
```

Parameters

in	*A_n*	(required, numpy array) - A_n[n] is nth value of timeseries A. Length is deduced from vector.
in	*B_n*	(optional, numpy array) - B_n[n] is nth value of timeseries B. Length is deduced from vector. If supplied, the cross-correlation of timeseries A and B will be estimated instead of the autocorrelation of timeseries A.

in	*fast*	(optional, boolean) - if True, will use faster (but less accurate) method to estimate correlation time, described in Ref. [1] (default: False)
in	*mintime*	(optional, int) - minimum amount of correlation function to compute (default: 3) The algorithm terminates after computing the correlation time out to mintime when the correlation function furst goes negative. Note that this time may need to be increased if there is a strong initial negative peak in the correlation function.

Returns

g The estimated statistical inefficiency (equal to 1 + 2 tau, where tau is the correlation time). We enforce g >= 1.0.

Definition at line 433 of file nifty.py.

7.21.2.36 def forcebalance.nifty.uncommadash (*s*)

Definition at line 91 of file nifty.py.

Here is the call graph for this function:

7.21.2.37 def forcebalance.nifty.warn_once (*warning*, *warnhash* = None)

Prints a warning but will only do so once in a given run.

Definition at line 887 of file nifty.py.

Here is the call graph for this function:

7.21.2.38 def forcebalance.nifty.warn_press_key (*warning*, *timeout* = 10)

Definition at line 872 of file nifty.py.

Here is the call graph for this function:

7.21.2.39 def forcebalance.nifty.which (*fnm*)

Definition at line 779 of file nifty.py.

Here is the call graph for this function:

7.21.2.40 def forcebalance.nifty.wq_wait (*wq*, *verbose* = `False`)

This function waits until the work queue is completely empty.

Definition at line 691 of file nifty.py.

Here is the call graph for this function:

7.21.2.41 def forcebalance.nifty.wq_wait1 (*wq*, *wait_time* = `10`, *verbose* = `False`)

This function waits ten seconds to see if a task in the Work Queue has finished.

Definition at line 640 of file nifty.py.

Here is the call graph for this function:

7.21.3 Variable Documentation

7.21.3.1 float forcebalance.nifty.bohrang = 0.529177249

One bohr equals this many angstroms.

Definition at line 44 of file nifty.py.

7.21.3.2 float forcebalance.nifty.eqcgmx = 2625.5002

Q-Chem to GMX unit conversion for energy.

Definition at line 40 of file nifty.py.

7.21.3.3 float forcebalance.nifty.fqcgmx = -49621.9

Q-Chem to GMX unit conversion for force.

Definition at line 42 of file nifty.py.

7.21.3.4 float forcebalance.nifty.kb = 0.0083144100163

Boltzmann constant.

Definition at line 38 of file nifty.py.

7.21.3.5 tuple forcebalance.nifty.logger = getLogger(__name__)

Definition at line 34 of file nifty.py.

7.21.3.6 tuple forcebalance.nifty.specific_dct = dict(list(itertools.chain(*[[(j,i[1]) for j in i[0]] for i in specific_lst]))

Definition at line 731 of file nifty.py.

7.21.3.7 list forcebalance.nifty.specific_lst

Initial value:

```
1 = [(['mdrun','grompp','trjconv','g_energy','g_traj'], "Make sure to install GROMACS and add it to your path
      (or set the gmxpath option)"),
2                ('force.mdin', 'stage.leap'], "This file is needed for setting up AMBER force matching
      targets"),
3                ('conf.pdb', 'mono.pdb'], "This file is needed for setting up OpenMM condensed phase
      property targets"),
4                ('liquid.xyz', 'liquid.key', 'mono.xyz', 'mono.key'], "This file is needed for setting up
      OpenMM condensed phase property targets"),
5                ('dynamic', 'analyze', 'minimize', 'testgrad', 'vibrate', 'optimize', 'polarize', '
     superpose'], "Make sure to install TINKER and add it to your path (or set the tinkerpath option)"),
6                ('runcuda.sh', 'npt.py', 'npt_tinker.py'], "This file belongs in the ForceBalance source
```

```
directory, not sure why it is missing"),
                ([’input.xyz’], "This file is needed for TINKER molecular property targets"),
                ([’.*key$’, ’.*xyz$’], "I am guessing this file is probably needed by TINKER"),
                ([’.*gro$’, ’.*top$’, ’.*itp$’, ’.*mdp$’, ’.*ndx$’], "I am guessing this file is probably
needed by GROMACS")
                 ]
```

Definition at line 719 of file nifty.py.

7.21.3.8 string forcebalance.nifty.XMLFILE = ’x’

Pickle uses ’flags’ to pickle and unpickle different variable types.

Here we use the letter ’x’ to signify that the variable type is an XML file.

Definition at line 495 of file nifty.py.

7.22 forcebalance.objective Namespace Reference

ForceBalance objective function.

Classes

- class Objective

  Objective function.
- class Penalty

  Penalty functions for regularizing the force field optimizer.

Variables

- tuple logger = getLogger(__name__)
- dictionary Implemented_Targets

  The table of implemented Targets.
- list Letters = ['X','G','H']

  This is the canonical lettering that corresponds to : objective function, gradient, Hessian.

7.22.1 Detailed Description

ForceBalance objective function.

7.22.2 Variable Documentation

7.22.2.1 dictionary forcebalance.objective.Implemented_Targets

Initial value:

```
= {
    ’ABINITIO_GMX’:AbInitio_GMX,
    ’ABINITIO_TINKER’:AbInitio_TINKER,
    ’ABINITIO_OPENMM’:AbInitio_OpenMM,
    ’ABINITIO_AMBER’:AbInitio_AMBER,
    ’ABINITIO_INTERNAL’:AbInitio_Internal,
    ’VIBRATION_TINKER’:Vibration_TINKER,
    ’LIQUID_OPENMM’:Liquid_OpenMM,
    ’LIQUID_TINKER’:Liquid_TINKER,
```

```
    'LIQUID_GMX':Liquid_GMX,
    'COUNTERPOISE':Counterpoise,
    'THCDF_PSI4':THCDF_Psi4,
    'RDVR3_PSI4':RDVR3_Psi4,
    'INTERACTION_TINKER':Interaction_TINKER,
    'INTERACTION_OPENMM':Interaction_OpenMM,
    'BINDINGENERGY_TINKER':BindingEnergy_TINKER,
    'MOMENTS_TINKER':Moments_TINKER,
    'MONOMER_QTPIE':Monomer_QTPIE,
    'REMOTE_TARGET':RemoteTarget,
    }
```

The table of implemented Targets.

Definition at line 74 of file objective.py.

7.22.2.2 list forcebalance.objective.Letters = ['X','G','H']

This is the canonical lettering that corresponds to : objective function, gradient, Hessian.

Definition at line 96 of file objective.py.

7.22.2.3 tuple forcebalance.objective.logger = getLogger(__name__)

Definition at line 17 of file objective.py.

7.23 forcebalance.openmmio Namespace Reference

OpenMM input/output.

Classes

- class OpenMM_Reader

  Class for parsing OpenMM force field files.
- class Liquid_OpenMM
- class AbInitio_OpenMM

  Subclass of AbInitio for force and energy matching using OpenMM.
- class Interaction_OpenMM

  Subclass of Target for interaction matching using OpenMM.

Functions

- def get_dipole

  Return the current dipole moment in Debye.
- def ResetVirtualSites

  Given a set of OpenMM-compatible positions and a System object, compute the correct virtual site positions according to the System.
- def CopyAmoebaBondParameters
- def CopyAmoebaOutOfPlaneBendParameters
- def CopyAmoebaAngleParameters
- def CopyAmoebaInPlaneAngleParameters
- def CopyAmoebaVdwParameters
- def CopyAmoebaMultipoleParameters
- def CopyHarmonicBondParameters

- def CopyHarmonicAngleParameters
- def CopyPeriodicTorsionParameters
- def CopyNonbondedParameters
- def do_nothing
- def CopySystemParameters

  Copy parameters from one system (i.e.
- def UpdateSimulationParameters
- def MTSVVVRIntegrator

  Create a multiple timestep velocity verlet with velocity randomization (VVVR) integrator.

Variables

- tuple logger = getLogger(__name__)
- dictionary suffix_dict
- string pdict = "XML_Override"

  pdict is a useless variable if the force field is XML.

7.23.1 Detailed Description

OpenMM input/output.

Author

Lee-Ping Wang

Date

04/2012

7.23.2 Function Documentation

7.23.2.1 def forcebalance.openmmio.CopyAmoebaAngleParameters (*src, dest*)

Definition at line 111 of file openmmio.py.

Here is the call graph for this function:

7.23.2.2 def forcebalance.openmmio.CopyAmoebaBondParameters (*src, dest*)

Definition at line 97 of file openmmio.py.

Here is the call graph for this function:

7.23.2.3 def forcebalance.openmmio.CopyAmoebaInPlaneAngleParameters (*src, dest*)

Definition at line 120 of file openmmio.py.

Here is the call graph for this function:

7.23.2.4 def forcebalance.openmmio.CopyAmoebaMultipoleParameters (*src, dest*)

Definition at line 133 of file openmmio.py.

Here is the call graph for this function:

7.23.2.5 def forcebalance.openmmio.CopyAmoebaOutOfPlaneBendParameters (*src, dest*)

Definition at line 103 of file openmmio.py.

Here is the call graph for this function:

7.23.2.6 def forcebalance.openmmio.CopyAmoebaVdwParameters (*src, dest*)

Definition at line 129 of file openmmio.py.

Here is the call graph for this function:

7.23.2.7 def forcebalance.openmmio.CopyHarmonicAngleParameters (*src, dest*)

Definition at line 141 of file openmmio.py.

Here is the call graph for this function:

7.23.2.8 def forcebalance.openmmio.CopyHarmonicBondParameters (*src, dest*)

Definition at line 137 of file openmmio.py.

Here is the call graph for this function:

7.23.2.9 def forcebalance.openmmio.CopyNonbondedParameters (*src, dest*)

Definition at line 149 of file openmmio.py.

Here is the call graph for this function:

7.23.2.10 def forcebalance.openmmio.CopyPeriodicTorsionParameters (*src, dest*)

Definition at line 145 of file openmmio.py.

Here is the call graph for this function:

7.23.2.11 def forcebalance.openmmio.CopySystemParameters (*src, dest*)

Copy parameters from one system (i.e.

that which is created by a new force field) sto another system (i.e. the one stored inside the Target object). DANGER: These need to be implemented manually!!!

Definition at line 163 of file openmmio.py.

Here is the call graph for this function:

7.23.2.12 def forcebalance.openmmio.do_nothing (*src, dest*)

Definition at line 156 of file openmmio.py.

7.23.2.13 def forcebalance.openmmio.get_dipole (*simulation, q* = None, *positions* = None)

Return the current dipole moment in Debye.

Note that this quantity is meaningless if the system carries a net charge.

Definition at line 36 of file openmmio.py.

Here is the call graph for this function:

7.23.2.14 def forcebalance.openmmio.MTSVVVRIntegrator (*temperature, collision_rate, timestep, system, ninnersteps* = 4)

Create a multiple timestep velocity verlet with velocity randomization (VVVR) integrator.

ARGUMENTS

temperature (numpy.unit.Quantity compatible with kelvin) - the temperature collision_rate (numpy.unit.Quantity compatible with 1/picoseconds) - the collision rate timestep (numpy.unit.Quantity compatible with femtoseconds) - the integration timestep system (simtk.openmm.System) - system whose forces will be partitioned ninnersteps (int) - number of inner timesteps (default: 4)

RETURNS

integrator (openmm.CustomIntegrator) - a VVVR integrator

NOTES

This integrator is equivalent to a Langevin integrator in the velocity Verlet discretization with a timestep correction to ensure that the field-free diffusion constant is timestep invariant. The inner velocity Verlet discretization is transformed into a multiple timestep algorithm.

REFERENCES

VVVR Langevin integrator:

- http://arxiv.org/abs/1301.3800
- http://arxiv.org/abs/1107.2967 (to appear in PRX 2013)

TODO

Move initialization of 'sigma' to setting the per-particle variables.

Definition at line 221 of file openmmio.py.

7.23.2.15 def forcebalance.openmmio.ResetVirtualSites (*positions, system*)

Given a set of OpenMM-compatible positions and a System object, compute the correct virtual site positions according to the System.

Definition at line 66 of file openmmio.py.

Here is the call graph for this function:

7.23.2.16 def forcebalance.openmmio.UpdateSimulationParameters (*src_system, dest_simulation*)

Definition at line 182 of file openmmio.py.

Here is the call graph for this function:

7.23.3 Variable Documentation

7.23.3.1 tuple forcebalance.openmmio.logger = getLogger(__name__)

Definition at line 24 of file openmmio.py.

7.23.3.2 string forcebalance.openmmio.pdict = "XML_Override"

pdict is a useless variable if the force field is XML.

Definition at line 300 of file openmmio.py.

7.23.3.3 dictionary forcebalance.openmmio.suffix_dict

Initial value:

```
= { "HarmonicBondForce" : {"Bond" : ["class1","class2"]},
                "HarmonicAngleForce" : {"Angle" : ["class1","class2","class3"],},
                "PeriodicTorsionForce" : {"Proper" : ["class1","class2","class3","class4"],},
                "NonbondedForce" : {"Atom": ["type"]},
                "AmoebaBondForce" : {"Bond" : ["class1","class2"]},
                "AmoebaAngleForce" : {"Angle" : ["class1","class2","class3"]},
                "AmoebaStretchBendForce" : {"StretchBend" : ["class1","class2","class3"]},
                "AmoebaVdwForce" : {"Vdw" : ["class"]},
                "AmoebaMultipoleForce" : {"Multipole" : ["type","kz","kx"], "Polarize" : ["type"]},
                 "AmoebaUreyBradleyForce" : {"UreyBradley" : ["class1","class2","class3"]},
                 "Residues.Residue" : {"VirtualSite" : ["index"]}
                 }
```

Definition at line 286 of file openmmio.py.

7.24 forcebalance.optimizer Namespace Reference

Optimization algorithms.

Classes

- class Optimizer

  Optimizer class.

Functions

- def Counter
- def GoodStep

Variables

- tuple logger = getLogger(__name__)
- int ITERATION_NUMBER = 0
- int GOODSTEP = 0

7.24.1 Detailed Description

Optimization algorithms. My current implementation is to have a single optimizer class with several methods contained inside.

Author

Lee-Ping Wang

Date

12/2011

7.24.2 Function Documentation

7.24.2.1 def forcebalance.optimizer.Counter ( )

Definition at line 29 of file optimizer.py.

Here is the call graph for this function:

7.24.2.2 def forcebalance.optimizer.GoodStep ( )

Definition at line 33 of file optimizer.py.

7.24.3 Variable Documentation

7.24.3.1 int forcebalance.optimizer.GOODSTEP = 0

Definition at line 27 of file optimizer.py.

7.24.3.2 int forcebalance.optimizer.ITERATION_NUMBER = 0

Definition at line 25 of file optimizer.py.

7.24.3.3 tuple forcebalance.optimizer.logger = getLogger(__name__)

Definition at line 22 of file optimizer.py.

7.25 forcebalance.output Namespace Reference

Classes

- class ForceBalanceLogger

  This logger starts out with a default handler that writes to stdout addHandler removes this default the first time another handler is added.
- class RawStreamHandler

  Exactly like output.StreamHandler except it does no extra formatting before sending logging messages to the stream.
- class RawFileHandler

  Exactly like output.FileHandler except it does no extra formatting before sending logging messages to the file.
- class CleanStreamHandler

  Similar to RawStreamHandler except it does not write terminal escape codes.
- class CleanFileHandler

  File handler that does not write terminal escape codes and carriage returns to files.

7.26 forcebalance.parser Namespace Reference

Input file parser for ForceBalance jobs.

Functions

- def read_mvals
- def read_pvals
- def read_priors
- def read_internals
- def printsection

  Print out a section of the input file in a parser-compliant and readable format.
- def parse_inputs

  Parse through the input file and read all user-supplied options.

Variables

- tuple logger = getLogger(__name__)
- dictionary gen_opts_types

  Default general options.
- dictionary tgt_opts_types

  Default fitting target options.
- dictionary gen_opts_defaults = {}

  Default general options - basically a collapsed veresion of gen_opts_types.
- dictionary subdict = {}
- dictionary tgt_opts_defaults = {}

  Default target options - basically a collapsed version of tgt_opts_types.
- dictionary bkwd = {"simtype" : "type"}

  Option maps for maintaining backward compatibility.
- list mainsections = ["SIMULATION","TARGET","OPTIONS","END","NONE"]

  Listing of sections in the input file.
- dictionary ParsTab

  ParsTab that refers to subsection parsers.

7.26.1 Detailed Description

Input file parser for ForceBalance jobs. Additionally, the location for all default options.

Although I will do my best to write good documentation, for many programs the input parser becomes the most up-to-date source for documentation. So this is a great place to write lots of comments for those who implement new functionality.

There are two types of sections for options - GENERAL and TARGET. Since there can be many fitting targets within a single job (i.e. we may wish to fit water trimers and hexamers, which constitutes two fitting targets) the input is organized into sections, like so:

$options

gen_option_1 Big

gen_option_2 Mao

$target

tgt_option_1 Sniffy

tgt_option_2 Schmao

$target

tgt_option_1 Nifty

tgt_option_2 Jiffy

$end

In this case, two sets of target options are generated in addition to the general option.

(Note: "Target" used to be called "Simulation". Backwards compatibility is maintained.)

Each option is meant to be parsed as a certain variable type.

- String option values are read in directly; note that only the first two words in the line are processed

- Some strings are capitalized when they are read in; this is mainly for function tables like OptTab and TgtTab
- List option types will pick up all of the words on the line and use them as values, plus if the option occurs more than once it will aggregate all of the values.
- Integer and float option types are read in a pretty straightforward way
- Boolean option types are always set to true, unless the second word is '0', 'no', or 'false' (not case sensitive)
- Section option types are meant to treat more elaborate inputs, such as the user pasting in output parameters from a previous job as input, or a specification of internal coordinate system. I imagine that for every section type I would have to write my own parser. Maybe a ParsTab of parsing functions would work. :)

To add a new option, simply add it to the dictionaries below and give it a default value if desired. If you add an entirely new type, make sure to implement the interpretation of that type in the parse_inputs function.

Author

Lee-Ping Wang

Date

11/2012

7.26.2 Function Documentation

7.26.2.1 def forcebalance.parser.parse_inputs (*input_file* = `None`)

Parse through the input file and read all user-supplied options.

```
This is usually the first thing that happens when an executable script is called.
Our parser first loads the default options, and then updates these options as it
encounters keywords.

Each keyword corresponds to a variable type; each variable type (e.g. string,
integer, float, boolean) is treated differently.  For more elaborate inputs,
there is a 'section' variable type.

There is only one set of general options, but multiple sets of target options.
Each target has its own section delimited by the \em $target keyword,
and we build a list of target options.
```

Parameters

in	*input_file*	The name of the input file.

Returns

options General options.
tgt_opts List of fitting target options.

Todo Implement internal coordinates.
Implement sampling correction.
Implement charge groups.

Definition at line 377 of file parser.py.

Here is the call graph for this function:

7.26.2.2 def forcebalance.parser.printsection (*heading, optdict, typedict*)

Print out a section of the input file in a parser-compliant and readable format.

```
At the time of writing of this function, it's mainly intended to be called by MakeInputFile.py.
The heading is printed first (it is something like $options or $target).  Then it loops
through the variable types (strings, allcaps, etc...) and the keys in each variable type.
The one-line description of each key is printed out as a comment, and then the key itself is
printed out along with the value provided in optdict.  If optdict is None, then the default
value is printed out instead.
```

Parameters

in	*heading*	Heading, either $options or $target
in	*optdict*	Options dictionary or None.
in	*typedict*	Option type dictionary, either gen_opts_types or tgt_opts_types specified in this file.

Returns

Answer List of strings for the section that we are printing out.

Definition at line 280 of file parser.py.

Here is the call graph for this function:

7.26.2.3 def forcebalance.parser.read_internals (*fobj*)

Definition at line 254 of file parser.py.

7.26.2.4 def forcebalance.parser.read_mvals (*fobj*)

Definition at line 229 of file parser.py.

Here is the call graph for this function:

7.26.2.5 def forcebalance.parser.read_priors (*fobj*)

Definition at line 245 of file parser.py.

Here is the call graph for this function:

7.26.2.6 def forcebalance.parser.read_pvals (*fobj*)

Definition at line 237 of file parser.py.

Here is the call graph for this function:

7.26.3 Variable Documentation

7.26.3.1 dictionary forcebalance.parser.bkwd = {"simtype" : "type"}

Option maps for maintaining backward compatibility.

Definition at line 224 of file parser.py.

7.26.3.2 dictionary forcebalance.parser.gen_opts_defaults = {}

Default general options - basically a collapsed veresion of gen_opts_types.

Definition at line 208 of file parser.py.

7.26.3.3 dictionary forcebalance.parser.gen_opts_types

Default general options.

Note that the documentation is included in part of the key; this will aid in automatic doc-extraction. :) In the 5-tuple we have: Default value, priority (larger number means printed first), short docstring, description of scope, list of filter strings for pulling out pertinent targets (MakeInputFile.py)

Definition at line 65 of file parser.py.

7.26.3.4 tuple forcebalance.parser.logger = getLogger(__name__)

Definition at line 60 of file parser.py.

7.26.3.5 list forcebalance.parser.mainsections = ["SIMULATION","TARGET","OPTIONS","END","NONE"]

Listing of sections in the input file.

Definition at line 227 of file parser.py.

7.26.3.6 dictionary forcebalance.parser.ParsTab

Initial value:

```
= {"read_mvals" : read_mvals,
            "read_pvals" : read_pvals,
            "priors"     : read_priors,
            "internal"   : read_internals
            }
```

ParsTab that refers to subsection parsers.

Definition at line 258 of file parser.py.

7.26.3.7 dictionary forcebalance.parser.subdict = $\{\}$

Definition at line 210 of file parser.py.

7.26.3.8 dictionary forcebalance.parser.tgt_opts_defaults = $\{\}$

Default target options - basically a collapsed version of tgt_opts_types.

Definition at line 216 of file parser.py.

7.26.3.9 dictionary forcebalance.parser.tgt_opts_types

Default fitting target options.

Definition at line 126 of file parser.py.

7.27 forcebalance.psi4io Namespace Reference

PSI4 force field input/output.

Classes

- class GBS_Reader

  Interaction type -> Parameter Dictionary.
- class THCDF_Psi4
- class Grid_Reader

  Finite state machine for parsing DVR grid files.
- class RDVR3_Psi4

  Subclass of Target for R-DVR3 grid fitting.

Variables

- tuple logger = getLogger(__name__)

7.27.1 Detailed Description

PSI4 force field input/output. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

Author

Lee-Ping Wang

Date

01/2012

7.27.2 Variable Documentation

7.27.2.1 tuple forcebalance.psi4io.logger = getLogger(__name__)

Definition at line 24 of file psi4io.py.

7.28 forcebalance.PT Namespace Reference

Variables

- dictionary PeriodicTable
- list Elements

7.28.1 Variable Documentation

7.28.1.1 list forcebalance.PT.Elements

Initial value:

```
= ["None",'H','He',
            'Li','Be','B','C','N','O','F','Ne',
            'Na','Mg','Al','Si','P','S','Cl','Ar',
            'K','Ca','Sc','Ti','V','Cr','Mn','Fe','Co','Ni','Cu','Zn','Ga','Ge','As','Se','Br','Kr',
            'Rb','Sr','Y','Zr','Nb','Mo','Tc','Ru','Rh','Pd','Ag','Cd','In','Sn','Sb','Te','I','Xe',
            'Cs','Ba','La','Ce','Pr','Nd','Pm','Sm','Eu','Gd','Tb','Dy','Ho','Er','Tm','Yb',
            'Lu','Hf','Ta','W','Re','Os','Ir','Pt','Au','Hg','Tl','Pb','Bi','Po','At','Rn',
            'Fr','Ra','Ac','Th','Pa','U','Np','Pu','Am','Cm','Bk','Cf','Es','Fm','Md','No','Lr','Rf','Db','Sg','Bh','Hs','Mt']
```

Definition at line 18 of file PT.py.

7.28.1.2 dictionary forcebalance.PT.PeriodicTable

Initial value:

```
= {'H' : 1.0079, 'He' : 4.0026,
                 'Li' : 6.941, 'Be' : 9.0122, 'B' : 10.811, 'C' : 12.0107, 'N' : 14.0067, 'O' : 15.9994, 'F' : 18.9984, 'Ne' : 20.1797,
                 'Na' : 22.9897, 'Mg' : 24.305, 'Al' : 26.9815, 'Si' : 28.0855, 'P' : 30.9738, 'S' : 32.065, 'Cl' : 35.453, 'Ar' : 39.948,
                 'K' : 39.0983, 'Ca' : 40.078, 'Sc' : 44.9559, 'Ti' : 47.867, 'V' : 50.9415, 'Cr' : 51.9961, 'Mn' : 54.938, 'Fe' : 55.845, 'Co' : 58.9332,
                 'Ni' : 58.6934, 'Cu' : 63.546, 'Zn' : 65.39, 'Ga' : 69.723, 'Ge' : 72.64, 'As' : 74.9216, 'Se' : 78.96, 'Br' : 79.904, 'Kr' : 83.8,
                 'Rb' : 85.4678, 'Sr' : 87.62, 'Y' : 88.9059, 'Zr' : 91.224, 'Nb' : 92.9064, 'Mo' : 95.94,
```

```
    'Tc' : 98, 'Ru' : 101.07, 'Rh' : 102.9055,
7                'Pd' : 106.42, 'Ag' : 107.8682, 'Cd' : 112.411, 'In' : 114.818, 'Sn' : 118.71, 'Sb' : 121.
    76, 'Te' : 127.6, 'I' : 126.9045, 'Xe' : 131.293,
8                'Cs' : 132.9055, 'Ba' : 137.327, 'La' : 138.9055, 'Ce' : 140.116, 'Pr' : 140.9077, 'Nd' :
    144.24, 'Pm' : 145, 'Sm' : 150.36,
9                'Eu' : 151.964, 'Gd' : 157.25, 'Tb' : 158.9253, 'Dy' : 162.5, 'Ho' : 164.9303, 'Er' : 167.
    259, 'Tm' : 168.9342, 'Yb' : 173.04,
10                'Lu' : 174.967, 'Hf' : 178.49, 'Ta' : 180.9479, 'W' : 183.84, 'Re' : 186.207, 'Os' : 190.2
    3, 'Ir' : 192.217, 'Pt' : 195.078,
11                'Au' : 196.9665, 'Hg' : 200.59, 'Tl' : 204.3833, 'Pb' : 207.2, 'Bi' : 208.9804, 'Po' : 209
    , 'At' : 210, 'Rn' : 222,
12                'Fr' : 223, 'Ra' : 226, 'Ac' : 227, 'Th' : 232.0381, 'Pa' : 231.0359, 'U' : 238.0289, 'Np'
     : 237, 'Pu' : 244,
13                'Am' : 243, 'Cm' : 247, 'Bk' : 247, 'Cf' : 251, 'Es' : 252, 'Fm' : 257, 'Md' : 258, 'No' :
     259,
14                'Lr' : 262, 'Rf' : 261, 'Db' : 262, 'Sg' : 266, 'Bh' : 264, 'Hs' : 277, 'Mt' : 268}
```

Definition at line 3 of file PT.py.

7.29 forcebalance.qchemio Namespace Reference

Q-Chem input file parser.

Classes

- class QCIn_Reader

  Finite state machine for parsing Q-Chem input files.

Functions

- def QChem_Dielectric_Energy

Variables

- tuple logger = getLogger(__name__)
- list ndtypes = [None]

  Types of counterpoise correction cptypes = [None, 'BASS', 'BASSP'] Types of NDDO correction.
- dictionary pdict

  Section -> Interaction type dictionary.

7.29.1 Detailed Description

Q-Chem input file parser.

7.29.2 Function Documentation

7.29.2.1 def forcebalance.qchemio.QChem_Dielectric_Energy (*fnm, wq*)

Definition at line 74 of file qchemio.py.

Here is the call graph for this function:

7.29.3 Variable Documentation

7.29.3.1 tuple forcebalance.qchemio.logger = getLogger(__name__)

Definition at line 11 of file qchemio.py.

7.29.3.2 list forcebalance.qchemio.ndtypes = [None]

Types of counterpoise correction cptypes = [None, 'BASS', 'BASSP'] Types of NDDO correction.

Definition at line 16 of file qchemio.py.

7.29.3.3 dictionary forcebalance.qchemio.pdict

Initial value:

```
1 = {'BASS':{0:'A', 1:'C'},
2          'BASSP' :{0:'A', 1:'B', 2:'C'}
3          }
```

Section -> Interaction type dictionary.

fdict = { 'basis' : bastypes } Interaction type -> Parameter Dictionary.

Definition at line 23 of file qchemio.py.

7.30 forcebalance.target Namespace Reference

Classes

- class Target

  Base class for all fitting targets.
- class RemoteTarget

Variables

- tuple logger = getLogger(__name__)

7.30.1 Variable Documentation

7.30.1.1 tuple forcebalance.target.logger = getLogger(__name__)

Definition at line 17 of file target.py.

7.31 forcebalance.tinkerio Namespace Reference

TINKER input/output.

Classes

- class Tinker_Reader

  Finite state machine for parsing TINKER force field files.
- class Liquid_TINKER
- class AbInitio_TINKER

  Subclass of Target for force and energy matching using TINKER.
- class Vibration_TINKER

  Subclass of Target for vibrational frequency matching using TINKER.
- class Moments_TINKER

  Subclass of Target for multipole moment matching using TINKER.
- class BindingEnergy_TINKER

  Subclass of BindingEnergy for binding energy matching using TINKER.
- class Interaction_TINKER

  Subclass of Target for interaction matching using TINKER.

Functions

- def write_key_with_prm

  Copies a TINKER .key file but changes the parameter keyword as necessary to reflect the ForceBalance settings.
- def modify_key

  Performs in-place modification of a TINKER .key file.

Variables

- tuple logger = getLogger(__name__)
- dictionary pdict

7.31.1 Detailed Description

TINKER input/output. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

Author

Lee-Ping Wang

Date

01/2012

7.31.2 Function Documentation

7.31.2.1 def forcebalance.tinkerio.modify_key (*src*, *in_dict*)

Performs in-place modification of a TINKER .key file.

```
The input dictionary contains key:value pairs such as
"polarization direct".  If the key exists in the TINKER file, then
that line is modified such that it contains the value in the
dictionary.  Note that this "key" is not to be confused with the
.key extension in the TINKER file that we're modifying.

Sometimes keys like 'archive' do not have a value, in which case
the dictionary should contain a None value or a blank space.

If the key doesn't exist in the TINKER file, then the key:value pair
will be printed at the end.
```

Parameters

in	*src*	Name of the TINKER file to be modified.
in	*in_dict*	Dictionary containing key-value pairs used to modify the TINKER file.

Definition at line 202 of file tinkerio.py.

7.31.2.2 def forcebalance.tinkerio.write_key_with_prm (*src*, *dest*, *prmfnm* = None, *ffobj* = None)

Copies a TINKER .key file but changes the parameter keyword as necessary to reflect the ForceBalance settings.

Definition at line 146 of file tinkerio.py.

Here is the call graph for this function:

7.31.3 Variable Documentation

7.31.3.1 tuple forcebalance.tinkerio.logger = getLogger(__name__)

Definition at line 32 of file tinkerio.py.

7.31.3.2 dictionary forcebalance.tinkerio.pdict

Initial value:

```
1 = {'VDW'          : {'Atom':[1], 2:'S',3:'T',4:'D'}, # Van der Waals distance, well depth, distance from
      bonded neighbor?
2         'BOND'         : {'Atom':[1,2], 3:'K',4:'B'},     # Bond force constant and equilibrium distance
      (Angstrom)
3         'ANGLE'        : {'Atom':[1,2,3], 4:'K',5:'B'},   # Angle force constant and equilibrium angle
4         'UREYBRAD'     : {'Atom':[1,2,3], 4:'K',5:'B'},   # Urey-Bradley force constant and equilibrium
```

```
distance (Angstrom)
    'MCHARGE'       : {'Atom':[1,2,3], 4:''},          # Atomic charge
    'DIPOLE'       : {0:'X',1:'Y',2:'Z'},             # Dipole moment in local frame
    'QUADX'        : {0:'X'},                         # Quadrupole moment, X component
    'QUADY'        : {0:'X',1:'Y'},                   # Quadrupole moment, Y component
    'QUADZ'        : {0:'X',1:'Y',2:'Z'},             # Quadrupole moment, Y component
     'POLARIZE'     : {'Atom':[1], 2:'A',3:'T'},       # Atomic dipole polarizability
     'BOND-CUBIC'   : {'Atom':[], 0:''},    # Below are global parameters.
     'BOND-QUARTIC' : {'Atom':[], 0:''},
     'ANGLE-CUBIC'  : {'Atom':[], 0:''},
     'ANGLE-QUARTIC': {'Atom':[], 0:''},
     'ANGLE-PENTIC' : {'Atom':[], 0:''},
     'ANGLE-SEXTIC' : {'Atom':[], 0:''},
     'DIELECTRIC'   : {'Atom':[], 0:''},
     'POLAR-SOR'    : {'Atom':[], 0:''}
                                            # Ignored for now: stretch/bend coupling, out-of-plane
bending,
                                            # torsional parameters, pi-torsion, torsion-torsion
     }
```

Definition at line 34 of file tinkerio.py.

7.32 forcebalance.vibration Namespace Reference

Vibrational mode fitting module.

Classes

- class Vibration

  Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).

Variables

- tuple logger = getLogger(__name__)

7.32.1 Detailed Description

Vibrational mode fitting module.

Author

Lee-Ping Wang

Date

08/2012

7.32.2 Variable Documentation

7.32.2.1 tuple forcebalance.vibration.logger = getLogger(__name__)

Definition at line 22 of file vibration.py.

8 Class Documentation

8.1 forcebalance.abinitio.AbInitio Class Reference

Subclass of Target for fitting force fields to ab initio data.

Inheritance diagram for forcebalance.abinitio.AbInitio:

Collaboration diagram for forcebalance.abinitio.AbInitio:

Public Member Functions

- def __init__

  Initialization; define a few core concepts.

- def read_topology
- def build_invdist
- def compute_netforce_torque
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def prepare_temp_directory

  Prepare the temporary directory, by default does nothing.
- def indicate
- def energy_force_transformer_all
- def energy_force_transformer
- def get_energy_force_

  LPW 06-30-2013.
- def get_resp_

  Electrostatic potential fitting.
- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- whamboltz_wts

  Initialize the base class.
- qmboltz_wts

  QM Boltzmann weights.
- eqm

  Reference (QM) energies.
- emd0

  Energies of the sampling simulation.
- fqm

  Reference (QM) forces.

- espxyz

  ESP grid points.
- espval

  ESP values.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- qmatoms

  The number of atoms in the QM calculation (Irrelevant if not fitting forces)
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)
- e_err_pct
- f_err

  Qualitative Indicator: average force error (fractional)
- f_err_pct
- esp_err

  Qualitative Indicator: "relative RMS" for electrostatic potential.
- nf_err
- nf_err_pct
- tq_err_pct
- use_nft

  Whether to compute net forces and torques, or not.
- ns

  Read in the trajectory file.
- traj
- nparticles

  The number of (atoms + drude particles + virtual sites)
- AtomLists

  This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.
- new_vsites

  Read in the topology.
- save_vmvals

  Save the mvals from the last time we updated the vsites.
- topology_flag
- force_map
- nnf
- ntq
- force
- w_force
- nesp
- fitatoms
- whamboltz
- nftqm
- fref
- w_energy
- w_netforce
- w_torque
- e_ref
- f_ref

- nf_ref
- tq_ref
- tq_err
- w_resp
- invdists
- respterm
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.1.1 Detailed Description

Subclass of Target for fitting force fields to ab initio data.

Currently Gromacs-X2, Gromacs, Tinker, OpenMM and AMBER are supported.

We introduce the following concepts:

- The number of snapshots
- The reference energies and forces (eqm, fqm) and the file they belong in (qdata.txt)
- The sampling simulation energies (emd0)
- The WHAM Boltzmann weights (these are computed externally and passed in)
- The QM Boltzmann weights (computed internally using the difference between eqm and emd0)

There are also these little details:

- Switches for whether to turn on certain Boltzmann weights (they stack)
- Temperature for the QM Boltzmann weights
- Whether to fit a subset of atoms

This subclass contains the 'get' method for building the objective function from any simulation software (a driver to run the program and read output is still required). The 'get' method can be overridden by subclasses like AbInitio_GMX.

Definition at line 47 of file abinitio.py.

8.1.2 Constructor & Destructor Documentation

8.1.2.1 def forcebalance.abinitio.AbInitio.__init__ (*self, options, tgt_opts, forcefield*)

Initialization; define a few core concepts.

Todo Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Definition at line 57 of file abinitio.py.

Here is the call graph for this function:

8.1.3 Member Function Documentation

8.1.3.1 def forcebalance.abinitio.AbInitio.build_invdist (*self, mvals*)

Definition at line 168 of file abinitio.py.

Here is the call graph for this function:

8.1.3.2 def forcebalance.abinitio.AbInitio.compute_netforce_torque (*self, xyz, force, QM* = `False`)

Definition at line 204 of file abinitio.py.

8.1.3.3 def forcebalance.abinitio.AbInitio.energy_force_transformer (*self, i*)

Definition at line 458 of file abinitio.py.

Here is the call graph for this function:

8.1.3.4 def forcebalance.abinitio.AbInitio.energy_force_transformer_all (*self*)

Definition at line 442 of file abinitio.py.

Here is the call graph for this function:

8.1.3.5 def forcebalance.abinitio.AbInitio.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Definition at line 1098 of file abinitio.py.

Here is the call graph for this function:

8.1.3.6 def forcebalance.abinitio.AbInitio.get_energy_force_ (*self, mvals, AGrad* = `False`, *AHess* = `False`)

LPW 06-30-2013.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general simulation software.  This is
in contrast to using GROMACS-X2, which computes the objective
function and prints it out; then 'get' only needs to call
GROMACS and read it in.

This subroutine interfaces with simulation software 'drivers'.
The driver is only expected to give the energy and forces.

Now this subroutine may sound trivial since the objective
function is simply a least-squares quantity (M-Q)^2 - but
there are a number of nontrivial considerations.  I will list
them here.

0) Polytensor formulation: Because there may exist covariance
between different components of the force (or covariance
between the energy and the force), we build the objective
function by taking outer products of vectors that have the
form [E F_1x F_1y F_1z F_2x F_2y ... ], and then we trace it
with the inverse of the covariance matrix to get the objective
function.

This version implements both the polytensor formulation and the standard formulation.

1) Boltzmann weights and normalization: Each snapshot has its
own Boltzmann weight, which may or may not be normalized.
This subroutine does the normalization automatically.

2) Subtracting out the mean energy gap: The zero-point energy
difference between reference data and simulation is
meaningless.  This subroutine subtracts it out.

3) Hybrid ensembles: This program builds a combined objective
function from both MM and QM ensembles, which is rigorously
better than using a single ensemble.

Note that this subroutine does not do EVERYTHING that
GROMACS-X2 can do, which includes:

1) Internal coordinate systems
2) 'Sampling correction' (deprecated, since it doesn't seem to work)
3) Analytic derivatives

In the previous code (ForTune) this subroutine used analytic
first derivatives of the energy and force to build the
derivatives of the objective function.  Here I will take a
simplified approach, because building the derivatives are
cumbersome.  For now we will return the objective function
ONLY.  A two-point central difference should give us the first
and diagonal second derivative anyhow.
```

Todo Parallelization over snapshots is not implemented yet

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 532 of file abinitio.py.

8.1.3.7 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.1.3.8 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.1.3.9 def forcebalance.abinitio.AbInitio.get_resp_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`)

Electrostatic potential fitting.

Implements the RESP objective function. (In Python so obviously not optimized.) This function takes the mathematical parameter values and returns the charges on the ATOMS (fancy mapping going on)

Definition at line 999 of file abinitio.py.

8.1.3.10 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.1.3.11 def forcebalance.abinitio.AbInitio.indicate (*self*)

Definition at line 422 of file abinitio.py.

Here is the call graph for this function:

8.1.3.12 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.1.3.13 def forcebalance.abinitio.AbInitio.prepare_temp_directory (*self, options, tgt_opts*)

Prepare the temporary directory, by default does nothing.

Definition at line 419 of file abinitio.py.

8.1.3.14 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.1.3.15 def forcebalance.abinitio.AbInitio.read_reference_data (*self*)

Read the reference ab initio data from a file such as qdata.txt.

Todo Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Todo Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

```
After reading in the information from qdata.txt, it is converted
into the GROMACS energy units (kind of an arbitrary choice);
forces (kind of a misnomer in qdata.txt) are multipled by -1
to convert gradients to forces.

We also subtract out the mean energies of all energy arrays
because energy/force matching does not account for zero-point
energy differences between MM and QM (i.e. energy of electrons
in core orbitals).

The configurations in force/energy matching typically come
from a the thermodynamic ensemble of the MM force field at
some temperature (by running MD, for example), and for many
reasons it is helpful to introduce non-Boltzmann weights in
front of these configurations.  There are two options: WHAM
Boltzmann weights (for combining the weights of several
simulations together) and QM Boltzmann weights (for converting
MM weights into QM weights).  Note that the two sets of weights
'stack'; i.e. they can be used at the same time.
```

```
A 'hybrid' ensemble is possible where we use 50% MM and 50% QM
weights.  Please read more in LPW and Troy Van Voorhis, JCP
Vol. 133, Pg. 231101 (2010), doi:10.1063/1.3519043.
```

Todo The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

```
Finally, note that using non-Boltzmann weights degrades the
statistical information content of the snapshots.  This
problem will generally become worse if the ensemble to which
we're reweighting is dramatically different from the one we're
sampling from.  We end up with a set of Boltzmann weights like
[1e-9, 1e-9, 1.0, 1e-9, 1e-9 ... ] and this is essentially just
one snapshot.  I believe Troy is working on something to cure
this problem.

Here, we have a measure for the information content of our snapshots,
which comes easily from the definition of information entropy:

S = -1*Sum_i(P_i*log(P_i))
InfoContent = exp(-S)

With uniform weights, InfoContent is equal to the number of snapshots;
with horrible weights, InfoContent is closer to one.
```

Definition at line 317 of file abinitio.py.

8.1.3.16 def forcebalance.abinitio.AbInitio.read_topology (*self*)

Definition at line 164 of file abinitio.py.

Here is the call graph for this function:

8.1.3.17 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.1.3.18 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.1.3.19 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.1.3.20 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.1.3.21 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.1.3.22 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.1.4 Member Data Documentation

8.1.4.1 forcebalance.abinitio.AbInitio.AtomLists

This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.

Definition at line 150 of file abinitio.py.

8.1.4.2 forcebalance.abinitio.AbInitio.e_err

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 128 of file abinitio.py.

8.1.4.3 forcebalance.abinitio.AbInitio.e_err_pct

Definition at line 129 of file abinitio.py.

8.1.4.4 forcebalance.abinitio.AbInitio.e_ref

Definition at line 976 of file abinitio.py.

8.1.4.5 forcebalance.abinitio.AbInitio.emd0

Energies of the sampling simulation.

Definition at line 116 of file abinitio.py.

8.1.4.6 forcebalance.abinitio.AbInitio.eqm

Reference (QM) energies.

Definition at line 114 of file abinitio.py.

8.1.4.7 forcebalance.abinitio.AbInitio.esp_err

Qualitative Indicator: "relative RMS" for electrostatic potential.

Definition at line 134 of file abinitio.py.

8.1.4.8 forcebalance.abinitio.AbInitio.espval

ESP values.

Definition at line 122 of file abinitio.py.

8.1.4.9 forcebalance.abinitio.AbInitio.espxyz

ESP grid points.

Definition at line 120 of file abinitio.py.

8.1.4.10 forcebalance.abinitio.AbInitio.f_err

Qualitative Indicator: average force error (fractional)

Definition at line 131 of file abinitio.py.

8.1.4.11 forcebalance.abinitio.AbInitio.f_err_pct

Definition at line 132 of file abinitio.py.

8.1.4.12 forcebalance.abinitio.AbInitio.f_ref

Definition at line 980 of file abinitio.py.

8.1.4.13 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.1.4.14 forcebalance.abinitio.AbInitio.fitatoms

Definition at line 354 of file abinitio.py.

8.1.4.15 forcebalance.abinitio.AbInitio.force

Definition at line 349 of file abinitio.py.

8.1.4.16 forcebalance.abinitio.AbInitio.force_map

Definition at line 212 of file abinitio.py.

8.1.4.17 forcebalance.abinitio.AbInitio.fqm

Reference (QM) forces.

Definition at line 118 of file abinitio.py.

8.1.4.18 forcebalance.abinitio.AbInitio.fref

Definition at line 413 of file abinitio.py.

8.1.4.19 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.1.4.20 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.1.4.21 forcebalance.abinitio.AbInitio.invdists

Definition at line 1007 of file abinitio.py.

8.1.4.22 forcebalance.abinitio.AbInitio.nesp

Definition at line 351 of file abinitio.py.

8.1.4.23 forcebalance.abinitio.AbInitio.new_vsites

Read in the topology.

Read in the reference data Prepare the temporary directory The below two options are related to whether we want to rebuild virtual site positions. Rebuild the distance matrix if virtual site positions have changed

Definition at line 159 of file abinitio.py.

8.1.4.24 forcebalance.abinitio.AbInitio.nf_err

Definition at line 135 of file abinitio.py.

8.1.4.25 forcebalance.abinitio.AbInitio.nf_err_pct

Definition at line 136 of file abinitio.py.

8.1.4.26 forcebalance.abinitio.AbInitio.nf_ref

Definition at line 984 of file abinitio.py.

8.1.4.27 forcebalance.abinitio.AbInitio.nftqm

Definition at line 409 of file abinitio.py.

8.1.4.28 forcebalance.abinitio.AbInitio.nnf

Definition at line 255 of file abinitio.py.

8.1.4.29 forcebalance.abinitio.AbInitio.nparticles

The number of (atoms + drude particles + virtual sites)

Definition at line 147 of file abinitio.py.

8.1.4.30 forcebalance.abinitio.AbInitio.ns

Read in the trajectory file.

Definition at line 141 of file abinitio.py.

8.1.4.31 forcebalance.abinitio.AbInitio.ntq

Definition at line 256 of file abinitio.py.

8.1.4.32 forcebalance.abinitio.AbInitio.objective

Definition at line 1118 of file abinitio.py.

8.1.4.33 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.1.4.34 forcebalance.abinitio.AbInitio.qfnm

The qdata.txt file that contains the QM energies and forces.

Definition at line 124 of file abinitio.py.

8.1.4.35 forcebalance.abinitio.AbInitio.qmatoms

The number of atoms in the QM calculation (Irrelevant if not fitting forces)

Definition at line 126 of file abinitio.py.

8.1.4.36 forcebalance.abinitio.AbInitio.qmboltz_wts

QM Boltzmann weights.

Definition at line 112 of file abinitio.py.

8.1.4.37 forcebalance.abinitio.AbInitio.respterm

Definition at line 1086 of file abinitio.py.

8.1.4.38 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.1.4.39 forcebalance.abinitio.AbInitio.save_vmvals

Save the mvals from the last time we updated the vsites.

Definition at line 161 of file abinitio.py.

8.1.4.40 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.1.4.41 forcebalance.abinitio.AbInitio.topology_flag

Definition at line 166 of file abinitio.py.

8.1.4.42 forcebalance.abinitio.AbInitio.tq_err

Definition at line 988 of file abinitio.py.

8.1.4.43 forcebalance.abinitio.AbInitio.tq_err_pct

Definition at line 137 of file abinitio.py.

8.1.4.44 forcebalance.abinitio.AbInitio.tq_ref

Definition at line 987 of file abinitio.py.

8.1.4.45 forcebalance.abinitio.AbInitio.traj

Definition at line 142 of file abinitio.py.

8.1.4.46 forcebalance.abinitio.AbInitio.use_nft

Whether to compute net forces and torques, or not.

Definition at line 139 of file abinitio.py.

8.1.4.47 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.1.4.48 forcebalance.abinitio.AbInitio.w_energy

Definition at line 571 of file abinitio.py.

8.1.4.49 forcebalance.abinitio.AbInitio.w_force

Definition at line 350 of file abinitio.py.

8.1.4.50 forcebalance.abinitio.AbInitio.w_netforce

Definition at line 571 of file abinitio.py.

8.1.4.51 forcebalance.abinitio.AbInitio.w_resp

Definition at line 1000 of file abinitio.py.

8.1.4.52 forcebalance.abinitio.AbInitio.w_torque

Definition at line 571 of file abinitio.py.

8.1.4.53 forcebalance.abinitio.AbInitio.whamboltz

Definition at line 370 of file abinitio.py.

8.1.4.54 forcebalance.abinitio.AbInitio.whamboltz_wts

Initialize the base class.

Number of snapshots Whether to use WHAM Boltzmann weights Whether to use the Sampling Correction Whether to match Absolute Energies (make sure you know what you're doing) Whether to use the Covariance Matrix Whether to use QM Boltzmann weights The temperature for QM Boltzmann weights Number of atoms that we are fitting Whether to fit Energies. Whether to fit Forces. Whether to fit Electrostatic Potential. Weights for the three components. Option for how much data to write to disk. Whether to do energy and force calculations for the whole trajectory, or to do one calculation per snapshot. OpenMM-only option - whether to run the energies and forces internally. Whether we have virtual sites (set at the global option level) WHAM Boltzmann weights

Definition at line 110 of file abinitio.py.

8.1.4.55 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- abinitio.py

8.2 forcebalance.amberio.AbInitio_AMBER Class Reference

Subclass of Target for force and energy matching using AMBER.

Inheritance diagram for forcebalance.amberio.AbInitio_AMBER:

Collaboration diagram for forcebalance.amberio.AbInitio_AMBER:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def energy_force_driver_all_external_
- def energy_force_driver_all
- def read_topology
- def build_invdist
- def compute_netforce_torque
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def indicate
- def energy_force_transformer_all
- def energy_force_transformer
- def get_energy_force_

  LPW 06-30-2013.
- def get_resp_

  Electrostatic potential fitting.

- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory, we need this BEFORE initializing the SuperClass.
- all_at_once

  all_at_once is not implemented.
- whamboltz_wts

  Initialize the base class.
- qmboltz_wts

  QM Boltzmann weights.
- eqm

  Reference (QM) energies.
- emd0

  Energies of the sampling simulation.
- fqm

  Reference (QM) forces.
- espxyz

  ESP grid points.
- espval

  ESP values.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- qmatoms

  The number of atoms in the QM calculation (Irrelevant if not fitting forces)
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)

- e_err_pct
- f_err

  Qualitative Indicator: average force error (fractional)
- f_err_pct
- esp_err

  Qualitative Indicator: "relative RMS" for electrostatic potential.
- nf_err
- nf_err_pct
- tq_err_pct
- use_nft

  Whether to compute net forces and torques, or not.
- ns

  Read in the trajectory file.
- traj
- nparticles

  The number of (atoms + drude particles + virtual sites)
- AtomLists

  This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.
- new_vsites

  Read in the topology.
- save_vmvals

  Save the mvals from the last time we updated the vsites.
- topology_flag
- force_map
- nnf
- ntq
- force
- w_force
- nesp
- fitatoms
- whamboltz
- nftqm
- fref
- w_energy
- w_netforce
- w_torque
- e_ref
- f_ref
- nf_ref
- tq_ref
- tq_err
- w_resp
- invdists
- respterm
- objective
- tempdir

  Root directory of the whole project.
- rundir

The directory in which the simulation is running - this can be updated.

- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.2.1 Detailed Description

Subclass of Target for force and energy matching using AMBER.

Implements the prepare and energy_force_driver methods. The get method is in the base class.

Definition at line 171 of file amberio.py.

8.2.2 Constructor & Destructor Documentation

8.2.2.1 def forcebalance.amberio.AbInitio_AMBER.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 174 of file amberio.py.

8.2.3 Member Function Documentation

8.2.3.1 def forcebalance.abinitio.AbInitio.build_invdist (*self, mvals*) [inherited]

Definition at line 168 of file abinitio.py.

Here is the call graph for this function:

8.2.3.2 def forcebalance.abinitio.AbInitio.compute_netforce_torque (*self, xyz, force, QM* = `False`) [inherited]

Definition at line 204 of file abinitio.py.

8.2.3.3 def forcebalance.amberio.AbInitio_AMBER.energy_force_driver_all (*self*)

Definition at line 228 of file amberio.py.

8.2.3.4 def forcebalance.amberio.AbInitio_AMBER.energy_force_driver_all_external_ (*self*)

Definition at line 190 of file amberio.py.

Here is the call graph for this function:

8.2.3.5 def forcebalance.abinitio.AbInitio.energy_force_transformer (*self, i*) [inherited]

Definition at line 458 of file abinitio.py.

Here is the call graph for this function:

8.2.3.6 def forcebalance.abinitio.AbInitio.energy_force_transformer_all (*self*) [inherited]

Definition at line 442 of file abinitio.py.

Here is the call graph for this function:

8.2.3.7 def forcebalance.abinitio.AbInitio.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Definition at line 1098 of file abinitio.py.

Here is the call graph for this function:

8.2.3.8 def forcebalance.abinitio.AbInitio.get_energy_force_ (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

LPW 06-30-2013.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general simulation software.  This is
in contrast to using GROMACS-X2, which computes the objective
function and prints it out; then 'get' only needs to call
GROMACS and read it in.

This subroutine interfaces with simulation software 'drivers'.
The driver is only expected to give the energy and forces.

Now this subroutine may sound trivial since the objective
function is simply a least-squares quantity (M-Q)^2 - but
there are a number of nontrivial considerations.  I will list
them here.

0) Polytensor formulation: Because there may exist covariance
between different components of the force (or covariance
between the energy and the force), we build the objective
function by taking outer products of vectors that have the
form [E F_1x F_1y F_1z F_2x F_2y ... ], and then we trace it
```

```
with the inverse of the covariance matrix to get the objective
function.

This version implements both the polytensor formulation and the standard formulation.

1) Boltzmann weights and normalization: Each snapshot has its
own Boltzmann weight, which may or may not be normalized.
This subroutine does the normalization automatically.

2) Subtracting out the mean energy gap: The zero-point energy
difference between reference data and simulation is
meaningless.  This subroutine subtracts it out.

3) Hybrid ensembles: This program builds a combined objective
function from both MM and QM ensembles, which is rigorously
better than using a single ensemble.

Note that this subroutine does not do EVERYTHING that
GROMACS-X2 can do, which includes:

1) Internal coordinate systems
2) 'Sampling correction' (deprecated, since it doesn't seem to work)
3) Analytic derivatives

In the previous code (ForTune) this subroutine used analytic
first derivatives of the energy and force to build the
derivatives of the objective function.  Here I will take a
simplified approach, because building the derivatives are
cumbersome.  For now we will return the objective function
ONLY.  A two-point central difference should give us the first
and diagonal second derivative anyhow.
```

Todo Parallelization over snapshots is not implemented yet

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 532 of file abinitio.py.

8.2.3.9 def forcebalance.target.Target.get_G (*self*, *mvals* = None) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.2.3.10 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.2.3.11 def forcebalance.abinitio.AbInitio.get_resp_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) [inherited]

Electrostatic potential fitting.

Implements the RESP objective function. (In Python so obviously not optimized.) This function takes the mathematical parameter values and returns the charges on the ATOMS (fancy mapping going on)

Definition at line 999 of file abinitio.py.

8.2.3.12 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.2.3.13 def forcebalance.abinitio.AbInitio.indicate (*self*) [inherited]

Definition at line 422 of file abinitio.py.

Here is the call graph for this function:

8.2.3.14 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.2.3.15 def forcebalance.amberio.AbInitio_AMBER.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 181 of file amberio.py.

Here is the call graph for this function:

8.2.3.16 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.2.3.17 def forcebalance.abinitio.AbInitio.read_reference_data (*self*) `[inherited]`

Read the reference ab initio data from a file such as qdata.txt.

Todo Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Todo Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

```
After reading in the information from qdata.txt, it is converted
into the GROMACS energy units (kind of an arbitrary choice);
forces (kind of a misnomer in qdata.txt) are multipled by -1
```

```
to convert gradients to forces.

We also subtract out the mean energies of all energy arrays
because energy/force matching does not account for zero-point
energy differences between MM and QM (i.e. energy of electrons
in core orbitals).

The configurations in force/energy matching typically come
from a the thermodynamic ensemble of the MM force field at
some temperature (by running MD, for example), and for many
reasons it is helpful to introduce non-Boltzmann weights in
front of these configurations.  There are two options: WHAM
Boltzmann weights (for combining the weights of several
simulations together) and QM Boltzmann weights (for converting
MM weights into QM weights).  Note that the two sets of weights
'stack'; i.e. they can be used at the same time.

A 'hybrid' ensemble is possible where we use 50% MM and 50% QM
weights.  Please read more in LPW and Troy Van Voorhis, JCP
Vol. 133, Pg. 231101 (2010), doi:10.1063/1.3519043.
```

Todo The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

```
Finally, note that using non-Boltzmann weights degrades the
statistical information content of the snapshots.  This
problem will generally become worse if the ensemble to which
we're reweighting is dramatically different from the one we're
sampling from.  We end up with a set of Boltzmann weights like
[1e-9, 1e-9, 1.0, 1e-9, 1e-9 ... ] and this is essentially just
one snapshot.  I believe Troy is working on something to cure
this problem.

Here, we have a measure for the information content of our snapshots,
which comes easily from the definition of information entropy:

S = -1*Sum_i(P_i*log(P_i))
InfoContent = exp(-S)

With uniform weights, InfoContent is equal to the number of snapshots;
with horrible weights, InfoContent is closer to one.
```

Definition at line 317 of file abinitio.py.

8.2.3.18 def forcebalance.abinitio.AbInitio.read_topology (*self*) [inherited]

Definition at line 164 of file abinitio.py.

Here is the call graph for this function:

8.2.3.19 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.2.3.20 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.2.3.21 def forcebalance.target.Target.sget (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.2.3.22 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.2.3.23 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.2.3.24 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.2.4 Member Data Documentation

8.2.4.1 forcebalance.amberio.AbInitio_AMBER.all_at_once

all_at_once is not implemented.

Definition at line 179 of file amberio.py.

8.2.4.2 forcebalance.abinitio.AbInitio.AtomLists [inherited]

This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.

Definition at line 150 of file abinitio.py.

8.2.4.3 forcebalance.abinitio.AbInitio.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 128 of file abinitio.py.

8.2.4.4 forcebalance.abinitio.AbInitio.e_err_pct [inherited]

Definition at line 129 of file abinitio.py.

8.2.4.5 forcebalance.abinitio.AbInitio.e_ref [inherited]

Definition at line 976 of file abinitio.py.

8.2.4.6 forcebalance.abinitio.AbInitio.emd0 [inherited]

Energies of the sampling simulation.

Definition at line 116 of file abinitio.py.

8.2.4.7 forcebalance.abinitio.AbInitio.eqm [inherited]

Reference (QM) energies.

Definition at line 114 of file abinitio.py.

8.2.4.8 forcebalance.abinitio.AbInitio.esp_err [inherited]

Qualitative Indicator: "relative RMS" for electrostatic potential.

Definition at line 134 of file abinitio.py.

8.2.4.9 forcebalance.abinitio.AbInitio.espval [inherited]

ESP values.

Definition at line 122 of file abinitio.py.

8.2.4.10 forcebalance.abinitio.AbInitio.espxyz [inherited]

ESP grid points.

Definition at line 120 of file abinitio.py.

8.2.4.11 forcebalance.abinitio.AbInitio.f_err [inherited]

Qualitative Indicator: average force error (fractional)

Definition at line 131 of file abinitio.py.

8.2.4.12 forcebalance.abinitio.AbInitio.f_err_pct [inherited]

Definition at line 132 of file abinitio.py.

8.2.4.13 forcebalance.abinitio.AbInitio.f_ref [inherited]

Definition at line 980 of file abinitio.py.

8.2.4.14 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.2.4.15 forcebalance.abinitio.AbInitio.fitatoms [inherited]

Definition at line 354 of file abinitio.py.

8.2.4.16 forcebalance.abinitio.AbInitio.force [inherited]

Definition at line 349 of file abinitio.py.

8.2.4.17 forcebalance.abinitio.AbInitio.force_map [inherited]

Definition at line 212 of file abinitio.py.

8.2.4.18 forcebalance.abinitio.AbInitio.fqm [inherited]

Reference (QM) forces.

Definition at line 118 of file abinitio.py.

8.2.4.19 forcebalance.abinitio.AbInitio.fref [inherited]

Definition at line 413 of file abinitio.py.

8.2.4.20 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.2.4.21 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.2.4.22 forcebalance.abinitio.AbInitio.invdists [inherited]

Definition at line 1007 of file abinitio.py.

8.2.4.23 forcebalance.abinitio.AbInitio.nesp [inherited]

Definition at line 351 of file abinitio.py.

8.2.4.24 forcebalance.abinitio.AbInitio.new_vsites [inherited]

Read in the topology.

Read in the reference data Prepare the temporary directory The below two options are related to whether we want to rebuild virtual site positions. Rebuild the distance matrix if virtual site positions have changed

Definition at line 159 of file abinitio.py.

8.2.4.25 forcebalance.abinitio.AbInitio.nf_err [inherited]

Definition at line 135 of file abinitio.py.

8.2.4.26 forcebalance.abinitio.AbInitio.nf_err_pct [inherited]

Definition at line 136 of file abinitio.py.

8.2.4.27 forcebalance.abinitio.AbInitio.nf_ref [inherited]

Definition at line 984 of file abinitio.py.

8.2.4.28 forcebalance.abinitio.AbInitio.nftqm [inherited]

Definition at line 409 of file abinitio.py.

8.2.4.29 forcebalance.abinitio.AbInitio.nnf [inherited]

Definition at line 255 of file abinitio.py.

8.2.4.30 forcebalance.abinitio.AbInitio.nparticles [inherited]

The number of (atoms + drude particles + virtual sites)

Definition at line 147 of file abinitio.py.

8.2.4.31 forcebalance.abinitio.AbInitio.ns [inherited]

Read in the trajectory file.

Definition at line 141 of file abinitio.py.

8.2.4.32 forcebalance.abinitio.AbInitio.ntq [inherited]

Definition at line 256 of file abinitio.py.

8.2.4.33 forcebalance.abinitio.AbInitio.objective [inherited]

Definition at line 1118 of file abinitio.py.

8.2.4.34 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.2.4.35 forcebalance.abinitio.AbInitio.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 124 of file abinitio.py.

8.2.4.36 forcebalance.abinitio.AbInitio.qmatoms [inherited]

The number of atoms in the QM calculation (Irrelevant if not fitting forces)

Definition at line 126 of file abinitio.py.

8.2.4.37 forcebalance.abinitio.AbInitio.qmboltz_wts [inherited]

QM Boltzmann weights.

Definition at line 112 of file abinitio.py.

8.2.4.38 forcebalance.abinitio.AbInitio.respterm [inherited]

Definition at line 1086 of file abinitio.py.

8.2.4.39 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.2.4.40 forcebalance.abinitio.AbInitio.save_vmvals [inherited]

Save the mvals from the last time we updated the vsites.

Definition at line 161 of file abinitio.py.

8.2.4.41 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.2.4.42 forcebalance.abinitio.AbInitio.topology_flag [inherited]

Definition at line 166 of file abinitio.py.

8.2.4.43 forcebalance.abinitio.AbInitio.tq_err [inherited]

Definition at line 988 of file abinitio.py.

8.2.4.44 forcebalance.abinitio.AbInitio.tq_err_pct [inherited]

Definition at line 137 of file abinitio.py.

8.2.4.45 forcebalance.abinitio.AbInitio.tq_ref [inherited]

Definition at line 987 of file abinitio.py.

8.2.4.46 forcebalance.abinitio.AbInitio.traj [inherited]

Definition at line 142 of file abinitio.py.

8.2.4.47 forcebalance.amberio.AbInitio_AMBER.trajfnm

Name of the trajectory, we need this BEFORE initializing the SuperClass.

Definition at line 176 of file amberio.py.

8.2.4.48 forcebalance.abinitio.AbInitio.use_nft [inherited]

Whether to compute net forces and torques, or not.

Definition at line 139 of file abinitio.py.

8.2.4.49 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.2.4.50 forcebalance.abinitio.AbInitio.w_energy [inherited]

Definition at line 571 of file abinitio.py.

8.2.4.51 forcebalance.abinitio.AbInitio.w_force [inherited]

Definition at line 350 of file abinitio.py.

8.2.4.52 forcebalance.abinitio.AbInitio.w_netforce [inherited]

Definition at line 571 of file abinitio.py.

8.2.4.53 forcebalance.abinitio.AbInitio.w_resp [inherited]

Definition at line 1000 of file abinitio.py.

8.2.4.54 forcebalance.abinitio.AbInitio.w_torque [inherited]

Definition at line 571 of file abinitio.py.

8.2.4.55 forcebalance.abinitio.AbInitio.whamboltz [inherited]

Definition at line 370 of file abinitio.py.

8.2.4.56 forcebalance.abinitio.AbInitio.whamboltz_wts [inherited]

Initialize the base class.

Number of snapshots Whether to use WHAM Boltzmann weights Whether to use the Sampling Correction Whether to match Absolute Energies (make sure you know what you're doing) Whether to use the Covariance Matrix Whether to use QM Boltzmann weights The temperature for QM Boltzmann weights Number of atoms that we are fitting Whether to fit Energies. Whether to fit Forces. Whether to fit Electrostatic Potential. Weights for the three components. Option for how much data to write to disk. Whether to do energy and force calculations for the whole trajectory, or to do one calculation per snapshot. OpenMM-only option - whether to run the energies and forces internally. Whether we have virtual sites (set at the global option level) WHAM Boltzmann weights

Definition at line 110 of file abinitio.py.

8.2.4.57 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- amberio.py

8.3 forcebalance.gmxio.AbInitio_GMX Class Reference

Subclass of AbInitio for force and energy matching using normal GROMACS.

Inheritance diagram for forcebalance.gmxio.AbInitio_GMX:

Collaboration diagram for forcebalance.gmxio.AbInitio_GMX:

Public Member Functions

- def __init__
- def read_topology

  This function parses the GROMACS topology file (.top) which contains a listing of the molecules in the simulation.
- def prepare_temp_directory
- def energy_force_driver

  Computes the energy and force using GROMACS for a single snapshot.
- def energy_force_driver_all

  Computes the energy and force using GROMACS for a trajectory.
- def generate_vsite_positions

  Call mdrun in order to update the virtual site positions.
- def build_invdist
- def compute_netforce_torque
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def indicate
- def energy_force_transformer_all

- def energy_force_transformer
- def get_energy_force_

  LPW 06-30-2013.
- def get_resp_

  Electrostatic potential fitting.
- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory.
- topfnm
- AtomMask
- topology_flag
- whamboltz_wts

  Initialize the base class.
- qmboltz_wts

  QM Boltzmann weights.
- eqm

  Reference (QM) energies.
- emd0

  Energies of the sampling simulation.
- fqm

  Reference (QM) forces.
- espxyz

  ESP grid points.
- espval

  ESP values.

- qfnm

  The qdata.txt file that contains the QM energies and forces.
- qmatoms

  The number of atoms in the QM calculation (Irrelevant if not fitting forces)
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)
- e_err_pct
- f_err

  Qualitative Indicator: average force error (fractional)
- f_err_pct
- esp_err

  Qualitative Indicator: "relative RMS" for electrostatic potential.
- nf_err
- nf_err_pct
- tq_err_pct
- use_nft

  Whether to compute net forces and torques, or not.
- ns

  Read in the trajectory file.
- traj
- nparticles

  The number of (atoms + drude particles + virtual sites)
- AtomLists

  This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.
- new_vsites

  Read in the topology.
- save_vmvals

  Save the mvals from the last time we updated the vsites.
- force_map
- nnf
- ntq
- force
- w_force
- nesp
- fitatoms
- whamboltz
- nftqm
- fref
- w_energy
- w_netforce
- w_torque
- e_ref
- f_ref
- nf_ref
- tq_ref
- tq_err
- w_resp
- invdists

- respterm
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.3.1 Detailed Description

Subclass of AbInitio for force and energy matching using normal GROMACS.

Implements the prepare_temp_directory and energy_force_driver methods.

Definition at line 438 of file gmxio.py.

8.3.2 Constructor & Destructor Documentation

8.3.2.1 def forcebalance.gmxio.AbInitio_GMX.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 440 of file gmxio.py.

Here is the call graph for this function:

8.3.3 Member Function Documentation

8.3.3.1 def forcebalance.abinitio.AbInitio.build_invdist (*self, mvals*) [inherited]

Definition at line 168 of file abinitio.py.

Here is the call graph for this function:

8.3.3.2 def forcebalance.abinitio.AbInitio.compute_netforce_torque (*self, xyz, force, QM* = `False`) [inherited]

Definition at line 204 of file abinitio.py.

8.3.3.3 def forcebalance.gmxio.AbInitio_GMX.energy_force_driver (*self, shot*)

Computes the energy and force using GROMACS for a single snapshot.

This does not require GROMACS-X2.

Definition at line 518 of file gmxio.py.

Here is the call graph for this function:

8.3.3.4 def forcebalance.gmxio.AbInitio_GMX.energy_force_driver_all (*self*)

Computes the energy and force using GROMACS for a trajectory.

This does not require GROMACS-X2.

Definition at line 539 of file gmxio.py.

Here is the call graph for this function:

8.3.3.5 def forcebalance.abinitio.AbInitio.energy_force_transformer (*self, i*) [inherited]

Definition at line 458 of file abinitio.py.

Here is the call graph for this function:

8.3.3.6 def forcebalance.abinitio.AbInitio.energy_force_transformer_all (*self*) [inherited]

Definition at line 442 of file abinitio.py.

Here is the call graph for this function:

8.3.3.7 def forcebalance.gmxio.AbInitio_GMX.generate_vsite_positions (*self*)

Call mdrun in order to update the virtual site positions.

Definition at line 562 of file gmxio.py.

Here is the call graph for this function:

8.3.3.8 def forcebalance.abinitio.AbInitio.get (*self*, *mvals*, *AGrad* = False, *AHess* = False) [inherited]

Definition at line 1098 of file abinitio.py.

Here is the call graph for this function:

8.3.3.9 def forcebalance.abinitio.AbInitio.get_energy_force_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) `[inherited]`

LPW 06-30-2013.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general simulation software.  This is
in contrast to using GROMACS-X2, which computes the objective
function and prints it out; then 'get' only needs to call
GROMACS and read it in.

This subroutine interfaces with simulation software 'drivers'.
The driver is only expected to give the energy and forces.

Now this subroutine may sound trivial since the objective
function is simply a least-squares quantity (M-Q)^2 - but
there are a number of nontrivial considerations.  I will list
them here.

0) Polytensor formulation: Because there may exist covariance
between different components of the force (or covariance
between the energy and the force), we build the objective
function by taking outer products of vectors that have the
form [E F_1x F_1y F_1z F_2x F_2y ... ], and then we trace it
with the inverse of the covariance matrix to get the objective
function.

This version implements both the polytensor formulation and the standard formulation.

1) Boltzmann weights and normalization: Each snapshot has its
own Boltzmann weight, which may or may not be normalized.
This subroutine does the normalization automatically.

2) Subtracting out the mean energy gap: The zero-point energy
difference between reference data and simulation is
meaningless.  This subroutine subtracts it out.

3) Hybrid ensembles: This program builds a combined objective
function from both MM and QM ensembles, which is rigorously
better than using a single ensemble.

Note that this subroutine does not do EVERYTHING that
GROMACS-X2 can do, which includes:

1) Internal coordinate systems
2) 'Sampling correction' (deprecated, since it doesn't seem to work)
3) Analytic derivatives
```

```
In the previous code (ForTune) this subroutine used analytic
first derivatives of the energy and force to build the
derivatives of the objective function.  Here I will take a
simplified approach, because building the derivatives are
cumbersome.  For now we will return the objective function
ONLY.  A two-point central difference should give us the first
and diagonal second derivative anyhow.
```

Todo Parallelization over snapshots is not implemented yet

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 532 of file abinitio.py.

8.3.3.10 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.3.3.11 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.3.3.12 def forcebalance.abinitio.AbInitio.get_resp_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) [inherited]

Electrostatic potential fitting.

Implements the RESP objective function. (In Python so obviously not optimized.) This function takes the mathematical parameter values and returns the charges on the ATOMS (fancy mapping going on)

Definition at line 999 of file abinitio.py.

8.3.3.13 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.3.3.14 def forcebalance.abinitio.AbInitio.indicate (*self*) [inherited]

Definition at line 422 of file abinitio.py.

Here is the call graph for this function:

8.3.3.15 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.3.3.16 def forcebalance.gmxio.AbInitio_GMX.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 493 of file gmxio.py.

Here is the call graph for this function:

8.3.3.17 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.3.3.18 def forcebalance.abinitio.AbInitio.read_reference_data (*self*) `[inherited]`

Read the reference ab initio data from a file such as qdata.txt.

Todo Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Todo Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

```
After reading in the information from qdata.txt, it is converted
into the GROMACS energy units (kind of an arbitrary choice);
forces (kind of a misnomer in qdata.txt) are multipled by -1
```

```
to convert gradients to forces.

We also subtract out the mean energies of all energy arrays
because energy/force matching does not account for zero-point
energy differences between MM and QM (i.e. energy of electrons
in core orbitals).

The configurations in force/energy matching typically come
from a the thermodynamic ensemble of the MM force field at
some temperature (by running MD, for example), and for many
reasons it is helpful to introduce non-Boltzmann weights in
front of these configurations.  There are two options: WHAM
Boltzmann weights (for combining the weights of several
simulations together) and QM Boltzmann weights (for converting
MM weights into QM weights).  Note that the two sets of weights
'stack'; i.e. they can be used at the same time.

A 'hybrid' ensemble is possible where we use 50% MM and 50% QM
weights.  Please read more in LPW and Troy Van Voorhis, JCP
Vol. 133, Pg. 231101 (2010), doi:10.1063/1.3519043.
```

Todo The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

```
Finally, note that using non-Boltzmann weights degrades the
statistical information content of the snapshots.  This
problem will generally become worse if the ensemble to which
we're reweighting is dramatically different from the one we're
sampling from.  We end up with a set of Boltzmann weights like
[1e-9, 1e-9, 1.0, 1e-9, 1e-9 ... ] and this is essentially just
one snapshot.  I believe Troy is working on something to cure
this problem.

Here, we have a measure for the information content of our snapshots,
which comes easily from the definition of information entropy:

S = -1*Sum_i(P_i*log(P_i))
InfoContent = exp(-S)

With uniform weights, InfoContent is equal to the number of snapshots;
with horrible weights, InfoContent is closer to one.
```

Definition at line 317 of file abinitio.py.

8.3.3.19 def forcebalance.gmxio.AbInitio_GMX.read_topology (*self*)

This function parses the GROMACS topology file (.top) which contains a listing of the molecules in the simulation.

For each molecule, it loads up a "FFMolecule" dictionary which contains information about each atom in the molecule - which residue (i.e. molecular fragment) the atom belongs in, the charge group, the particle type etc.

This allows us to do things like condense the gradients into net forces and torques, determine which particles are real atoms and which are virtual sites, and so on.

Definition at line 455 of file gmxio.py.

Here is the call graph for this function:

8.3.3.20 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.3.3.21 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.3.3.22 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.3.3.23 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.3.3.24 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.3.3.25 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.3.4 Member Data Documentation

8.3.4.1 forcebalance.abinitio.AbInitio.AtomLists [inherited]

This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.

Definition at line 150 of file abinitio.py.

8.3.4.2 forcebalance.gmxio.AbInitio_GMX.AtomMask

Definition at line 489 of file gmxio.py.

8.3.4.3 forcebalance.abinitio.AbInitio.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 128 of file abinitio.py.

8.3.4.4 forcebalance.abinitio.AbInitio.e_err_pct [inherited]

Definition at line 129 of file abinitio.py.

8.3.4.5 forcebalance.abinitio.AbInitio.e_ref [inherited]

Definition at line 976 of file abinitio.py.

8.3.4.6 forcebalance.abinitio.AbInitio.emd0 [inherited]

Energies of the sampling simulation.

Definition at line 116 of file abinitio.py.

8.3.4.7 forcebalance.abinitio.AbInitio.eqm [inherited]

Reference (QM) energies.

Definition at line 114 of file abinitio.py.

8.3.4.8 forcebalance.abinitio.AbInitio.esp_err [inherited]

Qualitative Indicator: "relative RMS" for electrostatic potential.

Definition at line 134 of file abinitio.py.

8.3.4.9 forcebalance.abinitio.AbInitio.espval [inherited]

ESP values.

Definition at line 122 of file abinitio.py.

8.3.4.10 forcebalance.abinitio.AbInitio.espxyz [inherited]

ESP grid points.

Definition at line 120 of file abinitio.py.

8.3.4.11 forcebalance.abinitio.AbInitio.f_err [inherited]

Qualitative Indicator: average force error (fractional)

Definition at line 131 of file abinitio.py.

8.3.4.12 forcebalance.abinitio.AbInitio.f_err_pct [inherited]

Definition at line 132 of file abinitio.py.

8.3.4.13 forcebalance.abinitio.AbInitio.f_ref [inherited]

Definition at line 980 of file abinitio.py.

8.3.4.14 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.3.4.15 forcebalance.abinitio.AbInitio.fitatoms [inherited]

Definition at line 354 of file abinitio.py.

8.3.4.16 forcebalance.abinitio.AbInitio.force [inherited]

Definition at line 349 of file abinitio.py.

8.3.4.17 forcebalance.abinitio.AbInitio.force_map [inherited]

Definition at line 212 of file abinitio.py.

8.3.4.18 forcebalance.abinitio.AbInitio.fqm [inherited]

Reference (QM) forces.

Definition at line 118 of file abinitio.py.

8.3.4.19 forcebalance.abinitio.AbInitio.fref [inherited]

Definition at line 413 of file abinitio.py.

8.3.4.20 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.3.4.21 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.3.4.22 forcebalance.abinitio.AbInitio.invdists [inherited]

Definition at line 1007 of file abinitio.py.

8.3.4.23 forcebalance.abinitio.AbInitio.nesp [inherited]

Definition at line 351 of file abinitio.py.

8.3.4.24 forcebalance.abinitio.AbInitio.new_vsites [inherited]

Read in the topology.

Read in the reference data Prepare the temporary directory The below two options are related to whether we want to rebuild virtual site positions. Rebuild the distance matrix if virtual site positions have changed

Definition at line 159 of file abinitio.py.

8.3.4.25 forcebalance.abinitio.AbInitio.nf_err [inherited]

Definition at line 135 of file abinitio.py.

8.3.4.26 forcebalance.abinitio.AbInitio.nf_err_pct [inherited]

Definition at line 136 of file abinitio.py.

8.3.4.27 forcebalance.abinitio.AbInitio.nf_ref [inherited]

Definition at line 984 of file abinitio.py.

8.3.4.28 forcebalance.abinitio.AbInitio.nftqm [inherited]

Definition at line 409 of file abinitio.py.

8.3.4.29 forcebalance.abinitio.AbInitio.nnf [inherited]

Definition at line 255 of file abinitio.py.

8.3.4.30 forcebalance.abinitio.AbInitio.nparticles [inherited]

The number of (atoms + drude particles + virtual sites)

Definition at line 147 of file abinitio.py.

8.3.4.31 forcebalance.abinitio.AbInitio.ns [inherited]

Read in the trajectory file.

Definition at line 141 of file abinitio.py.

8.3.4.32 forcebalance.abinitio.AbInitio.ntq [inherited]

Definition at line 256 of file abinitio.py.

8.3.4.33 forcebalance.abinitio.AbInitio.objective [inherited]

Definition at line 1118 of file abinitio.py.

8.3.4.34 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.3.4.35 forcebalance.abinitio.AbInitio.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 124 of file abinitio.py.

8.3.4.36 forcebalance.abinitio.AbInitio.qmatoms [inherited]

The number of atoms in the QM calculation (Irrelevant if not fitting forces)

Definition at line 126 of file abinitio.py.

8.3.4.37 forcebalance.abinitio.AbInitio.qmboltz_wts [inherited]

QM Boltzmann weights.

Definition at line 112 of file abinitio.py.

8.3.4.38 forcebalance.abinitio.AbInitio.respterm [inherited]

Definition at line 1086 of file abinitio.py.

8.3.4.39 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.3.4.40 forcebalance.abinitio.AbInitio.save_vmvals [inherited]

Save the mvals from the last time we updated the vsites.

Definition at line 161 of file abinitio.py.

8.3.4.41 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.3.4.42 forcebalance.gmxio.AbInitio_GMX.topfnm

Definition at line 443 of file gmxio.py.

8.3.4.43 forcebalance.gmxio.AbInitio_GMX.topology_flag

Definition at line 490 of file gmxio.py.

8.3.4.44 forcebalance.abinitio.AbInitio.tq_err [inherited]

Definition at line 988 of file abinitio.py.

8.3.4.45 forcebalance.abinitio.AbInitio.tq_err_pct [inherited]

Definition at line 137 of file abinitio.py.

8.3.4.46 forcebalance.abinitio.AbInitio.tq_ref [inherited]

Definition at line 987 of file abinitio.py.

8.3.4.47 forcebalance.abinitio.AbInitio.traj [inherited]

Definition at line 142 of file abinitio.py.

8.3.4.48 forcebalance.gmxio.AbInitio_GMX.trajfnm

Name of the trajectory.

Definition at line 442 of file gmxio.py.

8.3.4.49 forcebalance.abinitio.AbInitio.use_nft [inherited]

Whether to compute net forces and torques, or not.

Definition at line 139 of file abinitio.py.

8.3.4.50 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.3.4.51 forcebalance.abinitio.AbInitio.w_energy [inherited]

Definition at line 571 of file abinitio.py.

8.3.4.52 forcebalance.abinitio.AbInitio.w_force [inherited]

Definition at line 350 of file abinitio.py.

8.3.4.53 forcebalance.abinitio.AbInitio.w_netforce [inherited]

Definition at line 571 of file abinitio.py.

8.3.4.54 forcebalance.abinitio.AbInitio.w_resp [inherited]

Definition at line 1000 of file abinitio.py.

8.3.4.55 forcebalance.abinitio.AbInitio.w_torque [inherited]

Definition at line 571 of file abinitio.py.

8.3.4.56 forcebalance.abinitio.AbInitio.whamboltz [inherited]

Definition at line 370 of file abinitio.py.

8.3.4.57 forcebalance.abinitio.AbInitio.whamboltz_wts [inherited]

Initialize the base class.

Number of snapshots Whether to use WHAM Boltzmann weights Whether to use the Sampling Correction Whether to match Absolute Energies (make sure you know what you're doing) Whether to use the Covariance Matrix Whether to use QM Boltzmann weights The temperature for QM Boltzmann weights Number of atoms that we are fitting Whether to fit Energies. Whether to fit Forces. Whether to fit Electrostatic Potential. Weights for the three components. Option for how much data to write to disk. Whether to do energy and force calculations for the whole trajectory, or to do one calculation per snapshot. OpenMM-only option - whether to run the energies and forces internally. Whether we have virtual sites (set at the global option level) WHAM Boltzmann weights

Definition at line 110 of file abinitio.py.

8.3.4.58 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- gmxio.py

8.4 forcebalance.abinitio_internal.AbInitio_Internal Class Reference

Subclass of Target for force and energy matching using an internal implementation.

Inheritance diagram for forcebalance.abinitio_internal.AbInitio_Internal:

Collaboration diagram for forcebalance.abinitio_internal.AbInitio_Internal:

Public Member Functions

- def __init__
- def energy_force_driver_all

  Here we actually compute the interactions and return the energies and forces.
- def read_topology
- def build_invdist
- def compute_netforce_torque
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def prepare_temp_directory

  Prepare the temporary directory, by default does nothing.
- def indicate
- def energy_force_transformer_all
- def energy_force_transformer
- def get_energy_force_

  LPW 06-30-2013.

- def get_resp_

  Electrostatic potential fitting.
- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory, we need this BEFORE initializing the SuperClass.
- whamboltz_wts

  Initialize the base class.
- qmboltz_wts

  QM Boltzmann weights.
- eqm

  Reference (QM) energies.
- emd0

  Energies of the sampling simulation.
- fqm

  Reference (QM) forces.
- espxyz

  ESP grid points.
- espval

  ESP values.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- qmatoms

  The number of atoms in the QM calculation (Irrelevant if not fitting forces)
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)

- e_err_pct
- f_err

  Qualitative Indicator: average force error (fractional)
- f_err_pct
- esp_err

  Qualitative Indicator: "relative RMS" for electrostatic potential.
- nf_err
- nf_err_pct
- tq_err_pct
- use_nft

  Whether to compute net forces and torques, or not.
- ns

  Read in the trajectory file.
- traj
- nparticles

  The number of (atoms + drude particles + virtual sites)
- AtomLists

  This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.
- new_vsites

  Read in the topology.
- save_vmvals

  Save the mvals from the last time we updated the vsites.
- topology_flag
- force_map
- nnf
- ntq
- force
- w_force
- nesp
- fitatoms
- whamboltz
- nftqm
- fref
- w_energy
- w_netforce
- w_torque
- e_ref
- f_ref
- nf_ref
- tq_ref
- tq_err
- w_resp
- invdists
- respterm
- objective
- tempdir

  Root directory of the whole project.
- rundir

The directory in which the simulation is running - this can be updated.

- FF

  Need the forcefield (here for now)

- xct

  Counts how often the objective function was computed.

- gct

  Counts how often the gradient was computed.

- hct

  Counts how often the Hessian was computed.

- PrintOptionDict
- verbose_options

8.4.1 Detailed Description

Subclass of Target for force and energy matching using an internal implementation.

Implements the prepare and energy_force_driver methods. The get method is in the superclass.

The purpose of this class is to provide an extremely simple test case that does not require the user to install any external software. It only runs with one of the included sample test calculations (internal_tip3p), and the objective function is energy matching.

Warning

This class is only intended to work with a very specific test case (internal_tip3p). This is because the topology and ordering of the atoms is hard-coded (12 water molecules with 3 atoms each).
This class does energy matching only (no forces)

Definition at line 37 of file abinitio_internal.py.

8.4.2 Constructor & Destructor Documentation

8.4.2.1 def forcebalance.abinitio_internal.AbInitio_Internal.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 40 of file abinitio_internal.py.

8.4.3 Member Function Documentation

8.4.3.1 def forcebalance.abinitio.AbInitio.build_invdist (*self, mvals*) [inherited]

Definition at line 168 of file abinitio.py.

Here is the call graph for this function:

8.4.3.2 def forcebalance.abinitio.AbInitio.compute_netforce_torque (*self, xyz, force, QM* = `False`) [inherited]

Definition at line 204 of file abinitio.py.

8.4.3.3 def forcebalance.abinitio_internal.AbInitio_Internal.energy_force_driver_all (*self*)

Here we actually compute the interactions and return the energies and forces.

I verified this to give the same answer as GROMACS.

Definition at line 50 of file abinitio_internal.py.

8.4.3.4 def forcebalance.abinitio.AbInitio.energy_force_transformer (*self, i*) [inherited]

Definition at line 458 of file abinitio.py.

Here is the call graph for this function:

8.4.3.5 def forcebalance.abinitio.AbInitio.energy_force_transformer_all (*self*) [inherited]

Definition at line 442 of file abinitio.py.

Here is the call graph for this function:

8.4.3.6 def forcebalance.abinitio.AbInitio.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Definition at line 1098 of file abinitio.py.

Here is the call graph for this function:

8.4.3.7 def forcebalance.abinitio.AbInitio.get_energy_force_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) `[inherited]`

LPW 06-30-2013.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general simulation software.  This is
in contrast to using GROMACS-X2, which computes the objective
function and prints it out; then 'get' only needs to call
GROMACS and read it in.

This subroutine interfaces with simulation software 'drivers'.
The driver is only expected to give the energy and forces.

Now this subroutine may sound trivial since the objective
function is simply a least-squares quantity (M-Q)^2 - but
there are a number of nontrivial considerations.  I will list
them here.

0) Polytensor formulation: Because there may exist covariance
between different components of the force (or covariance
between the energy and the force), we build the objective
function by taking outer products of vectors that have the
form [E F_1x F_1y F_1z F_2x F_2y ... ], and then we trace it
with the inverse of the covariance matrix to get the objective
function.

This version implements both the polytensor formulation and the standard formulation.

1) Boltzmann weights and normalization: Each snapshot has its
own Boltzmann weight, which may or may not be normalized.
This subroutine does the normalization automatically.

2) Subtracting out the mean energy gap: The zero-point energy
difference between reference data and simulation is
meaningless.  This subroutine subtracts it out.

3) Hybrid ensembles: This program builds a combined objective
function from both MM and QM ensembles, which is rigorously
better than using a single ensemble.

Note that this subroutine does not do EVERYTHING that
GROMACS-X2 can do, which includes:

1) Internal coordinate systems
2) 'Sampling correction' (deprecated, since it doesn't seem to work)
3) Analytic derivatives
```

```
In the previous code (ForTune) this subroutine used analytic
first derivatives of the energy and force to build the
derivatives of the objective function.  Here I will take a
simplified approach, because building the derivatives are
cumbersome.  For now we will return the objective function
ONLY.  A two-point central difference should give us the first
and diagonal second derivative anyhow.
```

Todo Parallelization over snapshots is not implemented yet

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 532 of file abinitio.py.

8.4.3.8 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.4.3.9 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.4.3.10 def forcebalance.abinitio.AbInitio.get_resp_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) [inherited]

Electrostatic potential fitting.

Implements the RESP objective function. (In Python so obviously not optimized.) This function takes the mathematical parameter values and returns the charges on the ATOMS (fancy mapping going on)

Definition at line 999 of file abinitio.py.

8.4.3.11 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.4.3.12 def forcebalance.abinitio.AbInitio.indicate (*self*) [inherited]

Definition at line 422 of file abinitio.py.

Here is the call graph for this function:

8.4.3.13 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.4.3.14 def forcebalance.abinitio.AbInitio.prepare_temp_directory (*self, options, tgt_opts*) [inherited]

Prepare the temporary directory, by default does nothing.

Definition at line 419 of file abinitio.py.

8.4.3.15 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.4.3.16 def forcebalance.abinitio.AbInitio.read_reference_data (*self*) `[inherited]`

Read the reference ab initio data from a file such as qdata.txt.

Todo Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Todo Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

```
After reading in the information from qdata.txt, it is converted
into the GROMACS energy units (kind of an arbitrary choice);
forces (kind of a misnomer in qdata.txt) are multipled by -1
to convert gradients to forces.

We also subtract out the mean energies of all energy arrays
because energy/force matching does not account for zero-point
energy differences between MM and QM (i.e. energy of electrons
in core orbitals).

The configurations in force/energy matching typically come
from a the thermodynamic ensemble of the MM force field at
some temperature (by running MD, for example), and for many
reasons it is helpful to introduce non-Boltzmann weights in
front of these configurations.  There are two options: WHAM
Boltzmann weights (for combining the weights of several
simulations together) and QM Boltzmann weights (for converting
MM weights into QM weights).  Note that the two sets of weights
'stack'; i.e. they can be used at the same time.
```

```
A 'hybrid' ensemble is possible where we use 50% MM and 50% QM
weights.  Please read more in LPW and Troy Van Voorhis, JCP
Vol. 133, Pg. 231101 (2010), doi:10.1063/1.3519043.
```

Todo The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

```
Finally, note that using non-Boltzmann weights degrades the
statistical information content of the snapshots.  This
problem will generally become worse if the ensemble to which
we're reweighting is dramatically different from the one we're
sampling from.  We end up with a set of Boltzmann weights like
[1e-9, 1e-9, 1.0, 1e-9, 1e-9 ... ] and this is essentially just
one snapshot.  I believe Troy is working on something to cure
this problem.

Here, we have a measure for the information content of our snapshots,
which comes easily from the definition of information entropy:

S = -1*Sum_i(P_i*log(P_i))
InfoContent = exp(-S)

With uniform weights, InfoContent is equal to the number of snapshots;
with horrible weights, InfoContent is closer to one.
```

Definition at line 317 of file abinitio.py.

8.4.3.17 def forcebalance.abinitio.AbInitio.read_topology (*self*) [inherited]

Definition at line 164 of file abinitio.py.

Here is the call graph for this function:

8.4.3.18 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.4.3.19 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.4.3.20 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.4.3.21 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.4.3.22 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.4.3.23 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.4.4 Member Data Documentation

8.4.4.1 forcebalance.abinitio.AbInitio.AtomLists [inherited]

This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.

Definition at line 150 of file abinitio.py.

8.4.4.2 forcebalance.abinitio.AbInitio.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 128 of file abinitio.py.

8.4.4.3 forcebalance.abinitio.AbInitio.e_err_pct [inherited]

Definition at line 129 of file abinitio.py.

8.4.4.4 forcebalance.abinitio.AbInitio.e_ref [inherited]

Definition at line 976 of file abinitio.py.

8.4.4.5 forcebalance.abinitio.AbInitio.emd0 [inherited]

Energies of the sampling simulation.

Definition at line 116 of file abinitio.py.

8.4.4.6 forcebalance.abinitio.AbInitio.eqm [inherited]

Reference (QM) energies.

Definition at line 114 of file abinitio.py.

8.4.4.7 forcebalance.abinitio.AbInitio.esp_err [inherited]

Qualitative Indicator: "relative RMS" for electrostatic potential.

Definition at line 134 of file abinitio.py.

8.4.4.8 forcebalance.abinitio.AbInitio.espval [inherited]

ESP values.

Definition at line 122 of file abinitio.py.

8.4.4.9 forcebalance.abinitio.AbInitio.espxyz [inherited]

ESP grid points.

Definition at line 120 of file abinitio.py.

8.4.4.10 forcebalance.abinitio.AbInitio.f_err [inherited]

Qualitative Indicator: average force error (fractional)

Definition at line 131 of file abinitio.py.

8.4.4.11 forcebalance.abinitio.AbInitio.f_err_pct [inherited]

Definition at line 132 of file abinitio.py.

8.4.4.12 forcebalance.abinitio.AbInitio.f_ref [inherited]

Definition at line 980 of file abinitio.py.

8.4.4.13 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.4.4.14 forcebalance.abinitio.AbInitio.fitatoms [inherited]

Definition at line 354 of file abinitio.py.

8.4.4.15 forcebalance.abinitio.AbInitio.force [inherited]

Definition at line 349 of file abinitio.py.

8.4.4.16 forcebalance.abinitio.AbInitio.force_map [inherited]

Definition at line 212 of file abinitio.py.

8.4.4.17 forcebalance.abinitio.AbInitio.fqm [inherited]

Reference (QM) forces.

Definition at line 118 of file abinitio.py.

8.4.4.18 forcebalance.abinitio.AbInitio.fref [inherited]

Definition at line 413 of file abinitio.py.

8.4.4.19 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.4.4.20 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.4.4.21 forcebalance.abinitio.AbInitio.invdists [inherited]

Definition at line 1007 of file abinitio.py.

8.4.4.22 forcebalance.abinitio.AbInitio.nesp [inherited]

Definition at line 351 of file abinitio.py.

8.4.4.23 forcebalance.abinitio.AbInitio.new_vsites [inherited]

Read in the topology.

Read in the reference data Prepare the temporary directory The below two options are related to whether we want to rebuild virtual site positions. Rebuild the distance matrix if virtual site positions have changed

Definition at line 159 of file abinitio.py.

8.4.4.24 forcebalance.abinitio.AbInitio.nf_err [inherited]

Definition at line 135 of file abinitio.py.

8.4.4.25 forcebalance.abinitio.AbInitio.nf_err_pct [inherited]

Definition at line 136 of file abinitio.py.

8.4.4.26 forcebalance.abinitio.AbInitio.nf_ref [inherited]

Definition at line 984 of file abinitio.py.

8.4.4.27 forcebalance.abinitio.AbInitio.nftqm [inherited]

Definition at line 409 of file abinitio.py.

8.4.4.28 forcebalance.abinitio.AbInitio.nnf [inherited]

Definition at line 255 of file abinitio.py.

8.4.4.29 forcebalance.abinitio.AbInitio.nparticles [inherited]

The number of (atoms + drude particles + virtual sites)

Definition at line 147 of file abinitio.py.

8.4.4.30 forcebalance.abinitio.AbInitio.ns [inherited]

Read in the trajectory file.

Definition at line 141 of file abinitio.py.

8.4.4.31 forcebalance.abinitio.AbInitio.ntq [inherited]

Definition at line 256 of file abinitio.py.

8.4.4.32 forcebalance.abinitio.AbInitio.objective [inherited]

Definition at line 1118 of file abinitio.py.

8.4.4.33 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.4.4.34 forcebalance.abinitio.AbInitio.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 124 of file abinitio.py.

8.4.4.35 forcebalance.abinitio.AbInitio.qmatoms [inherited]

The number of atoms in the QM calculation (Irrelevant if not fitting forces)

Definition at line 126 of file abinitio.py.

8.4.4.36 forcebalance.abinitio.AbInitio.qmboltz_wts [inherited]

QM Boltzmann weights.

Definition at line 112 of file abinitio.py.

8.4.4.37 forcebalance.abinitio.AbInitio.respterm [inherited]

Definition at line 1086 of file abinitio.py.

8.4.4.38 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.4.4.39 forcebalance.abinitio.AbInitio.save_vmvals [inherited]

Save the mvals from the last time we updated the vsites.

Definition at line 161 of file abinitio.py.

8.4.4.40 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.4.4.41 forcebalance.abinitio.AbInitio.topology_flag [inherited]

Definition at line 166 of file abinitio.py.

8.4.4.42 forcebalance.abinitio.AbInitio.tq_err [inherited]

Definition at line 988 of file abinitio.py.

8.4.4.43 forcebalance.abinitio.AbInitio.tq_err_pct [inherited]

Definition at line 137 of file abinitio.py.

8.4.4.44 forcebalance.abinitio.AbInitio.tq_ref [inherited]

Definition at line 987 of file abinitio.py.

8.4.4.45 forcebalance.abinitio.AbInitio.traj [inherited]

Definition at line 142 of file abinitio.py.

8.4.4.46 forcebalance.abinitio_internal.AbInitio_Internal.trajfnm

Name of the trajectory, we need this BEFORE initializing the SuperClass.

Definition at line 42 of file abinitio_internal.py.

8.4.4.47 forcebalance.abinitio.AbInitio.use_nft [inherited]

Whether to compute net forces and torques, or not.

Definition at line 139 of file abinitio.py.

8.4.4.48 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.4.4.49 forcebalance.abinitio.AbInitio.w_energy [inherited]

Definition at line 571 of file abinitio.py.

8.4.4.50 forcebalance.abinitio.AbInitio.w_force [inherited]

Definition at line 350 of file abinitio.py.

8.4.4.51 forcebalance.abinitio.AbInitio.w_netforce [inherited]

Definition at line 571 of file abinitio.py.

8.4.4.52 forcebalance.abinitio.AbInitio.w_resp [inherited]

Definition at line 1000 of file abinitio.py.

8.4.4.53 forcebalance.abinitio.AbInitio.w_torque [inherited]

Definition at line 571 of file abinitio.py.

8.4.4.54 forcebalance.abinitio.AbInitio.whamboltz [inherited]

Definition at line 370 of file abinitio.py.

8.4.4.55 forcebalance.abinitio.AbInitio.whamboltz_wts [inherited]

Initialize the base class.

Number of snapshots Whether to use WHAM Boltzmann weights Whether to use the Sampling Correction Whether to match Absolute Energies (make sure you know what you're doing) Whether to use the Covariance Matrix Whether to use QM Boltzmann weights The temperature for QM Boltzmann weights Number of atoms that we are fitting Whether

to fit Energies. Whether to fit Forces. Whether to fit Electrostatic Potential. Weights for the three components. Option for how much data to write to disk. Whether to do energy and force calculations for the whole trajectory, or to do one calculation per snapshot. OpenMM-only option - whether to run the energies and forces internally. Whether we have virtual sites (set at the global option level) WHAM Boltzmann weights

Definition at line 110 of file abinitio.py.

8.4.4.56 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- abinitio_internal.py

8.5 forcebalance.openmmio.AbInitio_OpenMM Class Reference

Subclass of AbInitio for force and energy matching using OpenMM.

Inheritance diagram for forcebalance.openmmio.AbInitio_OpenMM:

Collaboration diagram for forcebalance.openmmio.AbInitio_OpenMM:

Public Member Functions

- def __init__
- def read_topology
- def prepare_temp_directory
- def energy_force_driver_all_external_
- def energy_force_driver_all_internal_

  Loop through the snapshots and compute the energies and forces using OpenMM.
- def energy_force_driver_all
- def build_invdist
- def compute_netforce_torque
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def indicate
- def energy_force_transformer_all
- def energy_force_transformer
- def get_energy_force_

  LPW 06-30-2013.

- def get_resp_

  Electrostatic potential fitting.
- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory, we need this BEFORE initializing the SuperClass.
- platform

  Initialize the SuperClass!
- simulation

  Create the simulation object within this class itself.
- xyz_omms
- topology_flag
- whamboltz_wts

  Initialize the base class.
- qmboltz_wts

  QM Boltzmann weights.
- eqm

  Reference (QM) energies.
- emd0

  Energies of the sampling simulation.
- fqm

  Reference (QM) forces.
- espxyz

  ESP grid points.
- espval

  ESP values.

- qfnm

  The qdata.txt file that contains the QM energies and forces.
- qmatoms

  The number of atoms in the QM calculation (Irrelevant if not fitting forces)
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)
- e_err_pct
- f_err

  Qualitative Indicator: average force error (fractional)
- f_err_pct
- esp_err

  Qualitative Indicator: "relative RMS" for electrostatic potential.
- nf_err
- nf_err_pct
- tq_err_pct
- use_nft

  Whether to compute net forces and torques, or not.
- ns

  Read in the trajectory file.
- traj
- nparticles

  The number of (atoms + drude particles + virtual sites)
- AtomLists

  This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.
- new_vsites

  Read in the topology.
- save_vmvals

  Save the mvals from the last time we updated the vsites.
- force_map
- nnf
- ntq
- force
- w_force
- nesp
- fitatoms
- whamboltz
- nftqm
- fref
- w_energy
- w_netforce
- w_torque
- e_ref
- f_ref
- nf_ref
- tq_ref
- tq_err
- w_resp
- invdists

- respterm
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.5.1 Detailed Description

Subclass of AbInitio for force and energy matching using OpenMM.

Implements the prepare and energy_force_driver methods. The get method is in the superclass.

Definition at line 400 of file openmmio.py.

8.5.2 Constructor & Destructor Documentation

8.5.2.1 def forcebalance.openmmio.AbInitio_OpenMM.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 403 of file openmmio.py.

8.5.3 Member Function Documentation

8.5.3.1 def forcebalance.abinitio.AbInitio.build_invdist (*self, mvals*) [inherited]

Definition at line 168 of file abinitio.py.

Here is the call graph for this function:

8.5.3.2 def forcebalance.abinitio.AbInitio.compute_netforce_torque (*self, xyz, force, QM* = `False`) [inherited]

Definition at line 204 of file abinitio.py.

8.5.3.3 def forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all (*self*)

Definition at line 527 of file openmmio.py.

Here is the call graph for this function:

8.5.3.4 def forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all_external_ (*self*)

Definition at line 471 of file openmmio.py.

8.5.3.5 def forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all_internal_ (*self*)

Loop through the snapshots and compute the energies and forces using OpenMM.

Definition at line 480 of file openmmio.py.

Here is the call graph for this function:

8.5.3.6 def forcebalance.abinitio.AbInitio.energy_force_transformer (*self, i*) [inherited]

Definition at line 458 of file abinitio.py.

Here is the call graph for this function:

8.5.3.7 def forcebalance.abinitio.AbInitio.energy_force_transformer_all (*self*) [inherited]

Definition at line 442 of file abinitio.py.

Here is the call graph for this function:

8.5.3.8 def forcebalance.abinitio.AbInitio.get (*self*, *mvals*, *AGrad* = False, *AHess* = False) [inherited]

Definition at line 1098 of file abinitio.py.

Here is the call graph for this function:

8.5.3.9 def forcebalance.abinitio.AbInitio.get_energy_force_ (*self*, *mvals*, *AGrad* = False, *AHess* = False) [inherited]

LPW 06-30-2013.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general simulation software.  This is
in contrast to using GROMACS-X2, which computes the objective
function and prints it out; then 'get' only needs to call
GROMACS and read it in.

This subroutine interfaces with simulation software 'drivers'.
The driver is only expected to give the energy and forces.

Now this subroutine may sound trivial since the objective
function is simply a least-squares quantity (M-Q)^2 - but
there are a number of nontrivial considerations.  I will list
them here.

0) Polytensor formulation: Because there may exist covariance
```

```
between different components of the force (or covariance
between the energy and the force), we build the objective
function by taking outer products of vectors that have the
form [E F_1x F_1y F_1z F_2x F_2y ... ], and then we trace it
with the inverse of the covariance matrix to get the objective
function.

This version implements both the polytensor formulation and the standard formulation.

1) Boltzmann weights and normalization: Each snapshot has its
own Boltzmann weight, which may or may not be normalized.
This subroutine does the normalization automatically.

2) Subtracting out the mean energy gap: The zero-point energy
difference between reference data and simulation is
meaningless.  This subroutine subtracts it out.

3) Hybrid ensembles: This program builds a combined objective
function from both MM and QM ensembles, which is rigorously
better than using a single ensemble.

Note that this subroutine does not do EVERYTHING that
GROMACS-X2 can do, which includes:

1) Internal coordinate systems
2) 'Sampling correction' (deprecated, since it doesn't seem to work)
3) Analytic derivatives

In the previous code (ForTune) this subroutine used analytic
first derivatives of the energy and force to build the
derivatives of the objective function.  Here I will take a
simplified approach, because building the derivatives are
cumbersome.  For now we will return the objective function
ONLY.  A two-point central difference should give us the first
and diagonal second derivative anyhow.
```

Todo Parallelization over snapshots is not implemented yet

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 532 of file abinitio.py.

8.5.3.10 def forcebalance.target.Target.get_G (*self*, *mvals* = None) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.5.3.11 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.5.3.12 def forcebalance.abinitio.AbInitio.get_resp_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) [inherited]

Electrostatic potential fitting.

Implements the RESP objective function. (In Python so obviously not optimized.) This function takes the mathematical parameter values and returns the charges on the ATOMS (fancy mapping going on)

Definition at line 999 of file abinitio.py.

8.5.3.13 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.5.3.14 def forcebalance.abinitio.AbInitio.indicate (*self*) [inherited]

Definition at line 422 of file abinitio.py.

Here is the call graph for this function:

8.5.3.15 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.5.3.16 def forcebalance.openmmio.AbInitio_OpenMM.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 466 of file openmmio.py.

Here is the call graph for this function:

8.5.3.17 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.5.3.18 def forcebalance.abinitio.AbInitio.read_reference_data (*self*) `[inherited]`

Read the reference ab initio data from a file such as qdata.txt.

Todo Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Todo Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

```
After reading in the information from qdata.txt, it is converted
into the GROMACS energy units (kind of an arbitrary choice);
forces (kind of a misnomer in qdata.txt) are multipled by -1
to convert gradients to forces.

We also subtract out the mean energies of all energy arrays
because energy/force matching does not account for zero-point
```

```
energy differences between MM and QM (i.e. energy of electrons
in core orbitals).

The configurations in force/energy matching typically come
from a the thermodynamic ensemble of the MM force field at
some temperature (by running MD, for example), and for many
reasons it is helpful to introduce non-Boltzmann weights in
front of these configurations.  There are two options: WHAM
Boltzmann weights (for combining the weights of several
simulations together) and QM Boltzmann weights (for converting
MM weights into QM weights).  Note that the two sets of weights
'stack'; i.e. they can be used at the same time.

A 'hybrid' ensemble is possible where we use 50% MM and 50% QM
weights.  Please read more in LPW and Troy Van Voorhis, JCP
Vol. 133, Pg. 231101 (2010), doi:10.1063/1.3519043.
```

Todo The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

```
Finally, note that using non-Boltzmann weights degrades the
statistical information content of the snapshots.  This
problem will generally become worse if the ensemble to which
we're reweighting is dramatically different from the one we're
sampling from.  We end up with a set of Boltzmann weights like
[1e-9, 1e-9, 1.0, 1e-9, 1e-9 ... ] and this is essentially just
one snapshot.  I believe Troy is working on something to cure
this problem.

Here, we have a measure for the information content of our snapshots,
which comes easily from the definition of information entropy:

S = -1*Sum_i(P_i*log(P_i))
InfoContent = exp(-S)

With uniform weights, InfoContent is equal to the number of snapshots;
with horrible weights, InfoContent is closer to one.
```

Definition at line 317 of file abinitio.py.

8.5.3.19 def forcebalance.openmmio.AbInitio_OpenMM.read_topology (*self*)

Definition at line 456 of file openmmio.py.

Here is the call graph for this function:

8.5.3.20 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.5.3.21 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.5.3.22 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.5.3.23 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.5.3.24 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.5.3.25 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.5.4 Member Data Documentation

8.5.4.1 forcebalance.abinitio.AbInitio.AtomLists [inherited]

This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.

Definition at line 150 of file abinitio.py.

8.5.4.2 forcebalance.abinitio.AbInitio.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 128 of file abinitio.py.

8.5.4.3 forcebalance.abinitio.AbInitio.e_err_pct [inherited]

Definition at line 129 of file abinitio.py.

8.5.4.4 forcebalance.abinitio.AbInitio.e_ref [inherited]

Definition at line 976 of file abinitio.py.

8.5.4.5 forcebalance.abinitio.AbInitio.emd0 [inherited]

Energies of the sampling simulation.

Definition at line 116 of file abinitio.py.

8.5.4.6 forcebalance.abinitio.AbInitio.eqm [inherited]

Reference (QM) energies.

Definition at line 114 of file abinitio.py.

8.5.4.7 forcebalance.abinitio.AbInitio.esp_err [inherited]

Qualitative Indicator: "relative RMS" for electrostatic potential.

Definition at line 134 of file abinitio.py.

8.5.4.8 forcebalance.abinitio.AbInitio.espval [inherited]

ESP values.

Definition at line 122 of file abinitio.py.

8.5.4.9 forcebalance.abinitio.AbInitio.espxyz [inherited]

ESP grid points.

Definition at line 120 of file abinitio.py.

8.5.4.10 forcebalance.abinitio.AbInitio.f_err [inherited]

Qualitative Indicator: average force error (fractional)

Definition at line 131 of file abinitio.py.

8.5.4.11 forcebalance.abinitio.AbInitio.f_err_pct [inherited]

Definition at line 132 of file abinitio.py.

8.5.4.12 forcebalance.abinitio.AbInitio.f_ref [inherited]

Definition at line 980 of file abinitio.py.

8.5.4.13 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.5.4.14 forcebalance.abinitio.AbInitio.fitatoms [inherited]

Definition at line 354 of file abinitio.py.

8.5.4.15 forcebalance.abinitio.AbInitio.force [inherited]

Definition at line 349 of file abinitio.py.

8.5.4.16 forcebalance.abinitio.AbInitio.force_map [inherited]

Definition at line 212 of file abinitio.py.

8.5.4.17 forcebalance.abinitio.AbInitio.fqm [inherited]

Reference (QM) forces.

Definition at line 118 of file abinitio.py.

8.5.4.18 forcebalance.abinitio.AbInitio.fref [inherited]

Definition at line 413 of file abinitio.py.

8.5.4.19 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.5.4.20 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.5.4.21 forcebalance.abinitio.AbInitio.invdists [inherited]

Definition at line 1007 of file abinitio.py.

8.5.4.22 forcebalance.abinitio.AbInitio.nesp [inherited]

Definition at line 351 of file abinitio.py.

8.5.4.23 forcebalance.abinitio.AbInitio.new_vsites [inherited]

Read in the topology.

Read in the reference data Prepare the temporary directory The below two options are related to whether we want to rebuild virtual site positions. Rebuild the distance matrix if virtual site positions have changed

Definition at line 159 of file abinitio.py.

8.5.4.24 forcebalance.abinitio.AbInitio.nf_err [inherited]

Definition at line 135 of file abinitio.py.

8.5.4.25 forcebalance.abinitio.AbInitio.nf_err_pct [inherited]

Definition at line 136 of file abinitio.py.

8.5.4.26 forcebalance.abinitio.AbInitio.nf_ref [inherited]

Definition at line 984 of file abinitio.py.

8.5.4.27 forcebalance.abinitio.AbInitio.nftqm [inherited]

Definition at line 409 of file abinitio.py.

8.5.4.28 forcebalance.abinitio.AbInitio.nnf [inherited]

Definition at line 255 of file abinitio.py.

8.5.4.29 forcebalance.abinitio.AbInitio.nparticles [inherited]

The number of (atoms + drude particles + virtual sites)

Definition at line 147 of file abinitio.py.

8.5.4.30 forcebalance.abinitio.AbInitio.ns [inherited]

Read in the trajectory file.

Definition at line 141 of file abinitio.py.

8.5.4.31 forcebalance.abinitio.AbInitio.ntq [inherited]

Definition at line 256 of file abinitio.py.

8.5.4.32 forcebalance.abinitio.AbInitio.objective [inherited]

Definition at line 1118 of file abinitio.py.

8.5.4.33 forcebalance.openmmio.AbInitio_OpenMM.platform

Initialize the SuperClass!

Set the device to the environment variable or zero otherwise.

Set the simulation platform

Definition at line 412 of file openmmio.py.

8.5.4.34 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.5.4.35 forcebalance.abinitio.AbInitio.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 124 of file abinitio.py.

8.5.4.36 forcebalance.abinitio.AbInitio.qmatoms [inherited]

The number of atoms in the QM calculation (Irrelevant if not fitting forces)

Definition at line 126 of file abinitio.py.

8.5.4.37 forcebalance.abinitio.AbInitio.qmboltz_wts [inherited]

QM Boltzmann weights.

Definition at line 112 of file abinitio.py.

8.5.4.38 forcebalance.abinitio.AbInitio.respterm [inherited]

Definition at line 1086 of file abinitio.py.

8.5.4.39 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.5.4.40 forcebalance.abinitio.AbInitio.save_vmvals [inherited]

Save the mvals from the last time we updated the vsites.

Definition at line 161 of file abinitio.py.

8.5.4.41 forcebalance.openmmio.AbInitio_OpenMM.simulation

Create the simulation object within this class itself.

Definition at line 444 of file openmmio.py.

8.5.4.42 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite

difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.5.4.43 forcebalance.openmmio.AbInitio_OpenMM.topology_flag

Definition at line 463 of file openmmio.py.

8.5.4.44 forcebalance.abinitio.AbInitio.tq_err [inherited]

Definition at line 988 of file abinitio.py.

8.5.4.45 forcebalance.abinitio.AbInitio.tq_err_pct [inherited]

Definition at line 137 of file abinitio.py.

8.5.4.46 forcebalance.abinitio.AbInitio.tq_ref [inherited]

Definition at line 987 of file abinitio.py.

8.5.4.47 forcebalance.abinitio.AbInitio.traj [inherited]

Definition at line 142 of file abinitio.py.

8.5.4.48 forcebalance.openmmio.AbInitio_OpenMM.trajfnm

Name of the trajectory, we need this BEFORE initializing the SuperClass.

Definition at line 405 of file openmmio.py.

8.5.4.49 forcebalance.abinitio.AbInitio.use_nft [inherited]

Whether to compute net forces and torques, or not.

Definition at line 139 of file abinitio.py.

8.5.4.50 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.5.4.51 forcebalance.abinitio.AbInitio.w_energy [inherited]

Definition at line 571 of file abinitio.py.

8.5.4.52 forcebalance.abinitio.AbInitio.w_force [inherited]

Definition at line 350 of file abinitio.py.

8.5.4.53 forcebalance.abinitio.AbInitio.w_netforce [inherited]

Definition at line 571 of file abinitio.py.

8.5.4.54 forcebalance.abinitio.AbInitio.w_resp [inherited]

Definition at line 1000 of file abinitio.py.

8.5.4.55 forcebalance.abinitio.AbInitio.w_torque [inherited]

Definition at line 571 of file abinitio.py.

8.5.4.56 forcebalance.abinitio.AbInitio.whamboltz [inherited]

Definition at line 370 of file abinitio.py.

8.5.4.57 forcebalance.abinitio.AbInitio.whamboltz_wts [inherited]

Initialize the base class.

Number of snapshots Whether to use WHAM Boltzmann weights Whether to use the Sampling Correction Whether to match Absolute Energies (make sure you know what you're doing) Whether to use the Covariance Matrix Whether to use QM Boltzmann weights The temperature for QM Boltzmann weights Number of atoms that we are fitting Whether to fit Energies. Whether to fit Forces. Whether to fit Electrostatic Potential. Weights for the three components. Option for how much data to write to disk. Whether to do energy and force calculations for the whole trajectory, or to do one calculation per snapshot. OpenMM-only option - whether to run the energies and forces internally. Whether we have virtual sites (set at the global option level) WHAM Boltzmann weights

Definition at line 110 of file abinitio.py.

8.5.4.58 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

8.5.4.59 forcebalance.openmmio.AbInitio_OpenMM.xyz_omms

Definition at line 446 of file openmmio.py.

The documentation for this class was generated from the following file:

- openmmio.py

8.6 forcebalance.tinkerio.AbInitio_TINKER Class Reference

Subclass of Target for force and energy matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.AbInitio_TINKER:

Collaboration diagram for forcebalance.tinkerio.AbInitio_TINKER:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def energy_force_driver
- def energy_driver_all
- def energy_force_driver_all
- def read_topology
- def build_invdist
- def compute_netforce_torque
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def indicate
- def energy_force_transformer_all
- def energy_force_transformer
- def get_energy_force_

  LPW 06-30-2013.
- def get_resp_

Electrostatic potential fitting.

- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory.
- all_at_once

  all_at_once is not implemented.
- whamboltz_wts

  Initialize the base class.
- qmboltz_wts

  QM Boltzmann weights.
- eqm

  Reference (QM) energies.
- emd0

  Energies of the sampling simulation.
- fqm

  Reference (QM) forces.
- espxyz

  ESP grid points.
- espval

  ESP values.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- qmatoms

  The number of atoms in the QM calculation (Irrelevant if not fitting forces)
- e_err

Qualitative Indicator: average energy error (in kJ/mol)

- e_err_pct
- f_err

  Qualitative Indicator: average force error (fractional)
- f_err_pct
- esp_err

  Qualitative Indicator: "relative RMS" for electrostatic potential.
- nf_err
- nf_err_pct
- tq_err_pct
- use_nft

  Whether to compute net forces and torques, or not.
- ns

  Read in the trajectory file.
- traj
- nparticles

  The number of (atoms + drude particles + virtual sites)
- AtomLists

  This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.
- new_vsites

  Read in the topology.
- save_vmvals

  Save the mvals from the last time we updated the vsites.
- topology_flag
- force_map
- nnf
- ntq
- force
- w_force
- nesp
- fitatoms
- whamboltz
- nftqm
- fref
- w_energy
- w_netforce
- w_torque
- e_ref
- f_ref
- nf_ref
- tq_ref
- tq_err
- w_resp
- invdists
- respterm
- objective
- tempdir

  Root directory of the whole project.

- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.6.1 Detailed Description

Subclass of Target for force and energy matching using TINKER.

Implements the prepare and energy_force_driver methods.

Definition at line 297 of file tinkerio.py.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 def forcebalance.tinkerio.AbInitio_TINKER.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 300 of file tinkerio.py.

Here is the call graph for this function:

8.6.3 Member Function Documentation

8.6.3.1 def forcebalance.abinitio.AbInitio.build_invdist (*self, mvals*) [inherited]

Definition at line 168 of file abinitio.py.

Here is the call graph for this function:

8.6.3.2 def forcebalance.abinitio.AbInitio.compute_netforce_torque (*self, xyz, force, QM* = `False`) [inherited]

Definition at line 204 of file abinitio.py.

8.6.3.3 def forcebalance.tinkerio.AbInitio_TINKER.energy_driver_all (*self*)

Definition at line 333 of file tinkerio.py.

Here is the call graph for this function:

8.6.3.4 def forcebalance.tinkerio.AbInitio_TINKER.energy_force_driver (*self, shot*)

Definition at line 317 of file tinkerio.py.

Here is the call graph for this function:

8.6.3.5 def forcebalance.tinkerio.AbInitio_TINKER.energy_force_driver_all (*self*)

Definition at line 346 of file tinkerio.py.

Here is the call graph for this function:

8.6.3.6 def forcebalance.abinitio.AbInitio.energy_force_transformer (*self, i*) [inherited]

Definition at line 458 of file abinitio.py.

Here is the call graph for this function:

8.6.3.7 def forcebalance.abinitio.AbInitio.energy_force_transformer_all (*self*) [inherited]

Definition at line 442 of file abinitio.py.

Here is the call graph for this function:

8.6.3.8 def forcebalance.abinitio.AbInitio.get (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) [inherited]

Definition at line 1098 of file abinitio.py.

Here is the call graph for this function:

8.6.3.9 def forcebalance.abinitio.AbInitio.get_energy_force_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) [inherited]

LPW 06-30-2013.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general simulation software.  This is
in contrast to using GROMACS-X2, which computes the objective
function and prints it out; then 'get' only needs to call
GROMACS and read it in.

This subroutine interfaces with simulation software 'drivers'.
The driver is only expected to give the energy and forces.

Now this subroutine may sound trivial since the objective
function is simply a least-squares quantity (M-Q)^2 - but
there are a number of nontrivial considerations.  I will list
them here.

0) Polytensor formulation: Because there may exist covariance
between different components of the force (or covariance
between the energy and the force), we build the objective
function by taking outer products of vectors that have the
form [E F_1x F_1y F_1z F_2x F_2y ... ], and then we trace it
with the inverse of the covariance matrix to get the objective
function.

This version implements both the polytensor formulation and the standard formulation.

1) Boltzmann weights and normalization: Each snapshot has its
own Boltzmann weight, which may or may not be normalized.
This subroutine does the normalization automatically.

2) Subtracting out the mean energy gap: The zero-point energy
difference between reference data and simulation is
meaningless.  This subroutine subtracts it out.

3) Hybrid ensembles: This program builds a combined objective
function from both MM and QM ensembles, which is rigorously
better than using a single ensemble.

Note that this subroutine does not do EVERYTHING that
GROMACS-X2 can do, which includes:

1) Internal coordinate systems
2) 'Sampling correction' (deprecated, since it doesn't seem to work)
3) Analytic derivatives

In the previous code (ForTune) this subroutine used analytic
first derivatives of the energy and force to build the
derivatives of the objective function.  Here I will take a
simplified approach, because building the derivatives are
cumbersome.  For now we will return the objective function
ONLY.  A two-point central difference should give us the first
and diagonal second derivative anyhow.
```

Todo Parallelization over snapshots is not implemented yet

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 532 of file abinitio.py.

8.6.3.10 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.6.3.11 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.6.3.12 def forcebalance.abinitio.AbInitio.get_resp_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`) [inherited]

Electrostatic potential fitting.

Implements the RESP objective function. (In Python so obviously not optimized.) This function takes the mathematical parameter values and returns the charges on the ATOMS (fancy mapping going on)

Definition at line 999 of file abinitio.py.

8.6.3.13 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.6.3.14 def forcebalance.abinitio.AbInitio.indicate (*self*) [inherited]

Definition at line 422 of file abinitio.py.

Here is the call graph for this function:

8.6.3.15 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.6.3.16 def forcebalance.tinkerio.AbInitio_TINKER.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 309 of file tinkerio.py.

Here is the call graph for this function:

8.6.3.17 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.6.3.18 def forcebalance.abinitio.AbInitio.read_reference_data (*self*) `[inherited]`

Read the reference ab initio data from a file such as qdata.txt.

Todo Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Todo Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

```
After reading in the information from qdata.txt, it is converted
into the GROMACS energy units (kind of an arbitrary choice);
forces (kind of a misnomer in qdata.txt) are multipled by -1
to convert gradients to forces.
```

```
We also subtract out the mean energies of all energy arrays
because energy/force matching does not account for zero-point
energy differences between MM and QM (i.e. energy of electrons
in core orbitals).

The configurations in force/energy matching typically come
from a the thermodynamic ensemble of the MM force field at
some temperature (by running MD, for example), and for many
reasons it is helpful to introduce non-Boltzmann weights in
front of these configurations.  There are two options: WHAM
Boltzmann weights (for combining the weights of several
simulations together) and QM Boltzmann weights (for converting
MM weights into QM weights).  Note that the two sets of weights
'stack'; i.e. they can be used at the same time.

A 'hybrid' ensemble is possible where we use 50% MM and 50% QM
weights.  Please read more in LPW and Troy Van Voorhis, JCP
Vol. 133, Pg. 231101 (2010), doi:10.1063/1.3519043.
```

Todo The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

```
Finally, note that using non-Boltzmann weights degrades the
statistical information content of the snapshots.  This
problem will generally become worse if the ensemble to which
we're reweighting is dramatically different from the one we're
sampling from.  We end up with a set of Boltzmann weights like
[1e-9, 1e-9, 1.0, 1e-9, 1e-9 ... ] and this is essentially just
one snapshot.  I believe Troy is working on something to cure
this problem.

Here, we have a measure for the information content of our snapshots,
which comes easily from the definition of information entropy:

S = -1*Sum_i(P_i*log(P_i))
InfoContent = exp(-S)

With uniform weights, InfoContent is equal to the number of snapshots;
with horrible weights, InfoContent is closer to one.
```

Definition at line 317 of file abinitio.py.

8.6.3.19 def forcebalance.abinitio.AbInitio.read_topology (*self*) [inherited]

Definition at line 164 of file abinitio.py.

Here is the call graph for this function:

8.6.3.20 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.6.3.21 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.6.3.22 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.6.3.23 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.6.3.24 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.6.3.25 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.6.4 Member Data Documentation

8.6.4.1 forcebalance.tinkerio.AbInitio_TINKER.all_at_once

all_at_once is not implemented.

Definition at line 307 of file tinkerio.py.

8.6.4.2 forcebalance.abinitio.AbInitio.AtomLists [inherited]

This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.

Definition at line 150 of file abinitio.py.

8.6.4.3 forcebalance.abinitio.AbInitio.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 128 of file abinitio.py.

8.6.4.4 forcebalance.abinitio.AbInitio.e_err_pct [inherited]

Definition at line 129 of file abinitio.py.

8.6.4.5 forcebalance.abinitio.AbInitio.e_ref [inherited]

Definition at line 976 of file abinitio.py.

8.6.4.6 forcebalance.abinitio.AbInitio.emd0 [inherited]

Energies of the sampling simulation.

Definition at line 116 of file abinitio.py.

8.6.4.7 forcebalance.abinitio.AbInitio.eqm [inherited]

Reference (QM) energies.

Definition at line 114 of file abinitio.py.

8.6.4.8 forcebalance.abinitio.AbInitio.esp_err [inherited]

Qualitative Indicator: "relative RMS" for electrostatic potential.

Definition at line 134 of file abinitio.py.

8.6.4.9 forcebalance.abinitio.AbInitio.espval [inherited]

ESP values.

Definition at line 122 of file abinitio.py.

8.6.4.10 forcebalance.abinitio.AbInitio.espxyz [inherited]

ESP grid points.

Definition at line 120 of file abinitio.py.

8.6.4.11 forcebalance.abinitio.AbInitio.f_err [inherited]

Qualitative Indicator: average force error (fractional)

Definition at line 131 of file abinitio.py.

8.6.4.12 forcebalance.abinitio.AbInitio.f_err_pct [inherited]

Definition at line 132 of file abinitio.py.

8.6.4.13 forcebalance.abinitio.AbInitio.f_ref [inherited]

Definition at line 980 of file abinitio.py.

8.6.4.14 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.6.4.15 forcebalance.abinitio.AbInitio.fitatoms [inherited]

Definition at line 354 of file abinitio.py.

8.6.4.16 forcebalance.abinitio.AbInitio.force [inherited]

Definition at line 349 of file abinitio.py.

8.6.4.17 forcebalance.abinitio.AbInitio.force_map [inherited]

Definition at line 212 of file abinitio.py.

8.6.4.18 forcebalance.abinitio.AbInitio.fqm [inherited]

Reference (QM) forces.

Definition at line 118 of file abinitio.py.

8.6.4.19 forcebalance.abinitio.AbInitio.fref [inherited]

Definition at line 413 of file abinitio.py.

8.6.4.20 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.6.4.21 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.6.4.22 forcebalance.abinitio.AbInitio.invdists [inherited]

Definition at line 1007 of file abinitio.py.

8.6.4.23 forcebalance.abinitio.AbInitio.nesp [inherited]

Definition at line 351 of file abinitio.py.

8.6.4.24 forcebalance.abinitio.AbInitio.new_vsites [inherited]

Read in the topology.

Read in the reference data Prepare the temporary directory The below two options are related to whether we want to rebuild virtual site positions. Rebuild the distance matrix if virtual site positions have changed

Definition at line 159 of file abinitio.py.

8.6.4.25 forcebalance.abinitio.AbInitio.nf_err [inherited]

Definition at line 135 of file abinitio.py.

8.6.4.26 forcebalance.abinitio.AbInitio.nf_err_pct [inherited]

Definition at line 136 of file abinitio.py.

8.6.4.27 forcebalance.abinitio.AbInitio.nf_ref [inherited]

Definition at line 984 of file abinitio.py.

8.6.4.28 forcebalance.abinitio.AbInitio.nftqm [inherited]

Definition at line 409 of file abinitio.py.

8.6.4.29 forcebalance.abinitio.AbInitio.nnf [inherited]

Definition at line 255 of file abinitio.py.

8.6.4.30 forcebalance.abinitio.AbInitio.nparticles [inherited]

The number of (atoms + drude particles + virtual sites)

Definition at line 147 of file abinitio.py.

8.6.4.31 forcebalance.abinitio.AbInitio.ns [inherited]

Read in the trajectory file.

Definition at line 141 of file abinitio.py.

8.6.4.32 forcebalance.abinitio.AbInitio.ntq [inherited]

Definition at line 256 of file abinitio.py.

8.6.4.33 forcebalance.abinitio.AbInitio.objective [inherited]

Definition at line 1118 of file abinitio.py.

8.6.4.34 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.6.4.35 forcebalance.abinitio.AbInitio.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 124 of file abinitio.py.

8.6.4.36 forcebalance.abinitio.AbInitio.qmatoms [inherited]

The number of atoms in the QM calculation (Irrelevant if not fitting forces)

Definition at line 126 of file abinitio.py.

8.6.4.37 forcebalance.abinitio.AbInitio.qmboltz_wts [inherited]

QM Boltzmann weights.

Definition at line 112 of file abinitio.py.

8.6.4.38 forcebalance.abinitio.AbInitio.respterm [inherited]

Definition at line 1086 of file abinitio.py.

8.6.4.39 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.6.4.40 forcebalance.abinitio.AbInitio.save_vmvals [inherited]

Save the mvals from the last time we updated the vsites.

Definition at line 161 of file abinitio.py.

8.6.4.41 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.6.4.42 forcebalance.abinitio.AbInitio.topology_flag [inherited]

Definition at line 166 of file abinitio.py.

8.6.4.43 forcebalance.abinitio.AbInitio.tq_err [inherited]

Definition at line 988 of file abinitio.py.

8.6.4.44 forcebalance.abinitio.AbInitio.tq_err_pct [inherited]

Definition at line 137 of file abinitio.py.

8.6.4.45 forcebalance.abinitio.AbInitio.tq_ref [inherited]

Definition at line 987 of file abinitio.py.

8.6.4.46 forcebalance.abinitio.AbInitio.traj [inherited]

Definition at line 142 of file abinitio.py.

8.6.4.47 forcebalance.tinkerio.AbInitio_TINKER.trajfnm

Name of the trajectory.

Definition at line 302 of file tinkerio.py.

8.6.4.48 forcebalance.abinitio.AbInitio.use_nft [inherited]

Whether to compute net forces and torques, or not.

Definition at line 139 of file abinitio.py.

8.6.4.49 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.6.4.50 forcebalance.abinitio.AbInitio.w_energy [inherited]

Definition at line 571 of file abinitio.py.

8.6.4.51 forcebalance.abinitio.AbInitio.w_force [inherited]

Definition at line 350 of file abinitio.py.

8.6.4.52 forcebalance.abinitio.AbInitio.w_netforce [inherited]

Definition at line 571 of file abinitio.py.

8.6.4.53 forcebalance.abinitio.AbInitio.w_resp [inherited]

Definition at line 1000 of file abinitio.py.

8.6.4.54 forcebalance.abinitio.AbInitio.w_torque [inherited]

Definition at line 571 of file abinitio.py.

8.6.4.55 forcebalance.abinitio.AbInitio.whamboltz [inherited]

Definition at line 370 of file abinitio.py.

8.6.4.56 forcebalance.abinitio.AbInitio.whamboltz_wts [inherited]

Initialize the base class.

Number of snapshots Whether to use WHAM Boltzmann weights Whether to use the Sampling Correction Whether to match Absolute Energies (make sure you know what you're doing) Whether to use the Covariance Matrix Whether to use QM Boltzmann weights The temperature for QM Boltzmann weights Number of atoms that we are fitting Whether to fit Energies. Whether to fit Forces. Whether to fit Electrostatic Potential. Weights for the three components. Option for how much data to write to disk. Whether to do energy and force calculations for the whole trajectory, or to do one calculation per snapshot. OpenMM-only option - whether to run the energies and forces internally. Whether we have virtual sites (set at the global option level) WHAM Boltzmann weights

Definition at line 110 of file abinitio.py.

8.6.4.57 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- tinkerio.py

8.7 forcebalance.forcefield.BackedUpDict Class Reference

Inheritance diagram for forcebalance.forcefield.BackedUpDict:

Collaboration diagram for forcebalance.forcefield.BackedUpDict:

Public Member Functions

- def __init__
- def __missing__

Public Attributes

- backup_dict

8.7.1 Detailed Description

Definition at line 175 of file forcefield.py.

8.7.2 Constructor & Destructor Documentation

8.7.2.1 def forcebalance.forcefield.BackedUpDict.__init__ (*self, backup_dict*)

Definition at line 176 of file forcefield.py.

Here is the call graph for this function:

8.7.3 Member Function Documentation

8.7.3.1 def forcebalance.forcefield.BackedUpDict.__missing__ (*self, key*)

Definition at line 179 of file forcefield.py.

Here is the call graph for this function:

8.7.4 Member Data Documentation

8.7.4.1 forcebalance.forcefield.BackedUpDict.backup_dict

Definition at line 178 of file forcefield.py.

The documentation for this class was generated from the following file:

- forcefield.py

8.8 forcebalance.BaseClass Class Reference

Provides some nifty functions that are common to all ForceBalance classes.

Inheritance diagram for forcebalance.BaseClass:

Collaboration diagram for forcebalance.BaseClass:

Public Member Functions

- def __init__
- def set_option

Public Attributes

- PrintOptionDict
- verbose_options

8.8.1 Detailed Description

Provides some nifty functions that are common to all ForceBalance classes.

Definition at line 29 of file __init__.py.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 def forcebalance.BaseClass.__init__ (*self, options*)

Definition at line 31 of file __init__.py.

8.8.3 Member Function Documentation

8.8.3.1 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`)

Definition at line 35 of file __init__.py.

8.8.4 Member Data Documentation

8.8.4.1 forcebalance.BaseClass.PrintOptionDict

Definition at line 32 of file __init__.py.

8.8.4.2 forcebalance.BaseClass.verbose_options

Definition at line 33 of file __init__.py.

The documentation for this class was generated from the following file:

- __init__.py

8.9 forcebalance.BaseReader Class Reference

The 'reader' class.

Inheritance diagram for forcebalance.BaseReader:

Collaboration diagram for forcebalance.BaseReader:

Public Member Functions

- def __init__
- def Split
- def Whites
- def feed
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- ln
- itype
- suffix
- pdict
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.9.1 Detailed Description

The 'reader' class.

It serves two main functions:

1) When parsing a text force field file, the 'feed' method is called once for every line. Calling the 'feed' method stores the internal variables that are needed for making the unique parameter identifier.

2) The 'reader' also stores the 'pdict' dictionary, which is needed for building the matrix of rescaling factors. This is not needed for the XML force fields, so in XML force fields pdict is replaced with a string called "XML_Override".

Definition at line 64 of file __init__.py.

8.9.2 Constructor & Destructor Documentation

8.9.2.1 def forcebalance.BaseReader.__init__ (*self, fnm*)

Definition at line 66 of file __init__.py.

8.9.3 Member Function Documentation

8.9.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*)

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.9.3.2 def forcebalance.BaseReader.feed (*self, line*)

Definition at line 88 of file __init__.py.

Here is the call graph for this function:

8.9.3.3 def forcebalance.BaseReader.Split (*self, line*)

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.9.3.4 def forcebalance.BaseReader.Whites (*self, line*)

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.9.4 Member Data Documentation

8.9.4.1 forcebalance.BaseReader.adict

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.9.4.2 forcebalance.BaseReader.AtomTypes

Definition at line 80 of file __init__.py.

8.9.4.3 forcebalance.BaseReader.itype

Definition at line 68 of file __init__.py.

8.9.4.4 forcebalance.BaseReader.ln

Definition at line 67 of file __init__.py.

8.9.4.5 forcebalance.BaseReader.molatom

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.9.4.6 forcebalance.BaseReader.Molecules

Definition at line 79 of file __init__.py.

8.9.4.7 forcebalance.BaseReader.pdict

Definition at line 70 of file __init__.py.

8.9.4.8 forcebalance.BaseReader.suffix

Definition at line 69 of file __init__.py.

The documentation for this class was generated from the following file:

- __init__.py

8.10 forcebalance.binding.BindingEnergy Class Reference

Improved subclass of Target for fitting force fields to binding energies.

Inheritance diagram for forcebalance.binding.BindingEnergy:

Collaboration diagram for forcebalance.binding.BindingEnergy:

Public Member Functions

- def __init__
- def indicate
- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- inter_opts
- PrintDict
- RMSDDict
- rmsd_part
- energy_part
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.10.1 Detailed Description

Improved subclass of Target for fitting force fields to binding energies.

Definition at line 125 of file binding.py.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 def forcebalance.binding.BindingEnergy.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 128 of file binding.py.

Here is the call graph for this function:

8.10.3 Member Function Documentation

8.10.3.1 def forcebalance.binding.BindingEnergy.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Definition at line 164 of file binding.py.

8.10.3.2 def forcebalance.target.Target.get_G (*self*, *mvals* = None) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.10.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = None) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.10.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.10.3.5 def forcebalance.binding.BindingEnergy.indicate (*self*)

Definition at line 158 of file binding.py.

Here is the call graph for this function:

8.10.3.6 def forcebalance.target.Target.link_from_tempdir (*self*, *absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.10.3.7 def forcebalance.target.Target.printcool_table (*self*, *data* = OrderedDict([]), *headings* = [], *banner* = None, *footnote* = None, *color* = 0) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.10.3.8 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.10.3.9 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.10.3.10 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.10.3.11 def forcebalance.target.Target.stage (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.10.3.12 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.10.3.13 def forcebalance.target.Target.wq_complete (*self*) `[inherited]`

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.10.4 Member Data Documentation

8.10.4.1 forcebalance.binding.BindingEnergy.energy_part

Definition at line 204 of file binding.py.

8.10.4.2 forcebalance.target.Target.FF `[inherited]`

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.10.4.3 forcebalance.target.Target.gct `[inherited]`

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.10.4.4 forcebalance.target.Target.hct `[inherited]`

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.10.4.5 forcebalance.binding.BindingEnergy.inter_opts

Definition at line 131 of file binding.py.

8.10.4.6 forcebalance.binding.BindingEnergy.objective

Definition at line 228 of file binding.py.

8.10.4.7 forcebalance.binding.BindingEnergy.PrintDict

Definition at line 166 of file binding.py.

8.10.4.8 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.10.4.9 forcebalance.binding.BindingEnergy.rmsd_part

Definition at line 202 of file binding.py.

8.10.4.10 forcebalance.binding.BindingEnergy.RMSDDict

Definition at line 167 of file binding.py.

8.10.4.11 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.10.4.12 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.10.4.13 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.10.4.14 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- binding.py

8.11 forcebalance.tinkerio.BindingEnergy_TINKER Class Reference

Subclass of BindingEnergy for binding energy matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.BindingEnergy_TINKER:

Collaboration diagram for forcebalance.tinkerio.BindingEnergy_TINKER:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def system_driver
- def indicate
- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

Stages the directory for the target, and then calls 'get'.

- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- optprog
- inter_opts
- PrintDict
- RMSDDict
- rmsd_part
- energy_part
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.11.1 Detailed Description

Subclass of BindingEnergy for binding energy matching using TINKER.

Definition at line 486 of file tinkerio.py.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 def forcebalance.tinkerio.BindingEnergy_TINKER.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 489 of file tinkerio.py.

Here is the call graph for this function:

8.11.3 Member Function Documentation

8.11.3.1 def forcebalance.binding.BindingEnergy.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Definition at line 164 of file binding.py.

8.11.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.11.3.3 def forcebalance.target.Target.get_H (*self, mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.11.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.11.3.5 def forcebalance.binding.BindingEnergy.indicate (*self*) [inherited]

Definition at line 158 of file binding.py.

Here is the call graph for this function:

8.11.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.11.3.7 def forcebalance.tinkerio.BindingEnergy_TINKER.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 493 of file tinkerio.py.

8.11.3.8 def forcebalance.target.Target.printcool_table (*self, data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.11.3.9 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.11.3.10 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.11.3.11 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.11.3.12 def forcebalance.target.Target.stage (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.11.3.13 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Definition at line 291 of file target.py.

8.11.3.14 def forcebalance.tinkerio.BindingEnergy_TINKER.system_driver (*self, sysname*)

Definition at line 547 of file tinkerio.py.

Here is the call graph for this function:

8.11.3.15 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.11.4 Member Data Documentation

8.11.4.1 forcebalance.binding.BindingEnergy.energy_part [inherited]

Definition at line 204 of file binding.py.

8.11.4.2 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.11.4.3 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.11.4.4 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.11.4.5 forcebalance.binding.BindingEnergy.inter_opts [inherited]

Definition at line 131 of file binding.py.

8.11.4.6 forcebalance.binding.BindingEnergy.objective [inherited]

Definition at line 228 of file binding.py.

8.11.4.7 forcebalance.tinkerio.BindingEnergy_TINKER.optprog

Definition at line 496 of file tinkerio.py.

8.11.4.8 forcebalance.binding.BindingEnergy.PrintDict [inherited]

Definition at line 166 of file binding.py.

8.11.4.9 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.11.4.10 forcebalance.binding.BindingEnergy.rmsd_part [inherited]

Definition at line 202 of file binding.py.

8.11.4.11 forcebalance.binding.BindingEnergy.RMSDDict [inherited]

Definition at line 167 of file binding.py.

8.11.4.12 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.11.4.13 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.11.4.14 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.11.4.15 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- tinkerio.py

8.12 forcebalance.output.CleanFileHandler Class Reference

File handler that does not write terminal escape codes and carriage returns to files.

Inheritance diagram for forcebalance.output.CleanFileHandler:

Collaboration diagram for forcebalance.output.CleanFileHandler:

Public Member Functions

- def emit

8.12.1 Detailed Description

File handler that does not write terminal escape codes and carriage returns to files.

Use this when writing to a file that will probably not be viewed in a terminal

Definition at line 69 of file output.py.

8.12.2 Member Function Documentation

8.12.2.1 def forcebalance.output.CleanFileHandler.emit (*self, record*)

Definition at line 70 of file output.py.

The documentation for this class was generated from the following file:

- output.py

8.13 forcebalance.output.CleanStreamHandler Class Reference

Similar to RawStreamHandler except it does not write terminal escape codes.

Inheritance diagram for forcebalance.output.CleanStreamHandler:

Collaboration diagram for forcebalance.output.CleanStreamHandler:

Public Member Functions

- def __init__
- def emit

8.13.1 Detailed Description

Similar to RawStreamHandler except it does not write terminal escape codes.

Use this for 'plain' terminal output without any fancy colors or formatting

Definition at line 56 of file output.py.

8.13.2 Constructor & Destructor Documentation

8.13.2.1 def forcebalance.output.CleanStreamHandler.__init__ (*self, stream* = `sys.stdout`)

Definition at line 57 of file output.py.

8.13.3 Member Function Documentation

8.13.3.1 def forcebalance.output.CleanStreamHandler.emit (*self, record*)

Definition at line 60 of file output.py.

The documentation for this class was generated from the following file:

- output.py

8.14 forcebalance.counterpoise.Counterpoise Class Reference

Target subclass for matching the counterpoise correction.

Inheritance diagram for forcebalance.counterpoise.Counterpoise:

Collaboration diagram for forcebalance.counterpoise.Counterpoise:

Public Member Functions

- def __init__

  To instantiate Counterpoise, we read the coordinates and counterpoise data.
- def loadxyz

  Parse an XYZ file which contains several xyz coordinates, and return their elements.
- def load_cp

  Load in the counterpoise data, which is easy; the file consists of floating point numbers separated by newlines.
- def get

  Gets the objective function for fitting the counterpoise correction.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs

- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- xyzs

  Number of snapshots.
- cpqm

  Counterpoise correction data.
- na

  Number of atoms.
- ns
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.14.1 Detailed Description

Target subclass for matching the counterpoise correction.

Definition at line 33 of file counterpoise.py.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 def forcebalance.counterpoise.Counterpoise.__init__ (*self, options, tgt_opts, forcefield*)

To instantiate Counterpoise, we read the coordinates and counterpoise data.

Definition at line 37 of file counterpoise.py.

Here is the call graph for this function:

8.14.3 Member Function Documentation

8.14.3.1 def forcebalance.counterpoise.Counterpoise.get (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`)

Gets the objective function for fitting the counterpoise correction.

```
As opposed to AbInitio_GMXX2, which calls an external program,
this script actually computes the empirical interaction given the
force field parameters.

It loops through the snapshots and atom pairs, and computes pairwise
contributions to an energy term according to hard-coded functional forms.

One potential issue is that we go through all atom pairs instead of
looking only at atom pairs between different fragments.  This means that
even for two infinitely separated fragments it will predict a finite
CP correction.  While it might be okay to apply such a potential in practice,
there will be some issues for the fitting.  Thus, we assume the last snapshot
to be CP-free and subtract that value of the potential back out.

Note that forces and parametric derivatives are not implemented.
```

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient (not implemented)
in	*AHess*	Switch to turn on analytic Hessian (not implemented)

Returns

Answer Contribution to the objective function

Definition at line 124 of file counterpoise.py.

8.14.3.2 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.14.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.14.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.14.3.5 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.14.3.6 def forcebalance.counterpoise.Counterpoise.load_cp (*self, fnm*)

Load in the counterpoise data, which is easy; the file consists of floating point numbers separated by newlines.

Definition at line 95 of file counterpoise.py.

8.14.3.7 def forcebalance.counterpoise.Counterpoise.loadxyz (*self, fnm*)

Parse an XYZ file which contains several xyz coordinates, and return their elements.

Parameters

in	*fnm*	The input XYZ file name

Returns

elem A list of chemical elements in the XYZ file
xyzs A list of XYZ coordinates (number of snapshots times number of atoms)

Todo I should probably put this into a more general library for reading coordinates.

Definition at line 63 of file counterpoise.py.

8.14.3.8 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.14.3.9 def forcebalance.target.Target.refresh_temp_directory (*self*) `[inherited]`

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.14.3.10 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.14.3.11 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.14.3.12 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.14.3.13 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.14.3.14 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.14.4 Member Data Documentation

8.14.4.1 forcebalance.counterpoise.Counterpoise.cpqm

Counterpoise correction data.

Definition at line 53 of file counterpoise.py.

8.14.4.2 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.14.4.3 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.14.4.4 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.14.4.5 forcebalance.counterpoise.Counterpoise.na

Number of atoms.

Definition at line 76 of file counterpoise.py.

8.14.4.6 forcebalance.counterpoise.Counterpoise.ns

Definition at line 89 of file counterpoise.py.

8.14.4.7 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.14.4.8 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.14.4.9 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.14.4.10 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.14.4.11 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

8.14.4.12 forcebalance.counterpoise.Counterpoise.xyzs

Number of snapshots.

XYZ elements and coordinates

Definition at line 51 of file counterpoise.py.

The documentation for this class was generated from the following file:

- counterpoise.py

8.15 forcebalance.forcefield.FF Class Reference

Force field class.

Inheritance diagram for forcebalance.forcefield.FF:

Collaboration diagram for forcebalance.forcefield.FF:

Public Member Functions

- def __init__

  Instantiation of force field class.
- def addff

  Parse a force field file and add it to the class.
- def addff_txt

  Parse a text force field and create several important instance variables.

- def addff_xml

  Parse an XML force field file and create important instance variables.
- def make

  Create a new force field using provided parameter values.
- def make_redirect
- def find_spacings
- def create_pvals

  Converts mathematical to physical parameters.
- def create_mvals

  Converts physical to mathematical parameters.
- def rsmake

  Create the rescaling factors for the coordinate transformation in parameter space.
- def mktransmat

  Create the transformation matrix to rescale and rotate the mathematical parameters.
- def list_map

  Create the plist, which is like a reversed version of the parameter map.
- def print_map

  Prints out the (physical or mathematical) parameter indices, IDs and values in a visually appealing way.
- def assign_p0

  Assign physical parameter values to the 'pvals0' array.
- def assign_field

  Record the locations of a parameter in a txt file; [[file name, line number, field number, and multiplier]].
- def __eq__
- def set_option

Public Attributes

- ffdata

  As these options proliferate, the force field class becomes less standalone.
- ffdata_isxml
- map

  The mapping of interaction type -> parameter number.
- plist

  The listing of parameter number -> interaction types.
- patoms

  A listing of parameter number -> atoms involved.
- pfields

  A list where pfields[pnum] = ['file',line,field,mult,cmd], basically a new way to modify force field files; when we modify the force field file, we go to the specific line/field in a given file and change the number.
- rs

  List of rescaling factors.
- tm

  The transformation matrix for mathematical -> physical parameters.
- tml

  The transpose of the transformation matrix.
- excision

  Indices to exclude from optimization / Hessian inversion.

- np

  The total number of parameters.
- pvals0

  Initial value of physical parameters.
- Readers

  A dictionary of force field reader classes.
- atomnames

  A list of atom names (this is new, for ESP fitting)
- FFAtomTypes

  WORK IN PROGRESS ## This is a dictionary of {'AtomType':{'Mass' : float, 'Charge' : float, 'ParticleType' : string ('A', 'S', or 'D'), 'AtomicNumber' : int}}.
- FFMolecules
- redirect

  Creates plist from map.
- linedestroy_save

  Destruction dictionary (experimental).
- parmdestroy_save
- linedestroy_this
- parmdestroy_this
- tinkerprm
- openmmxml
- qmap
- qid
- qid2
- PrintOptionDict
- verbose_options

8.15.1 Detailed Description

Force field class.

This class contains all methods for force field manipulation. To create an instance of this class, an input file is required containing the list of force field file names. Everything else inside this class pertaining to force field generation is self-contained.

For details on force field parsing, see the detailed documentation for addff.

Definition at line 196 of file forcefield.py.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 def forcebalance.forcefield.FF.__init__ (*self, options, verbose* = `True`)

Instantiation of force field class.

Many variables here are initialized to zero, but they are filled out by methods like addff, rsmake, and mktransmat.

Definition at line 204 of file forcefield.py.

Here is the call graph for this function:

8.15.3 Member Function Documentation

8.15.3.1 def forcebalance.forcefield.FF.__eq__ (*self, other*)

Definition at line 1145 of file forcefield.py.

8.15.3.2 def forcebalance.forcefield.FF.addff (*self, ffname*)

Parse a force field file and add it to the class.

```
First, figure out the type of force field file.  This is done
either by explicitly specifying the type using for example,
<tt> ffname force_field.xml:openmm </tt> or we figure it out
by looking at the file extension.

Next, parse the file.  Currently we support two classes of
files - text and XML.  The two types are treated very
differently; for XML we use the parsers in libxml (via the
python lxml module), and for text files we have our own
in-house parsing class.  Within text files, there is also a
specialized GROMACS and TINKER parser as well as a generic
text parser.

The job of the parser is to determine the following things:
1) Read the user-specified selection of parameters being fitted
2) Build a mapping (dictionary) of <tt> parameter identifier -> index in parameter vector </tt>
3) Build a list of physical parameter values
4) Figure out where to replace the parameter values in the force field file when the values are changed
5) Figure out which parameters need to be repeated or sign-flipped

Generally speaking, each parameter value in the force field
file has a <tt> unique parameter identifier <tt>.  The
identifier consists of three parts - the interaction type, the
parameter subtype (within that interaction type), and the
atoms involved.

--- If XML: ---

The force field file is read in using the lxml Python module.  Specify
which parameter you want to fit using by adding a 'parameterize' element
to the end of the force field XML file, like so.
```

```
<AmoebaVdwForce type="BUFFERED-14-7">
   <Vdw class="74" sigma="0.2655" epsilon="0.056484" reduction="0.910" parameterize="sigma, epsilon,
     reduction" />
```

In this example, the parameter identifier would look like `Vdw/74/epsilon` .

— If GROMACS (.itp) or TINKER (.prm) : —

Follow the rules in the ITP_Reader or Tinker_Reader derived class. Read the documentation in the class documentation or the 'feed' method to learn more. In all cases the parameter is tagged using `# PARM 3` (where # denotes a comment, the word PARM stays the same, and 3 is the field number starting from zero.)

— If normal text : —

The parameter identifier is simply built using the file name, line number, and field. Thus, the identifier is unique but completely noninformative (which is not ideal for our purposes, but it works.)

— Endif —

Warning

My program currently assumes that we are only using one MM program per job. If we use CHARMM and GROMACS to perform simulations as part of the same TARGET, we will get messed up. Maybe this needs to be fixed in the future, with program prefixes to parameters like C_ , G_ .. or simply unit conversions, you get the idea.
I don't think the multiplier actually works for analytic derivatives unless the interaction calculator knows the multiplier as well. I'm sure I can make this work in the future if necessary.

Parameters

in	*ffname*	Name of the force field file

Definition at line 395 of file forcefield.py.

Here is the call graph for this function:

8.15.3.3 def forcebalance.forcefield.FF.addff_txt (*self, ffname, fftype*)

Parse a text force field and create several important instance variables.

Each line is processed using the 'feed' method as implemented in the reader class. This essentially allows us to create the correct parameter identifier (pid), because the pid comes from more than the current line, it also depends on the section that we're in.

When 'PARM' or 'RPT' is encountered, we do several things:

- Build the parameter identifier and insert it into the map
- Point to the file name, line number, and field where the parameter may be modified

Additionally, when 'PARM' is encountered:

- Store the physical parameter value (this is permanent; it's the original value)
- Increment the total number of parameters

When 'RPT' is encountered we don't expand the parameter vector because this parameter is a copy of an existing one. If the parameter identifier is preceded by MINUS_, we chop off the prefix but remember that the sign needs to be flipped.

Definition at line 465 of file forcefield.py.

Here is the call graph for this function:

8.15.3.4 def forcebalance.forcefield.FF.addff_xml (*self, ffname*)

Parse an XML force field file and create important instance variables.

```
This was modeled after addff_txt, but XML and text files are
fundamentally different, necessitating two different methods.

We begin with an _ElementTree object.  We search through the tree
for the 'parameterize' and 'parameter_repeat' keywords.  Each time
the keyword is encountered, we do the same four actions that
I describe in addff_txt.

It's hard to specify precisely the location in an XML file to
change a force field parameter.  I can create a list of tree
elements (essentially pointers to elements within a tree), but
this method breaks down when I copy the tree because I have no
way to refer to the copied tree elements.  Fortunately, lxml
gives me a way to represent a tree using a flat list, and my
XML file 'locations' are represented using the positions in
the list.
```

Warning

The sign-flip hasn't been implemented yet. This shouldn't matter unless your calculation contains repeated parameters with opposite sign.

Definition at line 573 of file forcefield.py.

Here is the call graph for this function:

8.15.3.5 def forcebalance.forcefield.FF.assign_field (*self, idx, fnm, ln, pfld, mult, cmd* = `None`)

Record the locations of a parameter in a txt file; [[file name, line number, field number, and multiplier]].

```
Note that parameters can have multiple locations because of the repetition functionality.
```

Parameters

in	*idx*	The index of the parameter.
in	*fnm*	The file name of the parameter field.
in	*ln*	The line number within the file (or the node index in the flattened xml)
in	*pfld*	The field within the line (or the name of the attribute in the xml)
in	*mult*	The multiplier (this is usually 1.0)

Definition at line 1139 of file forcefield.py.

Here is the call graph for this function:

8.15.3.6 def forcebalance.forcefield.FF.assign_p0 (*self, idx, val*)

Assign physical parameter values to the 'pvals0' array.

Parameters

in	*idx*	The index to which we assign the parameter value.
in	*val*	The parameter value to be inserted.

Definition at line 1121 of file forcefield.py.

Here is the call graph for this function:

8.15.3.7 def forcebalance.forcefield.FF.create_mvals (*self, pvals*)

Converts physical to mathematical parameters.

```
We create the inverse transformation matrix using SVD.
```

Parameters

in	*pvals*	The physical parameters

Returns

mvals The mathematical parameters

Definition at line 854 of file forcefield.py.

Here is the call graph for this function:

8.15.3.8 def forcebalance.forcefield.FF.create_pvals (*self, mvals*)

Converts mathematical to physical parameters.

```
First, mathematical parameters are rescaled and rotated by
multiplying by the transformation matrix, followed by adding
the original physical parameters.
```

Parameters

in	*mvals*	The mathematical parameters

Returns

pvals The physical parameters

Definition at line 819 of file forcefield.py.

Here is the call graph for this function:

8.15.3.9 def forcebalance.forcefield.FF.find_spacings (*self*)

Definition at line 774 of file forcefield.py.

Here is the call graph for this function:

8.15.3.10 def forcebalance.forcefield.FF.list_map (*self*)

Create the plist, which is like a reversed version of the parameter map.

More convenient for printing.

Definition at line 1097 of file forcefield.py.

Here is the call graph for this function:

8.15.3.11 def forcebalance.forcefield.FF.make (*self*, *vals*, *use_pvals* = `False`, *printdir* = `None`, *precision* = `12`)

Create a new force field using provided parameter values.

```
This big kahuna does a number of things:
1) Creates the physical parameters from the mathematical parameters
2) Creates force fields with physical parameters substituted in
3) Prints the force fields to the specified file.

It does NOT store the mathematical parameters in the class state
(since we can only hold one set of parameters).
```

Parameters

in	*printdir*	The directory that the force fields are printed to; as usual this is relative to the project root directory.
in	*vals*	Input parameters. I previously had an option where it uses stored values in the class state, but I don't think that's a good idea anymore.
in	*use_pvals*	Switch for whether to bypass the coordinate transformation and use physical parameters directly.

Definition at line 621 of file forcefield.py.

Here is the call graph for this function:

8.15.3.12 def forcebalance.forcefield.FF.make_redirect (*self, mvals*)

Definition at line 733 of file forcefield.py.

Here is the call graph for this function:

8.15.3.13 def forcebalance.forcefield.FF.mktransmat (*self*)

Create the transformation matrix to rescale and rotate the mathematical parameters.

```
For point charge parameters, project out perturbations that
change the total charge.

First build these:

'qmap'    : Just a list of parameter indices that point to charges.

'qid'     : For each parameter in the qmap, a list of the affected atoms :)
            A potential target for the molecule-specific thang.

Then make this:

'qtrans2' : A transformation matrix that rotates the charge parameters.
            The first row is all zeros (because it corresponds to increasing the charge on all atoms)
            The other rows correspond to changing one of the parameters and decreasing all of the others
            equally such that the overall charge is preserved.

'qmat2'   : An identity matrix with 'qtrans2' pasted into the right place

'transmat': 'qmat2' with rows and columns scaled using self.rs

'excision': Parameter indices that need to be 'cut out' because they are irrelevant and
            mess with the matrix diagonalization
```

Todo Only project out changes in total charge of a molecule, and perhaps generalize to fragments of molecules or other types of parameters.

The AMOEBA selection of charge depends not only on the atom type, but what that atom is bonded to.

Definition at line 947 of file forcefield.py.

8.15.3.14 def forcebalance.forcefield.FF.print_map (*self*, *vals* = `None`, *precision* = `4`)

Prints out the (physical or mathematical) parameter indices, IDs and values in a visually appealing way.

Definition at line 1109 of file forcefield.py.

Here is the call graph for this function:

8.15.3.15 def forcebalance.forcefield.FF.rsmake (*self*, *printfacs* = `True`)

Create the rescaling factors for the coordinate transformation in parameter space.

```
The proper choice of rescaling factors (read: prior widths in maximum likelihood analysis)
is still a black art.  This is a topic of current research.
```

Todo Pass in rsfactors through the input file

Parameters

in	*printfacs*	List for printing out the resecaling factors

Definition at line 871 of file forcefield.py.

Here is the call graph for this function:

8.15.3.16 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.15.4 Member Data Documentation

8.15.4.1 forcebalance.forcefield.FF.atomnames

A list of atom names (this is new, for ESP fitting)

Definition at line 265 of file forcefield.py.

8.15.4.2 forcebalance.forcefield.FF.excision

Indices to exclude from optimization / Hessian inversion.

Some customized constraints here.

Quadrupoles must be traceless

Definition at line 257 of file forcefield.py.

8.15.4.3 forcebalance.forcefield.FF.FFAtomTypes

WORK IN PROGRESS ## This is a dictionary of {'AtomType':{'Mass' : float, 'Charge' : float, 'ParticleType' : string ('A', 'S', or 'D'), 'AtomicNumber' : int}}.

Definition at line 275 of file forcefield.py.

8.15.4.4 forcebalance.forcefield.FF.ffdata

As these options proliferate, the force field class becomes less standalone.

I need to think of a good solution here... The root directory of the project File names of force fields Directory containing force fields, relative to project directory Priors given by the user :) Whether to constrain the charges. Whether to constrain the charges. Switch for AMOEBA direct or mutual. Switch for rigid water molecules Bypass the transformation and use physical parameters directly The content of all force field files are stored in memory

Definition at line 237 of file forcefield.py.

8.15.4.5 forcebalance.forcefield.FF.ffdata_isxml

Definition at line 238 of file forcefield.py.

8.15.4.6 forcebalance.forcefield.FF.FFMolecules

Definition at line 288 of file forcefield.py.

8.15.4.7 forcebalance.forcefield.FF.linedestroy_save

Destruction dictionary (experimental).

Definition at line 316 of file forcefield.py.

8.15.4.8 forcebalance.forcefield.FF.linedestroy_this

Definition at line 318 of file forcefield.py.

8.15.4.9 forcebalance.forcefield.FF.map

The mapping of interaction type -> parameter number.

Definition at line 240 of file forcefield.py.

8.15.4.10 forcebalance.forcefield.FF.np

The total number of parameters.

Definition at line 259 of file forcefield.py.

8.15.4.11 forcebalance.forcefield.FF.openmmxml

Definition at line 411 of file forcefield.py.

8.15.4.12 forcebalance.forcefield.FF.parmdestroy_save

Definition at line 317 of file forcefield.py.

8.15.4.13 forcebalance.forcefield.FF.parmdestroy_this

Definition at line 319 of file forcefield.py.

8.15.4.14 forcebalance.forcefield.FF.patoms

A listing of parameter number -> atoms involved.

Definition at line 244 of file forcefield.py.

8.15.4.15 forcebalance.forcefield.FF.pfields

A list where pfields[pnum] = ['file',line,field,mult,cmd], basically a new way to modify force field files; when we modify the force field file, we go to the specific line/field in a given file and change the number.

Definition at line 249 of file forcefield.py.

8.15.4.16 forcebalance.forcefield.FF.plist

The listing of parameter number -> interaction types.

Definition at line 242 of file forcefield.py.

8.15.4.17 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.15.4.18 forcebalance.forcefield.FF.pvals0

Initial value of physical parameters.

Definition at line 261 of file forcefield.py.

8.15.4.19 forcebalance.forcefield.FF.qid

Definition at line 949 of file forcefield.py.

8.15.4.20 forcebalance.forcefield.FF.qid2

Definition at line 950 of file forcefield.py.

8.15.4.21 forcebalance.forcefield.FF.qmap

Definition at line 948 of file forcefield.py.

8.15.4.22 forcebalance.forcefield.FF.Readers

A dictionary of force field reader classes.

Definition at line 263 of file forcefield.py.

8.15.4.23 forcebalance.forcefield.FF.redirect

Creates plist from map.

Prints the plist to screen. Make the rescaling factors. Make the transformation matrix. Redirection dictionary (experi-

mental).

Definition at line 314 of file forcefield.py.

8.15.4.24 forcebalance.forcefield.FF.rs

List of rescaling factors.

Takes the dictionary 'BONDS':{3:'B', 4:'K'}, 'VDW':{4:'S', 5:'T'}, and turns it into a list of term types ['BONDSB','BOND-SK','VDWS','VDWT'].

The array of rescaling factors

Definition at line 251 of file forcefield.py.

8.15.4.25 forcebalance.forcefield.FF.tinkerprm

Definition at line 404 of file forcefield.py.

8.15.4.26 forcebalance.forcefield.FF.tm

The transformation matrix for mathematical -> physical parameters.

Definition at line 253 of file forcefield.py.

8.15.4.27 forcebalance.forcefield.FF.tml

The transpose of the transformation matrix.

Definition at line 255 of file forcefield.py.

8.15.4.28 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

The documentation for this class was generated from the following file:

- forcefield.py

8.16 forcebalance.output.ForceBalanceLogger Class Reference

This logger starts out with a default handler that writes to stdout addHandler removes this default the first time another handler is added.

Inheritance diagram for forcebalance.output.ForceBalanceLogger:

Collaboration diagram for forcebalance.output.ForceBalanceLogger:

Public Member Functions

- def __init__
- def addHandler
- def removeHandler

Public Attributes

- defaultHandler

8.16.1 Detailed Description

This logger starts out with a default handler that writes to stdout addHandler removes this default the first time another handler is added.

This is used for forcebalance package level logging, where a logger should always be present unless otherwise specified. To silence, add a NullHandler We also by default set the log level to INFO (logging.Logger starts at WARNING)

Definition at line 10 of file output.py.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 def forcebalance.output.ForceBalanceLogger.__init__ (*self, name*)

Definition at line 11 of file output.py.

8.16.3 Member Function Documentation

8.16.3.1 def forcebalance.output.ForceBalanceLogger.addHandler (*self, hdlr*)

Definition at line 17 of file output.py.

Here is the call graph for this function:

8.16.3.2 def forcebalance.output.ForceBalanceLogger.removeHandler (*self, hdlr*)

Definition at line 23 of file output.py.

Here is the call graph for this function:

8.16.4 Member Data Documentation

8.16.4.1 forcebalance.output.ForceBalanceLogger.defaultHandler

Definition at line 13 of file output.py.

The documentation for this class was generated from the following file:

- output.py

8.17 forcebalance.amberio.FrcMod_Reader Class Reference

Finite state machine for parsing FrcMod force field file.

Inheritance diagram for forcebalance.amberio.FrcMod_Reader:

Collaboration diagram for forcebalance.amberio.FrcMod_Reader:

Public Member Functions

- def __init__
- def Split
- def Whites
- def feed
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- pdict

  The parameter dictionary (defined in this file)
- atom

  The atom numbers in the interaction (stored in the parser)
- dihe

  Whether we're inside the dihedral section.
- adict

  The frcmod file never has any atoms in it.
- itype
- suffix
- ln
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.17.1 Detailed Description

Finite state machine for parsing FrcMod force field file.

Definition at line 99 of file amberio.py.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 def forcebalance.amberio.FrcMod_Reader.__init__ (*self, fnm*)

Definition at line 101 of file amberio.py.

8.17.3 Member Function Documentation

8.17.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.17.3.2 def forcebalance.amberio.FrcMod_Reader.feed (*self, line*)

Definition at line 119 of file amberio.py.

8.17.3.3 def forcebalance.amberio.FrcMod_Reader.Split (*self, line*)

Definition at line 113 of file amberio.py.

Here is the call graph for this function:

8.17.3.4 def forcebalance.amberio.FrcMod_Reader.Whites (*self, line*)

Definition at line 116 of file amberio.py.

Here is the call graph for this function:

8.17.4 Member Data Documentation

8.17.4.1 forcebalance.amberio.FrcMod_Reader.adict

The frcmod file never has any atoms in it.

Definition at line 111 of file amberio.py.

8.17.4.2 forcebalance.amberio.FrcMod_Reader.atom

The atom numbers in the interaction (stored in the parser)

Definition at line 107 of file amberio.py.

8.17.4.3 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.17.4.4 forcebalance.amberio.FrcMod_Reader.dihe

Whether we're inside the dihedral section.

Definition at line 109 of file amberio.py.

8.17.4.5 forcebalance.amberio.FrcMod_Reader.itype

Definition at line 130 of file amberio.py.

8.17.4.6 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.17.4.7 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.17.4.8 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.17.4.9 forcebalance.amberio.FrcMod_Reader.pdict

The parameter dictionary (defined in this file)

Definition at line 105 of file amberio.py.

8.17.4.10 forcebalance.amberio.FrcMod_Reader.suffix

Definition at line 165 of file amberio.py.

The documentation for this class was generated from the following file:

- amberio.py

8.18 forcebalance.psi4io.GBS_Reader Class Reference

Interaction type -> Parameter Dictionary.

Inheritance diagram for forcebalance.psi4io.GBS_Reader:

Collaboration diagram for forcebalance.psi4io.GBS_Reader:

Public Member Functions

- def __init__
- def build_pid
- def feed

  Feed in a line.

- def Split
- def Whites
- def feed

Public Attributes

- element
- amom
- last_amom
- basis_number
- contraction_number
- adict
- isdata
- destroy
- ln
- itype
- suffix
- pdict
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.18.1 Detailed Description

Interaction type -> Parameter Dictionary.

pdict = {'Exponent':{0:'A', 1:'C'}, 'BASSP' :{0:'A', 1:'B', 2:'C'} } Finite state machine for parsing basis set files.

Definition at line 35 of file psi4io.py.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 def forcebalance.psi4io.GBS_Reader.__init__ (*self, fnm* = `None`)

Definition at line 37 of file psi4io.py.

8.18.3 Member Function Documentation

8.18.3.1 def forcebalance.psi4io.GBS_Reader.build_pid (*self, pfld*)

Definition at line 48 of file psi4io.py.

Here is the call graph for this function:

8.18.3.2 def forcebalance.psi4io.GBS_Reader.feed (*self, line, linindep* = `False`)

Feed in a line.

Parameters

in	*line*	The line of data

Definition at line 61 of file psi4io.py.

Here is the call graph for this function:

8.18.3.3 def forcebalance.BaseReader.feed (*self, line*) [inherited]

Definition at line 88 of file __init__.py.

Here is the call graph for this function:

8.18.3.4 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.18.3.5 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.18.4 Member Data Documentation

8.18.4.1 forcebalance.psi4io.GBS_Reader.adict

Definition at line 44 of file psi4io.py.

8.18.4.2 forcebalance.psi4io.GBS_Reader.amom

Definition at line 40 of file psi4io.py.

8.18.4.3 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.18.4.4 forcebalance.psi4io.GBS_Reader.basis_number

Definition at line 42 of file psi4io.py.

8.18.4.5 forcebalance.psi4io.GBS_Reader.contraction_number

Definition at line 43 of file psi4io.py.

8.18.4.6 forcebalance.psi4io.GBS_Reader.destroy

Definition at line 46 of file psi4io.py.

8.18.4.7 forcebalance.psi4io.GBS_Reader.element

Definition at line 39 of file psi4io.py.

8.18.4.8 forcebalance.psi4io.GBS_Reader.isdata

Definition at line 45 of file psi4io.py.

8.18.4.9 forcebalance.BaseReader.itype [inherited]

Definition at line 68 of file __init__.py.

8.18.4.10 forcebalance.psi4io.GBS_Reader.last_amom

Definition at line 41 of file psi4io.py.

8.18.4.11 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.18.4.12 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.18.4.13 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.18.4.14 forcebalance.BaseReader.pdict [inherited]

Definition at line 70 of file __init__.py.

8.18.4.15 forcebalance.BaseReader.suffix [inherited]

Definition at line 69 of file __init__.py.

The documentation for this class was generated from the following file:

- psi4io.py

8.19 forcebalance.custom_io.Gen_Reader Class Reference

Finite state machine for parsing custom GROMACS force field files.

Inheritance diagram for forcebalance.custom_io.Gen_Reader:

Collaboration diagram for forcebalance.custom_io.Gen_Reader:

Public Member Functions

- def __init__
- def feed

  Feed in a line.
- def Split
- def Whites
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- sec

  The current section that we're in.
- pdict

  The parameter dictionary (defined in this file)
- itype
- suffix
- ln
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.19.1 Detailed Description

Finite state machine for parsing custom GROMACS force field files.

This class is instantiated when we begin to read in a file. The feed(line) method updates the state of the machine, giving it information like the residue we're currently on, the nonbonded interaction type, and the section that we're in. Using this information we can look up the interaction type and parameter type for building the parameter ID.

Definition at line 41 of file custom_io.py.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 def forcebalance.custom_io.Gen_Reader.__init__ (*self, fnm*)

Definition at line 43 of file custom_io.py.

8.19.3 Member Function Documentation

8.19.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.19.3.2 def forcebalance.custom_io.Gen_Reader.feed (*self, line*)

Feed in a line.

Parameters

in	*line*	The line of data

Definition at line 57 of file custom_io.py.

8.19.3.3 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.19.3.4 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.19.4 Member Data Documentation

8.19.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.19.4.2 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.19.4.3 forcebalance.custom_io.Gen_Reader.itype

Definition at line 60 of file custom_io.py.

8.19.4.4 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.19.4.5 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.19.4.6 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.19.4.7 forcebalance.custom_io.Gen_Reader.pdict

The parameter dictionary (defined in this file)

Definition at line 49 of file custom_io.py.

8.19.4.8 forcebalance.custom_io.Gen_Reader.sec

The current section that we're in.

Definition at line 47 of file custom_io.py.

8.19.4.9 forcebalance.custom_io.Gen_Reader.suffix

Definition at line 79 of file custom_io.py.

The documentation for this class was generated from the following file:

- custom_io.py

8.20 forcebalance.psi4io.Grid_Reader Class Reference

Finite state machine for parsing DVR grid files.

Inheritance diagram for forcebalance.psi4io.Grid_Reader:

Collaboration diagram for forcebalance.psi4io.Grid_Reader:

Public Member Functions

- def __init__
- def build_pid
- def feed

  Feed in a line.
- def Split
- def Whites
- def feed

Public Attributes

- element
- point
- radii
- isdata
- ln
- itype
- suffix
- pdict
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.20.1 Detailed Description

Finite state machine for parsing DVR grid files.

Definition at line 247 of file psi4io.py.

8.20.2 Constructor & Destructor Documentation

8.20.2.1 def forcebalance.psi4io.Grid_Reader.__init__ (*self, fnm* = `None`)

Definition at line 249 of file psi4io.py.

8.20.3 Member Function Documentation

8.20.3.1 def forcebalance.psi4io.Grid_Reader.build_pid (*self, pfld*)

Definition at line 255 of file psi4io.py.

Here is the call graph for this function:

8.20.3.2 def forcebalance.BaseReader.feed (*self, line*) `[inherited]`

Definition at line 88 of file __init__.py.

Here is the call graph for this function:

8.20.3.3 def forcebalance.psi4io.Grid_Reader.feed (*self, line, linindep* = `False`)

Feed in a line.

Parameters

in	*line*	The line of data

Definition at line 270 of file psi4io.py.

Here is the call graph for this function:

8.20.3.4 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.20.3.5 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.20.4 Member Data Documentation

8.20.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.20.4.2 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.20.4.3 forcebalance.psi4io.Grid_Reader.element

Definition at line 251 of file psi4io.py.

8.20.4.4 forcebalance.psi4io.Grid_Reader.isdata

Definition at line 281 of file psi4io.py.

8.20.4.5 forcebalance.BaseReader.itype [inherited]

Definition at line 68 of file __init__.py.

8.20.4.6 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.20.4.7 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.20.4.8 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.20.4.9 forcebalance.BaseReader.pdict [inherited]

Definition at line 70 of file __init__.py.

8.20.4.10 forcebalance.psi4io.Grid_Reader.point

Definition at line 252 of file psi4io.py.

8.20.4.11 forcebalance.psi4io.Grid_Reader.radii

Definition at line 253 of file psi4io.py.

8.20.4.12 forcebalance.BaseReader.suffix [inherited]

Definition at line 69 of file __init__.py.

The documentation for this class was generated from the following file:

- psi4io.py

8.21 forcebalance.interaction.Interaction Class Reference

Subclass of Target for fitting force fields to interaction energies.

Inheritance diagram for forcebalance.interaction.Interaction:

Collaboration diagram for forcebalance.interaction.Interaction:

Public Member Functions

- def __init__
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def prepare_temp_directory

  Prepare the temporary directory, by default does nothing.
- def indicate
- def get

  Evaluate objective function.
- def get_X

  Computes the objective function contribution without any parametric derivatives.

- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- select1

  Number of snapshots.
- select2

  Set fragment 2.
- eqm

  Set upper cutoff energy.
- label

  Snapshot label, useful for graphing.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)
- e_err_pct
- ns

  Read in the trajectory file.
- traj
- divisor

  Read in the reference data.
- prefactor
- weight
- emm
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

Need the forcefield (here for now)

- xct

  Counts how often the objective function was computed.

- gct

  Counts how often the gradient was computed.

- hct

  Counts how often the Hessian was computed.

- PrintOptionDict
- verbose_options

8.21.1 Detailed Description

Subclass of Target for fitting force fields to interaction energies.

Currently TINKER is supported.

We introduce the following concepts:

- The number of snapshots
- The reference interaction energies and the file they belong in (qdata.txt)

This subclass contains the 'get' method for building the objective function from any simulation software (a driver to run the program and read output is still required).

Definition at line 35 of file interaction.py.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 def forcebalance.interaction.Interaction.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 38 of file interaction.py.

Here is the call graph for this function:

8.21.3 Member Function Documentation

8.21.3.1 def forcebalance.interaction.Interaction.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Evaluate objective function.

Definition at line 163 of file interaction.py.

8.21.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.21.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.21.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.21.3.5 def forcebalance.interaction.Interaction.indicate (*self*)

Definition at line 147 of file interaction.py.

Here is the call graph for this function:

8.21.3.6 def forcebalance.target.Target.link_from_tempdir (*self*, *absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.21.3.7 def forcebalance.interaction.Interaction.prepare_temp_directory (*self*, *options*, *tgt_opts*)

Prepare the temporary directory, by default does nothing.

Definition at line 144 of file interaction.py.

8.21.3.8 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.21.3.9 def forcebalance.interaction.Interaction.read_reference_data (*self*)

Read the reference ab initio data from a file such as qdata.txt.

After reading in the information from qdata.txt, it is converted into the GROMACS/OpenMM units (kJ/mol for energy, kJ/mol/nm force).

Definition at line 125 of file interaction.py.

Here is the call graph for this function:

8.21.3.10 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.21.3.11 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.21.3.12 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.21.3.13 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.21.3.14 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.21.3.15 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.21.4 Member Data Documentation

8.21.4.1 forcebalance.interaction.Interaction.divisor

Read in the reference data.

Prepare the temporary directory

Definition at line 96 of file interaction.py.

8.21.4.2 forcebalance.interaction.Interaction.e_err

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 78 of file interaction.py.

8.21.4.3 forcebalance.interaction.Interaction.e_err_pct

Definition at line 79 of file interaction.py.

8.21.4.4 forcebalance.interaction.Interaction.emm

Definition at line 200 of file interaction.py.

8.21.4.5 forcebalance.interaction.Interaction.eqm

Set upper cutoff energy.

Reference (QM) interaction energies

Definition at line 72 of file interaction.py.

8.21.4.6 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.21.4.7 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.21.4.8 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.21.4.9 forcebalance.interaction.Interaction.label

Snapshot label, useful for graphing.

Definition at line 74 of file interaction.py.

8.21.4.10 forcebalance.interaction.Interaction.ns

Read in the trajectory file.

Definition at line 81 of file interaction.py.

8.21.4.11 forcebalance.interaction.Interaction.objective

Definition at line 201 of file interaction.py.

8.21.4.12 forcebalance.interaction.Interaction.prefactor

Definition at line 114 of file interaction.py.

8.21.4.13 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.21.4.14 forcebalance.interaction.Interaction.qfnm

The qdata.txt file that contains the QM energies and forces.

Definition at line 76 of file interaction.py.

8.21.4.15 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.21.4.16 forcebalance.interaction.Interaction.select1

Number of snapshots.

Do we call Q-Chem for dielectric energies? (Currently needs to be fixed) Do we put the reference energy into the denominator? Do we put the reference energy into the denominator? What is the energy denominator? Set fragment 1

Definition at line 60 of file interaction.py.

8.21.4.17 forcebalance.interaction.Interaction.select2

Set fragment 2.

Definition at line 65 of file interaction.py.

8.21.4.18 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.21.4.19 forcebalance.interaction.Interaction.traj

Definition at line 82 of file interaction.py.

8.21.4.20 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.21.4.21 forcebalance.interaction.Interaction.weight

Definition at line 167 of file interaction.py.

8.21.4.22 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- interaction.py

8.22 forcebalance.gmxio.Interaction_GMX Class Reference

Subclass of Interaction for interaction energy matching using GROMACS.

Inheritance diagram for forcebalance.gmxio.Interaction_GMX:

Collaboration diagram for forcebalance.gmxio.Interaction_GMX:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def interaction_driver

  Computes the energy and force using GROMACS for a single snapshot.
- def interaction_driver_all

  Computes the energy and force using GROMACS for a trajectory.
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def indicate
- def get

  Evaluate objective function.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

Computes the objective function contribution and its gradient / Hessian.

- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory.
- topfnm
- Dielectric
- Diel_B
- select1

  Number of snapshots.
- select2

  Set fragment 2.
- eqm

  Set upper cutoff energy.
- label

  Snapshot label, useful for graphing.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)
- e_err_pct
- ns

  Read in the trajectory file.
- traj
- divisor

  Read in the reference data.
- prefactor
- weight
- emm
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.

- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.22.1 Detailed Description

Subclass of Interaction for interaction energy matching using GROMACS.

Definition at line 624 of file gmxio.py.

8.22.2 Constructor & Destructor Documentation

8.22.2.1 def forcebalance.gmxio.Interaction_GMX.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 626 of file gmxio.py.

Here is the call graph for this function:

8.22.3 Member Function Documentation

8.22.3.1 def forcebalance.interaction.Interaction.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Evaluate objective function.

Definition at line 163 of file interaction.py.

8.22.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.22.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.22.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.22.3.5 def forcebalance.interaction.Interaction.indicate (*self*) [inherited]

Definition at line 147 of file interaction.py.

Here is the call graph for this function:

8.22.3.6 def forcebalance.gmxio.Interaction_GMX.interaction_driver (*self, shot*)

Computes the energy and force using GROMACS for a single snapshot.

This does not require GROMACS-X2.

Definition at line 657 of file gmxio.py.

8.22.3.7 def forcebalance.gmxio.Interaction_GMX.interaction_driver_all (*self, dielectric* = `False`)

Computes the energy and force using GROMACS for a trajectory.

This does not require GROMACS-X2.

Definition at line 662 of file gmxio.py.

Here is the call graph for this function:

8.22.3.8 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.22.3.9 def forcebalance.gmxio.Interaction_GMX.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 634 of file gmxio.py.

Here is the call graph for this function:

8.22.3.10 def forcebalance.target.Target.printcool_table (*self, data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.22.3.11 def forcebalance.interaction.Interaction.read_reference_data (*self*) [inherited]

Read the reference ab initio data from a file such as qdata.txt.

After reading in the information from qdata.txt, it is converted into the GROMACS/OpenMM units (kJ/mol for energy, kJ/mol/nm force).

Definition at line 125 of file interaction.py.

Here is the call graph for this function:

8.22.3.12 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.22.3.13 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.22.3.14 def forcebalance.target.Target.sget (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.22.3.15 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.22.3.16 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.22.3.17 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.22.4 Member Data Documentation

8.22.4.1 forcebalance.gmxio.Interaction_GMX.Diel_B

Definition at line 702 of file gmxio.py.

8.22.4.2 forcebalance.gmxio.Interaction_GMX.Dielectric

Definition at line 631 of file gmxio.py.

8.22.4.3 forcebalance.interaction.Interaction.divisor [inherited]

Read in the reference data.

Prepare the temporary directory

Definition at line 96 of file interaction.py.

8.22.4.4 forcebalance.interaction.Interaction.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 78 of file interaction.py.

8.22.4.5 forcebalance.interaction.Interaction.e_err_pct [inherited]

Definition at line 79 of file interaction.py.

8.22.4.6 forcebalance.interaction.Interaction.emm [inherited]

Definition at line 200 of file interaction.py.

8.22.4.7 forcebalance.interaction.Interaction.eqm [inherited]

Set upper cutoff energy.

Reference (QM) interaction energies

Definition at line 72 of file interaction.py.

8.22.4.8 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.22.4.9 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.22.4.10 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.22.4.11 forcebalance.interaction.Interaction.label [inherited]

Snapshot label, useful for graphing.

Definition at line 74 of file interaction.py.

8.22.4.12 forcebalance.interaction.Interaction.ns [inherited]

Read in the trajectory file.

Definition at line 81 of file interaction.py.

8.22.4.13 forcebalance.interaction.Interaction.objective [inherited]

Definition at line 201 of file interaction.py.

8.22.4.14 forcebalance.interaction.Interaction.prefactor [inherited]

Definition at line 114 of file interaction.py.

8.22.4.15 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.22.4.16 forcebalance.interaction.Interaction.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 76 of file interaction.py.

8.22.4.17 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.22.4.18 forcebalance.interaction.Interaction.select1 [inherited]

Number of snapshots.

Do we call Q-Chem for dielectric energies? (Currently needs to be fixed) Do we put the reference energy into the denominator? Do we put the reference energy into the denominator? What is the energy denominator? Set fragment 1

Definition at line 60 of file interaction.py.

8.22.4.19 forcebalance.interaction.Interaction.select2 [inherited]

Set fragment 2.

Definition at line 65 of file interaction.py.

8.22.4.20 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.22.4.21 forcebalance.gmxio.Interaction_GMX.topfnm

Definition at line 629 of file gmxio.py.

8.22.4.22 forcebalance.interaction.Interaction.traj [inherited]

Definition at line 82 of file interaction.py.

8.22.4.23 forcebalance.gmxio.Interaction_GMX.trajfnm

Name of the trajectory.

Definition at line 628 of file gmxio.py.

8.22.4.24 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.22.4.25 forcebalance.interaction.Interaction.weight [inherited]

Definition at line 167 of file interaction.py.

8.22.4.26 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- gmxio.py

8.23 forcebalance.openmmio.Interaction_OpenMM Class Reference

Subclass of Target for interaction matching using OpenMM.

Inheritance diagram for forcebalance.openmmio.Interaction_OpenMM:

Collaboration diagram for forcebalance.openmmio.Interaction_OpenMM:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def energy_driver_all
- def interaction_driver_all
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def indicate
- def get

  Evaluate objective function.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.

- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory file containing snapshots.
- simulations

  Dictionary of simulation objects (dimer, fraga, fragb)
- platform

  Set up three OpenMM System objects.
- select1

  Number of snapshots.
- select2

  Set fragment 2.
- eqm

  Set upper cutoff energy.
- label

  Snapshot label, useful for graphing.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)
- e_err_pct
- ns

  Read in the trajectory file.
- traj
- divisor

  Read in the reference data.
- prefactor
- weight
- emm
- objective
- tempdir

  Root directory of the whole project.
- rundir

The directory in which the simulation is running - this can be updated.

- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.23.1 Detailed Description

Subclass of Target for interaction matching using OpenMM.

Definition at line 536 of file openmmio.py.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 def forcebalance.openmmio.Interaction_OpenMM.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 539 of file openmmio.py.

8.23.3 Member Function Documentation

8.23.3.1 def forcebalance.openmmio.Interaction_OpenMM.energy_driver_all (*self, mode*)

Definition at line 600 of file openmmio.py.

Here is the call graph for this function:

8.23.3.2 def forcebalance.interaction.Interaction.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Evaluate objective function.

Definition at line 163 of file interaction.py.

8.23.3.3 def forcebalance.target.Target.get_G (*self, mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.23.3.4 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.23.3.5 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.23.3.6 def forcebalance.interaction.Interaction.indicate (*self*) [inherited]

Definition at line 147 of file interaction.py.

Here is the call graph for this function:

8.23.3.7 def forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all (*self*, *dielectric* = `False`)

Definition at line 653 of file openmmio.py.

8.23.3.8 def forcebalance.target.Target.link_from_tempdir (*self*, *absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.23.3.9 def forcebalance.openmmio.Interaction_OpenMM.prepare_temp_directory (*self*, *options*, *tgt_opts*)

Definition at line 547 of file openmmio.py.

8.23.3.10 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.23.3.11 def forcebalance.interaction.Interaction.read_reference_data (*self*) [inherited]

Read the reference ab initio data from a file such as qdata.txt.

After reading in the information from qdata.txt, it is converted into the GROMACS/OpenMM units (kJ/mol for energy, kJ/mol/nm force).

Definition at line 125 of file interaction.py.

Here is the call graph for this function:

8.23.3.12 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.23.3.13 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.23.3.14 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.23.3.15 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.23.3.16 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.23.3.17 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.23.4 Member Data Documentation

8.23.4.1 forcebalance.interaction.Interaction.divisor [inherited]

Read in the reference data.

Prepare the temporary directory

Definition at line 96 of file interaction.py.

8.23.4.2 forcebalance.interaction.Interaction.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 78 of file interaction.py.

8.23.4.3 forcebalance.interaction.Interaction.e_err_pct [inherited]

Definition at line 79 of file interaction.py.

8.23.4.4 forcebalance.interaction.Interaction.emm [inherited]

Definition at line 200 of file interaction.py.

8.23.4.5 forcebalance.interaction.Interaction.eqm [inherited]

Set upper cutoff energy.

Reference (QM) interaction energies

Definition at line 72 of file interaction.py.

8.23.4.6 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.23.4.7 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.23.4.8 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.23.4.9 forcebalance.interaction.Interaction.label [inherited]

Snapshot label, useful for graphing.

Definition at line 74 of file interaction.py.

8.23.4.10 forcebalance.interaction.Interaction.ns [inherited]

Read in the trajectory file.

Definition at line 81 of file interaction.py.

8.23.4.11 forcebalance.interaction.Interaction.objective [inherited]

Definition at line 201 of file interaction.py.

8.23.4.12 forcebalance.openmmio.Interaction_OpenMM.platform

Set up three OpenMM System objects.

Set the device to the environment variable or zero otherwise.

TODO: The following code should not be repeated everywhere. Set the simulation platform

Definition at line 565 of file openmmio.py.

8.23.4.13 forcebalance.interaction.Interaction.prefactor [inherited]

Definition at line 114 of file interaction.py.

8.23.4.14 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.23.4.15 forcebalance.interaction.Interaction.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 76 of file interaction.py.

8.23.4.16 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.23.4.17 forcebalance.interaction.Interaction.select1 [inherited]

Number of snapshots.

Do we call Q-Chem for dielectric energies? (Currently needs to be fixed) Do we put the reference energy into the denominator? Do we put the reference energy into the denominator? What is the energy denominator? Set fragment 1

Definition at line 60 of file interaction.py.

8.23.4.18 forcebalance.interaction.Interaction.select2 [inherited]

Set fragment 2.

Definition at line 65 of file interaction.py.

8.23.4.19 forcebalance.openmmio.Interaction_OpenMM.simulations

Dictionary of simulation objects (dimer, fraga, fragb)

Definition at line 543 of file openmmio.py.

8.23.4.20 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.23.4.21 forcebalance.interaction.Interaction.traj [inherited]

Definition at line 82 of file interaction.py.

8.23.4.22 forcebalance.openmmio.Interaction_OpenMM.trajfnm

Name of the trajectory file containing snapshots.

Definition at line 541 of file openmmio.py.

8.23.4.23 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.23.4.24 forcebalance.interaction.Interaction.weight [inherited]

Definition at line 167 of file interaction.py.

8.23.4.25 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- openmmio.py

8.24 forcebalance.tinkerio.Interaction_TINKER Class Reference

Subclass of Target for interaction matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.Interaction_TINKER:

Collaboration diagram for forcebalance.tinkerio.Interaction_TINKER:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def energy_driver_all
- def interaction_driver_all
- def read_reference_data

  Read the reference ab initio data from a file such as qdata.txt.
- def indicate
- def get

  Evaluate objective function.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.

- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- trajfnm

  Name of the trajectory.
- select1

  Number of snapshots.
- select2

  Set fragment 2.
- eqm

  Set upper cutoff energy.
- label

  Snapshot label, useful for graphing.
- qfnm

  The qdata.txt file that contains the QM energies and forces.
- e_err

  Qualitative Indicator: average energy error (in kJ/mol)
- e_err_pct
- ns

  Read in the trajectory file.
- traj
- divisor

  Read in the reference data.
- prefactor
- weight
- emm
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

Counts how often the objective function was computed.

- gct

  Counts how often the gradient was computed.

- hct

  Counts how often the Hessian was computed.

- PrintOptionDict
- verbose_options

8.24.1 Detailed Description

Subclass of Target for interaction matching using TINKER.

Definition at line 585 of file tinkerio.py.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 def forcebalance.tinkerio.Interaction_TINKER.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 588 of file tinkerio.py.

Here is the call graph for this function:

8.24.3 Member Function Documentation

8.24.3.1 def forcebalance.tinkerio.Interaction_TINKER.energy_driver_all (*self, select* = `None`)

Definition at line 601 of file tinkerio.py.

Here is the call graph for this function:

8.24.3.2 def forcebalance.interaction.Interaction.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Evaluate objective function.

Definition at line 163 of file interaction.py.

8.24.3.3 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.24.3.4 def forcebalance.target.Target.get_H (*self*, *mvals* = None) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.24.3.5 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.24.3.6 def forcebalance.interaction.Interaction.indicate (*self*) [inherited]

Definition at line 147 of file interaction.py.

Here is the call graph for this function:

8.24.3.7 def forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all (*self*, *dielectric* = `False`)

Definition at line 618 of file tinkerio.py.

8.24.3.8 def forcebalance.target.Target.link_from_tempdir (*self*, *absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.24.3.9 def forcebalance.tinkerio.Interaction_TINKER.prepare_temp_directory (*self*, *options*, *tgt_opts*)

Definition at line 593 of file tinkerio.py.

8.24.3.10 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.24.3.11 def forcebalance.interaction.Interaction.read_reference_data (*self*) [inherited]

Read the reference ab initio data from a file such as qdata.txt.

After reading in the information from qdata.txt, it is converted into the GROMACS/OpenMM units (kJ/mol for energy, kJ/mol/nm force).

Definition at line 125 of file interaction.py.

Here is the call graph for this function:

8.24.3.12 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.24.3.13 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.24.3.14 def forcebalance.target.Target.sget (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.24.3.15 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.24.3.16 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.24.3.17 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.24.4 Member Data Documentation

8.24.4.1 forcebalance.interaction.Interaction.divisor [inherited]

Read in the reference data.

Prepare the temporary directory

Definition at line 96 of file interaction.py.

8.24.4.2 forcebalance.interaction.Interaction.e_err [inherited]

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 78 of file interaction.py.

8.24.4.3 forcebalance.interaction.Interaction.e_err_pct [inherited]

Definition at line 79 of file interaction.py.

8.24.4.4 forcebalance.interaction.Interaction.emm [inherited]

Definition at line 200 of file interaction.py.

8.24.4.5 forcebalance.interaction.Interaction.eqm [inherited]

Set upper cutoff energy.

Reference (QM) interaction energies

Definition at line 72 of file interaction.py.

8.24.4.6 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.24.4.7 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.24.4.8 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.24.4.9 forcebalance.interaction.Interaction.label [inherited]

Snapshot label, useful for graphing.

Definition at line 74 of file interaction.py.

8.24.4.10 forcebalance.interaction.Interaction.ns [inherited]

Read in the trajectory file.

Definition at line 81 of file interaction.py.

8.24.4.11 forcebalance.interaction.Interaction.objective [inherited]

Definition at line 201 of file interaction.py.

8.24.4.12 forcebalance.interaction.Interaction.prefactor [inherited]

Definition at line 114 of file interaction.py.

8.24.4.13 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.24.4.14 forcebalance.interaction.Interaction.qfnm [inherited]

The qdata.txt file that contains the QM energies and forces.

Definition at line 76 of file interaction.py.

8.24.4.15 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.24.4.16 forcebalance.interaction.Interaction.select1 [inherited]

Number of snapshots.

Do we call Q-Chem for dielectric energies? (Currently needs to be fixed) Do we put the reference energy into the denominator? Do we put the reference energy into the denominator? What is the energy denominator? Set fragment 1

Definition at line 60 of file interaction.py.

8.24.4.17 forcebalance.interaction.Interaction.select2 [inherited]

Set fragment 2.

Definition at line 65 of file interaction.py.

8.24.4.18 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.24.4.19 forcebalance.interaction.Interaction.traj [inherited]

Definition at line 82 of file interaction.py.

8.24.4.20 forcebalance.tinkerio.Interaction_TINKER.trajfnm

Name of the trajectory.

Definition at line 590 of file tinkerio.py.

8.24.4.21 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.24.4.22 forcebalance.interaction.Interaction.weight [inherited]

Definition at line 167 of file interaction.py.

8.24.4.23 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- tinkerio.py

8.25 forcebalance.gmxio.ITP_Reader Class Reference

Finite state machine for parsing GROMACS force field files.

Inheritance diagram for forcebalance.gmxio.ITP_Reader:

Collaboration diagram for forcebalance.gmxio.ITP_Reader:

Public Member Functions

- def __init__
- def feed

  Given a line, determine the interaction type and the atoms involved (the suffix).
- def Split

- def Whites
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- sec

  The current section that we're in.
- nbtype

  Nonbonded type.
- mol

  The current molecule (set by the moleculetype keyword)
- pdict

  The parameter dictionary (defined in this file)
- atomnames

  Listing of all atom names in the file, (probably unnecessary)
- atomtypes

  Listing of all atom types in the file, (probably unnecessary)
- atomtype_to_mass

  A dictionary of atomic masses.
- itype
- suffix
- molatom
- ln
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- Molecules
- AtomTypes

8.25.1 Detailed Description

Finite state machine for parsing GROMACS force field files.

```
We open the force field file and read all of its lines.  As we loop
through the force field file, we look for two types of tags: (1) section
markers, in GMX indicated by [ section_name ], which allows us to determine
the section, and (2) parameter tags, indicated by the 'PARM' or 'RPT' keywords.

As we go through the file, we figure out the atoms involved in the interaction
described on each line.

When a 'PARM' keyword is indicated, it is followed by a number which is the field
in the line to be modified, starting with zero.  Based on the field number and the
section name, we can figure out the parameter type.  With the parameter type
and the atoms in hand, we construct a 'parameter identifier' or pid which uniquely
identifies that parameter.  We also store the physical parameter value in an array
called 'pvals0' and the precise location of that parameter (by filename, line number,
and field number) in a list called 'pfields'.

An example: Suppose in 'my_ff.itp' I encounter the following on lines 146 and 147:
```

```
1 [ angletypes ]
2 CA   CB   O   1   109.47  350.00  ; PARM 4 5
```

From reading `[ angletypes ]` I know I'm in the 'angletypes' section.

On the next line, I notice two parameters on fields 4 and 5.

From the atom types, section type and field number I know the parameter IDs are `'ANGLESBCACBO'` and `'ANGLESKCACBO'`.

After building `map={'ANGLESBCACBO':1,'ANGLESKCACBO':2}`, I store the values in an array: `pvals0=array([109.47,350.00])`, and I put the parameter locations in pfields: `pfields=[['my_ff.itp',147,4,1.0],['my_ff.itp',146,5,1.0]]`. The 1.0 is a 'multiplier' and I will explain it below.

Note that in the creation of parameter IDs, we run into the issue that the atoms involved in the interaction may be labeled in reverse order (e.g. `OCACB`). Thus, we store both the normal and the reversed parameter ID in the map.

Parameter repetition and multiplier:

If `'RPT'` is encountered in the line, it is always in the syntax: `'RPT 4 ANGLESBCACAH 5 MINUS_ANGLESKCACAH /RPT'`. In this case, field 4 is replaced by the stored parameter value corresponding to `ANGLESBCACAH` and field 5 is replaced by -1 times the stored value of `ANGLESKCACAH`. Now I just picked this as an example, I don't think people actually want a negative angle force constant .. :) the `MINUS` keyword does come in handy for assigning atomic charges and virtual site positions. In order to achieve this, a multiplier of -1.0 is stored into pfields instead of 1.0.

Todo Note that I can also create the opposite virtual site position by changing the atom labeling, woo!

Definition at line 275 of file gmxio.py.

8.25.2 Constructor & Destructor Documentation

8.25.2.1 def forcebalance.gmxio.ITP_Reader.__init__ (*self, fnm*)

Definition at line 278 of file gmxio.py.

8.25.3 Member Function Documentation

8.25.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.line_num.field_num'

Definition at line 107 of file __init__.py.

8.25.3.2 def forcebalance.gmxio.ITP_Reader.feed (*self, line*)

Given a line, determine the interaction type and the atoms involved (the suffix).

For example, we want

```
H O H 5 1.231258497536e+02 4.269161426840e+02 -1.033397697685e-02 1.304674117410e+04
; PARM 4 5 6 7
```

to give us itype = 'UREY_BRADLEY' and suffix = 'HOH'

If we are in a TypeSection, it returns a list of atom types;

If we are in a TopolSection, it returns a list of atom names.

The section is essentially a case statement that picks out the appropriate interaction type and makes a list of the atoms involved

Note that we can call gmxdump for this as well, but I prefer to read the force field file directly.

ToDo: [atoms] section might need to be more flexible to accommodate optional fields

Definition at line 316 of file gmxio.py.

8.25.3.3 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.25.3.4 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.25.4 Member Data Documentation

8.25.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.25.4.2 forcebalance.gmxio.ITP_Reader.atomnames

Listing of all atom names in the file, (probably unnecessary)

Definition at line 290 of file gmxio.py.

8.25.4.3 forcebalance.gmxio.ITP_Reader.atomtype_to_mass

A dictionary of atomic masses.

Definition at line 294 of file gmxio.py.

8.25.4.4 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.25.4.5 forcebalance.gmxio.ITP_Reader.atomtypes

Listing of all atom types in the file, (probably unnecessary)

Definition at line 292 of file gmxio.py.

8.25.4.6 forcebalance.gmxio.ITP_Reader.itype

Definition at line 319 of file gmxio.py.

8.25.4.7 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.25.4.8 forcebalance.gmxio.ITP_Reader.mol

The current molecule (set by the moleculetype keyword)

Definition at line 286 of file gmxio.py.

8.25.4.9 forcebalance.gmxio.ITP_Reader.molatom

Definition at line 426 of file gmxio.py.

8.25.4.10 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.25.4.11 forcebalance.gmxio.ITP_Reader.nbtype

Nonbonded type.

Definition at line 284 of file gmxio.py.

8.25.4.12 forcebalance.gmxio.ITP_Reader.pdict

The parameter dictionary (defined in this file)

Definition at line 288 of file gmxio.py.

8.25.4.13 forcebalance.gmxio.ITP_Reader.sec

The current section that we're in.

Definition at line 282 of file gmxio.py.

8.25.4.14 forcebalance.gmxio.ITP_Reader.suffix

Definition at line 421 of file gmxio.py.

The documentation for this class was generated from the following file:

- gmxio.py

8.26 forcebalance.leastsq.LeastSquares Class Reference

Subclass of Target for general least squares fitting.

Inheritance diagram for forcebalance.leastsq.LeastSquares:

Collaboration diagram for forcebalance.leastsq.LeastSquares:

Public Member Functions

- def __init__
- def indicate
- def get

  LPW 05-30-2012.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.

- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- call_derivatives

  Number of snapshots.
- MAQ

  Dictionary for derivative terms.
- D
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.26.1 Detailed Description

Subclass of Target for general least squares fitting.

Definition at line 35 of file leastsq.py.

8.26.2 Constructor & Destructor Documentation

8.26.2.1 def forcebalance.leastsq.LeastSquares.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 38 of file leastsq.py.

Here is the call graph for this function:

8.26.3 Member Function Documentation

8.26.3.1 def forcebalance.leastsq.LeastSquares.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

LPW 05-30-2012.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general software.

This subroutine interfaces with simulation software 'drivers'.
The driver is expected to give exact values, fitting values, and weights.
```

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 74 of file leastsq.py.

Here is the call graph for this function:

8.26.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.26.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.26.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.26.3.5 def forcebalance.leastsq.LeastSquares.indicate (*self*)

Definition at line 53 of file leastsq.py.

Here is the call graph for this function:

8.26.3.6 def forcebalance.target.Target.link_from_tempdir (*self*, *absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.26.3.7 def forcebalance.target.Target.printcool_table (*self*, *data* = OrderedDict([]), *headings* = [], *banner* = None, *footnote* = None, *color* = 0) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.26.3.8 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.26.3.9 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.26.3.10 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.26.3.11 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.26.3.12 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Definition at line 291 of file target.py.

8.26.3.13 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.26.4 Member Data Documentation

8.26.4.1 forcebalance.leastsq.LeastSquares.call_derivatives

Number of snapshots.

Which parameters are differentiated?

Definition at line 51 of file leastsq.py.

8.26.4.2 forcebalance.leastsq.LeastSquares.D

Definition at line 137 of file leastsq.py.

8.26.4.3 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.26.4.4 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.26.4.5 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.26.4.6 forcebalance.leastsq.LeastSquares.MAQ

Dictionary for derivative terms.

Definition at line 100 of file leastsq.py.

8.26.4.7 forcebalance.leastsq.LeastSquares.objective

Definition at line 138 of file leastsq.py.

8.26.4.8 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.26.4.9 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.26.4.10 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.26.4.11 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.26.4.12 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- leastsq.py

8.27 forcebalance.liquid.Liquid Class Reference

Subclass of Target for liquid property matching.

Inheritance diagram for forcebalance.liquid.Liquid:

Collaboration diagram for forcebalance.liquid.Liquid:

Public Member Functions

- def __init__

  Create an instance of the class.
- def read_data
- def npt_simulation

  Submit a NPT simulation to the Work Queue.
- def indicate
- def objective_term
- def submit_jobs
- def get

  Fitting of liquid bulk properties.
- def polarization_correction

Return the self-polarization correction as described in Berendsen et al., JPC 1987.

- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- do_self_pol
- extra_output

  Read the reference data.
- SavedMVal

  Saved force field mvals for all iterations.
- SavedTraj

  Saved trajectories for all iterations and all temperatures :)
- MBarEnergy

  Evaluated energies for all trajectories (i.e.
- nptpfx
- nptfiles
- nptsfx
- last_traj
- RefData
- PhasePoints
- Labels
- w_rho

  Density.
- w_hvap
- w_alpha
- w_kappa
- w_cp
- w_eps0
- tempdir

  Root directory of the whole project.
- rundir

The directory in which the simulation is running - this can be updated.

- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.27.1 Detailed Description

Subclass of Target for liquid property matching.

Definition at line 51 of file liquid.py.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 def forcebalance.liquid.Liquid.__init__ (*self, options, tgt_opts, forcefield*)

Create an instance of the class.

Definition at line 56 of file liquid.py.

Here is the call graph for this function:

8.27.3 Member Function Documentation

8.27.3.1 def forcebalance.liquid.Liquid.get (*self, mvals, AGrad* = `True`, *AHess* = `True`)

Fitting of liquid bulk properties.

This is the current major direction of development for ForceBalance. Basically, fitting the QM energies / forces alone does not always give us the best simulation behavior. In many cases it makes more sense to try and reproduce some experimentally known data as well.

In order to reproduce experimentally known data, we need to run a simulation and compare the simulation result to experiment. The main challenge here is that the simulations are computationally intensive (i.e. they require energy and force evaluations), and furthermore the results are noisy. We need to run the simulations automatically and remotely (i.e. on clusters) and a good way to calculate the derivatives of the simulation results with respect to the parameter values.

This function contains some experimentally known values of the density and enthalpy of vaporization (Hvap) of liquid water. It launches the density and Hvap calculations on the cluster, and gathers the results / derivatives. The actual calculation of results / derivatives is done in a separate file.

After the results come back, they are gathered together to form an objective function.

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 401 of file liquid.py.

Here is the call graph for this function:

8.27.3.2 def forcebalance.target.Target.get_G (*self, mvals =* `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.27.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.27.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.27.3.5 def forcebalance.liquid.Liquid.indicate (*self*)

Definition at line 254 of file liquid.py.

8.27.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.27.3.7 def forcebalance.liquid.Liquid.npt_simulation (*self, temperature, pressure, simnum*)

Submit a NPT simulation to the Work Queue.

Definition at line 233 of file liquid.py.

Here is the call graph for this function:

8.27.3.8 def forcebalance.liquid.Liquid.objective_term (*self, points, expname, calc, err, grad, name* = "Quantity", *SubAverage* = False)

Definition at line 258 of file liquid.py.

Here is the call graph for this function:

8.27.3.9 def forcebalance.liquid.Liquid.polarization_correction (*self, mvals*)

Return the self-polarization correction as described in Berendsen et al., JPC 1987.

Definition at line 738 of file liquid.py.

8.27.3.10 def forcebalance.target.Target.printcool_table (*self, data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.27.3.11 def forcebalance.liquid.Liquid.read_data (*self*)

Definition at line 141 of file liquid.py.

8.27.3.12 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.27.3.13 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.27.3.14 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.27.3.15 def forcebalance.target.Target.stage (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.27.3.16 def forcebalance.liquid.Liquid.submit_jobs (*self*, *mvals*, *AGrad* = `True`, *AHess* = `True`)

Definition at line 336 of file liquid.py.

Here is the call graph for this function:

8.27.3.17 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.27.4 Member Data Documentation

8.27.4.1 forcebalance.liquid.Liquid.do_self_pol

Definition at line 99 of file liquid.py.

8.27.4.2 forcebalance.liquid.Liquid.extra_output

Read the reference data.

Prepare the temporary directory Extra platform-dependent data to send back

Definition at line 114 of file liquid.py.

8.27.4.3 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.27.4.4 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.27.4.5 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.27.4.6 forcebalance.liquid.Liquid.Labels

Definition at line 221 of file liquid.py.

8.27.4.7 forcebalance.liquid.Liquid.last_traj

Definition at line 139 of file liquid.py.

8.27.4.8 forcebalance.liquid.Liquid.MBarEnergy

Evaluated energies for all trajectories (i.e.

all iterations and all temperatures), using all mvals

Definition at line 126 of file liquid.py.

8.27.4.9 forcebalance.liquid.Liquid.nptfiles

Definition at line 130 of file liquid.py.

8.27.4.10 forcebalance.liquid.Liquid.nptpfx

Definition at line 128 of file liquid.py.

8.27.4.11 forcebalance.liquid.Liquid.nptsfx

Definition at line 132 of file liquid.py.

8.27.4.12 forcebalance.liquid.Liquid.PhasePoints

Definition at line 217 of file liquid.py.

8.27.4.13 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.27.4.14 forcebalance.liquid.Liquid.RefData

Definition at line 151 of file liquid.py.

8.27.4.15 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.27.4.16 forcebalance.liquid.Liquid.SavedMVal

Saved force field mvals for all iterations.

Definition at line 122 of file liquid.py.

8.27.4.17 forcebalance.liquid.Liquid.SavedTraj

Saved trajectories for all iterations and all temperatures :)

Definition at line 124 of file liquid.py.

8.27.4.18 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.27.4.19 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.27.4.20 forcebalance.liquid.Liquid.w_alpha

Definition at line 650 of file liquid.py.

8.27.4.21 forcebalance.liquid.Liquid.w_cp

Definition at line 652 of file liquid.py.

8.27.4.22 forcebalance.liquid.Liquid.w_eps0

Definition at line 653 of file liquid.py.

8.27.4.23 forcebalance.liquid.Liquid.w_hvap

Definition at line 649 of file liquid.py.

8.27.4.24 forcebalance.liquid.Liquid.w_kappa

Definition at line 651 of file liquid.py.

8.27.4.25 forcebalance.liquid.Liquid.w_rho

Density.

Enthalpy of vaporization. Thermal expansion coefficient. Isothermal compressibility. Isobaric heat capacity. Static dielectric constant. Estimation of errors.

Definition at line 648 of file liquid.py.

8.27.4.26 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- liquid.py

8.28 forcebalance.gmxio.Liquid_GMX Class Reference

Inheritance diagram for forcebalance.gmxio.Liquid_GMX:

Collaboration diagram for forcebalance.gmxio.Liquid_GMX:

Public Member Functions

- def __init__
- def prepare_temp_directory

  Prepare the temporary directory by copying in important files.
- def polarization_correction
- def read_data
- def npt_simulation

  Submit a NPT simulation to the Work Queue.
- def indicate
- def objective_term
- def submit_jobs
- def get

  Fitting of liquid bulk properties.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

Computes the objective function contribution and its gradient / Hessian.

- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- liquid_fnm
- liquid_conf
- liquid_traj
- gas_fnm
- nptpfx
- engine
- extra_output
- do_self_pol
- SavedMVal

  Saved force field mvals for all iterations.
- SavedTraj

  Saved trajectories for all iterations and all temperatures :)
- MBarEnergy

  Evaluated energies for all trajectories (i.e.
- nptfiles
- nptsfx
- last_traj
- RefData
- PhasePoints
- Labels
- w_rho

  Density.
- w_hvap
- w_alpha
- w_kappa
- w_cp
- w_eps0
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

Need the forcefield (here for now)

- xct

  Counts how often the objective function was computed.

- gct

  Counts how often the gradient was computed.

- hct

  Counts how often the Hessian was computed.

- PrintOptionDict
- verbose_options

8.28.1 Detailed Description

Definition at line 573 of file gmxio.py.

8.28.2 Constructor & Destructor Documentation

8.28.2.1 def forcebalance.gmxio.Liquid_GMX.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 574 of file gmxio.py.

Here is the call graph for this function:

8.28.3 Member Function Documentation

8.28.3.1 def forcebalance.liquid.Liquid.get (*self, mvals, AGrad* = `True`, *AHess* = `True`) `[inherited]`

Fitting of liquid bulk properties.

This is the current major direction of development for ForceBalance. Basically, fitting the QM energies / forces alone does not always give us the best simulation behavior. In many cases it makes more sense to try and reproduce some experimentally known data as well.

In order to reproduce experimentally known data, we need to run a simulation and compare the simulation result to experiment. The main challenge here is that the simulations are computationally intensive (i.e. they require energy and force evaluations), and furthermore the results are noisy. We need to run the simulations automatically and remotely (i.e. on clusters) and a good way to calculate the derivatives of the simulation results with respect to the parameter values.

This function contains some experimentally known values of the density and enthalpy of vaporization (Hvap) of liquid water. It launches the density and Hvap calculations on the cluster, and gathers the results / derivatives. The actual calculation of results / derivatives is done in a separate file.

After the results come back, they are gathered together to form an objective function.

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 401 of file liquid.py.

Here is the call graph for this function:

8.28.3.2 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.28.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.28.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.28.3.5 def forcebalance.liquid.Liquid.indicate (*self*) [inherited]

Definition at line 254 of file liquid.py.

8.28.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.28.3.7 def forcebalance.liquid.Liquid.npt_simulation (*self, temperature, pressure, simnum*) [inherited]

Submit a NPT simulation to the Work Queue.

Definition at line 233 of file liquid.py.

Here is the call graph for this function:

8.28.3.8 def forcebalance.liquid.Liquid.objective_term (*self, points, expname, calc, err, grad, name* = "Quantity", *SubAverage* = False) [inherited]

Definition at line 258 of file liquid.py.

Here is the call graph for this function:

8.28.3.9 def forcebalance.gmxio.Liquid_GMX.polarization_correction (*self, mvals*)

Definition at line 618 of file gmxio.py.

Here is the call graph for this function:

8.28.3.10 def forcebalance.gmxio.Liquid_GMX.prepare_temp_directory (*self, options, tgt_opts*)

Prepare the temporary directory by copying in important files.

Definition at line 601 of file gmxio.py.

Here is the call graph for this function:

8.28.3.11 def forcebalance.target.Target.printcool_table (*self, data =* `OrderedDict([])`, *headings =* `[]`, *banner =* `None`, *footnote =* `None`, *color =* `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.28.3.12 def forcebalance.liquid.Liquid.read_data (*self*) [inherited]

Definition at line 141 of file liquid.py.

8.28.3.13 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.28.3.14 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.28.3.15 def forcebalance.target.Target.sget (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.28.3.16 def forcebalance.target.Target.stage (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.28.3.17 def forcebalance.liquid.Liquid.submit_jobs (*self, mvals, AGrad* = `True`, *AHess* = `True`) `[inherited]`

Definition at line 336 of file liquid.py.

Here is the call graph for this function:

8.28.3.18 def forcebalance.target.Target.wq_complete (*self*) `[inherited]`

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.28.4 Member Data Documentation

8.28.4.1 forcebalance.liquid.Liquid.do_self_pol `[inherited]`

Definition at line 99 of file liquid.py.

8.28.4.2 forcebalance.gmxio.Liquid_GMX.engine

Definition at line 594 of file gmxio.py.

8.28.4.3 forcebalance.gmxio.Liquid_GMX.extra_output

Definition at line 597 of file gmxio.py.

8.28.4.4 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.28.4.5 forcebalance.gmxio.Liquid_GMX.gas_fnm

Definition at line 581 of file gmxio.py.

8.28.4.6 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.28.4.7 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.28.4.8 forcebalance.liquid.Liquid.Labels [inherited]

Definition at line 221 of file liquid.py.

8.28.4.9 forcebalance.liquid.Liquid.last_traj [inherited]

Definition at line 139 of file liquid.py.

8.28.4.10 forcebalance.gmxio.Liquid_GMX.liquid_conf

Definition at line 579 of file gmxio.py.

8.28.4.11 forcebalance.gmxio.Liquid_GMX.liquid_fnm

Definition at line 578 of file gmxio.py.

8.28.4.12 forcebalance.gmxio.Liquid_GMX.liquid_traj

Definition at line 580 of file gmxio.py.

8.28.4.13 forcebalance.liquid.Liquid.MBarEnergy [inherited]

Evaluated energies for all trajectories (i.e.

all iterations and all temperatures), using all mvals

Definition at line 126 of file liquid.py.

8.28.4.14 forcebalance.liquid.Liquid.nptfiles [inherited]

Definition at line 130 of file liquid.py.

8.28.4.15 forcebalance.gmxio.Liquid_GMX.nptpfx

Definition at line 588 of file gmxio.py.

8.28.4.16 forcebalance.liquid.Liquid.nptsfx [inherited]

Definition at line 132 of file liquid.py.

8.28.4.17 forcebalance.liquid.Liquid.PhasePoints [inherited]

Definition at line 217 of file liquid.py.

8.28.4.18 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.28.4.19 forcebalance.liquid.Liquid.RefData [inherited]

Definition at line 151 of file liquid.py.

8.28.4.20 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.28.4.21 forcebalance.liquid.Liquid.SavedMVal [inherited]

Saved force field mvals for all iterations.

Definition at line 122 of file liquid.py.

8.28.4.22 forcebalance.liquid.Liquid.SavedTraj [inherited]

Saved trajectories for all iterations and all temperatures :)

Definition at line 124 of file liquid.py.

8.28.4.23 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.28.4.24 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.28.4.25 forcebalance.liquid.Liquid.w_alpha [inherited]

Definition at line 650 of file liquid.py.

8.28.4.26 forcebalance.liquid.Liquid.w_cp [inherited]

Definition at line 652 of file liquid.py.

8.28.4.27 forcebalance.liquid.Liquid.w_eps0 [inherited]

Definition at line 653 of file liquid.py.

8.28.4.28 forcebalance.liquid.Liquid.w_hvap [inherited]

Definition at line 649 of file liquid.py.

8.28.4.29 forcebalance.liquid.Liquid.w_kappa [inherited]

Definition at line 651 of file liquid.py.

8.28.4.30 forcebalance.liquid.Liquid.w_rho [inherited]

Density.

Enthalpy of vaporization. Thermal expansion coefficient. Isothermal compressibility. Isobaric heat capacity. Static dielectric constant. Estimation of errors.

Definition at line 648 of file liquid.py.

8.28.4.31 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- gmxio.py

8.29 forcebalance.openmmio.Liquid_OpenMM Class Reference

Inheritance diagram for forcebalance.openmmio.Liquid_OpenMM:

Collaboration diagram for forcebalance.openmmio.Liquid_OpenMM:

Public Member Functions

- def __init__
- def prepare_temp_directory

  Prepare the temporary directory by copying in important files.
- def polarization_correction
- def read_data
- def npt_simulation

  Submit a NPT simulation to the Work Queue.
- def indicate
- def objective_term
- def submit_jobs
- def get

  Fitting of liquid bulk properties.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.

- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- mpdb
- platform
- msim
- liquid_fnm
- liquid_conf
- liquid_traj
- gas_fnm
- engine
- do_self_pol
- extra_output

  Read the reference data.
- SavedMVal

  Saved force field mvals for all iterations.
- SavedTraj

  Saved trajectories for all iterations and all temperatures :)
- MBarEnergy

  Evaluated energies for all trajectories (i.e.
- nptpfx
- nptfiles
- nptsfx
- last_traj
- RefData
- PhasePoints
- Labels
- w_rho

  Density.
- w_hvap
- w_alpha
- w_kappa
- w_cp
- w_eps0

- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.29.1 Detailed Description

Definition at line 329 of file openmmio.py.

8.29.2 Constructor & Destructor Documentation

8.29.2.1 def forcebalance.openmmio.Liquid_OpenMM.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 330 of file openmmio.py.

8.29.3 Member Function Documentation

8.29.3.1 def forcebalance.liquid.Liquid.get (*self, mvals, AGrad* = `True`, *AHess* = `True`) `[inherited]`

Fitting of liquid bulk properties.

This is the current major direction of development for ForceBalance. Basically, fitting the QM energies / forces alone does not always give us the best simulation behavior. In many cases it makes more sense to try and reproduce some experimentally known data as well.

In order to reproduce experimentally known data, we need to run a simulation and compare the simulation result to experiment. The main challenge here is that the simulations are computationally intensive (i.e. they require energy and force evaluations), and furthermore the results are noisy. We need to run the simulations automatically and remotely (i.e. on clusters) and a good way to calculate the derivatives of the simulation results with respect to the parameter values.

This function contains some experimentally known values of the density and enthalpy of vaporization (Hvap) of liquid water. It launches the density and Hvap calculations on the cluster, and gathers the results / derivatives. The actual calculation of results / derivatives is done in a separate file.

After the results come back, they are gathered together to form an objective function.

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 401 of file liquid.py.

Here is the call graph for this function:

8.29.3.2 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.29.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.29.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.29.3.5 def forcebalance.liquid.Liquid.indicate (*self*) [inherited]

Definition at line 254 of file liquid.py.

8.29.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.29.3.7 def forcebalance.liquid.Liquid.npt_simulation (*self, temperature, pressure, simnum*) [inherited]

Submit a NPT simulation to the Work Queue.

Definition at line 233 of file liquid.py.

Here is the call graph for this function:

8.29.3.8 def forcebalance.liquid.Liquid.objective_term (*self, points, expname, calc, err, grad, name* = "Quantity", *SubAverage* = False) [inherited]

Definition at line 258 of file liquid.py.

Here is the call graph for this function:

8.29.3.9 def forcebalance.openmmio.Liquid_OpenMM.polarization_correction (*self, mvals*)

Definition at line 377 of file openmmio.py.

Here is the call graph for this function:

8.29.3.10 def forcebalance.openmmio.Liquid_OpenMM.prepare_temp_directory (*self, options, tgt_opts*)

Prepare the temporary directory by copying in important files.

Definition at line 370 of file openmmio.py.

Here is the call graph for this function:

8.29.3.11 def forcebalance.target.Target.printcool_table (*self, data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.

banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.29.3.12 def forcebalance.liquid.Liquid.read_data (*self*) [inherited]

Definition at line 141 of file liquid.py.

8.29.3.13 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.29.3.14 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.29.3.15 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.29.3.16 def forcebalance.target.Target.stage (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.29.3.17 def forcebalance.liquid.Liquid.submit_jobs (*self, mvals, AGrad* = `True`, *AHess* = `True`) `[inherited]`

Definition at line 336 of file liquid.py.

Here is the call graph for this function:

8.29.3.18 def forcebalance.target.Target.wq_complete (*self*) `[inherited]`

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.29.4 Member Data Documentation

8.29.4.1 forcebalance.liquid.Liquid.do_self_pol `[inherited]`

Definition at line 99 of file liquid.py.

8.29.4.2 forcebalance.openmmio.Liquid_OpenMM.engine

Definition at line 366 of file openmmio.py.

8.29.4.3 forcebalance.liquid.Liquid.extra_output [inherited]

Read the reference data.

Prepare the temporary directory Extra platform-dependent data to send back

Definition at line 114 of file liquid.py.

8.29.4.4 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.29.4.5 forcebalance.openmmio.Liquid_OpenMM.gas_fnm

Definition at line 353 of file openmmio.py.

8.29.4.6 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.29.4.7 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.29.4.8 forcebalance.liquid.Liquid.Labels [inherited]

Definition at line 221 of file liquid.py.

8.29.4.9 forcebalance.liquid.Liquid.last_traj [inherited]

Definition at line 139 of file liquid.py.

8.29.4.10 forcebalance.openmmio.Liquid_OpenMM.liquid_conf

Definition at line 351 of file openmmio.py.

8.29.4.11 forcebalance.openmmio.Liquid_OpenMM.liquid_fnm

Definition at line 350 of file openmmio.py.

8.29.4.12 forcebalance.openmmio.Liquid_OpenMM.liquid_traj

Definition at line 352 of file openmmio.py.

8.29.4.13 forcebalance.liquid.Liquid.MBarEnergy [inherited]

Evaluated energies for all trajectories (i.e.

all iterations and all temperatures), using all mvals

Definition at line 126 of file liquid.py.

8.29.4.14 forcebalance.openmmio.Liquid_OpenMM.mpdb

Definition at line 338 of file openmmio.py.

8.29.4.15 forcebalance.openmmio.Liquid_OpenMM.msim

Definition at line 348 of file openmmio.py.

8.29.4.16 forcebalance.liquid.Liquid.nptfiles [inherited]

Definition at line 130 of file liquid.py.

8.29.4.17 forcebalance.liquid.Liquid.nptpfx [inherited]

Definition at line 128 of file liquid.py.

8.29.4.18 forcebalance.liquid.Liquid.nptsfx [inherited]

Definition at line 132 of file liquid.py.

8.29.4.19 forcebalance.liquid.Liquid.PhasePoints [inherited]

Definition at line 217 of file liquid.py.

8.29.4.20 forcebalance.openmmio.Liquid_OpenMM.platform

Definition at line 346 of file openmmio.py.

8.29.4.21 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.29.4.22 forcebalance.liquid.Liquid.RefData [inherited]

Definition at line 151 of file liquid.py.

8.29.4.23 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.29.4.24 forcebalance.liquid.Liquid.SavedMVal [inherited]

Saved force field mvals for all iterations.

Definition at line 122 of file liquid.py.

8.29.4.25 forcebalance.liquid.Liquid.SavedTraj [inherited]

Saved trajectories for all iterations and all temperatures :)

Definition at line 124 of file liquid.py.

8.29.4.26 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.29.4.27 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.29.4.28 forcebalance.liquid.Liquid.w_alpha [inherited]

Definition at line 650 of file liquid.py.

8.29.4.29 forcebalance.liquid.Liquid.w_cp [inherited]

Definition at line 652 of file liquid.py.

8.29.4.30 forcebalance.liquid.Liquid.w_eps0 [inherited]

Definition at line 653 of file liquid.py.

8.29.4.31 forcebalance.liquid.Liquid.w_hvap [inherited]

Definition at line 649 of file liquid.py.

8.29.4.32 forcebalance.liquid.Liquid.w_kappa [inherited]

Definition at line 651 of file liquid.py.

8.29.4.33 forcebalance.liquid.Liquid.w_rho [inherited]

Density.

Enthalpy of vaporization. Thermal expansion coefficient. Isothermal compressibility. Isobaric heat capacity. Static dielectric constant. Estimation of errors.

Definition at line 648 of file liquid.py.

8.29.4.34 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- openmmio.py

8.30 forcebalance.tinkerio.Liquid_TINKER Class Reference

Inheritance diagram for forcebalance.tinkerio.Liquid_TINKER:

Collaboration diagram for forcebalance.tinkerio.Liquid_TINKER:

Public Member Functions

- def __init__
- def prepare_temp_directory

  Prepare the temporary directory by copying in important files.
- def npt_simulation

  Submit a NPT simulation to the Work Queue.
- def read_data
- def indicate
- def objective_term
- def submit_jobs
- def get

  Fitting of liquid bulk properties.
- def polarization_correction

  Return the self-polarization correction as described in Berendsen et al., JPC 1987.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

Computes the objective function contribution and its gradient.

- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- DynDict
- DynDict_New
- do_self_pol
- extra_output

  Read the reference data.
- SavedMVal

  Saved force field mvals for all iterations.
- SavedTraj

  Saved trajectories for all iterations and all temperatures :)
- MBarEnergy

  Evaluated energies for all trajectories (i.e.
- nptpfx
- nptfiles
- nptsfx
- last_traj
- RefData
- PhasePoints
- Labels
- w_rho

  Density.
- w_hvap
- w_alpha
- w_kappa
- w_cp
- w_eps0
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

Need the forcefield (here for now)

- xct

  Counts how often the objective function was computed.

- gct

  Counts how often the gradient was computed.

- hct

  Counts how often the Hessian was computed.

- PrintOptionDict
- verbose_options

8.30.1 Detailed Description

Definition at line 235 of file tinkerio.py.

8.30.2 Constructor & Destructor Documentation

8.30.2.1 def forcebalance.tinkerio.Liquid_TINKER.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 236 of file tinkerio.py.

Here is the call graph for this function:

8.30.3 Member Function Documentation

8.30.3.1 def forcebalance.liquid.Liquid.get (*self, mvals, AGrad* = `True`, *AHess* = `True`) `[inherited]`

Fitting of liquid bulk properties.

This is the current major direction of development for ForceBalance. Basically, fitting the QM energies / forces alone does not always give us the best simulation behavior. In many cases it makes more sense to try and reproduce some experimentally known data as well.

In order to reproduce experimentally known data, we need to run a simulation and compare the simulation result to experiment. The main challenge here is that the simulations are computationally intensive (i.e. they require energy and force evaluations), and furthermore the results are noisy. We need to run the simulations automatically and remotely (i.e. on clusters) and a good way to calculate the derivatives of the simulation results with respect to the parameter values.

This function contains some experimentally known values of the density and enthalpy of vaporization (Hvap) of liquid water. It launches the density and Hvap calculations on the cluster, and gathers the results / derivatives. The actual calculation of results / derivatives is done in a separate file.

After the results come back, they are gathered together to form an objective function.

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 401 of file liquid.py.

Here is the call graph for this function:

8.30.3.2 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.30.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.30.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.30.3.5 def forcebalance.liquid.Liquid.indicate (*self*) [inherited]

Definition at line 254 of file liquid.py.

8.30.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.30.3.7 def forcebalance.tinkerio.Liquid_TINKER.npt_simulation (*self, temperature, pressure, simnum*)

Submit a NPT simulation to the Work Queue.

Definition at line 263 of file tinkerio.py.

Here is the call graph for this function:

8.30.3.8 def forcebalance.liquid.Liquid.objective_term (*self, points, expname, calc, err, grad, name* = "Quantity", *SubAverage* = False) [inherited]

Definition at line 258 of file liquid.py.

Here is the call graph for this function:

8.30.3.9 def forcebalance.liquid.Liquid.polarization_correction (*self, mvals*) [inherited]

Return the self-polarization correction as described in Berendsen et al., JPC 1987.

Definition at line 738 of file liquid.py.

8.30.3.10 def forcebalance.tinkerio.Liquid_TINKER.prepare_temp_directory (*self, options, tgt_opts*)

Prepare the temporary directory by copying in important files.

Definition at line 245 of file tinkerio.py.

Here is the call graph for this function:

8.30.3.11 def forcebalance.target.Target.printcool_table (*self, data* = OrderedDict([]), *headings* = [], *banner* = None, *footnote* = None, *color* = 0) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.30.3.12 def forcebalance.liquid.Liquid.read_data (*self*) [inherited]

Definition at line 141 of file liquid.py.

8.30.3.13 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.30.3.14 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.30.3.15 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.30.3.16 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.30.3.17 def forcebalance.liquid.Liquid.submit_jobs (*self, mvals, AGrad* = `True`, *AHess* = `True`) `[inherited]`

Definition at line 336 of file liquid.py.

Here is the call graph for this function:

8.30.3.18 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.30.4 Member Data Documentation

8.30.4.1 forcebalance.liquid.Liquid.do_self_pol [inherited]

Definition at line 99 of file liquid.py.

8.30.4.2 forcebalance.tinkerio.Liquid_TINKER.DynDict

Definition at line 238 of file tinkerio.py.

8.30.4.3 forcebalance.tinkerio.Liquid_TINKER.DynDict_New

Definition at line 239 of file tinkerio.py.

8.30.4.4 forcebalance.liquid.Liquid.extra_output [inherited]

Read the reference data.

Prepare the temporary directory Extra platform-dependent data to send back

Definition at line 114 of file liquid.py.

8.30.4.5 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.30.4.6 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.30.4.7 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.30.4.8 forcebalance.liquid.Liquid.Labels [inherited]

Definition at line 221 of file liquid.py.

8.30.4.9 forcebalance.liquid.Liquid.last_traj [inherited]

Definition at line 139 of file liquid.py.

8.30.4.10 forcebalance.liquid.Liquid.MBarEnergy [inherited]

Evaluated energies for all trajectories (i.e.

all iterations and all temperatures), using all mvals

Definition at line 126 of file liquid.py.

8.30.4.11 forcebalance.liquid.Liquid.nptfiles [inherited]

Definition at line 130 of file liquid.py.

8.30.4.12 forcebalance.liquid.Liquid.nptpfx [inherited]

Definition at line 128 of file liquid.py.

8.30.4.13 forcebalance.liquid.Liquid.nptsfx [inherited]

Definition at line 132 of file liquid.py.

8.30.4.14 forcebalance.liquid.Liquid.PhasePoints [inherited]

Definition at line 217 of file liquid.py.

8.30.4.15 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.30.4.16 forcebalance.liquid.Liquid.RefData [inherited]

Definition at line 151 of file liquid.py.

8.30.4.17 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.30.4.18 forcebalance.liquid.Liquid.SavedMVal [inherited]

Saved force field mvals for all iterations.

Definition at line 122 of file liquid.py.

8.30.4.19 forcebalance.liquid.Liquid.SavedTraj [inherited]

Saved trajectories for all iterations and all temperatures :)

Definition at line 124 of file liquid.py.

8.30.4.20 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.30.4.21 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.30.4.22 forcebalance.liquid.Liquid.w_alpha [inherited]

Definition at line 650 of file liquid.py.

8.30.4.23 forcebalance.liquid.Liquid.w_cp [inherited]

Definition at line 652 of file liquid.py.

8.30.4.24 forcebalance.liquid.Liquid.w_eps0 [inherited]

Definition at line 653 of file liquid.py.

8.30.4.25 forcebalance.liquid.Liquid.w_hvap [inherited]

Definition at line 649 of file liquid.py.

8.30.4.26 forcebalance.liquid.Liquid.w_kappa [inherited]

Definition at line 651 of file liquid.py.

8.30.4.27 forcebalance.liquid.Liquid.w_rho [inherited]

Density.

Enthalpy of vaporization. Thermal expansion coefficient. Isothermal compressibility. Isobaric heat capacity. Static dielectric constant. Estimation of errors.

Definition at line 648 of file liquid.py.

8.30.4.28 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- tinkerio.py

8.31 forcebalance.Mol2.mol2 Class Reference

This is to manage one mol2 series of lines on the form:

Public Member Functions

- def __init__
- def __repr__
- def out
- def set_mol_name

  bond identifier (integer, starting from 1)
- def set_num_atoms

  number of atoms (integer)
- def set_num_bonds

  number of bonds (integer)
- def set_num_subst

  number of substructures (integer)
- def set_num_feat

  number of features (integer)
- def set_num_sets

  number of sets (integer)
- def set_mol_type

  bond identifier (integer, starting from 1)
- def set_charge_type

  bond identifier (integer, starting from 1)
- def parse

  Parse a series of text lines, and setup compound information.
- def get_atom

  return the atom instance given its atom identifier
- def get_bonded_atoms

  return a dictionnary of atom instances bonded to the atom, and their types
- def set_donnor_acceptor_atoms

  modify atom types to specify donnor, acceptor, or both

Public Attributes

- mol_name
- num_atoms
- num_bonds
- num_subst
- num_feat
- num_sets

- mol_type
- charge_type
- comments
- atoms
- bonds

8.31.1 Detailed Description

This is to manage one mol2 series of lines on the form:

```
@<TRIPOS>MOLECULE
CDK2.xray.inh1.1E9H
 34 37 0 0 0
SMALL
GASTEIGER
Energy = 0

@<TRIPOS>ATOM
      1 C1          5.4790   42.2880   49.5910 C.ar    1  <1>         0.0424
      2 C2          4.4740   42.6430   50.5070 C.ar    1  <1>         0.0447
@<TRIPOS>BOND
     1     1     2   ar
     2     1     6   ar
```

Definition at line 288 of file Mol2.py.

8.31.2 Constructor & Destructor Documentation

8.31.2.1 def forcebalance.Mol2.mol2.__init__ (*self, data*)

Definition at line 289 of file Mol2.py.

8.31.3 Member Function Documentation

8.31.3.1 def forcebalance.Mol2.mol2.__repr__ (*self*)

Definition at line 305 of file Mol2.py.

Here is the call graph for this function:

8.31.3.2 def forcebalance.Mol2.mol2.get_atom (*self, id*)

return the atom instance given its atom identifier

Definition at line 461 of file Mol2.py.

Here is the call graph for this function:

8.31.3.3 def forcebalance.Mol2.mol2.get_bonded_atoms (*self, id*)

return a dictionnary of atom instances bonded to the atom, and their types

Definition at line 475 of file Mol2.py.

8.31.3.4 def forcebalance.Mol2.mol2.out (*self, f* = `sys.stdout`)

Definition at line 325 of file Mol2.py.

Here is the call graph for this function:

8.31.3.5 def forcebalance.Mol2.mol2.parse (*self, data*)

Parse a series of text lines, and setup compound information.

Definition at line 405 of file Mol2.py.

Here is the call graph for this function:

8.31.3.6 def forcebalance.Mol2.mol2.set_charge_type (*self, charge_type* = `None`)

bond identifier (integer, starting from 1)

Definition at line 395 of file Mol2.py.

Here is the call graph for this function:

8.31.3.7 def forcebalance.Mol2.mol2.set_donnor_acceptor_atoms (*self, verbose* = 0)

modify atom types to specify donnor, acceptor, or both

Definition at line 490 of file Mol2.py.

Here is the call graph for this function:

8.31.3.8 def forcebalance.Mol2.mol2.set_mol_name (*self, mol_name* = None)

bond identifier (integer, starting from 1)

Definition at line 332 of file Mol2.py.

Here is the call graph for this function:

8.31.3.9 def forcebalance.Mol2.mol2.set_mol_type (*self, mol_type* = None)

bond identifier (integer, starting from 1)

Definition at line 386 of file Mol2.py.

Here is the call graph for this function:

8.31.3.10 def forcebalance.Mol2.mol2.set_num_atoms (*self, num_atoms* = `None`)

number of atoms (integer)

Definition at line 341 of file Mol2.py.

Here is the call graph for this function:

8.31.3.11 def forcebalance.Mol2.mol2.set_num_bonds (*self, num_bonds* = `None`)

number of bonds (integer)

Definition at line 350 of file Mol2.py.

Here is the call graph for this function:

8.31.3.12 def forcebalance.Mol2.mol2.set_num_feat (*self, num_feat* = `None`)

number of features (integer)

Definition at line 368 of file Mol2.py.

Here is the call graph for this function:

8.31.3.13 def forcebalance.Mol2.mol2.set_num_sets (*self, num_sets* = None)

number of sets (integer)

Definition at line 377 of file Mol2.py.

Here is the call graph for this function:

8.31.3.14 def forcebalance.Mol2.mol2.set_num_subst (*self, num_subst* = None)

number of substructures (integer)

Definition at line 359 of file Mol2.py.

Here is the call graph for this function:

8.31.4 Member Data Documentation

8.31.4.1 forcebalance.Mol2.mol2.atoms

Definition at line 300 of file Mol2.py.

8.31.4.2 forcebalance.Mol2.mol2.bonds

Definition at line 301 of file Mol2.py.

8.31.4.3 forcebalance.Mol2.mol2.charge_type

Definition at line 297 of file Mol2.py.

8.31.4.4 forcebalance.Mol2.mol2.comments

Definition at line 298 of file Mol2.py.

8.31.4.5 forcebalance.Mol2.mol2.mol_name

Definition at line 290 of file Mol2.py.

8.31.4.6 forcebalance.Mol2.mol2.mol_type

Definition at line 296 of file Mol2.py.

8.31.4.7 forcebalance.Mol2.mol2.num_atoms

Definition at line 291 of file Mol2.py.

8.31.4.8 forcebalance.Mol2.mol2.num_bonds

Definition at line 292 of file Mol2.py.

8.31.4.9 forcebalance.Mol2.mol2.num_feat

Definition at line 294 of file Mol2.py.

8.31.4.10 forcebalance.Mol2.mol2.num_sets

Definition at line 295 of file Mol2.py.

8.31.4.11 forcebalance.Mol2.mol2.num_subst

Definition at line 293 of file Mol2.py.

The documentation for this class was generated from the following file:

- Mol2.py

8.32 forcebalance.Mol2.mol2_atom Class Reference

This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.

Public Member Functions

- def __init__

  if data is passed, it will be installed
- def parse

  split the text line into a series of properties
- def __repr__

  assemble the properties as a text line, and return it
- def set_atom_id

  atom identifier (integer, starting from 1)

- def set_atom_name

  The name of the atom (string)
- def set_crds

  the coordinates of the atom
- def set_atom_type

  The mol2 type of the atom.
- def set_subst_id

  substructure identifier
- def set_subst_name

  substructure name
- def set_charge

  atomic charge
- def set_status_bit

  Never to use (in theory)

Public Attributes

- atom_id
- atom_name
- x
- y
- z
- atom_type
- subst_id
- subst_name
- charge
- status_bit

8.32.1 Detailed Description

This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.

Definition at line 32 of file Mol2.py.

8.32.2 Constructor & Destructor Documentation

8.32.2.1 def forcebalance.Mol2.mol2_atom.__init__ (*self, data =* `None`)

if data is passed, it will be installed

Definition at line 37 of file Mol2.py.

8.32.3 Member Function Documentation

8.32.3.1 def forcebalance.Mol2.mol2_atom.__repr__ (*self*)

assemble the properties as a text line, and return it

Definition at line 78 of file Mol2.py.

Here is the call graph for this function:

8.32.3.2 def forcebalance.Mol2.mol2_atom.parse (*self, data*)

split the text line into a series of properties

Definition at line 56 of file Mol2.py.

Here is the call graph for this function:

8.32.3.3 def forcebalance.Mol2.mol2_atom.set_atom_id (*self, atom_id =* `None`)

atom identifier (integer, starting from 1)

Definition at line 90 of file Mol2.py.

Here is the call graph for this function:

8.32.3.4 def forcebalance.Mol2.mol2_atom.set_atom_name (*self, atom_name =* `None`)

The name of the atom (string)

Definition at line 99 of file Mol2.py.

Here is the call graph for this function:

8.32.3.5 def forcebalance.Mol2.mol2_atom.set_atom_type (*self*, *atom_type* = None)

The mol2 type of the atom.

Definition at line 119 of file Mol2.py.

Here is the call graph for this function:

8.32.3.6 def forcebalance.Mol2.mol2_atom.set_charge (*self*, *charge* = None)

atomic charge

Definition at line 146 of file Mol2.py.

8.32.3.7 def forcebalance.Mol2.mol2_atom.set_crds (*self*, *x* = None, *y* = None, *z* = None)

the coordinates of the atom

Definition at line 108 of file Mol2.py.

Here is the call graph for this function:

8.32.3.8 def forcebalance.Mol2.mol2_atom.set_status_bit (*self*, *status_bit* = None)

Never to use (in theory)

Definition at line 155 of file Mol2.py.

Here is the call graph for this function:

8.32.3.9 def forcebalance.Mol2.mol2_atom.set_subst_id (*self*, *subst_id* = None)

substructure identifier

Definition at line 128 of file Mol2.py.

Here is the call graph for this function:

8.32.3.10 def forcebalance.Mol2.mol2_atom.set_subst_name (*self*, *subst_name* = None)

substructure name

Definition at line 137 of file Mol2.py.

Here is the call graph for this function:

8.32.4 Member Data Documentation

8.32.4.1 forcebalance.Mol2.mol2_atom.atom_id

Definition at line 38 of file Mol2.py.

8.32.4.2 forcebalance.Mol2.mol2_atom.atom_name

Definition at line 39 of file Mol2.py.

8.32.4.3 forcebalance.Mol2.mol2_atom.atom_type

Definition at line 43 of file Mol2.py.

8.32.4.4 forcebalance.Mol2.mol2_atom.charge

Definition at line 46 of file Mol2.py.

8.32.4.5 forcebalance.Mol2.mol2_atom.status_bit

Definition at line 47 of file Mol2.py.

8.32.4.6 forcebalance.Mol2.mol2_atom.subst_id

Definition at line 44 of file Mol2.py.

8.32.4.7 forcebalance.Mol2.mol2_atom.subst_name

Definition at line 45 of file Mol2.py.

8.32.4.8 forcebalance.Mol2.mol2_atom.x

Definition at line 40 of file Mol2.py.

8.32.4.9 forcebalance.Mol2.mol2_atom.y

Definition at line 41 of file Mol2.py.

8.32.4.10 forcebalance.Mol2.mol2_atom.z

Definition at line 42 of file Mol2.py.

The documentation for this class was generated from the following file:

- Mol2.py

8.33 forcebalance.Mol2.mol2_bond Class Reference

This is to manage mol2 bond lines on the form: 1 1 2 ar.

Public Member Functions

- def __init__

  if data is passed, it will be installed
- def __repr__
- def parse

  split the text line into a series of properties
- def set_bond_id

  bond identifier (integer, starting from 1)
- def set_origin_atom_id

  the origin atom identifier (integer)
- def set_target_atom_id

  the target atom identifier (integer)
- def set_bond_type

  bond type (string) one of: 1 = single 2 = double 3 = triple am = amide ar = aromatic du = dummy un = unknown nc = not connected
- def set_status_bit

  Never to use (in theory)

Public Attributes

- bond_id
- origin_atom_id
- target_atom_id
- bond_type
- status_bit

8.33.1 Detailed Description

This is to manage mol2 bond lines on the form: 1 1 2 ar.

Definition at line 172 of file Mol2.py.

8.33.2 Constructor & Destructor Documentation

8.33.2.1 def forcebalance.Mol2.mol2_bond.__init__ (*self, data* = `None`)

if data is passed, it will be installed

Definition at line 177 of file Mol2.py.

8.33.3 Member Function Documentation

8.33.3.1 def forcebalance.Mol2.mol2_bond.__repr__ (*self*)

Definition at line 186 of file Mol2.py.

Here is the call graph for this function:

8.33.3.2 def forcebalance.Mol2.mol2_bond.parse (*self, data*)

split the text line into a series of properties

Definition at line 197 of file Mol2.py.

Here is the call graph for this function:

8.33.3.3 def forcebalance.Mol2.mol2_bond.set_bond_id (*self, bond_id* = None)

bond identifier (integer, starting from 1)

Definition at line 214 of file Mol2.py.

Here is the call graph for this function:

8.33.3.4 def forcebalance.Mol2.mol2_bond.set_bond_type (*self, bond_type* = None)

bond type (string) one of: 1 = single 2 = double 3 = triple am = amide ar = aromatic du = dummy un = unknown nc = not connected

Definition at line 250 of file Mol2.py.

8.33.3.5 def forcebalance.Mol2.mol2_bond.set_origin_atom_id (*self, origin_atom_id* = None)

the origin atom identifier (integer)

Definition at line 223 of file Mol2.py.

8.33.3.6 def forcebalance.Mol2.mol2_bond.set_status_bit (*self, status_bit* = None)

Never to use (in theory)

Definition at line 259 of file Mol2.py.

Here is the call graph for this function:

8.33.3.7 def forcebalance.Mol2.mol2_bond.set_target_atom_id (*self, target_atom_id =* `None`)

the target atom identifier (integer)

Definition at line 232 of file Mol2.py.

Here is the call graph for this function:

8.33.4 Member Data Documentation

8.33.4.1 forcebalance.Mol2.mol2_bond.bond_id

Definition at line 178 of file Mol2.py.

8.33.4.2 forcebalance.Mol2.mol2_bond.bond_type

Definition at line 181 of file Mol2.py.

8.33.4.3 forcebalance.Mol2.mol2_bond.origin_atom_id

Definition at line 179 of file Mol2.py.

8.33.4.4 forcebalance.Mol2.mol2_bond.status_bit

Definition at line 206 of file Mol2.py.

8.33.4.5 forcebalance.Mol2.mol2_bond.target_atom_id

Definition at line 180 of file Mol2.py.

The documentation for this class was generated from the following file:

- Mol2.py

8.34 forcebalance.amberio.Mol2_Reader Class Reference

Finite state machine for parsing Mol2 force field file.

Inheritance diagram for forcebalance.amberio.Mol2_Reader:

Collaboration diagram for forcebalance.amberio.Mol2_Reader:

Public Member Functions

- def __init__

- def feed
- def Split
- def Whites
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- pdict

  The parameter dictionary (defined in this file)
- atom

  The atom numbers in the interaction (stored in the parser)
- atomnames

  The mol2 file provides a list of atom names.
- section

  The section that we're in.
- mol
- itype
- suffix
- molatom
- ln
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- Molecules
- AtomTypes

8.34.1 Detailed Description

Finite state machine for parsing Mol2 force field file.

(just for parameterizing the charges)

Definition at line 43 of file amberio.py.

8.34.2 Constructor & Destructor Documentation

8.34.2.1 def forcebalance.amberio.Mol2_Reader.__init__ (*self, fnm*)

Definition at line 45 of file amberio.py.

8.34.3 Member Function Documentation

8.34.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.34.3.2 def forcebalance.amberio.Mol2_Reader.feed (*self, line*)

Definition at line 59 of file amberio.py.

8.34.3.3 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.34.3.4 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.34.4 Member Data Documentation

8.34.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.34.4.2 forcebalance.amberio.Mol2_Reader.atom

The atom numbers in the interaction (stored in the parser)

Definition at line 51 of file amberio.py.

8.34.4.3 forcebalance.amberio.Mol2_Reader.atomnames

The mol2 file provides a list of atom names.

Definition at line 53 of file amberio.py.

8.34.4.4 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.34.4.5 forcebalance.amberio.Mol2_Reader.itype

Definition at line 64 of file amberio.py.

8.34.4.6 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.34.4.7 forcebalance.amberio.Mol2_Reader.mol

Definition at line 57 of file amberio.py.

8.34.4.8 forcebalance.amberio.Mol2_Reader.molatom

Definition at line 95 of file amberio.py.

8.34.4.9 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.34.4.10 forcebalance.amberio.Mol2_Reader.pdict

The parameter dictionary (defined in this file)

Definition at line 49 of file amberio.py.

8.34.4.11 forcebalance.amberio.Mol2_Reader.section

The section that we're in.

Definition at line 55 of file amberio.py.

8.34.4.12 forcebalance.amberio.Mol2_Reader.suffix

Definition at line 93 of file amberio.py.

The documentation for this class was generated from the following file:

- amberio.py

8.35 forcebalance.mol2io.Mol2_Reader Class Reference

Finite state machine for parsing Mol2 force field file.

Inheritance diagram for forcebalance.mol2io.Mol2_Reader:

Collaboration diagram for forcebalance.mol2io.Mol2_Reader:

Public Member Functions

- def __init__
- def feed
- def Split
- def Whites

- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- pdict

  The parameter dictionary (defined in this file)
- atom

  The atom numbers in the interaction (stored in the parser)
- itype
- suffix
- ln
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.35.1 Detailed Description

Finite state machine for parsing Mol2 force field file.

(just for parameterizing the charges)

Definition at line 22 of file mol2io.py.

8.35.2 Constructor & Destructor Documentation

8.35.2.1 def forcebalance.mol2io.Mol2_Reader.__init__ (*self, fnm*)

Definition at line 24 of file mol2io.py.

8.35.3 Member Function Documentation

8.35.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.35.3.2 def forcebalance.mol2io.Mol2_Reader.feed (*self, line*)

Definition at line 32 of file mol2io.py.

8.35.3.3 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.35.3.4 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.35.4 Member Data Documentation

8.35.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.35.4.2 forcebalance.mol2io.Mol2_Reader.atom

The atom numbers in the interaction (stored in the parser)

Definition at line 30 of file mol2io.py.

8.35.4.3 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.35.4.4 forcebalance.mol2io.Mol2_Reader.itype

Definition at line 36 of file mol2io.py.

8.35.4.5 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.35.4.6 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.35.4.7 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.35.4.8 forcebalance.mol2io.Mol2_Reader.pdict

The parameter dictionary (defined in this file)

Definition at line 28 of file mol2io.py.

8.35.4.9 forcebalance.mol2io.Mol2_Reader.suffix

Definition at line 44 of file mol2io.py.

The documentation for this class was generated from the following file:

- mol2io.py

8.36 forcebalance.Mol2.mol2_set Class Reference

Public Member Functions

- def __init__

  A collection is organized as a dictionnary of compounds self.num_compounds : the number of compounds self.-compounds : the dictionnary of compounds data : the data to setup the set subset: it is possible to specify a subset of the compounds to load, based on their mol_name identifiers.
- def parse

  parse a list of lines, detect compounds, load them only load the subset if specified.

Public Attributes

- num_compounds
- comments
- compounds

8.36.1 Detailed Description

Definition at line 568 of file Mol2.py.

8.36.2 Constructor & Destructor Documentation

8.36.2.1 def forcebalance.Mol2.mol2_set.__init__ (*self*, *data* = None, *subset* = None)

A collection is organized as a dictionnary of compounds self.num_compounds : the number of compounds self.-compounds : the dictionnary of compounds data : the data to setup the set subset: it is possible to specify a subset of

the compounds to load, based on their mol_name identifiers.

Definition at line 577 of file Mol2.py.

8.36.3 Member Function Documentation

8.36.3.1 def forcebalance.Mol2.mol2_set.parse (*self, data, subset* = `None`)

parse a list of lines, detect compounds, load them only load the subset if specified.

Definition at line 621 of file Mol2.py.

8.36.4 Member Data Documentation

8.36.4.1 forcebalance.Mol2.mol2_set.comments

Definition at line 579 of file Mol2.py.

8.36.4.2 forcebalance.Mol2.mol2_set.compounds

Definition at line 580 of file Mol2.py.

8.36.4.3 forcebalance.Mol2.mol2_set.num_compounds

Definition at line 578 of file Mol2.py.

The documentation for this class was generated from the following file:

- Mol2.py

8.37 forcebalance.molecule.Molecule Class Reference

Lee-Ping's general file format conversion class.

Inheritance diagram for forcebalance.molecule.Molecule:

Collaboration diagram for forcebalance.molecule.Molecule:

Public Member Functions

- def __len__

  Return the number of frames in the trajectory.
- def __getattr__

  Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.
- def __setattr__

  Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.
- def __getitem__

  The Molecule class has list-like behavior, so we can get slices of it.
- def __delitem__

  Similarly, in order to delete a frame, we simply perform item deletion on framewise variables.
- def __iter__

  List-like behavior for looping over trajectories.
- def __add__

  Add method for Molecule objects.
- def __iadd__

  Add method for Molecule objects.
- def append
- def __init__

  To create the Molecule object, we simply define the table of file reading/writing functions and read in a file if it is provided.
- def require
- def write
- def center_of_mass
- def radius_of_gyration
- def load_frames
- def edit_qcrems
- def add_quantum
- def add_virtual_site

  Add a virtual site to the system.
- def replace_peratom

  Replace all of the data for a certain attribute in the system from orig to want.

- def replace_peratom_conditional

  Replace all of the data for a attribute key2 from orig to want, contingent on key1 being equal to cond.
- def atom_select

  Return a copy of the object with certain atoms selected.
- def atom_stack

  Return a copy of the object with another molecule object appended.
- def align_by_moments

  Align molecules using the "moment of inertia." Note that we're following the MSMBuilder convention of using all ones for the masses.
- def align

  Align molecules.
- def build_topology

  A bare-bones implementation of the bond graph capability in the nanoreactor code.
- def measure_dihedrals

  Return a series of dihedral angles, given four atom indices numbered from zero.
- def all_pairwise_rmsd

  Find pairwise RMSD (super slow, not like the one in MSMBuilder.)
- def align_center
- def openmm_positions

  Returns the Cartesian coordinates in the Molecule object in a list of OpenMM-compatible positions, so it is possible to type simulation.context.setPositions(Mol.openmm_positions()[0]) or something like that.
- def openmm_boxes

  Returns the periodic box vectors in the Molecule object in a list of OpenMM-compatible boxes, so it is possible to type simulation.context.setPeriodicBoxVectors(Mol.openmm_boxes()[0]) or something like that.
- def split

  Split the molecule object into a number of separate files (chunks), either by specifying the number of frames per chunk or the number of chunks.
- def read_xyz

  Parse a .xyz file which contains several xyz coordinates, and return their elements.
- def read_mdcrd

  Parse an AMBER .mdcrd file.
- def read_qdata
- def read_mol2
- def read_dcd
- def read_com

  Parse a Gaussian .com file and return a SINGLE-ELEMENT list of xyz coordinates (no multiple file support)
- def read_arc

  Read a TINKER .arc file.
- def read_gro

  Read a GROMACS .gro file.
- def read_charmm

  Read a CHARMM .cor (or .crd) file.
- def read_qcin

  Read a Q-Chem input file.
- def read_pdb

  Loads a PDB and returns a dictionary containing its data.
- def read_qcesp
- def read_qcout

Q-Chem output file reader, adapted for our parser.

- def write_qcin
- def write_xyz
- def write_molproq
- def write_mdcrd
- def write_arc
- def write_gro
- def write_dcd
- def write_pdb

  Save to a PDB.
- def write_qdata

  Text quantum data format.
- def require_resid
- def require_resname
- def require_boxes

Public Attributes

- Read_Tab

  The table of file readers.
- Write_Tab

  The table of file writers.
- Funnel

  A funnel dictionary that takes redundant file types and maps them down to a few.
- positive_resid

  Creates entries like 'gromacs' : 'gromacs' and 'xyz' : 'xyz' in the Funnel.
- Data
- comms

  Read in stuff if we passed in a file name, otherwise return an empty instance.
- topology

  Make sure the comment line isn't too long for i in range(len(self.comms)): self.comms[i] = self.comms[i][:100] if len(self.comms[i]) > 100 else self.comms[i] Attempt to build the topology for small systems.
- molecules
- fout

  Fill in comments.
- resid
- resname
- boxes

8.37.1 Detailed Description

Lee-Ping's general file format conversion class.

The purpose of this class is to read and write chemical file formats in a way that is convenient for research. There are highly general file format converters out there (e.g. catdcd, openbabel) but I find that writing my own class can be very helpful for specific purposes. Here are some things this class can do:

- Convert a .gro file to a .xyz file, or a .pdb file to a .dcd file. Data is stored internally, so any readable file can be converted into any writable file as long as there is sufficient information to write that file.

- Accumulate information from different files. For example, we may read A.gro to get a list of coordinates, add quantum settings from a B.in file, and write A.in (this gives us a file that we can use to run QM calculations)

- Concatenate two trajectories together as long as they're compatible. This is done by creating two Molecule objects and then simply adding them. Addition means two things: (1) Information fields missing from each class, but present in the other, are added to the sum, and (2) Appendable or per-frame fields (i.e. coordinates) are concatenated together.

- Slice trajectories using reasonable Python language. That is to say, MyMolecule[1:10] returns a new Molecule object that contains frames 2 through 10.

Next step: Read in Q-Chem output data using this too!

Special variables: These variables cannot be set manually because there is a special method associated with getting them.

na = The number of atoms. You'll get this if you use MyMol.na or MyMol['na']. na = The number of snapshots. You'll get this if you use MyMol.ns or MyMol['ns'].

Unit system: Angstroms.

Definition at line 645 of file molecule.py.

8.37.2 Constructor & Destructor Documentation

8.37.2.1 def forcebalance.molecule.Molecule.__init__ (*self*, *fnm* = `None`, *ftype* = `None`, *positive_resid* = `True`, *build_topology* = `True`)

To create the Molecule object, we simply define the table of file reading/writing functions and read in a file if it is provided.

Definition at line 815 of file molecule.py.

8.37.3 Member Function Documentation

8.37.3.1 def forcebalance.molecule.Molecule.__add__ (*self*, *other*)

Add method for Molecule objects.

Definition at line 754 of file molecule.py.

Here is the call graph for this function:

8.37.3.2 def forcebalance.molecule.Molecule.__delitem__ (*self, key*)

Similarly, in order to delete a frame, we simply perform item deletion on framewise variables.

Definition at line 734 of file molecule.py.

Here is the call graph for this function:

8.37.3.3 def forcebalance.molecule.Molecule.__getattr__ (*self, key*)

Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.

Definition at line 665 of file molecule.py.

Here is the call graph for this function:

8.37.3.4 def forcebalance.molecule.Molecule.__getitem__ (*self, key*)

The Molecule class has list-like behavior, so we can get slices of it.

If we say MyMolecule[0:10], then we'll return a copy of MyMolecule with frames 0 through 9.

Definition at line 713 of file molecule.py.

Here is the call graph for this function:

8.37.3.5 def forcebalance.molecule.Molecule.__iadd__ (*self, other*)

Add method for Molecule objects.

Definition at line 784 of file molecule.py.

Here is the call graph for this function:

8.37.3.6 def forcebalance.molecule.Molecule.__iter__ (*self*)

List-like behavior for looping over trajectories.

Note that these values are returned by reference. Note that this is intended to be more efficient than **getitem**, so when we loop over a trajectory, it's best to go "for m in M" instead of "for i in range(len(M)): m = M[i]"

Definition at line 743 of file molecule.py.

Here is the call graph for this function:

8.37.3.7 def forcebalance.molecule.Molecule.__len__ (*self*)

Return the number of frames in the trajectory.

Definition at line 649 of file molecule.py.

8.37.3.8 def forcebalance.molecule.Molecule.__setattr__ (*self, key, value*)

Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.

Definition at line 701 of file molecule.py.

Here is the call graph for this function:

8.37.3.9 def forcebalance.molecule.Molecule.add_quantum (*self, other*)

Definition at line 995 of file molecule.py.

Here is the call graph for this function:

8.37.3.10 def forcebalance.molecule.Molecule.add_virtual_site (*self, idx, kwargs*)

Add a virtual site to the system.

This does NOT set the position of the virtual site; it sits at the origin.

Definition at line 1007 of file molecule.py.

8.37.3.11 def forcebalance.molecule.Molecule.align (*self, smooth* = `True`, *center* = `True`)

Align molecules.

Has the option to create smooth trajectories (align each frame to the previous one) or to align each frame to the first one.

Also has the option to remove the center of mass.

Definition at line 1143 of file molecule.py.

Here is the call graph for this function:

8.37.3.12 def forcebalance.molecule.Molecule.align_by_moments (*self*)

Align molecules using the "moment of inertia." Note that we're following the MSMBuilder convention of using all ones for the masses.

Definition at line 1124 of file molecule.py.

8.37.3.13 def forcebalance.molecule.Molecule.align_center (*self*)

Definition at line 1247 of file molecule.py.

8.37.3.14 def forcebalance.molecule.Molecule.all_pairwise_rmsd (*self*)

Find pairwise RMSD (super slow, not like the one in MSMBuilder.)

Definition at line 1231 of file molecule.py.

Here is the call graph for this function:

8.37.3.15 def forcebalance.molecule.Molecule.append (*self, other*)

Definition at line 808 of file molecule.py.

Here is the call graph for this function:

8.37.3.16 def forcebalance.molecule.Molecule.atom_select (*self, atomslice*)

Return a copy of the object with certain atoms selected.

Takes an integer, list or array as argument.

Definition at line 1045 of file molecule.py.

8.37.3.17 def forcebalance.molecule.Molecule.atom_stack (*self, other*)

Return a copy of the object with another molecule object appended.

WARNING: This function may invalidate stuff like QM energies.

Definition at line 1080 of file molecule.py.

Here is the call graph for this function:

8.37.3.18 def forcebalance.molecule.Molecule.build_topology (*self, sn* = `None`, *Fac* = `1.2`)

A bare-bones implementation of the bond graph capability in the nanoreactor code.

Returns a NetworkX graph that depicts the molecular topology, which might be useful for stuff. Provide, optionally, the frame number used to compute the topology.

Definition at line 1162 of file molecule.py.

Here is the call graph for this function:

8.37.3.19 def forcebalance.molecule.Molecule.center_of_mass (*self*)

Definition at line 952 of file molecule.py.

Here is the call graph for this function:

8.37.3.20 def forcebalance.molecule.Molecule.edit_qcrems (*self, in_dict, subcalc* = `None`)

Definition at line 986 of file molecule.py.

Here is the call graph for this function:

8.37.3.21 def forcebalance.molecule.Molecule.load_frames (*self, fnm*)

Definition at line 979 of file molecule.py.

8.37.3.22 def forcebalance.molecule.Molecule.measure_dihedrals (*self, i, j, k, l*)

Return a series of dihedral angles, given four atom indices numbered from zero.

Definition at line 1204 of file molecule.py.

Here is the call graph for this function:

8.37.3.23 def forcebalance.molecule.Molecule.openmm_boxes (*self*)

Returns the periodic box vectors in the Molecule object in a list of OpenMM-compatible boxes, so it is possible to type simulation.context.setPeriodicBoxVectors(Mol.openmm_boxes()[0]) or something like that.

Definition at line 1273 of file molecule.py.

8.37.3.24 def forcebalance.molecule.Molecule.openmm_positions (*self*)

Returns the Cartesian coordinates in the Molecule object in a list of OpenMM-compatible positions, so it is possible to type simulation.context.setPositions(Mol.openmm_positions()[0]) or something like that.

Definition at line 1256 of file molecule.py.

Here is the call graph for this function:

8.37.3.25 def forcebalance.molecule.Molecule.radius_of_gyration (*self*)

Definition at line 956 of file molecule.py.

Here is the call graph for this function:

8.37.3.26 def forcebalance.molecule.Molecule.read_arc (*self, fnm*)

Read a TINKER .arc file.

Parameters

in	*fnm*	The input file name

Returns

xyzs A list for the XYZ coordinates.
resid The residue ID numbers. These are not easy to get!
elem A list of chemical elements in the XYZ file
comms A single-element list for the comment.
tinkersuf The suffix that comes after lines in the XYZ coordinates; this is usually topology info

Definition at line 1551 of file molecule.py.

Here is the call graph for this function:

8.37.3.27 def forcebalance.molecule.Molecule.read_charmm (*self, fnm*)

Read a CHARMM .cor (or .crd) file.

Definition at line 1690 of file molecule.py.

Here is the call graph for this function:

8.37.3.28 def forcebalance.molecule.Molecule.read_com (*self, fnm*)

Parse a Gaussian .com file and return a SINGLE-ELEMENT list of xyz coordinates (no multiple file support)

Parameters

in	*fnm*	The input file name

Returns

elem A list of chemical elements in the XYZ file
comms A single-element list for the comment.
xyzs A single-element list for the XYZ coordinates.
charge The total charge of the system.
mult The spin multiplicity of the system.

Definition at line 1508 of file molecule.py.

Here is the call graph for this function:

8.37.3.29 def forcebalance.molecule.Molecule.read_dcd (*self, fnm*)

Definition at line 1469 of file molecule.py.

Here is the call graph for this function:

8.37.3.30 def forcebalance.molecule.Molecule.read_gro (*self, fnm*)

Read a GROMACS .gro file.

Definition at line 1614 of file molecule.py.

Here is the call graph for this function:

8.37.3.31 def forcebalance.molecule.Molecule.read_mdcrd (*self, fnm*)

Parse an AMBER .mdcrd file.

This requires at least the number of atoms. This will FAIL for monatomic trajectories (but who the heck makes those?)

Parameters

in	*fnm*	The input file name

Returns

xyzs A list of XYZ coordinates (number of snapshots times number of atoms)
boxes Boxes (if present.)

Definition at line 1360 of file molecule.py.

Here is the call graph for this function:

8.37.3.32 def forcebalance.molecule.Molecule.read_mol2 (*self, fnm*)

Definition at line 1420 of file molecule.py.

Here is the call graph for this function:

8.37.3.33 def forcebalance.molecule.Molecule.read_pdb (*self, fnm*)

Loads a PDB and returns a dictionary containing its data.

Definition at line 1870 of file molecule.py.

Here is the call graph for this function:

8.37.3.34 def forcebalance.molecule.Molecule.read_qcesp (*self, fnm*)

Definition at line 1945 of file molecule.py.

8.37.3.35 def forcebalance.molecule.Molecule.read_qcin (*self, fnm*)

Read a Q-Chem input file.

These files can be very complicated, and I can't write a completely general parser for them. It is important to keep our goal in mind:

1) The main goal is to convert a trajectory to Q-Chem input files with identical calculation settings.

2) When we print the Q-Chem file, we should preserve the line ordering of the 'rem' section, but also be able to add 'rem' options at the end.

3) We should accommodate the use case that the Q-Chem file may have follow-up calculations delimited by '@@'.

4) We can read in all of the xyz's as a trajectory, but only the Q-Chem settings belonging to the first xyz will be saved.

Definition at line 1758 of file molecule.py.

Here is the call graph for this function:

8.37.3.36 def forcebalance.molecule.Molecule.read_qcout (*self, fnm*)

Q-Chem output file reader, adapted for our parser.

Q-Chem output files are very flexible and there's no way I can account for all of them. Here's what I am able to account for:

A list of:

- Coordinates
- Energies
- Forces

Note that each step in a geometry optimization counts as a frame.

As with all Q-Chem output files, note that successive calculations can have different numbers of atoms.

Definition at line 1974 of file molecule.py.

Here is the call graph for this function:

8.37.3.37 def forcebalance.molecule.Molecule.read_qdata (*self, fnm*)

Definition at line 1385 of file molecule.py.

8.37.3.38 def forcebalance.molecule.Molecule.read_xyz (*self, fnm*)

Parse a .xyz file which contains several xyz coordinates, and return their elements.

Parameters

in	*fnm*	The input file name

Returns

elem A list of chemical elements in the XYZ file
comms A list of comments.
xyzs A list of XYZ coordinates (number of snapshots times number of atoms)

Definition at line 1314 of file molecule.py.

Here is the call graph for this function:

8.37.3.39 def forcebalance.molecule.Molecule.replace_peratom (*self, key, orig, want*)

Replace all of the data for a certain attribute in the system from orig to want.

Definition at line 1024 of file molecule.py.

8.37.3.40 def forcebalance.molecule.Molecule.replace_peratom_conditional (*self, key1, cond, key2, orig, want*)

Replace all of the data for a attribute key2 from orig to want, contingent on key1 being equal to cond.

For instance: replace H1 with H2 if resname is SOL.

Definition at line 1035 of file molecule.py.

Here is the call graph for this function:

8.37.3.41 def forcebalance.molecule.Molecule.require (*self, args*)

Definition at line 903 of file molecule.py.

8.37.3.42 def forcebalance.molecule.Molecule.require_boxes (*self*)

Definition at line 2447 of file molecule.py.

Here is the call graph for this function:

8.37.3.43 def forcebalance.molecule.Molecule.require_resid (*self*)

Definition at line 2434 of file molecule.py.

8.37.3.44 def forcebalance.molecule.Molecule.require_resname (*self*)

Definition at line 2442 of file molecule.py.

Here is the call graph for this function:

8.37.3.45 def forcebalance.molecule.Molecule.split (*self*, *fnm* = `None`, *ftype* = `None`, *method* = `"chunks"`, *num* = `None`)

Split the molecule object into a number of separate files (chunks), either by specifying the number of frames per chunk or the number of chunks.

Only relevant for "trajectories". The type of file may be specified; if they aren't specified then the original file type is used.

The output file names are [name].[numbers].[extension] where [name] can be specified by passing 'fnm' or taken from the object's 'fnm' attribute by default. [numbers] are integers ranging from the lowest to the highest chunk number, prepended by zeros.

If the number of chunks / frames is not specified, then one file is written for each frame.

Returns

fnms A list of the file names that were written.

Definition at line 1296 of file molecule.py.

Here is the call graph for this function:

8.37.3.46 def forcebalance.molecule.Molecule.write (*self, fnm* = `None`, *ftype* = `None`, *append* = `False`, *select* = `None`)

Definition at line 918 of file molecule.py.

Here is the call graph for this function:

8.37.3.47 def forcebalance.molecule.Molecule.write_arc (*self, select*)

Definition at line 2231 of file molecule.py.

Here is the call graph for this function:

8.37.3.48 def forcebalance.molecule.Molecule.write_dcd (*self, select*)

Definition at line 2267 of file molecule.py.

Here is the call graph for this function:

8.37.3.49 def forcebalance.molecule.Molecule.write_gro (*self, select*)

Definition at line 2243 of file molecule.py.

8.37.3.50 def forcebalance.molecule.Molecule.write_mdcrd (*self, select*)

Definition at line 2220 of file molecule.py.

8.37.3.51 def forcebalance.molecule.Molecule.write_molproq (*self, select*)

Definition at line 2208 of file molecule.py.

Here is the call graph for this function:

8.37.3.52 def forcebalance.molecule.Molecule.write_pdb (*self, select*)

Save to a PDB.

Copied wholesale from MSMBuilder.

COLUMNS TYPE FIELD DEFINITION

7-11 int serial Atom serial number. 13-16 string name Atom name. 17 string altLoc Alternate location indicator. 18-20 (17-21 KAB) string resName Residue name. 22 string chainID Chain identifier. 23-26 int resSeq Residue sequence number. 27 string iCode Code for insertion of residues. 31-38 float x Orthogonal coordinates for X in Angstroms. 39-46 float y Orthogonal coordinates for Y in Angstroms. 47-54 float z Orthogonal coordinates for Z in Angstroms. 55-60 float occupancy Occupancy. 61-66 float tempFactor Temperature factor. 73-76 string segID Segment identifier, left-justified. 77-78 string element Element symbol, right-justified. 79-80 string charge Charge on the atom.

CRYST1 line, added by Lee-Ping

COLUMNS TYPE FIELD DEFINITION

7-15 float a a (Angstroms). 16-24 float b b (Angstroms). 25-33 float c c (Angstroms). 34-40 float alpha alpha (degrees). 41-47 float beta beta (degrees). 48-54 float gamma gamma (degrees). 56-66 string sGroup Space group. 67-70 int z Z value.

Definition at line 2323 of file molecule.py.

Here is the call graph for this function:

8.37.3.53 def forcebalance.molecule.Molecule.write_qcin (*self, select*)

Definition at line 2151 of file molecule.py.

Here is the call graph for this function:

8.37.3.54 def forcebalance.molecule.Molecule.write_qdata (*self, select*)

Text quantum data format.

Definition at line 2413 of file molecule.py.

Here is the call graph for this function:

8.37.3.55 def forcebalance.molecule.Molecule.write_xyz (*self, select*)

Definition at line 2197 of file molecule.py.

Here is the call graph for this function:

8.37.4 Member Data Documentation

8.37.4.1 forcebalance.molecule.Molecule.boxes

Definition at line 2495 of file molecule.py.

8.37.4.2 forcebalance.molecule.Molecule.comms

Read in stuff if we passed in a file name, otherwise return an empty instance.

Try to determine from the file name using the extension. Actually read the file. Set member variables. Create a list of comment lines if we don't already have them from reading the file.

Definition at line 882 of file molecule.py.

8.37.4.3 forcebalance.molecule.Molecule.Data

Definition at line 865 of file molecule.py.

8.37.4.4 forcebalance.molecule.Molecule.fout

Fill in comments.

I needed to add in this line because the DCD writer requires the file name, but the other methods don't.

Definition at line 929 of file molecule.py.

8.37.4.5 forcebalance.molecule.Molecule.Funnel

A funnel dictionary that takes redundant file types and maps them down to a few.

Definition at line 847 of file molecule.py.

8.37.4.6 forcebalance.molecule.Molecule.molecules

Definition at line 893 of file molecule.py.

8.37.4.7 forcebalance.molecule.Molecule.positive_resid

Creates entries like 'gromacs' : 'gromacs' and 'xyz' : 'xyz' in the Funnel.

Definition at line 861 of file molecule.py.

8.37.4.8 forcebalance.molecule.Molecule.Read_Tab

The table of file readers.

Definition at line 822 of file molecule.py.

8.37.4.9 forcebalance.molecule.Molecule.resid

Definition at line 2438 of file molecule.py.

8.37.4.10 forcebalance.molecule.Molecule.resname

Definition at line 2445 of file molecule.py.

8.37.4.11 forcebalance.molecule.Molecule.topology

Make sure the comment line isn't too long for i in range(len(self.comms)): self.comms[i] = self.comms[i][:100] if len(self.comms[i]) $>$ 100 else self.comms[i] Attempt to build the topology for small systems.

:)

Definition at line 892 of file molecule.py.

8.37.4.12 forcebalance.molecule.Molecule.Write_Tab

The table of file writers.

Definition at line 836 of file molecule.py.

The documentation for this class was generated from the following file:

- molecule.py

8.38 forcebalance.molecule.MolfileTimestep Class Reference

Wrapper for the timestep C structure used in molfile plugins.

Inheritance diagram for forcebalance.molecule.MolfileTimestep:

Collaboration diagram for forcebalance.molecule.MolfileTimestep:

8.38.1 Detailed Description

Wrapper for the timestep C structure used in molfile plugins.

Definition at line 469 of file molecule.py.

The documentation for this class was generated from the following file:

- molecule.py

8.39 forcebalance.moments.Moments Class Reference

Subclass of Target for fitting force fields to multipole moments (from experiment or theory).

Inheritance diagram for forcebalance.moments.Moments:

Collaboration diagram for forcebalance.moments.Moments:

Public Member Functions

- def __init__

  Initialization.
- def read_reference_data

  Read the reference data from a file.
- def prepare_temp_directory

  Prepare the temporary directory.
- def indicate

  Print qualitative indicator.
- def unpack_moments
- def get

  Evaluate objective function.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

Stages the directory for the target, and then calls 'get'.

- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- denoms
- mfnm

  The mdata.txt file that contains the moments.
- ref_moments
- na

  Number of atoms.
- ref_eigvals
- ref_eigvecs
- calc_moments
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.39.1 Detailed Description

Subclass of Target for fitting force fields to multipole moments (from experiment or theory).

Currently Tinker is supported.

Definition at line 30 of file moments.py.

8.39.2 Constructor & Destructor Documentation

8.39.2.1 def forcebalance.moments.Moments.__init__ (*self, options, tgt_opts, forcefield*)

Initialization.

Definition at line 35 of file moments.py.

Here is the call graph for this function:

8.39.3 Member Function Documentation

8.39.3.1 def forcebalance.moments.Moments.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Evaluate objective function.

Definition at line 173 of file moments.py.

Here is the call graph for this function:

8.39.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.39.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.39.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.39.3.5 def forcebalance.moments.Moments.indicate (*self*)

Print qualitative indicator.

Definition at line 141 of file moments.py.

Here is the call graph for this function:

8.39.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.39.3.7 def forcebalance.moments.Moments.prepare_temp_directory (*self, options, tgt_opts*)

Prepare the temporary directory.

Definition at line 136 of file moments.py.

8.39.3.8 def forcebalance.target.Target.printcool_table (*self, data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.39.3.9 def forcebalance.moments.Moments.read_reference_data (*self*)

Read the reference data from a file.

Definition at line 67 of file moments.py.

8.39.3.10 def forcebalance.target.Target.refresh_temp_directory (*self*) `[inherited]`

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.39.3.11 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.39.3.12 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.39.3.13 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.39.3.14 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.39.3.15 def forcebalance.moments.Moments.unpack_moments (*self, moment_dict*)

Definition at line 167 of file moments.py.

Here is the call graph for this function:

8.39.3.16 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.39.4 Member Data Documentation

8.39.4.1 forcebalance.moments.Moments.calc_moments

Definition at line 201 of file moments.py.

8.39.4.2 forcebalance.moments.Moments.denoms

Definition at line 48 of file moments.py.

8.39.4.3 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.39.4.4 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.39.4.5 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.39.4.6 forcebalance.moments.Moments.mfnm

The mdata.txt file that contains the moments.

Definition at line 57 of file moments.py.

8.39.4.7 forcebalance.moments.Moments.na

Number of atoms.

Definition at line 69 of file moments.py.

8.39.4.8 forcebalance.moments.Moments.objective

Definition at line 202 of file moments.py.

8.39.4.9 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.39.4.10 forcebalance.moments.Moments.ref_eigvals

Definition at line 70 of file moments.py.

8.39.4.11 forcebalance.moments.Moments.ref_eigvecs

Definition at line 71 of file moments.py.

8.39.4.12 forcebalance.moments.Moments.ref_moments

Definition at line 58 of file moments.py.

8.39.4.13 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.39.4.14 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.39.4.15 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.39.4.16 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- moments.py

8.40 forcebalance.tinkerio.Moments_TINKER Class Reference

Subclass of Target for multipole moment matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.Moments_TINKER:

Collaboration diagram for forcebalance.tinkerio.Moments_TINKER:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def moments_driver
- def read_reference_data

  Read the reference data from a file.
- def indicate

  Print qualitative indicator.
- def unpack_moments
- def get

  Evaluate objective function.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.

- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- denoms
- mfnm

  The mdata.txt file that contains the moments.
- ref_moments
- na

  Number of atoms.
- ref_eigvals
- ref_eigvecs
- calc_moments
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.40.1 Detailed Description

Subclass of Target for multipole moment matching using TINKER.

Definition at line 405 of file tinkerio.py.

8.40.2 Constructor & Destructor Documentation

8.40.2.1 def forcebalance.tinkerio.Moments_TINKER.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 408 of file tinkerio.py.

Here is the call graph for this function:

8.40.3 Member Function Documentation

8.40.3.1 def forcebalance.moments.Moments.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Evaluate objective function.

Definition at line 173 of file moments.py.

Here is the call graph for this function:

8.40.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.40.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.40.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.40.3.5 def forcebalance.moments.Moments.indicate (*self*) [inherited]

Print qualitative indicator.

Definition at line 141 of file moments.py.

Here is the call graph for this function:

8.40.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.40.3.7 def forcebalance.tinkerio.Moments_TINKER.moments_driver (*self*)

Definition at line 424 of file tinkerio.py.

Here is the call graph for this function:

8.40.3.8 def forcebalance.tinkerio.Moments_TINKER.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 413 of file tinkerio.py.

Here is the call graph for this function:

8.40.3.9 def forcebalance.target.Target.printcool_table (*self, data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.

banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.40.3.10 def forcebalance.moments.Moments.read_reference_data (*self*) [inherited]

Read the reference data from a file.

Definition at line 67 of file moments.py.

8.40.3.11 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.40.3.12 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.40.3.13 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.40.3.14 def forcebalance.target.Target.stage (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.40.3.15 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

Definition at line 291 of file target.py.

8.40.3.16 def forcebalance.moments.Moments.unpack_moments (*self, moment_dict*) [inherited]

Definition at line 167 of file moments.py.

Here is the call graph for this function:

8.40.3.17 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.40.4 Member Data Documentation

8.40.4.1 forcebalance.moments.Moments.calc_moments [inherited]

Definition at line 201 of file moments.py.

8.40.4.2 forcebalance.moments.Moments.denoms [inherited]

Definition at line 48 of file moments.py.

8.40.4.3 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.40.4.4 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.40.4.5 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.40.4.6 forcebalance.moments.Moments.mfnm [inherited]

The mdata.txt file that contains the moments.

Definition at line 57 of file moments.py.

8.40.4.7 forcebalance.moments.Moments.na [inherited]

Number of atoms.

Definition at line 69 of file moments.py.

8.40.4.8 forcebalance.moments.Moments.objective [inherited]

Definition at line 202 of file moments.py.

8.40.4.9 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.40.4.10 forcebalance.moments.Moments.ref_eigvals [inherited]

Definition at line 70 of file moments.py.

8.40.4.11 forcebalance.moments.Moments.ref_eigvecs [inherited]

Definition at line 71 of file moments.py.

8.40.4.12 forcebalance.moments.Moments.ref_moments [inherited]

Definition at line 58 of file moments.py.

8.40.4.13 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.40.4.14 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.40.4.15 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.40.4.16 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- tinkerio.py

8.41 forcebalance.gmxqpio.Monomer_QTPIE Class Reference

Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).

Inheritance diagram for forcebalance.gmxqpio.Monomer_QTPIE:

Collaboration diagram for forcebalance.gmxqpio.Monomer_QTPIE:

Public Member Functions

- def __init__
- def indicate

  Print qualitative indicator.
- def prepare_temp_directory
- def unpack_moments
- def get
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.

- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- ref_moments
- weights
- calc_moments
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.41.1 Detailed Description

Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).

Definition at line 130 of file gmxqpio.py.

8.41.2 Constructor & Destructor Documentation

8.41.2.1 def forcebalance.gmxqpio.Monomer_QTPIE.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 132 of file gmxqpio.py.

Here is the call graph for this function:

8.41.3 Member Function Documentation

8.41.3.1 def forcebalance.gmxqpio.Monomer_QTPIE.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Definition at line 221 of file gmxqpio.py.

Here is the call graph for this function:

8.41.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.41.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.41.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.41.3.5 def forcebalance.gmxqpio.Monomer_QTPIE.indicate (*self*)

Print qualitative indicator.

Definition at line 179 of file gmxqpio.py.

Here is the call graph for this function:

8.41.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.41.3.7 def forcebalance.gmxqpio.Monomer_QTPIE.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 200 of file gmxqpio.py.

Here is the call graph for this function:

8.41.3.8 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.41.3.9 def forcebalance.target.Target.refresh_temp_directory (*self*) `[inherited]`

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.41.3.10 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.41.3.11 def forcebalance.target.Target.sget (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.41.3.12 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.41.3.13 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.41.3.14 def forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments (*self, moment_dict*)

Definition at line 217 of file gmxqpio.py.

Here is the call graph for this function:

8.41.3.15 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.41.4 Member Data Documentation

8.41.4.1 forcebalance.gmxqpio.Monomer_QTPIE.calc_moments

Definition at line 258 of file gmxqpio.py.

8.41.4.2 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.41.4.3 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.41.4.4 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.41.4.5 forcebalance.gmxqpio.Monomer_QTPIE.objective

Definition at line 259 of file gmxqpio.py.

8.41.4.6 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.41.4.7 forcebalance.gmxqpio.Monomer_QTPIE.ref_moments

Definition at line 151 of file gmxqpio.py.

8.41.4.8 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.41.4.9 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.41.4.10 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.41.4.11 forcebalance.gmxqpio.Monomer_QTPIE.weights

Definition at line 165 of file gmxqpio.py.

8.41.4.12 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- gmxqpio.py

8.42 forcebalance.objective.Objective Class Reference

Objective function.

Inheritance diagram for forcebalance.objective.Objective:

Collaboration diagram for forcebalance.objective.Objective:

Public Member Functions

- def __init__
- def Target_Terms
- def Indicate

  Print objective function contributions.
- def Full
- def set_option

Public Attributes

- Targets

  Work Queue Port (The specific target itself may or may not actually use this.)
- FF

  The force field (it seems to be everywhere)
- Penalty

  Initialize the penalty function.
- WTot

  Obtain the denominator.
- ObjDict
- ObjDict_Last
- PrintOptionDict
- verbose_options

8.42.1 Detailed Description

Objective function.

```
The objective function is a combination of contributions from the different
fitting targets.  Basically, it loops through the targets, gets their
contributions to the objective function and then sums all of them
(although more elaborate schemes are conceivable).  The return value is the
same data type as calling the target itself: a dictionary containing
the objective function, the gradient and the Hessian.

The penalty function is also computed here; it keeps the parameters from straying
too far from their initial values.
```

Parameters

in	*mvals*	The mathematical parameters that enter into computing the objective function
in	*Order*	The requested order of differentiation

Definition at line 114 of file objective.py.

8.42.2 Constructor & Destructor Documentation

8.42.2.1 def forcebalance.objective.Objective.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 115 of file objective.py.

Here is the call graph for this function:

8.42.3 Member Function Documentation

8.42.3.1 def forcebalance.objective.Objective.Full (*self, mvals, Order* = `0`, *verbose* = `False`)

Definition at line 260 of file objective.py.

Here is the call graph for this function:

8.42.3.2 def forcebalance.objective.Objective.Indicate (*self*)

Print objective function contributions.

Definition at line 222 of file objective.py.

Here is the call graph for this function:

8.42.3.3 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.42.3.4 def forcebalance.objective.Objective.Target_Terms (*self, mvals, Order* = `0`, *verbose* = `False`)

Definition at line 162 of file objective.py.

Here is the call graph for this function:

8.42.4 Member Data Documentation

8.42.4.1 forcebalance.objective.Objective.FF

The force field (it seems to be everywhere)

Definition at line 141 of file objective.py.

8.42.4.2 forcebalance.objective.Objective.ObjDict

Definition at line 151 of file objective.py.

8.42.4.3 forcebalance.objective.Objective.ObjDict_Last

Definition at line 152 of file objective.py.

8.42.4.4 forcebalance.objective.Objective.Penalty

Initialize the penalty function.

Definition at line 143 of file objective.py.

8.42.4.5 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.42.4.6 forcebalance.objective.Objective.Targets

Work Queue Port (The specific target itself may or may not actually use this.)

Asynchronous objective function evaluation (i.e. execute Work Queue and local objective concurrently.) The list of fitting targets

Definition at line 130 of file objective.py.

8.42.4.7 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.42.4.8 forcebalance.objective.Objective.WTot

Obtain the denominator.

Definition at line 148 of file objective.py.

The documentation for this class was generated from the following file:

- objective.py

8.43 forcebalance.openmmio.OpenMM_Reader Class Reference

Class for parsing OpenMM force field files.

Inheritance diagram for forcebalance.openmmio.OpenMM_Reader:

Collaboration diagram for forcebalance.openmmio.OpenMM_Reader:

Public Member Functions

- def __init__
- def build_pid

  Build the parameter identifier (see link for an example)
- def Split

- def Whites
- def feed
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- pdict

  Initialize the superclass.
- ln
- itype
- suffix
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.43.1 Detailed Description

Class for parsing OpenMM force field files.

Definition at line 304 of file openmmio.py.

8.43.2 Constructor & Destructor Documentation

8.43.2.1 def forcebalance.openmmio.OpenMM_Reader.__init__ (*self, fnm*)

Definition at line 305 of file openmmio.py.

Here is the call graph for this function:

8.43.3 Member Function Documentation

8.43.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.43.3.2 def forcebalance.openmmio.OpenMM_Reader.build_pid (*self, element, parameter*)

Build the parameter identifier (see *link* for an example)

Todo Add a link here

Definition at line 314 of file openmmio.py.

Here is the call graph for this function:

8.43.3.3 def forcebalance.BaseReader.feed (*self, line*) [inherited]

Definition at line 88 of file __init__.py.

Here is the call graph for this function:

8.43.3.4 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.43.3.5 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.43.4 Member Data Documentation

8.43.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.43.4.2 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.43.4.3 forcebalance.BaseReader.itype [inherited]

Definition at line 68 of file __init__.py.

8.43.4.4 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.43.4.5 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.43.4.6 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.43.4.7 forcebalance.openmmio.OpenMM_Reader.pdict

Initialize the superclass.

:) The parameter dictionary (defined in this file)

Definition at line 309 of file openmmio.py.

8.43.4.8 forcebalance.BaseReader.suffix [inherited]

Definition at line 69 of file __init__.py.

The documentation for this class was generated from the following file:

- openmmio.py

8.44 forcebalance.optimizer.Optimizer Class Reference

Optimizer class.

Inheritance diagram for forcebalance.optimizer.Optimizer:

Collaboration diagram for forcebalance.optimizer.Optimizer:

Public Member Functions

- def __init__

  Create an Optimizer object.
- def Run

  Call the appropriate optimizer.
- def MainOptimizer

  The main ForceBalance adaptive trust-radius pseudo-Newton optimizer.
- def step

  Computes the next step in the parameter space.
- def NewtonRaphson

  Optimize the force field parameters using the Newton-Raphson method (.
- def BFGS

  Optimize the force field parameters using the BFGS method; currently the recommended choice (.
- def ScipyOptimizer

  Driver for SciPy optimizations.
- def GeneticAlgorithm

  Genetic algorithm, under development.
- def Simplex

  Use SciPy's built-in simplex algorithm to optimize the parameters.
- def Powell

  Use SciPy's built-in Powell direction-set algorithm to optimize the parameters.
- def Anneal

  Use SciPy's built-in simulated annealing algorithm to optimize the parameters.
- def ConjugateGradient

  Use SciPy's built-in simulated annealing algorithm to optimize the parameters.
- def Scan_Values

  Scan through parameter values.
- def ScanMVals

  Scan through the mathematical parameter space.
- def ScanPVals

  Scan through the physical parameter space.
- def SinglePoint

  A single-point objective function computation.
- def Gradient

  A single-point gradient computation.
- def Hessian

  A single-point Hessian computation.
- def FDCheckG

  Finite-difference checker for the objective function gradient.
- def FDCheckH

  Finite-difference checker for the objective function Hessian.
- def readchk

  Read the checkpoint file for the main optimizer.
- def writechk

  Write the checkpoint file for the main optimizer.
- def set_option

Public Attributes

- OptTab

  A list of all the things we can ask the optimizer to do.
- Objective

  The root directory.
- bhyp

  Whether the penalty function is hyperbolic.
- FF

  The force field itself.
- excision

  The indices to be excluded from the Hessian update.
- np

  Number of parameters.
- mvals0

  The original parameter values.
- chk

  Put data into the checkpoint file.
- H
- dx
- Val
- Grad
- Hess
- Penalty
- PrintOptionDict
- verbose_options

8.44.1 Detailed Description

Optimizer class.

Contains several methods for numerical optimization.

For various reasons, the optimizer depends on the force field and fitting targets (i.e. we cannot treat it as a fully independent numerical optimizer). The dependency is rather weak which suggests that I can remove it someday.

Definition at line 44 of file optimizer.py.

8.44.2 Constructor & Destructor Documentation

8.44.2.1 def forcebalance.optimizer.Optimizer.__init__ (*self, options, Objective, FF*)

Create an Optimizer object.

The optimizer depends on both the FF and the fitting targets so there is a chain of dependencies: FF –> FitSim –> Optimizer, and FF –> Optimizer

Here's what we do:

- Take options from the parser
- Pass in the objective function, force field, all fitting targets

Definition at line 57 of file optimizer.py.

Here is the call graph for this function:

8.44.3 Member Function Documentation

8.44.3.1 def forcebalance.optimizer.Optimizer.Anneal (*self*)

Use SciPy's built-in simulated annealing algorithm to optimize the parameters.

See Also

Optimizer::ScipyOptimizer

Definition at line 784 of file optimizer.py.

8.44.3.2 def forcebalance.optimizer.Optimizer.BFGS (*self*)

Optimize the force field parameters using the BFGS method; currently the recommended choice (.

See Also

MainOptimizer)

Definition at line 605 of file optimizer.py.

8.44.3.3 def forcebalance.optimizer.Optimizer.ConjugateGradient (*self*)

Use SciPy's built-in simulated annealing algorithm to optimize the parameters.

See Also

Optimizer::ScipyOptimizer

Definition at line 789 of file optimizer.py.

8.44.3.4 def forcebalance.optimizer.Optimizer.FDCheckG (*self*)

Finite-difference checker for the objective function gradient.

For each element in the gradient, use a five-point finite difference stencil to compute a finite-difference derivative, and compare it to the analytic result.

Definition at line 885 of file optimizer.py.

Here is the call graph for this function:

8.44.3.5 def forcebalance.optimizer.Optimizer.FDCheckH (*self*)

Finite-difference checker for the objective function Hessian.

For each element in the Hessian, use a five-point stencil in both parameter indices to compute a finite-difference derivative, and compare it to the analytic result.

This is meant to be a foolproof checker, so it is pretty slow. We could write a faster checker if we assumed we had accurate first derivatives, but it's better to not make that assumption.

The second derivative is computed by double-wrapping the objective function via the 'wrap2' function.

Definition at line 917 of file optimizer.py.

Here is the call graph for this function:

8.44.3.6 def forcebalance.optimizer.Optimizer.GeneticAlgorithm (*self*)

Genetic algorithm, under development.

It currently works but a genetic algorithm is more like a concept; i.e. there is no single way to implement it.

Todo Massive parallelization hasn't been implemented yet

Definition at line 681 of file optimizer.py.

Here is the call graph for this function:

8.44.3.7 def forcebalance.optimizer.Optimizer.Gradient (*self*)

A single-point gradient computation.

Definition at line 863 of file optimizer.py.

Here is the call graph for this function:

8.44.3.8 def forcebalance.optimizer.Optimizer.Hessian (*self*)

A single-point Hessian computation.

Definition at line 871 of file optimizer.py.

Here is the call graph for this function:

8.44.3.9 def forcebalance.optimizer.Optimizer.MainOptimizer (*self, b_BFGS* = 0)

The main ForceBalance adaptive trust-radius pseudo-Newton optimizer.

Tried and true in many situations. :)

```
Usually this function is called with the BFGS or NewtonRaphson
method.  The NewtonRaphson method is consistently the best
method I have, because I always provide at least an
approximate Hessian to the objective function.  The BFGS
method is vestigial and currently does not work.

BFGS is a pseudo-Newton method in the sense that it builds an
approximate Hessian matrix from the gradient information in previous
steps; Newton-Raphson requires the actual Hessian matrix.
However, the algorithms are similar in that they both compute the
step by inverting the Hessian and multiplying by the gradient.

The method adaptively changes the step size.  If the step is
sufficiently good (i.e. the objective function goes down by a
large fraction of the predicted decrease), then the step size
is increased; if the step is bad, then it rejects the step and
tries again.

The optimization is terminated after either a function value or
step size tolerance is reached.
```

Parameters

in	*b_BFGS*	Switch to use BFGS (True) or Newton-Raphson (False)

Definition at line 229 of file optimizer.py.

Here is the call graph for this function:

8.44.3.10 def forcebalance.optimizer.Optimizer.NewtonRaphson (*self*)

Optimize the force field parameters using the Newton-Raphson method (.

See Also

MainOptimizer)

Definition at line 600 of file optimizer.py.

Here is the call graph for this function:

8.44.3.11 def forcebalance.optimizer.Optimizer.Powell (*self*)

Use SciPy's built-in Powell direction-set algorithm to optimize the parameters.

See Also

Optimizer::ScipyOptimizer

Definition at line 779 of file optimizer.py.

8.44.3.12 def forcebalance.optimizer.Optimizer.readchk (*self*)

Read the checkpoint file for the main optimizer.

Definition at line 945 of file optimizer.py.

8.44.3.13 def forcebalance.optimizer.Optimizer.Run (*self*)

Call the appropriate optimizer.

This is the method we might want to call from an executable.

Definition at line 168 of file optimizer.py.

Here is the call graph for this function:

8.44.3.14 def forcebalance.optimizer.Optimizer.Scan_Values (*self*, *MathPhys* = 1)

Scan through parameter values.

```
This option is activated using the inputs:
```

```
scan[mp]vals
scan_vals low:hi:nsteps
scan_idxnum (number) -or-
scan_idxname (name)
```

This method goes to the specified parameter indices and scans through the supplied values, evaluating the objective function at every step.

I hope this method will be useful for people who just want to look at changing one or two parameters and seeing how it affects the force field performance.

Todo Maybe a multidimensional grid can be done.

Parameters

in	*MathPhys*	Switch to use mathematical (True) or physical (False) parameters.

Definition at line 815 of file optimizer.py.

Here is the call graph for this function:

8.44.3.15 def forcebalance.optimizer.Optimizer.ScanMVals (*self*)

Scan through the mathematical parameter space.

See Also

Optimizer::ScanValues

Definition at line 847 of file optimizer.py.

Here is the call graph for this function:

8.44.3.16 def forcebalance.optimizer.Optimizer.ScanPVals (*self*)

Scan through the physical parameter space.

See Also

Optimizer::ScanValues

Definition at line 852 of file optimizer.py.

8.44.3.17 def forcebalance.optimizer.Optimizer.ScipyOptimizer (*self, Algorithm* = "None")

Driver for SciPy optimizations.

```
Using any of the SciPy optimizers requires that SciPy is installed.
This method first defines several wrappers around the objective function that the SciPy
optimizers can use.  Then it calls the algorith mitself.
```

Parameters

in	*Algorithm*	The optimization algorithm to use, for example 'powell', 'simplex' or 'anneal'

Definition at line 618 of file optimizer.py.

Here is the call graph for this function:

8.44.3.18 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.44.3.19 def forcebalance.optimizer.Optimizer.Simplex (*self*)

Use SciPy's built-in simplex algorithm to optimize the parameters.

See Also

Optimizer::ScipyOptimizer

Definition at line 774 of file optimizer.py.

Here is the call graph for this function:

8.44.3.20 def forcebalance.optimizer.Optimizer.SinglePoint (*self*)

A single-point objective function computation.

Definition at line 857 of file optimizer.py.

Here is the call graph for this function:

8.44.3.21 def forcebalance.optimizer.Optimizer.step (*self, xk, data, trust*)

Computes the next step in the parameter space.

There are lots of tricks here that I will document later.

Parameters

in	*G*	The gradient
in	*H*	The Hessian
in	*trust*	The trust radius

Definition at line 421 of file optimizer.py.

Here is the call graph for this function:

8.44.3.22 def forcebalance.optimizer.Optimizer.writechk (*self*)

Write the checkpoint file for the main optimizer.

Definition at line 957 of file optimizer.py.

8.44.4 Member Data Documentation

8.44.4.1 forcebalance.optimizer.Optimizer.bhyp

Whether the penalty function is hyperbolic.

Definition at line 141 of file optimizer.py.

8.44.4.2 forcebalance.optimizer.Optimizer.chk

Put data into the checkpoint file.

Definition at line 272 of file optimizer.py.

8.44.4.3 forcebalance.optimizer.Optimizer.dx

Definition at line 450 of file optimizer.py.

8.44.4.4 forcebalance.optimizer.Optimizer.excision

The indices to be excluded from the Hessian update.

Definition at line 149 of file optimizer.py.

8.44.4.5 forcebalance.optimizer.Optimizer.FF

The force field itself.

Definition at line 143 of file optimizer.py.

8.44.4.6 forcebalance.optimizer.Optimizer.Grad

Definition at line 452 of file optimizer.py.

8.44.4.7 forcebalance.optimizer.Optimizer.H

Definition at line 449 of file optimizer.py.

8.44.4.8 forcebalance.optimizer.Optimizer.Hess

Definition at line 453 of file optimizer.py.

8.44.4.9 forcebalance.optimizer.Optimizer.mvals0

The original parameter values.

Sometimes the optimizer doesn't return anything (i.e.

in the case of a single point calculation) In these situations, don't do anything Check derivatives by finite difference after the optimization is over (for good measure)

Definition at line 155 of file optimizer.py.

8.44.4.10 forcebalance.optimizer.Optimizer.np

Number of parameters.

Definition at line 152 of file optimizer.py.

8.44.4.11 forcebalance.optimizer.Optimizer.Objective

The root directory.

The job type Initial step size trust radius Minimum trust radius (for noisy objective functions) Lower bound on Hessian eigenvalue (below this, we add in steepest descent) Guess value for Brent Step size for numerical finite difference Number of steps to average over Function value convergence threshold Step size convergence threshold Gradient convergence threshold Maximum number of optimization steps For scan[mp]vals: The parameter index to scan over For scan[mp]vals: The parameter name to scan over, it just looks up an index For scan[mp]vals: The values that are fed into the scanner Name of the checkpoint file that we're reading in Name of the checkpoint file that we're writing out Whether to write the checkpoint file at every step Adaptive trust radius adjustment factor Adaptive trust radius adjustment damping Whether to print gradient during each step of the optimization Whether to print Hessian during each step of the optimization Whether to print parameters during each step of the optimization Error tolerance (if objective function rises by less than this, then the optimizer will forge ahead!) Search tolerance (The nonlinear search will stop if the change is below this threshold) The objective function (needs to pass in when I instantiate)

Definition at line 139 of file optimizer.py.

8.44.4.12 forcebalance.optimizer.Optimizer.OptTab

A list of all the things we can ask the optimizer to do.

Definition at line 61 of file optimizer.py.

8.44.4.13 forcebalance.optimizer.Optimizer.Penalty

Definition at line 454 of file optimizer.py.

8.44.4.14 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.44.4.15 forcebalance.optimizer.Optimizer.Val

Definition at line 451 of file optimizer.py.

8.44.4.16 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

The documentation for this class was generated from the following file:

- optimizer.py

8.45 forcebalance.objective.Penalty Class Reference

Penalty functions for regularizing the force field optimizer.

Public Member Functions

- def __init__
- def compute
- def L2_norm

  Harmonic L2-norm constraints.
- def HYP

  Hyperbolic constraints.
- def FUSE
- def FUSE_BARRIER
- def FUSE_L0

Public Attributes

- fadd
- fmul
- a
- b
- FF
- ptyp
- Pen_Tab
- spacings

  Find exponential spacings.

Static Public Attributes

- dictionary Pen_Names

8.45.1 Detailed Description

Penalty functions for regularizing the force field optimizer.

The purpose for this module is to improve the behavior of our optimizer; essentially, our problem is fraught with 'linear dependencies', a.k.a. directions in the parameter space that the objective function does not respond to. This would happen if a parameter is just plain useless, or if there are two or more parameters that describe the same thing.

To accomplish these objectives, a penalty function is added to the objective function. Generally, the more the parameters change (i.e. the greater the norm of the parameter vector), the greater the penalty. Note that this is added on after all of the other contributions have been computed. This only matters if the penalty 'multiplies' the objective function: Obj + Obj∗Penalty, but we also have the option of an additive penalty: Obj + Penalty.

Statistically, this is called regularization. If the penalty function is the norm squared of the parameter vector, it is called ridge regression. There is also the option of using simply the norm, and this is called lasso, but I think it presents problems for the optimizer that I need to work out.

Note that the penalty functions can be considered as part of a 'maximum likelihood' framework in which we assume a PRIOR PROBABILITY of the force field parameters around their initial values. The penalty function is related to the prior by an exponential. Ridge regression corresponds to a Gaussian prior and lasso corresponds to an exponential prior. There is also 'elastic net regression' which interpolates between Gaussian and exponential using a tuning parameter.

Our priors are adjustable too - there is one parameter, which is the width of the distribution. We can even use a noninformative prior for the distribution widths (hyperprior!). These are all important things to consider later.

Importantly, note that here there is no code that treats the distribution width. That is because the distribution width is wrapped up in the rescaling factors, which is essentially a coordinate transformation on the parameter space. More documentation on this will follow, perhaps in the 'rsmake' method.

Definition at line 317 of file objective.py.

8.45.2 Constructor & Destructor Documentation

8.45.2.1 def forcebalance.objective.Penalty.__init__ (*self, User_Option, ForceField, Factor_Add* = `0.0`, *Factor_Mult* = `0.0`, *Factor_B* = `0.1`, *Alpha* = `1.0`)

Definition at line 323 of file objective.py.

8.45.3 Member Function Documentation

8.45.3.1 def forcebalance.objective.Penalty.compute (*self, mvals, Objective*)

Definition at line 349 of file objective.py.

Here is the call graph for this function:

8.45.3.2 def forcebalance.objective.Penalty.FUSE (*self, mvals*)

Definition at line 409 of file objective.py.

Here is the call graph for this function:

8.45.3.3 def forcebalance.objective.Penalty.FUSE_BARRIER (*self, mvals*)

Definition at line 450 of file objective.py.

Here is the call graph for this function:

8.45.3.4 def forcebalance.objective.Penalty.FUSE_L0 (*self, mvals*)

Definition at line 492 of file objective.py.

Here is the call graph for this function:

8.45.3.5 def forcebalance.objective.Penalty.HYP (*self, mvals*)

Hyperbolic constraints.

Depending on the 'b' parameter, the smaller it is, the closer we are to an L1-norm constraint. If we use these, we expect a properly-behaving optimizer to make several of the parameters very nearly zero (which would be cool).

Parameters

in	*mvals*	The parameter vector

Returns

DC0 The hyperbolic penalty
DC1 The gradient
DC2 The Hessian

Definition at line 401 of file objective.py.

Here is the call graph for this function:

8.45.3.6 def forcebalance.objective.Penalty.L2_norm (*self, mvals*)

Harmonic L2-norm constraints.

These are the ones that I use the most often to regularize my optimization.

Parameters

in	*mvals*	The parameter vector

Returns

DC0 The norm squared of the vector
DC1 The gradient of DC0
DC2 The Hessian (just a constant)

Definition at line 381 of file objective.py.

Here is the call graph for this function:

8.45.4 Member Data Documentation

8.45.4.1 forcebalance.objective.Penalty.a

Definition at line 326 of file objective.py.

8.45.4.2 forcebalance.objective.Penalty.b

Definition at line 327 of file objective.py.

8.45.4.3 forcebalance.objective.Penalty.fadd

Definition at line 324 of file objective.py.

8.45.4.4 forcebalance.objective.Penalty.FF

Definition at line 328 of file objective.py.

8.45.4.5 forcebalance.objective.Penalty.fmul

Definition at line 325 of file objective.py.

8.45.4.6 dictionary forcebalance.objective.Penalty.Pen_Names [static]

Initial value:

```
1 = {'HYP' : 1, 'HYPER' : 1, 'HYPERBOLIC' : 1, 'L1' : 1, 'HYPERBOLA' : 1,
2                        'PARA' : 2, 'PARABOLA' : 2, 'PARABOLIC' : 2, 'L2': 2, 'QUADRATIC' : 2,
3                        'FUSE' : 3, 'FUSION' : 3, 'FUSE_L0' : 4, 'FUSION_L0' : 4, 'FUSION-L0' : 4,
4                        'FUSE-BARRIER' : 5, 'FUSE-BARRIER' : 5, 'FUSE_BARRIER' : 5, 'FUSION_BARRIER' : 5}
```

Definition at line 318 of file objective.py.

8.45.4.7 forcebalance.objective.Penalty.Pen_Tab

Definition at line 330 of file objective.py.

8.45.4.8 forcebalance.objective.Penalty.ptyp

Definition at line 329 of file objective.py.

8.45.4.9 forcebalance.objective.Penalty.spacings

Find exponential spacings.

Definition at line 346 of file objective.py.

The documentation for this class was generated from the following file:

- objective.py

8.46 forcebalance.nifty.Pickler_LP Class Reference

A subclass of the python Pickler that implements pickling of _ElementTree types.

Inheritance diagram for forcebalance.nifty.Pickler_LP:

Collaboration diagram for forcebalance.nifty.Pickler_LP:

Public Member Functions

- def __init__

8.46.1 Detailed Description

A subclass of the python Pickler that implements pickling of _ElementTree types.

Definition at line 499 of file nifty.py.

8.46.2 Constructor & Destructor Documentation

8.46.2.1 def forcebalance.nifty.Pickler_LP.__init__ (*self, file, protocol* = `None`)

Definition at line 500 of file nifty.py.

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- nifty.py

8.47 forcebalance.qchemio.QCIn_Reader Class Reference

Finite state machine for parsing Q-Chem input files.

Inheritance diagram for forcebalance.qchemio.QCIn_Reader:

Collaboration diagram for forcebalance.qchemio.QCIn_Reader:

Public Member Functions

- def __init__

- def feed

  Feed in a line.
- def Split
- def Whites
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- atom
- snum
- cnum
- shell
- pdict
- sec
- itype
- suffix
- ln
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.47.1 Detailed Description

Finite state machine for parsing Q-Chem input files.

Definition at line 31 of file qchemio.py.

8.47.2 Constructor & Destructor Documentation

8.47.2.1 def forcebalance.qchemio.QCIn_Reader.__init__ (*self, fnm*)

Definition at line 33 of file qchemio.py.

8.47.3 Member Function Documentation

8.47.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.47.3.2 def forcebalance.qchemio.QCIn_Reader.feed (*self, line*)

Feed in a line.

Parameters

in	*line*	The line of data

Definition at line 48 of file qchemio.py.

8.47.3.3 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.47.3.4 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.47.4 Member Data Documentation

8.47.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.47.4.2 forcebalance.qchemio.QCIn_Reader.atom

Definition at line 36 of file qchemio.py.

8.47.4.3 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.47.4.4 forcebalance.qchemio.QCIn_Reader.cnum

Definition at line 38 of file qchemio.py.

8.47.4.5 forcebalance.qchemio.QCIn_Reader.itype

Definition at line 67 of file qchemio.py.

8.47.4.6 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.47.4.7 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.47.4.8 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.47.4.9 forcebalance.qchemio.QCIn_Reader.pdict

Definition at line 40 of file qchemio.py.

8.47.4.10 forcebalance.qchemio.QCIn_Reader.sec

Definition at line 58 of file qchemio.py.

8.47.4.11 forcebalance.qchemio.QCIn_Reader.shell

Definition at line 39 of file qchemio.py.

8.47.4.12 forcebalance.qchemio.QCIn_Reader.snum

Definition at line 37 of file qchemio.py.

8.47.4.13 forcebalance.qchemio.QCIn_Reader.suffix

Definition at line 72 of file qchemio.py.

The documentation for this class was generated from the following file:

- qchemio.py

8.48 forcebalance.output.RawFileHandler Class Reference

Exactly like output.FileHandler except it does no extra formatting before sending logging messages to the file.

Inheritance diagram for forcebalance.output.RawFileHandler:

Collaboration diagram for forcebalance.output.RawFileHandler:

Public Member Functions

- def emit

8.48.1 Detailed Description

Exactly like output.FileHandler except it does no extra formatting before sending logging messages to the file.

This is more compatible with how output has been displayed in ForceBalance.

Definition at line 47 of file output.py.

8.48.2 Member Function Documentation

8.48.2.1 def forcebalance.output.RawFileHandler.emit (*self, record*)

Definition at line 48 of file output.py.

The documentation for this class was generated from the following file:

- output.py

8.49 forcebalance.output.RawStreamHandler Class Reference

Exactly like output.StreamHandler except it does no extra formatting before sending logging messages to the stream.

Inheritance diagram for forcebalance.output.RawStreamHandler:

Collaboration diagram for forcebalance.output.RawStreamHandler:

Public Member Functions

- def __init__
- def emit

8.49.1 Detailed Description

Exactly like output.StreamHandler except it does no extra formatting before sending logging messages to the stream.

This is more compatible with how output has been displayed in ForceBalance. Default stream has also been changed from stderr to stdout

Definition at line 34 of file output.py.

8.49.2 Constructor & Destructor Documentation

8.49.2.1 def forcebalance.output.RawStreamHandler.__init__ (*self, stream =* `sys.stdout`)

Definition at line 35 of file output.py.

Here is the call graph for this function:

8.49.3 Member Function Documentation

8.49.3.1 def forcebalance.output.RawStreamHandler.emit (*self, record*)

Definition at line 38 of file output.py.

The documentation for this class was generated from the following file:

- output.py

8.50 forcebalance.psi4io.RDVR3_Psi4 Class Reference

Subclass of Target for R-DVR3 grid fitting.

Inheritance diagram for forcebalance.psi4io.RDVR3_Psi4:

Collaboration diagram for forcebalance.psi4io.RDVR3_Psi4:

Public Member Functions

- def __init__
- def indicate
- def submit_jobs

  Create a grid file and a psi4 input file in the absolute path and submit it to the work queue.
- def driver
- def get

  LPW 04-17-2013.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- objfiles

  Which parameters are differentiated?
- objvals
- elements
- molecules
- callderivs
- factor
- tdir
- objd
- gradd
- hdiagd
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)

- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.50.1 Detailed Description

Subclass of Target for R-DVR3 grid fitting.

Main features:

- Multiple molecules are treated as a single target.
- R-DVR3 can only print out the objective function, it cannot print out the residual vector.
- We should be smart enough to mask derivatives.

Definition at line 296 of file psi4io.py.

8.50.2 Constructor & Destructor Documentation

8.50.2.1 def forcebalance.psi4io.RDVR3_Psi4.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 299 of file psi4io.py.

8.50.3 Member Function Documentation

8.50.3.1 def forcebalance.psi4io.RDVR3_Psi4.driver (*self, mvals, d*)

Definition at line 416 of file psi4io.py.

Here is the call graph for this function:

8.50.3.2 def forcebalance.psi4io.RDVR3_Psi4.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

LPW 04-17-2013.

```
This subroutine builds the objective function from Psi4.
```

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 452 of file psi4io.py.

Here is the call graph for this function:

8.50.3.3 def forcebalance.target.Target.get_G (*self, mvals =* None) [inherited]

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.50.3.4 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.50.3.5 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.50.3.6 def forcebalance.psi4io.RDVR3_Psi4.indicate (*self*)

Definition at line 338 of file psi4io.py.

Here is the call graph for this function:

8.50.3.7 def forcebalance.target.Target.link_from_tempdir (*self*, *absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.50.3.8 def forcebalance.target.Target.printcool_table (*self*, *data* = OrderedDict([]), *headings* = [], *banner* = None, *footnote* = None, *color* = 0) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.50.3.9 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.50.3.10 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.50.3.11 def forcebalance.target.Target.sget (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.50.3.12 def forcebalance.target.Target.stage (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.50.3.13 def forcebalance.psi4io.RDVR3_Psi4.submit_jobs (*self, mvals, AGrad* = `True`, *AHess* = `True`)

Create a grid file and a psi4 input file in the absolute path and submit it to the work queue.

Definition at line 347 of file psi4io.py.

8.50.3.14 def forcebalance.target.Target.wq_complete (*self*) `[inherited]`

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.50.4 Member Data Documentation

8.50.4.1 forcebalance.psi4io.RDVR3_Psi4.callderivs

Definition at line 309 of file psi4io.py.

8.50.4.2 forcebalance.psi4io.RDVR3_Psi4.elements

Definition at line 307 of file psi4io.py.

8.50.4.3 forcebalance.psi4io.RDVR3_Psi4.factor

Definition at line 310 of file psi4io.py.

8.50.4.4 forcebalance.target.Target.FF `[inherited]`

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.50.4.5 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.50.4.6 forcebalance.psi4io.RDVR3_Psi4.gradd

Definition at line 462 of file psi4io.py.

8.50.4.7 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.50.4.8 forcebalance.psi4io.RDVR3_Psi4.hdiagd

Definition at line 463 of file psi4io.py.

8.50.4.9 forcebalance.psi4io.RDVR3_Psi4.molecules

Definition at line 308 of file psi4io.py.

8.50.4.10 forcebalance.psi4io.RDVR3_Psi4.objd

Definition at line 461 of file psi4io.py.

8.50.4.11 forcebalance.psi4io.RDVR3_Psi4.objective

Definition at line 539 of file psi4io.py.

8.50.4.12 forcebalance.psi4io.RDVR3_Psi4.objfiles

Which parameters are differentiated?

Definition at line 305 of file psi4io.py.

8.50.4.13 forcebalance.psi4io.RDVR3_Psi4.objvals

Definition at line 306 of file psi4io.py.

8.50.4.14 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.50.4.15 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.50.4.16 forcebalance.psi4io.RDVR3_Psi4.tdir

Definition at line 351 of file psi4io.py.

8.50.4.17 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.50.4.18 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.50.4.19 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- psi4io.py

8.51 forcebalance.target.RemoteTarget Class Reference

Inheritance diagram for forcebalance.target.RemoteTarget:

Collaboration diagram for forcebalance.target.RemoteTarget:

Public Member Functions

- def __init__
- def submit_jobs
- def get
- def indicate
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- r_options
- r_tgt_opts
- remote_indicate
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.51.1 Detailed Description

Definition at line 406 of file target.py.

8.51.2 Constructor & Destructor Documentation

8.51.2.1 def forcebalance.target.RemoteTarget.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 407 of file target.py.

8.51.3 Member Function Documentation

8.51.3.1 def forcebalance.target.RemoteTarget.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Definition at line 443 of file target.py.

8.51.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.51.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.51.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.51.3.5 def forcebalance.target.RemoteTarget.indicate (*self*)

Definition at line 449 of file target.py.

8.51.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.51.3.7 def forcebalance.target.Target.printcool_table (*self, data* = OrderedDict([]), *headings* = [], *banner* = None, *footnote* = None, *color* = 0) [inherited]

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.51.3.8 def forcebalance.target.Target.refresh_temp_directory (*self*) [inherited]

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.51.3.9 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.51.3.10 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.51.3.11 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) [inherited]

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.51.3.12 def forcebalance.target.RemoteTarget.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Definition at line 422 of file target.py.

Here is the call graph for this function:

8.51.3.13 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.51.4 Member Data Documentation

8.51.4.1 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.51.4.2 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.51.4.3 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.51.4.4 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.51.4.5 forcebalance.target.RemoteTarget.r_options

Definition at line 410 of file target.py.

8.51.4.6 forcebalance.target.RemoteTarget.r_tgt_opts

Definition at line 413 of file target.py.

8.51.4.7 forcebalance.target.RemoteTarget.remote_indicate

Definition at line 420 of file target.py.

8.51.4.8 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.51.4.9 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.51.4.10 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.51.4.11 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- target.py

8.52 forcebalance.target.Target Class Reference

Base class for all fitting targets.

Inheritance diagram for forcebalance.target.Target:

Collaboration diagram for forcebalance.target.Target:

Public Member Functions

- def __init__

  All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def get

  Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.
- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.52.1 Detailed Description

Base class for all fitting targets.

In ForceBalance a Target is defined as a set of reference data plus a corresponding method to simulate that data using the force field.

The 'computable quantities' may include energies and forces where the reference values come from QM calculations (energy and force matching), energies from an EDA analysis (Maybe in the future, FDA?), molecular properties (like polarizability, refractive indices, multipole moments or vibrational frequencies), relative entropies, and bulk properties. Single-molecule or bulk properties can even come from the experiment!

The central idea in ForceBalance is that each quantity makes a contribution to the overall objective function. So we can build force fields that fit several quantities at once, rather than putting all of our chips behind energy and force matching. In the future ForceBalance may even include multiobjective optimization into the optimizer.

The optimization is done by way of minimizing an 'objective function', which is comprised of squared differences between the computed and reference values. These differences are not computed in this file, but rather in subclasses that use Target as a base class. Thus, the contents of Target itself are meant to be as general as possible, because the pertinent variables apply to all types of fitting targets.

An important node: Target requires that all subclasses have a method get(self,mvals,AGrad=False,AHess=False) that does the following:

Inputs: mvals = The parameter vector, which modifies the force field (Note to self: We include mvals with each Target because we can create copies of the force field and do finite difference derivatives) AGrad, AHess = Boolean switches for computing analytic gradients and Hessians

Outputs: Answer = {'X': Number, 'G': numpy.array(np), 'H': numpy.array((np,np)) } 'X' = The objective function itself 'G' = The gradient, elements not computed analytically are zero 'H' = The Hessian, elements not computed analytically are zero

This is the only global requirement of a Target. Obviously 'get' itself is not defined here, because its calculation will depend entirely on specifically which target we wish to use. However, this should give us a unified framework which will faciliate rapid implementation of Targets.

Future work: Robert suggested that I could enable automatic detection of which parameters need to be computed by finite difference. Not a bad idea. :)

Definition at line 75 of file target.py.

8.52.2 Constructor & Destructor Documentation

8.52.2.1 def forcebalance.target.Target.__init__ (*self, options, tgt_opts, forcefield*)

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Definition at line 92 of file target.py.

Here is the call graph for this function:

8.52.3 Member Function Documentation

8.52.3.1 def forcebalance.target.Target.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.

See abinitio for an example.

Definition at line 254 of file target.py.

Here is the call graph for this function:

8.52.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`)

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.52.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`)

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.52.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`)

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.52.3.5 def forcebalance.target.Target.link_from_tempdir (*self*, *absdestdir*)

Definition at line 213 of file target.py.

8.52.3.6 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`)

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.52.3.7 def forcebalance.target.Target.refresh_temp_directory (*self*)

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.52.3.8 def forcebalance.BaseClass.set_option (*self*, *in_dict*, *src_key*, *dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.52.3.9 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = False, *AHess* = False, *customdir* = None)

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.52.3.10 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`)

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.52.3.11 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Definition at line 291 of file target.py.

8.52.3.12 def forcebalance.target.Target.wq_complete (*self*)

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.52.4 Member Data Documentation

8.52.4.1 forcebalance.target.Target.FF

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.52.4.2 forcebalance.target.Target.gct

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.52.4.3 forcebalance.target.Target.hct

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.52.4.4 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.52.4.5 forcebalance.target.Target.rundir

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.52.4.6 forcebalance.target.Target.tempdir

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.52.4.7 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.52.4.8 forcebalance.target.Target.xct

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- target.py

8.53 forcebalance.psi4io.THCDF_Psi4 Class Reference

Inheritance diagram for forcebalance.psi4io.THCDF_Psi4:

Collaboration diagram for forcebalance.psi4io.THCDF_Psi4:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def indicate
- def write_nested_destroy
- def driver
- def get

  LPW 05-30-2012.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.

- def sget

  Stages the directory for the target, and then calls 'get'.
- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- Molecules
- throw_outs
- Elements
- GBSfnm

  Psi4 basis set file.
- DATfnm

  Psi4 input file for calculation of linear dependencies This is actually a file in 'forcefield' until we can figure out a better system.
- MP2_Energy

  Actually run PSI4.
- DF_Energy
- call_derivatives

  Number of snapshots.
- MAQ

  Dictionary for derivative terms.
- D
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.53.1 Detailed Description

Definition at line 97 of file psi4io.py.

8.53.2 Constructor & Destructor Documentation

8.53.2.1 def forcebalance.psi4io.THCDF_Psi4.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 99 of file psi4io.py.

8.53.3 Member Function Documentation

8.53.3.1 def forcebalance.psi4io.THCDF_Psi4.driver (*self*)

Definition at line 172 of file psi4io.py.

Here is the call graph for this function:

8.53.3.2 def forcebalance.leastsq.LeastSquares.get (*self, mvals, AGrad* = `False`, *AHess* = `False`) [inherited]

LPW 05-30-2012.

```
This subroutine builds the objective function (and optionally
its derivatives) from a general software.

This subroutine interfaces with simulation software 'drivers'.
The driver is expected to give exact values, fitting values, and weights.
```

Parameters

in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 74 of file leastsq.py.

Here is the call graph for this function:

8.53.3.3 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.53.3.4 def forcebalance.target.Target.get_H (*self*, *mvals* = None) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.53.3.5 def forcebalance.target.Target.get_X (*self*, *mvals* = None) [inherited]

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.53.3.6 def forcebalance.psi4io.THCDF_Psi4.indicate (*self*)

Definition at line 152 of file psi4io.py.

Here is the call graph for this function:

8.53.3.7 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.53.3.8 def forcebalance.psi4io.THCDF_Psi4.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 141 of file psi4io.py.

Here is the call graph for this function:

8.53.3.9 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.53.3.10 def forcebalance.target.Target.refresh_temp_directory (*self*) `[inherited]`

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.53.3.11 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = `None`, *val* = `None`, *default* = `None`, *forceprint* = `False`) `[inherited]`

Definition at line 35 of file __init__.py.

8.53.3.12 def forcebalance.target.Target.sget (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.53.3.13 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.53.3.14 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.53.3.15 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.53.3.16 def forcebalance.psi4io.THCDF_Psi4.write_nested_destroy (*self, fnm, linedestroy*)

Definition at line 157 of file psi4io.py.

Here is the call graph for this function:

8.53.4 Member Data Documentation

8.53.4.1 forcebalance.leastsq.LeastSquares.call_derivatives [inherited]

Number of snapshots.

Which parameters are differentiated?

Definition at line 51 of file leastsq.py.

8.53.4.2 forcebalance.leastsq.LeastSquares.D [inherited]

Definition at line 137 of file leastsq.py.

8.53.4.3 forcebalance.psi4io.THCDF_Psi4.DATfnm

Psi4 input file for calculation of linear dependencies This is actually a file in 'forcefield' until we can figure out a better system.

Definition at line 137 of file psi4io.py.

8.53.4.4 forcebalance.psi4io.THCDF_Psi4.DF_Energy

Definition at line 238 of file psi4io.py.

8.53.4.5 forcebalance.psi4io.THCDF_Psi4.Elements

Definition at line 117 of file psi4io.py.

8.53.4.6 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.53.4.7 forcebalance.psi4io.THCDF_Psi4.GBSfnm

Psi4 basis set file.

Definition at line 130 of file psi4io.py.

8.53.4.8 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.53.4.9 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.53.4.10 forcebalance.leastsq.LeastSquares.MAQ [inherited]

Dictionary for derivative terms.

Definition at line 100 of file leastsq.py.

8.53.4.11 forcebalance.psi4io.THCDF_Psi4.Molecules

Definition at line 105 of file psi4io.py.

8.53.4.12 forcebalance.psi4io.THCDF_Psi4.MP2_Energy

Actually run PSI4.

Read in the commented linindep.gbs file and ensure that these same lines are commented in the new .gbs file Now build a "Frankenstein" .gbs file composed of the original .gbs file but with data from the linindep.gbs file!

Definition at line 236 of file psi4io.py.

8.53.4.13 forcebalance.leastsq.LeastSquares.objective [inherited]

Definition at line 138 of file leastsq.py.

8.53.4.14 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.53.4.15 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.53.4.16 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.53.4.17 forcebalance.psi4io.THCDF_Psi4.throw_outs

Definition at line 106 of file psi4io.py.

8.53.4.18 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.53.4.19 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- psi4io.py

8.54 forcebalance.tinkerio.Tinker_Reader Class Reference

Finite state machine for parsing TINKER force field files.

Inheritance diagram for forcebalance.tinkerio.Tinker_Reader:

Collaboration diagram for forcebalance.tinkerio.Tinker_Reader:

Public Member Functions

- def __init__
- def feed

  Given a line, determine the interaction type and the atoms involved (the suffix).
- def Split

- def Whites
- def build_pid

  Returns the parameter type (e.g.

Public Attributes

- pdict

  The parameter dictionary (defined in this file)
- atom

  The atom numbers in the interaction (stored in the TINKER parser)
- itype
- suffix
- ln
- adict

  The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.
- molatom

  The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.
- Molecules
- AtomTypes

8.54.1 Detailed Description

Finite state machine for parsing TINKER force field files.

This class is instantiated when we begin to read in a file. The feed(line) method updates the state of the machine, informing it of the current interaction type. Using this information we can look up the interaction type and parameter type for building the parameter ID.

Definition at line 68 of file tinkerio.py.

8.54.2 Constructor & Destructor Documentation

8.54.2.1 def forcebalance.tinkerio.Tinker_Reader.__init__ (*self, fnm*)

Definition at line 70 of file tinkerio.py.

8.54.3 Member Function Documentation

8.54.3.1 def forcebalance.BaseReader.build_pid (*self, pfld*) [inherited]

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Definition at line 107 of file __init__.py.

8.54.3.2 def forcebalance.tinkerio.Tinker_Reader.feed (*self, line*)

Given a line, determine the interaction type and the atoms involved (the suffix).

```
TINKER generally has stuff like this:


    bond-cubic              -2.55
    bond-quartic            3.793125

    vdw           1               3.4050     0.1100
    vdw           2               2.6550     0.0135      0.910 # PARM 4

    multipole     2    1    2               0.25983
                                            -0.03859    0.00000   -0.05818
                                            -0.03673
                                             0.00000   -0.10739
                                            -0.00203    0.00000    0.14412
```

The '#PARM 4' has no effect on TINKER but it indicates that we are tuning the fourth field on the line (the 0.910 value).

Todo Put the rescaling factors for TINKER parameters in here. Currently we're using the initial value to determine the rescaling factor which is not very good.

```
Every parameter line is prefaced by the interaction type
except for 'multipole' which is on multiple lines.  Because
the lines that come after 'multipole' are predictable, we just
determine the current line using the previous line.

Random note: Unit of force is kcal / mole / angstrom squared.
```

Definition at line 111 of file tinkerio.py.

8.54.3.3 def forcebalance.BaseReader.Split (*self, line*) [inherited]

Definition at line 82 of file __init__.py.

Here is the call graph for this function:

8.54.3.4 def forcebalance.BaseReader.Whites (*self, line*) [inherited]

Definition at line 85 of file __init__.py.

Here is the call graph for this function:

8.54.4 Member Data Documentation

8.54.4.1 forcebalance.BaseReader.adict [inherited]

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [bonds], [angles] etc.

Definition at line 72 of file __init__.py.

8.54.4.2 forcebalance.tinkerio.Tinker_Reader.atom

The atom numbers in the interaction (stored in the TINKER parser)

Definition at line 75 of file tinkerio.py.

8.54.4.3 forcebalance.BaseReader.AtomTypes [inherited]

Definition at line 80 of file __init__.py.

8.54.4.4 forcebalance.tinkerio.Tinker_Reader.itype

Definition at line 119 of file tinkerio.py.

8.54.4.5 forcebalance.BaseReader.ln [inherited]

Definition at line 67 of file __init__.py.

8.54.4.6 forcebalance.BaseReader.molatom [inherited]

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Definition at line 77 of file __init__.py.

8.54.4.7 forcebalance.BaseReader.Molecules [inherited]

Definition at line 79 of file __init__.py.

8.54.4.8 forcebalance.tinkerio.Tinker_Reader.pdict

The parameter dictionary (defined in this file)

Definition at line 73 of file tinkerio.py.

8.54.4.9 forcebalance.tinkerio.Tinker_Reader.suffix

Definition at line 141 of file tinkerio.py.

The documentation for this class was generated from the following file:

- tinkerio.py

8.55 forcebalance.nifty.Unpickler_LP Class Reference

A subclass of the python Unpickler that implements unpickling of _ElementTree types.

Inheritance diagram for forcebalance.nifty.Unpickler_LP:

Collaboration diagram for forcebalance.nifty.Unpickler_LP:

Public Member Functions

- def __init__

8.55.1 Detailed Description

A subclass of the python Unpickler that implements unpickling of _ElementTree types.

Definition at line 523 of file nifty.py.

8.55.2 Constructor & Destructor Documentation

8.55.2.1 def forcebalance.nifty.Unpickler_LP.__init__ (*self, file*)

Definition at line 524 of file nifty.py.

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- nifty.py

8.56 forcebalance.vibration.Vibration Class Reference

Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).

Inheritance diagram for forcebalance.vibration.Vibration:

Collaboration diagram for forcebalance.vibration.Vibration:

Public Member Functions

- def __init__

  Initialization.
- def read_reference_data

  Read the reference vibrational data from a file.
- def prepare_temp_directory

  Prepare the temporary directory, by default does nothing.
- def indicate

  Print qualitative indicator.
- def get

  Evaluate objective function.
- def get_X

  Computes the objective function contribution without any parametric derivatives.
- def get_G

  Computes the objective function contribution and its gradient.
- def get_H

  Computes the objective function contribution and its gradient / Hessian.
- def link_from_tempdir
- def refresh_temp_directory

  Back up the temporary directory if desired, delete it and then create a new one.
- def sget

  Stages the directory for the target, and then calls 'get'.

- def submit_jobs
- def stage

  Stages the directory for the target, and then launches Work Queue processes if any.
- def wq_complete

  This method determines whether the Work Queue tasks for the current target have completed.
- def printcool_table

  Print target information in an organized table format.
- def set_option

Public Attributes

- vfnm

  The vdata.txt file that contains the vibrations.
- na

  Number of atoms.
- ref_eigvals
- ref_eigvecs
- calc_eigvals
- objective
- tempdir

  Root directory of the whole project.
- rundir

  The directory in which the simulation is running - this can be updated.
- FF

  Need the forcefield (here for now)
- xct

  Counts how often the objective function was computed.
- gct

  Counts how often the gradient was computed.
- hct

  Counts how often the Hessian was computed.
- PrintOptionDict
- verbose_options

8.56.1 Detailed Description

Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).

Currently Tinker is supported.

Definition at line 30 of file vibration.py.

8.56.2 Constructor & Destructor Documentation

8.56.2.1 def forcebalance.vibration.Vibration.__init__ (*self, options, tgt_opts, forcefield*)

Initialization.

Definition at line 35 of file vibration.py.

Here is the call graph for this function:

8.56.3 Member Function Documentation

8.56.3.1 def forcebalance.vibration.Vibration.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Evaluate objective function.

Definition at line 111 of file vibration.py.

Here is the call graph for this function:

8.56.3.2 def forcebalance.target.Target.get_G (*self, mvals* = `None`) `[inherited]`

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_-pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 172 of file target.py.

Here is the call graph for this function:

8.56.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`) [inherited]

Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 195 of file target.py.

Here is the call graph for this function:

8.56.3.4 def forcebalance.target.Target.get_X (*self*, *mvals* = `None`) `[inherited]`

Computes the objective function contribution without any parametric derivatives.

Definition at line 155 of file target.py.

Here is the call graph for this function:

8.56.3.5 def forcebalance.vibration.Vibration.indicate (*self*)

Print qualitative indicator.

Definition at line 102 of file vibration.py.

Here is the call graph for this function:

8.56.3.6 def forcebalance.target.Target.link_from_tempdir (*self, absdestdir*) [inherited]

Definition at line 213 of file target.py.

8.56.3.7 def forcebalance.vibration.Vibration.prepare_temp_directory (*self, options, tgt_opts*)

Prepare the temporary directory, by default does nothing.

Definition at line 97 of file vibration.py.

8.56.3.8 def forcebalance.target.Target.printcool_table (*self*, *data* = `OrderedDict([])`, *headings* = `[]`, *banner* = `None`, *footnote* = `None`, *color* = `0`) `[inherited]`

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

Parameters

data	Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended).
headings	Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line.
banner	Optional heading line, which will be printed at the top in the title.
footnote	Optional footnote line, which will be printed at the bottom.

Definition at line 367 of file target.py.

Here is the call graph for this function:

8.56.3.9 def forcebalance.vibration.Vibration.read_reference_data (*self*)

Read the reference vibrational data from a file.

Definition at line 57 of file vibration.py.

8.56.3.10 def forcebalance.target.Target.refresh_temp_directory (*self*) `[inherited]`

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 219 of file target.py.

Here is the call graph for this function:

8.56.3.11 def forcebalance.BaseClass.set_option (*self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False) [inherited]

Definition at line 35 of file __init__.py.

8.56.3.12 def forcebalance.target.Target.sget (*self, mvals, AGrad* = False, *AHess* = False, *customdir* = None) [inherited]

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 266 of file target.py.

Here is the call graph for this function:

8.56.3.13 def forcebalance.target.Target.stage (*self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None`) `[inherited]`

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 301 of file target.py.

Here is the call graph for this function:

8.56.3.14 def forcebalance.target.Target.submit_jobs (*self, mvals, AGrad* = `False`, *AHess* = `False`) `[inherited]`

Definition at line 291 of file target.py.

8.56.3.15 def forcebalance.target.Target.wq_complete (*self*) [inherited]

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 331 of file target.py.

Here is the call graph for this function:

8.56.4 Member Data Documentation

8.56.4.1 forcebalance.vibration.Vibration.calc_eigvals

Definition at line 140 of file vibration.py.

8.56.4.2 forcebalance.target.Target.FF [inherited]

Need the forcefield (here for now)

Definition at line 139 of file target.py.

8.56.4.3 forcebalance.target.Target.gct [inherited]

Counts how often the gradient was computed.

Definition at line 143 of file target.py.

8.56.4.4 forcebalance.target.Target.hct [inherited]

Counts how often the Hessian was computed.

Definition at line 145 of file target.py.

8.56.4.5 forcebalance.vibration.Vibration.na

Number of atoms.

Definition at line 59 of file vibration.py.

8.56.4.6 forcebalance.vibration.Vibration.objective

Definition at line 141 of file vibration.py.

8.56.4.7 forcebalance.BaseClass.PrintOptionDict [inherited]

Definition at line 32 of file __init__.py.

8.56.4.8 forcebalance.vibration.Vibration.ref_eigvals

Definition at line 60 of file vibration.py.

8.56.4.9 forcebalance.vibration.Vibration.ref_eigvecs

Definition at line 61 of file vibration.py.

8.56.4.10 forcebalance.target.Target.rundir [inherited]

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 137 of file target.py.

8.56.4.11 forcebalance.target.Target.tempdir [inherited]

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 135 of file target.py.

8.56.4.12 forcebalance.BaseClass.verbose_options [inherited]

Definition at line 33 of file __init__.py.

8.56.4.13 forcebalance.vibration.Vibration.vfnm

The vdata.txt file that contains the vibrations.

Definition at line 49 of file vibration.py.

8.56.4.14 forcebalance.target.Target.xct [inherited]

Counts how often the objective function was computed.

Definition at line 141 of file target.py.

The documentation for this class was generated from the following file:

- vibration.py

8.57 forcebalance.tinkerio.Vibration_TINKER Class Reference

Subclass of Target for vibrational frequency matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.Vibration_TINKER:

Collaboration diagram for forcebalance.tinkerio.Vibration_TINKER:

Public Member Functions

- def __init__
- def prepare_temp_directory
- def vibration_driver

8.57.1 Detailed Description

Subclass of Target for vibrational frequency matching using TINKER.

Provides optimized geometry, vibrational frequencies (in cm-1), and eigenvectors.

Definition at line 355 of file tinkerio.py.

8.57.2 Constructor & Destructor Documentation

8.57.2.1 def forcebalance.tinkerio.Vibration_TINKER.__init__ (*self, options, tgt_opts, forcefield*)

Definition at line 358 of file tinkerio.py.

Here is the call graph for this function:

8.57.3 Member Function Documentation

8.57.3.1 def forcebalance.tinkerio.Vibration_TINKER.prepare_temp_directory (*self, options, tgt_opts*)

Definition at line 363 of file tinkerio.py.

Here is the call graph for this function:

8.57.3.2 def forcebalance.tinkerio.Vibration_TINKER.vibration_driver (*self*)

Definition at line 373 of file tinkerio.py.

Here is the call graph for this function:

The documentation for this class was generated from the following file:

- tinkerio.py

9 File Documentation

9.1 __init__.py File Reference

Classes

- class forcebalance.BaseClass

  Provides some nifty functions that are common to all ForceBalance classes.
- class forcebalance.BaseReader

  The 'reader' class.

Namespaces

- namespace forcebalance

Variables

- tuple forcebalance.__version__ = pkg_resources.get_distribution("forcebalance")
- forcebalance.WORK_QUEUE = None
- tuple forcebalance.WQIDS = defaultdict(list)

9.2 abinitio.py File Reference

Classes

- class forcebalance.abinitio.AbInitio

  Subclass of Target for fitting force fields to ab initio data.

Namespaces

- namespace forcebalance.abinitio

  Ab-initio fitting module (energies, forces, resp).

Functions

- def forcebalance.abinitio.weighted_variance

  A more generalized version of build_objective which is callable for derivatives, but the covariance is not there anymore.
- def forcebalance.abinitio.weighted_variance2

  A bit of a hack, since we have to subtract out two mean quantities to get Hessian elements.
- def forcebalance.abinitio.build_objective

  This function builds an objective function (number) from the complicated polytensor and covariance matrices.

Variables

- tuple forcebalance.abinitio.logger = getLogger(__name__)

9.3 abinitio_internal.py File Reference

Classes

- class forcebalance.abinitio_internal.AbInitio_Internal

  Subclass of Target for force and energy matching using an internal implementation.

Namespaces

- namespace forcebalance.abinitio_internal

  Internal implementation of energy matching (for TIP3P water only)

9.4 amberio.py File Reference

Classes

- class forcebalance.amberio.Mol2_Reader

  Finite state machine for parsing Mol2 force field file.
- class forcebalance.amberio.FrcMod_Reader

  Finite state machine for parsing FrcMod force field file.
- class forcebalance.amberio.AbInitio_AMBER

  Subclass of Target for force and energy matching using AMBER.

Namespaces

- namespace forcebalance.amberio

  AMBER force field input/output.

Functions

- def forcebalance.amberio.is_mol2_atom

Variables

- tuple forcebalance.amberio.logger = getLogger(__name__)
- dictionary forcebalance.amberio.mol2_pdict = {'COUL':{'Atom':[1], 8:''}}
- dictionary forcebalance.amberio.frcmod_pdict

9.5 api.dox File Reference

9.6 binding.py File Reference

Classes

- class forcebalance.binding.BindingEnergy

  Improved subclass of Target for fitting force fields to binding energies.

Namespaces

- namespace forcebalance.binding

  Binding energy fitting module.

Functions

- def forcebalance.binding.parse_interactions

  Parse through the interactions input file.

Variables

- tuple forcebalance.binding.logger = getLogger(__name__)

9.7 chemistry.py File Reference

Namespaces

- namespace forcebalance.chemistry

Functions

- def forcebalance.chemistry.LookupByMass
- def forcebalance.chemistry.BondStrengthByLength

Variables

- tuple forcebalance.chemistry.BondEnergies = defaultdict(lambda:defaultdict(dict))
- list forcebalance.chemistry.Radii

  Covalent radii from Cordero et al.
- dictionary forcebalance.chemistry.PeriodicTable
- list forcebalance.chemistry.Elements
- list forcebalance.chemistry.BondChars = ['-','=','3']
- string forcebalance.chemistry.data_from_web
- tuple forcebalance.chemistry.line = line.expandtabs()
- tuple forcebalance.chemistry.BE = float(line.split()[1])
- tuple forcebalance.chemistry.L = float(line.split()[2])
- tuple forcebalance.chemistry.atoms = re.split('[-=3]', line.split()[0])
- list forcebalance.chemistry.A = atoms[0]
- list forcebalance.chemistry.B = atoms[1]
- tuple forcebalance.chemistry.bo = BondChars.index(re.findall('[-=3]', line.split()[0])[0])

9.8 contact.py File Reference

Namespaces

- namespace forcebalance.contact

Functions

- def forcebalance.contact.atom_distances

  For each frame in xyzlist, compute the (euclidean) distance between pairs of atoms whos indices are given in contacts.
- def forcebalance.contact.residue_distances

  For each frame in xyzlist, and for each pair of residues in the array contact, compute the distance between the closest pair of atoms such that one of them belongs to each residue.

9.9 counterpoise.py File Reference

Classes

- class forcebalance.counterpoise.Counterpoise

  Target subclass for matching the counterpoise correction.

Namespaces

- namespace forcebalance.counterpoise

  Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE).

Variables

- tuple forcebalance.counterpoise.logger = getLogger(__name__)

9.10 custom_io.py File Reference

Classes

- class forcebalance.custom_io.Gen_Reader

  Finite state machine for parsing custom GROMACS force field files.

Namespaces

- namespace forcebalance.custom_io

  Custom force field parser.

Variables

- list forcebalance.custom_io.cptypes = [None, 'CPGAUSS', 'CPEXPG', 'CPGEXP']

  Types of counterpoise correction.
- list forcebalance.custom_io.ndtypes = [None]

  Types of NDDO correction.
- dictionary forcebalance.custom_io.fdict

  Section -> Interaction type dictionary.
- dictionary forcebalance.custom_io.pdict

  Interaction type -> Parameter Dictionary.

9.11 finite_difference.py File Reference

Namespaces

- namespace forcebalance.finite_difference

Functions

- def forcebalance.finite_difference.f1d2p

  A two-point finite difference stencil.
- def forcebalance.finite_difference.f1d5p

  A highly accurate five-point finite difference stencil for computing derivatives of a function.
- def forcebalance.finite_difference.f1d7p

  A highly accurate seven-point finite difference stencil for computing derivatives of a function.
- def forcebalance.finite_difference.f12d7p
- def forcebalance.finite_difference.f12d3p

  A three-point finite difference stencil.
- def forcebalance.finite_difference.in_fd

  Invoking this function from anywhere will tell us whether we're being called by a finite-difference function.
- def forcebalance.finite_difference.fdwrap

  A function wrapper for finite difference designed for differentiating 'get'-type functions.
- def forcebalance.finite_difference.fdwrap_G

  A driver to fdwrap for gradients (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.
- def forcebalance.finite_difference.fdwrap_H

  A driver to fdwrap for Hessians (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.

Variables

- tuple forcebalance.finite_difference.logger = getLogger(__name__)

9.12 forcefield.py File Reference

Classes

- class forcebalance.forcefield.BackedUpDict
- class forcebalance.forcefield.FF

  Force field class.

Namespaces

- namespace forcebalance.forcefield

  Force field module.

Functions

- def forcebalance.forcefield.determine_fftype

  Determine the type of a force field file.
- def forcebalance.forcefield.rs_override

  This function takes in a dictionary (rsfactors) and a string (termtype).

Variables

- tuple forcebalance.forcefield.logger = getLogger(__name__)
- dictionary forcebalance.forcefield.FF_Extensions
- dictionary forcebalance.forcefield.FF_IOModules

9.13 gmxio.py File Reference

Classes

- class forcebalance.gmxio.ITP_Reader

  Finite state machine for parsing GROMACS force field files.
- class forcebalance.gmxio.AbInitio_GMX

  Subclass of AbInitio for force and energy matching using normal GROMACS.
- class forcebalance.gmxio.Liquid_GMX
- class forcebalance.gmxio.Interaction_GMX

  Subclass of Interaction for interaction energy matching using GROMACS.

Namespaces

- namespace forcebalance.gmxio

  GROMACS input/output.

Functions

- def forcebalance.gmxio.edit_mdp

  Create or edit a Gromacs MDP file.
- def forcebalance.gmxio.parse_atomtype_line

  Parses the 'atomtype' line.
- def forcebalance.gmxio.rm_gmx_baks

Variables

- tuple forcebalance.gmxio.logger = getLogger(__name__)
- list forcebalance.gmxio.nftypes = [None, 'VDW', 'VDW_BHAM']

  VdW interaction function types.
- list forcebalance.gmxio.pftypes = [None, 'VPAIR', 'VPAIR_BHAM']

  Pairwise interaction function types.
- list forcebalance.gmxio.bftypes = [None, 'BONDS', 'G96BONDS', 'MORSE']

  Bonded interaction function types.

- list forcebalance.gmxio.aftypes

  Angle interaction function types.
- list forcebalance.gmxio.dftypes = [None, 'PDIHS', 'IDIHS', 'RBDIHS', 'PIMPDIHS', 'FOURDIHS', None, None, 'TABDIHS', 'PDIHMULS']

  Dihedral interaction function types.
- dictionary forcebalance.gmxio.fdict

  Section -> Interaction type dictionary.
- dictionary forcebalance.gmxio.pdict

  Interaction type -> Parameter Dictionary.

9.14 gmxqpio.py File Reference

Classes

- class forcebalance.gmxqpio.Monomer_QTPIE

  Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).

Namespaces

- namespace forcebalance.gmxqpio
- namespace forcebalance.gmxio

  GROMACS input/output.

Functions

- def forcebalance.gmxqpio.get_monomer_properties

Variables

- tuple forcebalance.gmxqpio.logger = getLogger(__name__)

9.15 interaction.py File Reference

Classes

- class forcebalance.interaction.Interaction

  Subclass of Target for fitting force fields to interaction energies.

Namespaces

- namespace forcebalance.interaction

  Interaction energy fitting module.

Variables

- tuple forcebalance.interaction.logger = getLogger(__name__)

9.16 leastsq.py File Reference

Classes

- class forcebalance.leastsq.LeastSquares

  Subclass of Target for general least squares fitting.

Namespaces

- namespace forcebalance.leastsq
- namespace forcebalance.abinitio

  Ab-initio fitting module (energies, forces, resp).

Functions

- def forcebalance.leastsq.CheckBasis
- def forcebalance.leastsq.LastMvals

Variables

- tuple forcebalance.leastsq.logger = getLogger(__name__)
- forcebalance.leastsq.CHECK_BASIS = False
- forcebalance.leastsq.LAST_MVALS = None

9.17 liquid.py File Reference

Classes

- class forcebalance.liquid.Liquid

  Subclass of Target for liquid property matching.

Namespaces

- namespace forcebalance.liquid

  Matching of liquid bulk properties.

Functions

- def forcebalance.liquid.weight_info

Variables

- tuple forcebalance.liquid.logger = getLogger(__name__)

9.18 Mol2.py File Reference

Classes

- class forcebalance.Mol2.mol2_atom

  This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.
- class forcebalance.Mol2.mol2_bond

  This is to manage mol2 bond lines on the form: 1 1 2 ar.
- class forcebalance.Mol2.mol2

  This is to manage one mol2 series of lines on the form:
- class forcebalance.Mol2.mol2_set

Namespaces

- namespace forcebalance.Mol2

Variables

- tuple forcebalance.Mol2.data = mol2_set(sys.argv[1], subset=["RNAse.xray.inh8.1QHC"])

9.19 mol2io.py File Reference

Classes

- class forcebalance.mol2io.Mol2_Reader

  Finite state machine for parsing Mol2 force field file.

Namespaces

- namespace forcebalance.mol2io

  Mol2 I/O.

Variables

- dictionary forcebalance.mol2io.mol2_pdict = {'COUL':{'Atom':[1], 6:''}}

9.20 molecule.py File Reference

Classes

- class forcebalance.molecule.MolfileTimestep

  Wrapper for the timestep C structure used in molfile plugins.
- class forcebalance.molecule.Molecule

  Lee-Ping's general file format conversion class.

Namespaces

- namespace forcebalance.molecule

Functions

- def forcebalance.molecule.getElement
- def forcebalance.molecule.nodematch
- def forcebalance.molecule.isint

  ONLY matches integers! If you have a decimal point? None shall pass!
- def forcebalance.molecule.isfloat

  Matches ANY number; it can be a decimal, scientific notation, integer, or what have you.
- def forcebalance.molecule.BuildLatticeFromLengthsAngles

  This function takes in three lattice lengths and three lattice angles, and tries to return a complete box specification.
- def forcebalance.molecule.BuildLatticeFromVectors

  This function takes in three lattice vectors and tries to return a complete box specification.
- def forcebalance.molecule.format_xyz_coord

  Print a line consisting of (element, x, y, z) in accordance with .xyz file format.
- def forcebalance.molecule.format_gro_coord

  Print a line in accordance with .gro file format, with six decimal points of precision.
- def forcebalance.molecule.format_xyzgen_coord

  Print a line consisting of (element, p, q, r, s, t, ...) where (p, q, r) are arbitrary atom-wise data (this might happen, for instance, with atomic charges)
- def forcebalance.molecule.format_gro_box

  Print a line corresponding to the box vector in accordance with .gro file format.
- def forcebalance.molecule.is_gro_coord

  Determines whether a line contains GROMACS data or not.
- def forcebalance.molecule.is_charmm_coord

  Determines whether a line contains CHARMM data or not.
- def forcebalance.molecule.is_gro_box

  Determines whether a line contains a GROMACS box vector or not.
- def forcebalance.molecule.add_strip_to_mat
- def forcebalance.molecule.pvec
- def forcebalance.molecule.grouper

  Groups a big long iterable into groups of ten or what have you.
- def forcebalance.molecule.even_list

  Creates a list of number sequences divided as evenly as possible.
- def forcebalance.molecule.both
- def forcebalance.molecule.diff
- def forcebalance.molecule.either
- def forcebalance.molecule.EulerMatrix

  Constructs an Euler matrix from three Euler angles.
- def forcebalance.molecule.ComputeOverlap

  Computes an 'overlap' between two molecules based on some fictitious density.
- def forcebalance.molecule.AlignToDensity

  Computes a "overlap density" from two frames.
- def forcebalance.molecule.AlignToMoments

  Pre-aligns molecules to 'moment of inertia'.
- def forcebalance.molecule.get_rotate_translate
- def forcebalance.molecule.main

Variables

- tuple forcebalance.molecule.FrameVariableNames
- tuple forcebalance.molecule.AtomVariableNames = set(['elem', 'partial_charge', 'atomname', 'atomtype', 'tinkersuf', 'resid', 'resname', 'qcsuf', 'qm_ghost', 'chain', 'altloc', 'icode'])
- tuple forcebalance.molecule.MetaVariableNames = set(['fnm', 'ftype', 'qcrems', 'qctemplate', 'charge', 'mult', 'bonds'])
- tuple forcebalance.molecule.QuantumVariableNames = set(['qcrems', 'qctemplate', 'charge', 'mult', 'qcsuf', 'qm_ghost'])
- forcebalance.molecule.AllVariableNames = QuantumVariableNames|AtomVariableNames|MetaVariableNames|FrameVariableNames
- list forcebalance.molecule.Radii
- list forcebalance.molecule.Elements
- tuple forcebalance.molecule.PeriodicTable
- float forcebalance.molecule.bohrang = 0.529177249

  One bohr equals this many angstroms.
- tuple forcebalance.molecule.splitter = re.compile(r'(\s+|\S+)')
- tuple forcebalance.molecule.Box = namedtuple('Box',['a','b','c','alpha','beta','gamma','A','B','C','V'])
- int forcebalance.molecule.radian = 180
- forcebalance.molecule.Alive

9.21 moments.py File Reference

Classes

- class forcebalance.moments.Moments

  Subclass of Target for fitting force fields to multipole moments (from experiment or theory).

Namespaces

- namespace forcebalance.moments

  Multipole moment fitting module.

Variables

- tuple forcebalance.moments.logger = getLogger(__name__)

9.22 nifty.py File Reference

Classes

- class forcebalance.nifty.Pickler_LP

  A subclass of the python Pickler that implements pickling of _ElementTree types.
- class forcebalance.nifty.Unpickler_LP

  A subclass of the python Unpickler that implements unpickling of _ElementTree types.

Namespaces

- namespace forcebalance.nifty

  Nifty functions, intended to be imported by any module within ForceBalance.

Functions

- def forcebalance.nifty.pvec1d

  Printout of a 1-D vector.
- def forcebalance.nifty.pmat2d

  Printout of a 2-D matrix.
- def forcebalance.nifty.encode
- def forcebalance.nifty.segments
- def forcebalance.nifty.commadash
- def forcebalance.nifty.uncommadash
- def forcebalance.nifty.printcool

  Cool-looking printout for slick formatting of output.
- def forcebalance.nifty.printcool_dictionary

  See documentation for printcool; this is a nice way to print out keys/values in a dictionary.
- def forcebalance.nifty.isint

  ONLY matches integers! If you have a decimal point? None shall pass!
- def forcebalance.nifty.isfloat

  Matches ANY number; it can be a decimal, scientific notation, what have you CAUTION - this will also match an integer.
- def forcebalance.nifty.isdecimal

  Matches things with a decimal only; see isint and isfloat.
- def forcebalance.nifty.floatornan

  Returns a big number if we encounter NaN.
- def forcebalance.nifty.col

  Given any list, array, or matrix, return a 1-column matrix.
- def forcebalance.nifty.row

  Given any list, array, or matrix, return a 1-row matrix.
- def forcebalance.nifty.flat

  Given any list, array, or matrix, return a single-index array.
- def forcebalance.nifty.orthogonalize

  Given two vectors vec1 and vec2, project out the component of vec1 that is along the vec2-direction.
- def forcebalance.nifty.invert_svd

  Invert a matrix using singular value decomposition.
- def forcebalance.nifty.get_least_squares
- def forcebalance.nifty.statisticalInefficiency

  Compute the (cross) statistical inefficiency of (two) timeseries.
- def forcebalance.nifty.lp_dump

  Use this instead of pickle.dump for pickling anything that contains _ElementTree types.
- def forcebalance.nifty.lp_load

  Use this instead of pickle.load for unpickling anything that contains _ElementTree types.
- def forcebalance.nifty.getWorkQueue
- def forcebalance.nifty.getWQIds
- def forcebalance.nifty.createWorkQueue
- def forcebalance.nifty.queue_up

Submit a job to the Work Queue.

- def forcebalance.nifty.queue_up_src_dest

  Submit a job to the Work Queue.
- def forcebalance.nifty.wq_wait1

  This function waits ten seconds to see if a task in the Work Queue has finished.
- def forcebalance.nifty.wq_wait

  This function waits until the work queue is completely empty.
- def forcebalance.nifty.GoInto
- def forcebalance.nifty.allsplit
- def forcebalance.nifty.Leave
- def forcebalance.nifty.MissingFileInspection
- def forcebalance.nifty.LinkFile
- def forcebalance.nifty.CopyFile
- def forcebalance.nifty.link_dir_contents
- def forcebalance.nifty.remove_if_exists

  Remove the file if it exists (doesn't return an error).
- def forcebalance.nifty.which
- def forcebalance.nifty.warn_press_key
- def forcebalance.nifty.warn_once

  Prints a warning but will only do so once in a given run.
- def forcebalance.nifty.concurrent_map

  Similar to the bultin function map().
- def forcebalance.nifty.multiopen

  This function be given any of several variable types (single file name, file object, or list of lines, or a list of the above) and give a list of files:

Variables

- tuple forcebalance.nifty.logger = getLogger(__name__)
- float forcebalance.nifty.kb = 0.0083144100163

  Boltzmann constant.
- float forcebalance.nifty.eqcgmx = 2625.5002

  Q-Chem to GMX unit conversion for energy.
- float forcebalance.nifty.fqcgmx = -49621.9

  Q-Chem to GMX unit conversion for force.
- float forcebalance.nifty.bohrang = 0.529177249

  One bohr equals this many angstroms.
- string forcebalance.nifty.XMLFILE = 'x'

  Pickle uses 'flags' to pickle and unpickle different variable types.
- list forcebalance.nifty.specific_lst
- tuple forcebalance.nifty.specific_dct = dict(list(itertools.chain(∗[[(j,i[1]) for j in i[0]] for i in specific_lst])))

9.23 objective.py File Reference

Classes

- class forcebalance.objective.Objective

  Objective function.
- class forcebalance.objective.Penalty

  Penalty functions for regularizing the force field optimizer.

Namespaces

- namespace forcebalance.objective

  ForceBalance objective function.

Variables

- tuple forcebalance.objective.logger = getLogger(__name__)
- dictionary forcebalance.objective.Implemented_Targets

  The table of implemented Targets.
- list forcebalance.objective.Letters = ['X','G','H']

  This is the canonical lettering that corresponds to : objective function, gradient, Hessian.

9.24 openmmio.py File Reference

Classes

- class forcebalance.openmmio.OpenMM_Reader

  Class for parsing OpenMM force field files.
- class forcebalance.openmmio.Liquid_OpenMM
- class forcebalance.openmmio.AbInitio_OpenMM

  Subclass of AbInitio for force and energy matching using OpenMM.
- class forcebalance.openmmio.Interaction_OpenMM

  Subclass of Target for interaction matching using OpenMM.

Namespaces

- namespace forcebalance.openmmio

  OpenMM input/output.

Functions

- def forcebalance.openmmio.get_dipole

  Return the current dipole moment in Debye.
- def forcebalance.openmmio.ResetVirtualSites

  Given a set of OpenMM-compatible positions and a System object, compute the correct virtual site positions according to the System.
- def forcebalance.openmmio.CopyAmoebaBondParameters
- def forcebalance.openmmio.CopyAmoebaOutOfPlaneBendParameters
- def forcebalance.openmmio.CopyAmoebaAngleParameters
- def forcebalance.openmmio.CopyAmoebaInPlaneAngleParameters
- def forcebalance.openmmio.CopyAmoebaVdwParameters
- def forcebalance.openmmio.CopyAmoebaMultipoleParameters
- def forcebalance.openmmio.CopyHarmonicBondParameters
- def forcebalance.openmmio.CopyHarmonicAngleParameters
- def forcebalance.openmmio.CopyPeriodicTorsionParameters
- def forcebalance.openmmio.CopyNonbondedParameters
- def forcebalance.openmmio.do_nothing

- def forcebalance.openmmio.CopySystemParameters

  Copy parameters from one system (i.e.
- def forcebalance.openmmio.UpdateSimulationParameters
- def forcebalance.openmmio.MTSVVVRIntegrator

  Create a multiple timestep velocity verlet with velocity randomization (VVVR) integrator.

Variables

- tuple forcebalance.openmmio.logger = getLogger(__name__)
- dictionary forcebalance.openmmio.suffix_dict
- string forcebalance.openmmio.pdict = "XML_Override"

  pdict is a useless variable if the force field is XML.

9.25 optimizer.py File Reference

Classes

- class forcebalance.optimizer.Optimizer

  Optimizer class.

Namespaces

- namespace forcebalance.optimizer

  Optimization algorithms.

Functions

- def forcebalance.optimizer.Counter
- def forcebalance.optimizer.GoodStep

Variables

- tuple forcebalance.optimizer.logger = getLogger(__name__)
- int forcebalance.optimizer.ITERATION_NUMBER = 0
- int forcebalance.optimizer.GOODSTEP = 0

9.26 output.py File Reference

Classes

- class forcebalance.output.ForceBalanceLogger

  This logger starts out with a default handler that writes to stdout addHandler removes this default the first time another handler is added.
- class forcebalance.output.RawStreamHandler

  Exactly like output.StreamHandler except it does no extra formatting before sending logging messages to the stream.
- class forcebalance.output.RawFileHandler

  Exactly like output.FileHandler except it does no extra formatting before sending logging messages to the file.

- class forcebalance.output.CleanStreamHandler

  Similar to RawStreamHandler except it does not write terminal escape codes.
- class forcebalance.output.CleanFileHandler

  File handler that does not write terminal escape codes and carriage returns to files.

Namespaces

- namespace forcebalance.output

9.27 parser.py File Reference

Namespaces

- namespace forcebalance.parser

  Input file parser for ForceBalance jobs.

Functions

- def forcebalance.parser.read_mvals
- def forcebalance.parser.read_pvals
- def forcebalance.parser.read_priors
- def forcebalance.parser.read_internals
- def forcebalance.parser.printsection

  Print out a section of the input file in a parser-compliant and readable format.
- def forcebalance.parser.parse_inputs

  Parse through the input file and read all user-supplied options.

Variables

- tuple forcebalance.parser.logger = getLogger(__name__)
- dictionary forcebalance.parser.gen_opts_types

  Default general options.
- dictionary forcebalance.parser.tgt_opts_types

  Default fitting target options.
- dictionary forcebalance.parser.gen_opts_defaults = {}

  Default general options - basically a collapsed veresion of gen_opts_types.
- dictionary forcebalance.parser.subdict = {}
- dictionary forcebalance.parser.tgt_opts_defaults = {}

  Default target options - basically a collapsed version of tgt_opts_types.
- dictionary forcebalance.parser.bkwd = {"simtype" : "type"}

  Option maps for maintaining backward compatibility.
- list forcebalance.parser.mainsections = ["SIMULATION","TARGET","OPTIONS","END","NONE"]

  Listing of sections in the input file.
- dictionary forcebalance.parser.ParsTab

  ParsTab that refers to subsection parsers.

9.28 psi4io.py File Reference

Classes

- class forcebalance.psi4io.GBS_Reader

  Interaction type -> Parameter Dictionary.
- class forcebalance.psi4io.THCDF_Psi4
- class forcebalance.psi4io.Grid_Reader

  Finite state machine for parsing DVR grid files.
- class forcebalance.psi4io.RDVR3_Psi4

  Subclass of Target for R-DVR3 grid fitting.

Namespaces

- namespace forcebalance.psi4io

  PSI4 force field input/output.

Variables

- tuple forcebalance.psi4io.logger = getLogger(__name__)

9.29 PT.py File Reference

Namespaces

- namespace forcebalance.PT

Variables

- dictionary forcebalance.PT.PeriodicTable
- list forcebalance.PT.Elements

9.30 qchemio.py File Reference

Classes

- class forcebalance.qchemio.QCIn_Reader

  Finite state machine for parsing Q-Chem input files.

Namespaces

- namespace forcebalance.qchemio

  Q-Chem input file parser.

Functions

- def forcebalance.qchemio.QChem_Dielectric_Energy

Variables

- tuple forcebalance.qchemio.logger = getLogger(__name__)
- list forcebalance.qchemio.ndtypes = [None]

  Types of counterpoise correction cptypes = [None, 'BASS', 'BASSP'] Types of NDDO correction.
- dictionary forcebalance.qchemio.pdict

  Section -> Interaction type dictionary.

9.31 target.py File Reference

Classes

- class forcebalance.target.Target

  Base class for all fitting targets.
- class forcebalance.target.RemoteTarget

Namespaces

- namespace forcebalance.target

Variables

- tuple forcebalance.target.logger = getLogger(__name__)

9.32 tinkerio.py File Reference

Classes

- class forcebalance.tinkerio.Tinker_Reader

  Finite state machine for parsing TINKER force field files.
- class forcebalance.tinkerio.Liquid_TINKER
- class forcebalance.tinkerio.AbInitio_TINKER

  Subclass of Target for force and energy matching using TINKER.
- class forcebalance.tinkerio.Vibration_TINKER

  Subclass of Target for vibrational frequency matching using TINKER.
- class forcebalance.tinkerio.Moments_TINKER

  Subclass of Target for multipole moment matching using TINKER.
- class forcebalance.tinkerio.BindingEnergy_TINKER

  Subclass of BindingEnergy for binding energy matching using TINKER.
- class forcebalance.tinkerio.Interaction_TINKER

  Subclass of Target for interaction matching using TINKER.

Namespaces

- namespace forcebalance.tinkerio

  TINKER input/output.

Functions

- def forcebalance.tinkerio.write_key_with_prm

  Copies a TINKER .key file but changes the parameter keyword as necessary to reflect the ForceBalance settings.
- def forcebalance.tinkerio.modify_key

  Performs in-place modification of a TINKER .key file.

Variables

- tuple forcebalance.tinkerio.logger = getLogger(__name__)
- dictionary forcebalance.tinkerio.pdict

9.33 vibration.py File Reference

Classes

- class forcebalance.vibration.Vibration

  Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).

Namespaces

- namespace forcebalance.vibration

  Vibrational mode fitting module.

Variables

- tuple forcebalance.vibration.logger = getLogger(__name__)

Index

__add__
forcebalance::molecule::Molecule, 436
__delitem__
forcebalance::molecule::Molecule, 436
__eq__
forcebalance::forcefield::FF, 254
__getattr__
forcebalance::molecule::Molecule, 437
__getitem__
forcebalance::molecule::Molecule, 437
__iadd__
forcebalance::molecule::Molecule, 438
__init__
forcebalance::abinitio::AbInitio, 91
forcebalance::abinitio_internal::AbInitio_Internal, 151
forcebalance::amberio::AbInitio_AMBER, 110
forcebalance::amberio::FrcMod_Reader, 268
forcebalance::amberio::Mol2_Reader, 425
forcebalance::BaseClass, 211
forcebalance::BaseReader, 214
forcebalance::binding::BindingEnergy, 218
forcebalance::counterpoise::Counterpoise, 241
forcebalance::custom_io::Gen_Reader, 277
forcebalance::forcefield::BackedUpDict, 208
forcebalance::forcefield::FF, 253
forcebalance::gmxio::AbInitio_GMX, 130
forcebalance::gmxio::Interaction_GMX, 299
forcebalance::gmxio::ITP_Reader, 338
forcebalance::gmxio::Liquid_GMX, 368
forcebalance::gmxqpio::Monomer_QTPIE, 484
forcebalance::interaction::Interaction, 286
forcebalance::leastsq::LeastSquares, 343
forcebalance::liquid::Liquid, 354
forcebalance::Mol2::mol2, 410
forcebalance::Mol2::mol2_atom, 416
forcebalance::Mol2::mol2_bond, 421
forcebalance::Mol2::mol2_set, 431
forcebalance::mol2io::Mol2_Reader, 429
forcebalance::molecule::Molecule, 436
forcebalance::moments::Moments, 460
forcebalance::nifty::Pickler_LP, 518
forcebalance::nifty::Unpickler_LP, 574
forcebalance::objective::Objective, 495
forcebalance::objective::Penalty, 514
forcebalance::openmmio::AbInitio_OpenMM, 170
forcebalance::openmmio::Interaction_OpenMM, 313
forcebalance::openmmio::Liquid_OpenMM, 383
forcebalance::openmmio::OpenMM_Reader, 499
forcebalance::optimizer::Optimizer, 504
forcebalance::output::CleanStreamHandler, 238
forcebalance::output::ForceBalanceLogger, 266
forcebalance::output::RawStreamHandler, 525
forcebalance::psi4io::GBS_Reader, 272
forcebalance::psi4io::Grid_Reader, 281
forcebalance::psi4io::RDVR3_Psi4, 528
forcebalance::psi4io::THCDF_Psi4, 560
forcebalance::qchemio::QCIn_Reader, 520
forcebalance::target::RemoteTarget, 539
forcebalance::target::Target, 549
forcebalance::tinkerio::AbInitio_TINKER, 191
forcebalance::tinkerio::BindingEnergy_TINKER, 228
forcebalance::tinkerio::Interaction_TINKER, 326
forcebalance::tinkerio::Liquid_TINKER, 398
forcebalance::tinkerio::Moments_TINKER, 472
forcebalance::tinkerio::Tinker_Reader, 571
forcebalance::tinkerio::Vibration_TINKER, 587
forcebalance::vibration::Vibration, 577
__init__.py, 589
__iter__
forcebalance::molecule::Molecule, 438
__len__
forcebalance::molecule::Molecule, 439
__missing__
forcebalance::forcefield::BackedUpDict, 209
__repr__
forcebalance::Mol2::mol2, 410
forcebalance::Mol2::mol2_atom, 416
forcebalance::Mol2::mol2_bond, 421
__setattr__
forcebalance::molecule::Molecule, 439
__version__
forcebalance, 12

A
forcebalance::chemistry, 18
a
forcebalance::objective::Penalty, 516
abinitio.py, 589
abinitio_internal.py, 590
add_quantum
forcebalance::molecule::Molecule, 439
add_strip_to_mat
forcebalance::molecule, 40
add_virtual_site
forcebalance::molecule::Molecule, 440
addHandler
forcebalance::output::ForceBalanceLogger, 266
addff
forcebalance::forcefield::FF, 254
addff_txt
forcebalance::forcefield::FF, 255
addff_xml

forcebalance::forcefield::FF, 256
adict
forcebalance::amberio::FrcMod_Reader, 269
forcebalance::amberio::Mol2_Reader, 426
forcebalance::BaseReader, 215
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 339
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 274
forcebalance::psi4io::Grid_Reader, 282
forcebalance::qchemio::QCIn_Reader, 521
forcebalance::tinkerio::Tinker_Reader, 573
aftypes
forcebalance::gmxio, 33
align
forcebalance::molecule::Molecule, 440
align_by_moments
forcebalance::molecule::Molecule, 440
align_center
forcebalance::molecule::Molecule, 440
AlignToDensity
forcebalance::molecule, 40
AlignToMoments
forcebalance::molecule, 41
Alive
forcebalance::molecule, 47
all_at_once
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::tinkerio::AbInitio_TINKER, 202
all_pairwise_rmsd
forcebalance::molecule::Molecule, 440
AllVariableNames
forcebalance::molecule, 47
allsplit
forcebalance::nifty, 52
amberio.py, 590
amom
forcebalance::psi4io::GBS_Reader, 274
Anneal
forcebalance::optimizer::Optimizer, 505
api.dox, 590
append
forcebalance::molecule::Molecule, 441
assign_field
forcebalance::forcefield::FF, 256
assign_p0
forcebalance::forcefield::FF, 257
atom
forcebalance::amberio::FrcMod_Reader, 269
forcebalance::amberio::Mol2_Reader, 426
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::qchemio::QCIn_Reader, 521
forcebalance::tinkerio::Tinker_Reader, 573
atom_distances
forcebalance::contact, 20
atom_id
forcebalance::Mol2::mol2_atom, 419
atom_name
forcebalance::Mol2::mol2_atom, 419
atom_select
forcebalance::molecule::Molecule, 441
atom_stack
forcebalance::molecule::Molecule, 441
atom_type
forcebalance::Mol2::mol2_atom, 419
AtomLists
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::tinkerio::AbInitio_TINKER, 202
AtomMask
forcebalance::gmxio::AbInitio_GMX, 141
AtomTypes
forcebalance::amberio::FrcMod_Reader, 269
forcebalance::amberio::Mol2_Reader, 426
forcebalance::BaseReader, 215
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 339
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 274
forcebalance::psi4io::Grid_Reader, 282
forcebalance::qchemio::QCIn_Reader, 521
forcebalance::tinkerio::Tinker_Reader, 573
AtomVariableNames
forcebalance::molecule, 47
atomnames
forcebalance::amberio::Mol2_Reader, 426
forcebalance::forcefield::FF, 261
forcebalance::gmxio::ITP_Reader, 339
atoms
forcebalance::chemistry, 18
forcebalance::Mol2::mol2, 414
atomtype_to_mass
forcebalance::gmxio::ITP_Reader, 339
atomtypes
forcebalance::gmxio::ITP_Reader, 340

B
forcebalance::chemistry, 18
b
forcebalance::objective::Penalty, 516
BE
forcebalance::chemistry, 18
BFGS

forcebalance::optimizer::Optimizer, 505
backup_dict
forcebalance::forcefield::BackedUpDict, 209
basis_number
forcebalance::psi4io::GBS_Reader, 274
bftypes
forcebalance::gmxio, 33
bhyp
forcebalance::optimizer::Optimizer, 511
binding.py, 590
bkwd
forcebalance::parser, 79
bo
forcebalance::chemistry, 18
bohrang
forcebalance::molecule, 47
forcebalance::nifty, 66
bond_id
forcebalance::Mol2::mol2_bond, 423
bond_type
forcebalance::Mol2::mol2_bond, 423
BondChars
forcebalance::chemistry, 18
BondEnergies
forcebalance::chemistry, 18
BondStrengthByLength
forcebalance::chemistry, 18
bonds
forcebalance::Mol2::mol2, 414
both
forcebalance::molecule, 41
Box
forcebalance::molecule, 47
boxes
forcebalance::molecule::Molecule, 454
build_invdist
forcebalance::abinitio::AbInitio, 91
forcebalance::abinitio_internal::AbInitio_Internal, 151
forcebalance::amberio::AbInitio_AMBER, 110
forcebalance::gmxio::AbInitio_GMX, 130
forcebalance::openmmio::AbInitio_OpenMM, 170
forcebalance::tinkerio::AbInitio_TINKER, 191
build_objective
forcebalance::abinitio, 13
build_pid
forcebalance::amberio::FrcMod_Reader, 268
forcebalance::amberio::Mol2_Reader, 425
forcebalance::BaseReader, 214
forcebalance::custom_io::Gen_Reader, 277
forcebalance::gmxio::ITP_Reader, 338
forcebalance::mol2io::Mol2_Reader, 429
forcebalance::openmmio::OpenMM_Reader, 499, 500
forcebalance::psi4io::GBS_Reader, 272
forcebalance::psi4io::Grid_Reader, 281
forcebalance::qchemio::QCIn_Reader, 520
forcebalance::tinkerio::Tinker_Reader, 571
build_topology
forcebalance::molecule::Molecule, 441
BuildLatticeFromLengthsAngles
forcebalance::molecule, 41
BuildLatticeFromVectors
forcebalance::molecule, 41

CHECK_BASIS
forcebalance::leastsq, 36
calc_eigvals
forcebalance::vibration::Vibration, 585
calc_moments
forcebalance::gmxqpio::Monomer_QTPIE, 492
forcebalance::moments::Moments, 467
forcebalance::tinkerio::Moments_TINKER, 479
call_derivatives
forcebalance::leastsq::LeastSquares, 350
forcebalance::psi4io::THCDF_Psi4, 567
callderivs
forcebalance::psi4io::RDVR3_Psi4, 535
center_of_mass
forcebalance::molecule::Molecule, 442
charge
forcebalance::Mol2::mol2_atom, 419
charge_type
forcebalance::Mol2::mol2, 414
CheckBasis
forcebalance::leastsq, 36
chemistry.py, 591
chk
forcebalance::optimizer::Optimizer, 511
cnum
forcebalance::qchemio::QCIn_Reader, 521
col
forcebalance::nifty, 52
commadash
forcebalance::nifty, 53
comments
forcebalance::Mol2::mol2, 415
forcebalance::Mol2::mol2_set, 432
comms
forcebalance::molecule::Molecule, 454
compounds
forcebalance::Mol2::mol2_set, 432
compute
forcebalance::objective::Penalty, 514
compute_netforce_torque
forcebalance::abinitio::AbInitio, 91
forcebalance::abinitio_internal::AbInitio_Internal, 151
forcebalance::amberio::AbInitio_AMBER, 110
forcebalance::gmxio::AbInitio_GMX, 131

forcebalance::openmmio::AbInitio_OpenMM, 170
forcebalance::tinkerio::AbInitio_TINKER, 191
ComputeOverlap
forcebalance::molecule, 42
concurrent_map
forcebalance::nifty, 53
ConjugateGradient
forcebalance::optimizer::Optimizer, 505
contact.py, 591
contraction_number
forcebalance::psi4io::GBS_Reader, 274
CopyAmoebaAngleParameters
forcebalance::openmmio, 69
CopyAmoebaBondParameters
forcebalance::openmmio, 69
CopyAmoebaInPlaneAngleParameters
forcebalance::openmmio, 69
CopyAmoebaMultipoleParameters
forcebalance::openmmio, 70
CopyAmoebaOutOfPlaneBendParameters
forcebalance::openmmio, 70
CopyAmoebaVdwParameters
forcebalance::openmmio, 70
CopyFile
forcebalance::nifty, 53
CopyHarmonicAngleParameters
forcebalance::openmmio, 70
CopyHarmonicBondParameters
forcebalance::openmmio, 70
CopyNonbondedParameters
forcebalance::openmmio, 71
CopyPeriodicTorsionParameters
forcebalance::openmmio, 71
CopySystemParameters
forcebalance::openmmio, 71
Counter
forcebalance::optimizer, 74
counterpoise.py, 592
cpqm
forcebalance::counterpoise::Counterpoise, 249
cptypes
forcebalance::custom_io, 22
create_mvals
forcebalance::forcefield::FF, 257
create_pvals
forcebalance::forcefield::FF, 258
createWorkQueue
forcebalance::nifty, 53
custom_io.py, 592

D
forcebalance::leastsq::LeastSquares, 350
forcebalance::psi4io::THCDF_Psi4, 567
DATfnm
forcebalance::psi4io::THCDF_Psi4, 567
DF_Energy
forcebalance::psi4io::THCDF_Psi4, 567
Data
forcebalance::molecule::Molecule, 454
data
forcebalance::Mol2, 38
data_from_web
forcebalance::chemistry, 19
defaultHandler
forcebalance::output::ForceBalanceLogger, 266
denoms
forcebalance::moments::Moments, 467
forcebalance::tinkerio::Moments_TINKER, 480
destroy
forcebalance::psi4io::GBS_Reader, 274
determine_fftype
forcebalance::forcefield, 29
dftypes
forcebalance::gmxio, 33
Diel_B
forcebalance::gmxio::Interaction_GMX, 307
Dielectric
forcebalance::gmxio::Interaction_GMX, 307
diff
forcebalance::molecule, 42
dihe
forcebalance::amberio::FrcMod_Reader, 269
divisor
forcebalance::gmxio::Interaction_GMX, 307
forcebalance::interaction::Interaction, 293
forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::tinkerio::Interaction_TINKER, 333
do_nothing
forcebalance::openmmio, 72
do_self_pol
forcebalance::gmxio::Liquid_GMX, 376
forcebalance::liquid::Liquid, 362
forcebalance::openmmio::Liquid_OpenMM, 391
forcebalance::tinkerio::Liquid_TINKER, 406
driver
forcebalance::psi4io::RDVR3_Psi4, 528
forcebalance::psi4io::THCDF_Psi4, 560
dx
forcebalance::optimizer::Optimizer, 511
DynDict
forcebalance::tinkerio::Liquid_TINKER, 406
DynDict_New
forcebalance::tinkerio::Liquid_TINKER, 406

e_err
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121

forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::gmxio::Interaction_GMX, 307
forcebalance::interaction::Interaction, 293
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::tinkerio::AbInitio_TINKER, 202
forcebalance::tinkerio::Interaction_TINKER, 333
e_err_pct
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::gmxio::Interaction_GMX, 307
forcebalance::interaction::Interaction, 293
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::tinkerio::AbInitio_TINKER, 202
forcebalance::tinkerio::Interaction_TINKER, 333
e_ref
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::tinkerio::AbInitio_TINKER, 202
edit_mdp
forcebalance::gmxio, 32
edit_qcrems
forcebalance::molecule::Molecule, 442
either
forcebalance::molecule, 42
element
forcebalance::psi4io::GBS_Reader, 274
forcebalance::psi4io::Grid_Reader, 282
Elements
forcebalance::chemistry, 19
forcebalance::molecule, 47
forcebalance::psi4io::THCDF_Psi4, 567
forcebalance::PT, 81
elements
forcebalance::psi4io::RDVR3_Psi4, 535
emd0
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::tinkerio::AbInitio_TINKER, 202
emit
forcebalance::output::CleanFileHandler, 237
forcebalance::output::CleanStreamHandler, 239
forcebalance::output::RawFileHandler, 523
forcebalance::output::RawStreamHandler, 525
emm
forcebalance::gmxio::Interaction_GMX, 307
forcebalance::interaction::Interaction, 293
forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::tinkerio::Interaction_TINKER, 333
encode
forcebalance::nifty, 54
energy_driver_all
forcebalance::openmmio::Interaction_OpenMM, 313
forcebalance::tinkerio::AbInitio_TINKER, 192
forcebalance::tinkerio::Interaction_TINKER, 326
energy_force_driver
forcebalance::gmxio::AbInitio_GMX, 131
forcebalance::tinkerio::AbInitio_TINKER, 192
energy_force_driver_all
forcebalance::abinitio_internal::AbInitio_Internal, 152
forcebalance::amberio::AbInitio_AMBER, 110
forcebalance::gmxio::AbInitio_GMX, 131
forcebalance::openmmio::AbInitio_OpenMM, 170
forcebalance::tinkerio::AbInitio_TINKER, 192
energy_force_driver_all_external_
forcebalance::amberio::AbInitio_AMBER, 110
forcebalance::openmmio::AbInitio_OpenMM, 171
energy_force_driver_all_internal_
forcebalance::openmmio::AbInitio_OpenMM, 171
energy_force_transformer
forcebalance::abinitio::AbInitio, 91
forcebalance::abinitio_internal::AbInitio_Internal, 152
forcebalance::amberio::AbInitio_AMBER, 111
forcebalance::gmxio::AbInitio_GMX, 131
forcebalance::openmmio::AbInitio_OpenMM, 171
forcebalance::tinkerio::AbInitio_TINKER, 192
energy_force_transformer_all
forcebalance::abinitio::AbInitio, 92
forcebalance::abinitio_internal::AbInitio_Internal, 152
forcebalance::amberio::AbInitio_AMBER, 111
forcebalance::gmxio::AbInitio_GMX, 132
forcebalance::openmmio::AbInitio_OpenMM, 171
forcebalance::tinkerio::AbInitio_TINKER, 193
energy_part
forcebalance::binding::BindingEnergy, 224
forcebalance::tinkerio::BindingEnergy_TINKER, 234
engine
forcebalance::gmxio::Liquid_GMX, 376
forcebalance::openmmio::Liquid_OpenMM, 391
eqcgmx
forcebalance::nifty, 66
eqm
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::gmxio::Interaction_GMX, 307
forcebalance::interaction::Interaction, 293
forcebalance::openmmio::AbInitio_OpenMM, 181

forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::tinkerio::AbInitio_TINKER, 202
forcebalance::tinkerio::Interaction_TINKER, 333
esp_err
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::tinkerio::AbInitio_TINKER, 202
espval
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 121
forcebalance::gmxio::AbInitio_GMX, 141
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::tinkerio::AbInitio_TINKER, 202
espxyz
forcebalance::abinitio::AbInitio, 101
forcebalance::abinitio_internal::AbInitio_Internal, 161
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 181
forcebalance::tinkerio::AbInitio_TINKER, 203
EulerMatrix
forcebalance::molecule, 42
even_list
forcebalance::molecule, 42
excision
forcebalance::forcefield::FF, 261
forcebalance::optimizer::Optimizer, 511
extra_output
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::liquid::Liquid, 362
forcebalance::openmmio::Liquid_OpenMM, 392
forcebalance::tinkerio::Liquid_TINKER, 406

f12d3p
forcebalance::finite_difference, 24
f12d7p
forcebalance::finite_difference, 24
f1d2p
forcebalance::finite_difference, 25
f1d5p
forcebalance::finite_difference, 25
f1d7p
forcebalance::finite_difference, 26
f_err
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
f_err_pct
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
f_ref
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
FDCheckG
forcebalance::optimizer::Optimizer, 505
FDCheckH
forcebalance::optimizer::Optimizer, 506
FF
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::binding::BindingEnergy, 224
forcebalance::counterpoise::Counterpoise, 249
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::gmxio::Interaction_GMX, 307
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::gmxqpio::Monomer_QTPIE, 492
forcebalance::interaction::Interaction, 293
forcebalance::leastsq::LeastSquares, 350
forcebalance::liquid::Liquid, 362
forcebalance::moments::Moments, 467
forcebalance::objective::Objective, 496
forcebalance::objective::Penalty, 516
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::openmmio::Liquid_OpenMM, 392
forcebalance::optimizer::Optimizer, 512
forcebalance::psi4io::RDVR3_Psi4, 535
forcebalance::psi4io::THCDF_Psi4, 568
forcebalance::target::RemoteTarget, 545
forcebalance::target::Target, 556
forcebalance::tinkerio::AbInitio_TINKER, 203
forcebalance::tinkerio::BindingEnergy_TINKER, 235
forcebalance::tinkerio::Interaction_TINKER, 333
forcebalance::tinkerio::Liquid_TINKER, 406
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585
FF_Extensions
forcebalance::forcefield, 30
FF_IOModules
forcebalance::forcefield, 30
FFAtomTypes
forcebalance::forcefield::FF, 262

FFMolecules
forcebalance::forcefield::FF, 262
FUSE
forcebalance::objective::Penalty, 514
FUSE_BARRIER
forcebalance::objective::Penalty, 515
FUSE_L0
forcebalance::objective::Penalty, 515
factor
forcebalance::psi4io::RDVR3_Psi4, 535
fadd
forcebalance::objective::Penalty, 516
fdict
forcebalance::custom_io, 22
forcebalance::gmxio, 33
fdwrap
forcebalance::finite_difference, 26
fdwrap_G
forcebalance::finite_difference, 27
fdwrap_H
forcebalance::finite_difference, 27
feed
forcebalance::amberio::FrcMod_Reader, 269
forcebalance::amberio::Mol2_Reader, 426
forcebalance::BaseReader, 214
forcebalance::custom_io::Gen_Reader, 277
forcebalance::gmxio::ITP_Reader, 338
forcebalance::mol2io::Mol2_Reader, 429
forcebalance::openmmio::OpenMM_Reader, 500
forcebalance::psi4io::GBS_Reader, 273
forcebalance::psi4io::Grid_Reader, 281
forcebalance::qchemio::QCIn_Reader, 521
forcebalance::tinkerio::Tinker_Reader, 571
ffdata
forcebalance::forcefield::FF, 262
ffdata_isxml
forcebalance::forcefield::FF, 262
find_spacings
forcebalance::forcefield::FF, 258
finite_difference.py, 593
fitatoms
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
flat
forcebalance::nifty, 54
floatornan
forcebalance::nifty, 54
fmul
forcebalance::objective::Penalty, 516
force
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
force_map
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
forcebalance, 11
__version__, 12
WORK_QUEUE, 12
WQIDS, 12
forcebalance.abinitio, 13
forcebalance.abinitio.AbInitio, 87
forcebalance.abinitio_internal, 14
forcebalance.abinitio_internal.AbInitio_Internal, 146
forcebalance.amberio, 15
forcebalance.amberio.AbInitio_AMBER, 106
forcebalance.amberio.FrcMod_Reader, 267
forcebalance.amberio.Mol2_Reader, 424
forcebalance.BaseClass, 209
forcebalance.BaseReader, 211
forcebalance.binding, 16
forcebalance.binding.BindingEnergy, 216
forcebalance.chemistry, 17
forcebalance.contact, 20
forcebalance.counterpoise, 21
forcebalance.counterpoise.Counterpoise, 239
forcebalance.custom_io, 22
forcebalance.custom_io.Gen_Reader, 275
forcebalance.finite_difference, 23
forcebalance.forcefield, 28
forcebalance.forcefield.BackedUpDict, 207
forcebalance.forcefield.FF, 250
forcebalance.gmxio, 30
forcebalance.gmxio.AbInitio_GMX, 126
forcebalance.gmxio.ITP_Reader, 335
forcebalance.gmxio.Interaction_GMX, 295
forcebalance.gmxio.Liquid_GMX, 365
forcebalance.gmxqpio, 34
forcebalance.gmxqpio.Monomer_QTPIE, 481
forcebalance.interaction, 35
forcebalance.interaction.Interaction, 283
forcebalance.leastsq, 36
forcebalance.leastsq.LeastSquares, 341
forcebalance.liquid, 37
forcebalance.liquid.Liquid, 351
forcebalance.Mol2, 38
forcebalance.Mol2.mol2, 409

forcebalance.Mol2.mol2_atom, 415
forcebalance.Mol2.mol2_bond, 420
forcebalance.Mol2.mol2_set, 431
forcebalance.mol2io, 38
forcebalance.mol2io.Mol2_Reader, 427
forcebalance.molecule, 39
forcebalance.molecule.Molecule, 432
forcebalance.molecule.MolfileTimestep, 455
forcebalance.moments, 49
forcebalance.moments.Moments, 456
forcebalance.nifty, 49
forcebalance.nifty.Pickler_LP, 517
forcebalance.nifty.Unpickler_LP, 573
forcebalance.objective, 67
forcebalance.objective.Objective, 493
forcebalance.objective.Penalty, 513
forcebalance.openmmio, 68
forcebalance.openmmio.AbInitio_OpenMM, 166
forcebalance.openmmio.Interaction_OpenMM, 309
forcebalance.openmmio.Liquid_OpenMM, 380
forcebalance.openmmio.OpenMM_Reader, 497
forcebalance.optimizer, 74
forcebalance.optimizer.Optimizer, 502
forcebalance.output, 75
forcebalance.output.CleanFileHandler, 236
forcebalance.output.CleanStreamHandler, 237
forcebalance.output.ForceBalanceLogger, 264
forcebalance.output.RawFileHandler, 522
forcebalance.output.RawStreamHandler, 524
forcebalance.PT, 81
forcebalance.parser, 75
forcebalance.psi4io, 80
forcebalance.psi4io.GBS_Reader, 270
forcebalance.psi4io.Grid_Reader, 279
forcebalance.psi4io.RDVR3_Psi4, 525
forcebalance.psi4io.THCDF_Psi4, 557
forcebalance.qchemio, 82
forcebalance.qchemio.QCIn_Reader, 519
forcebalance.target, 83
forcebalance.target.RemoteTarget, 537
forcebalance.target.Target, 546
forcebalance.tinkerio, 84
forcebalance.tinkerio.AbInitio_TINKER, 186
forcebalance.tinkerio.BindingEnergy_TINKER, 226
forcebalance.tinkerio.Interaction_TINKER, 322
forcebalance.tinkerio.Liquid_TINKER, 395
forcebalance.tinkerio.Moments_TINKER, 469
forcebalance.tinkerio.Tinker_Reader, 569
forcebalance.tinkerio.Vibration_TINKER, 586
forcebalance.vibration, 86
forcebalance.vibration.Vibration, 575
forcebalance::BaseClass
    __init__, 211
    PrintOptionDict, 211
    set_option, 211
    verbose_options, 211
forcebalance::BaseReader
    __init__, 214
    adict, 215
    AtomTypes, 215
    build_pid, 214
    feed, 214
    itype, 215
    ln, 215
    molatom, 215
    Molecules, 215
    pdict, 215
    Split, 214
    suffix, 215
    Whites, 214
forcebalance::Mol2
    data, 38
forcebalance::Mol2::mol2
    __init__, 410
    __repr__, 410
    atoms, 414
    bonds, 414
    charge_type, 414
    comments, 415
    get_atom, 410
    get_bonded_atoms, 411
    mol_name, 415
    mol_type, 415
    num_atoms, 415
    num_bonds, 415
    num_feat, 415
    num_sets, 415
    num_subst, 415
    out, 411
    parse, 411
    set_charge_type, 411
    set_donnor_acceptor_atoms, 412
    set_mol_name, 412
    set_mol_type, 412
    set_num_atoms, 413
    set_num_bonds, 413
    set_num_feat, 413
    set_num_sets, 414
    set_num_subst, 414
forcebalance::Mol2::mol2_atom
    __init__, 416
    __repr__, 416
    atom_id, 419
    atom_name, 419
    atom_type, 419
    charge, 419
    parse, 417
    set_atom_id, 417

set_atom_name, 417
set_atom_type, 417
set_charge, 418
set_crds, 418
set_status_bit, 418
set_subst_id, 418
set_subst_name, 419
status_bit, 419
subst_id, 420
subst_name, 420
x, 420
y, 420
z, 420
forcebalance::Mol2::mol2_bond
__init__, 421
__repr__, 421
bond_id, 423
bond_type, 423
origin_atom_id, 423
parse, 421
set_bond_id, 422
set_bond_type, 422
set_origin_atom_id, 422
set_status_bit, 422
set_target_atom_id, 423
status_bit, 423
target_atom_id, 423
forcebalance::Mol2::mol2_set
__init__, 431
comments, 432
compounds, 432
num_compounds, 432
parse, 432
forcebalance::PT
Elements, 81
PeriodicTable, 81
forcebalance::abinitio
build_objective, 13
logger, 14
weighted_variance, 13
weighted_variance2, 14
forcebalance::abinitio::AbInitio
__init__, 91
AtomLists, 101
build_invdist, 91
compute_netforce_torque, 91
e_err, 101
e_err_pct, 101
e_ref, 101
emd0, 101
energy_force_transformer, 91
energy_force_transformer_all, 92
eqm, 101
esp_err, 101
espval, 101
espxyz, 101
f_err, 102
f_err_pct, 102
f_ref, 102
FF, 102
fitatoms, 102
force, 102
force_map, 102
fqm, 102
fref, 102
gct, 102
get, 92
get_G, 94
get_H, 94
get_X, 95
get_energy_force_, 92
get_resp_, 95
hct, 102
indicate, 96
invdists, 102
link_from_tempdir, 96
nesp, 103
new_vsites, 103
nf_err, 103
nf_err_pct, 103
nf_ref, 103
nftqm, 103
nnf, 103
nparticles, 103
ns, 103
ntq, 103
objective, 103
prepare_temp_directory, 96
PrintOptionDict, 103
printcool_table, 96
qfnm, 104
qmatoms, 104
qmboltz_wts, 104
read_reference_data, 97
read_topology, 98
refresh_temp_directory, 98
respterm, 104
rundir, 104
save_vmvals, 104
set_option, 99
sget, 99
stage, 100
submit_jobs, 100
tempdir, 104
topology_flag, 104
tq_err, 104
tq_err_pct, 104
tq_ref, 105

traj, 105
use_nft, 105
verbose_options, 105
w_energy, 105
w_force, 105
w_netforce, 105
w_resp, 105
w_torque, 105
whamboltz, 105
whamboltz_wts, 105
wq_complete, 100
xct, 105
forcebalance::abinitio_internal::AbInitio_Internal
__init__, 151
AtomLists, 161
build_invdist, 151
compute_netforce_torque, 151
e_err, 161
e_err_pct, 161
e_ref, 161
emd0, 161
energy_force_driver_all, 152
energy_force_transformer, 152
energy_force_transformer_all, 152
eqm, 161
esp_err, 161
espval, 161
espxyz, 161
f_err, 162
f_err_pct, 162
f_ref, 162
FF, 162
fitatoms, 162
force, 162
force_map, 162
fqm, 162
fref, 162
gct, 162
get, 152
get_G, 154
get_H, 155
get_X, 155
get_energy_force_, 153
get_resp_, 155
hct, 162
indicate, 156
invdists, 162
link_from_tempdir, 156
nesp, 163
new_vsites, 163
nf_err, 163
nf_err_pct, 163
nf_ref, 163
nftqm, 163
nnf, 163
nparticles, 163
ns, 163
ntq, 163
objective, 163
prepare_temp_directory, 156
PrintOptionDict, 163
printcool_table, 156
qfnm, 164
qmatoms, 164
qmboltz_wts, 164
read_reference_data, 157
read_topology, 158
refresh_temp_directory, 158
respterm, 164
rundir, 164
save_vmvals, 164
set_option, 159
sget, 159
stage, 160
submit_jobs, 160
tempdir, 164
topology_flag, 164
tq_err, 164
tq_err_pct, 164
tq_ref, 165
traj, 165
trajfnm, 165
use_nft, 165
verbose_options, 165
w_energy, 165
w_force, 165
w_netforce, 165
w_resp, 165
w_torque, 165
whamboltz, 165
whamboltz_wts, 165
wq_complete, 160
xct, 166
forcebalance::amberio
frcmod_pdict, 16
is_mol2_atom, 15
logger, 16
mol2_pdict, 16
forcebalance::amberio::AbInitio_AMBER
__init__, 110
all_at_once, 121
AtomLists, 121
build_invdist, 110
compute_netforce_torque, 110
e_err, 121
e_err_pct, 121
e_ref, 121
emd0, 121

energy_force_driver_all, 110
energy_force_transformer, 111
energy_force_transformer_all, 111
eqm, 121
esp_err, 121
espval, 121
espxyz, 122
f_err, 122
f_err_pct, 122
f_ref, 122
FF, 122
fitatoms, 122
force, 122
force_map, 122
fqm, 122
fref, 122
gct, 122
get, 112
get_G, 113
get_H, 114
get_X, 115
get_energy_force_, 112
get_resp_, 115
hct, 122
indicate, 116
invdists, 123
link_from_tempdir, 116
nesp, 123
new_vsites, 123
nf_err, 123
nf_err_pct, 123
nf_ref, 123
nftqm, 123
nnf, 123
nparticles, 123
ns, 123
ntq, 123
objective, 123
prepare_temp_directory, 116
PrintOptionDict, 124
printcool_table, 117
qfnm, 124
qmatoms, 124
qmboltz_wts, 124
read_reference_data, 117
read_topology, 118
refresh_temp_directory, 118
respterm, 124
rundir, 124
save_vmvals, 124
set_option, 119
sget, 119
stage, 120
submit_jobs, 120
tempdir, 124
topology_flag, 124
tq_err, 124
tq_err_pct, 125
tq_ref, 125
traj, 125
trajfnm, 125
use_nft, 125
verbose_options, 125
w_energy, 125
w_force, 125
w_netforce, 125
w_resp, 125
w_torque, 125
whamboltz, 125
whamboltz_wts, 125
wq_complete, 120
xct, 126
forcebalance::amberio::FrcMod_Reader
__init__, 268
adict, 269
atom, 269
AtomTypes, 269
build_pid, 268
dihe, 269
feed, 269
itype, 269
ln, 270
molatom, 270
Molecules, 270
pdict, 270
Split, 269
suffix, 270
Whites, 269
forcebalance::amberio::Mol2_Reader
__init__, 425
adict, 426
atom, 426
AtomTypes, 426
atomnames, 426
build_pid, 425
feed, 426
itype, 427
ln, 427
mol, 427
molatom, 427
Molecules, 427
pdict, 427
section, 427
Split, 426
suffix, 427
Whites, 426
forcebalance::binding
logger, 17

parse_interactions, 17
forcebalance::binding::BindingEnergy
__init__, 218
energy_part, 224
FF, 224
gct, 224
get, 218
get_G, 218
get_H, 219
get_X, 220
hct, 224
indicate, 221
inter_opts, 224
link_from_tempdir, 221
objective, 225
PrintDict, 225
PrintOptionDict, 225
printcool_table, 221
RMSDDict, 225
refresh_temp_directory, 222
rmsd_part, 225
rundir, 225
set_option, 222
sget, 222
stage, 223
submit_jobs, 224
tempdir, 225
verbose_options, 225
wq_complete, 224
xct, 225
forcebalance::chemistry
A, 18
atoms, 18
B, 18
BE, 18
bo, 18
BondChars, 18
BondEnergies, 18
BondStrengthByLength, 18
data_from_web, 19
Elements, 19
L, 19
line, 19
LookupByMass, 18
PeriodicTable, 19
Radii, 19
forcebalance::contact
atom_distances, 20
residue_distances, 20
forcebalance::counterpoise
logger, 22
forcebalance::counterpoise::Counterpoise
__init__, 241
cpqm, 249
FF, 249
gct, 249
get, 242
get_G, 242
get_H, 243
get_X, 244
hct, 249
link_from_tempdir, 245
load_cp, 245
loadxyz, 245
na, 249
ns, 249
PrintOptionDict, 249
printcool_table, 245
refresh_temp_directory, 246
rundir, 249
set_option, 247
sget, 247
stage, 248
submit_jobs, 248
tempdir, 250
verbose_options, 250
wq_complete, 248
xct, 250
xyzs, 250
forcebalance::custom_io
cptypes, 22
fdict, 22
ndtypes, 23
pdict, 23
forcebalance::custom_io::Gen_Reader
__init__, 277
adict, 278
AtomTypes, 278
build_pid, 277
feed, 277
itype, 278
ln, 278
molatom, 278
Molecules, 278
pdict, 278
sec, 278
Split, 277
suffix, 278
Whites, 277
forcebalance::finite_difference
f12d3p, 24
f12d7p, 24
f1d2p, 25
f1d5p, 25
f1d7p, 26
fdwrap, 26
fdwrap_G, 27
fdwrap_H, 27

in_fd, 27
logger, 27
forcebalance::forcefield
determine_fftype, 29
FF_Extensions, 30
FF_IOModules, 30
logger, 30
rs_override, 29
forcebalance::forcefield::BackedUpDict
__init__, 208
__missing__, 209
backup_dict, 209
forcebalance::forcefield::FF
__eq__, 254
__init__, 253
addff, 254
addff_txt, 255
addff_xml, 256
assign_field, 256
assign_p0, 257
atomnames, 261
create_mvals, 257
create_pvals, 258
excision, 261
FFAtomTypes, 262
FFMolecules, 262
ffdata, 262
ffdata_isxml, 262
find_spacings, 258
linedestroy_save, 262
linedestroy_this, 262
list_map, 259
make, 259
make_redirect, 260
map, 262
mktransmat, 260
np, 262
openmmxml, 262
parmdestroy_save, 262
parmdestroy_this, 263
patoms, 263
pfields, 263
plist, 263
print_map, 260
PrintOptionDict, 263
pvals0, 263
qid, 263
qid2, 263
qmap, 263
Readers, 263
redirect, 263
rs, 264
rsmake, 261
set_option, 261
tinkerprm, 264
tm, 264
tml, 264
verbose_options, 264
forcebalance::gmxio
aftypes, 33
bftypes, 33
dftypes, 33
edit_mdp, 32
fdict, 33
logger, 34
nftypes, 34
parse_atomtype_line, 32
pdict, 34
pftypes, 34
rm_gmx_baks, 33
forcebalance::gmxio::AbInitio_GMX
__init__, 130
AtomLists, 141
AtomMask, 141
build_invdist, 130
compute_netforce_torque, 131
e_err, 141
e_err_pct, 141
e_ref, 141
emd0, 141
energy_force_driver, 131
energy_force_driver_all, 131
energy_force_transformer, 131
energy_force_transformer_all, 132
eqm, 141
esp_err, 141
espval, 141
espxyz, 142
f_err, 142
f_err_pct, 142
f_ref, 142
FF, 142
fitatoms, 142
force, 142
force_map, 142
fqm, 142
fref, 142
gct, 142
generate_vsite_positions, 132
get, 132
get_G, 134
get_H, 135
get_X, 135
get_energy_force_, 133
get_resp_, 135
hct, 142
indicate, 136
invdists, 143

link_from_tempdir, 136
nesp, 143
new_vsites, 143
nf_err, 143
nf_err_pct, 143
nf_ref, 143
nftqm, 143
nnf, 143
nparticles, 143
ns, 143
ntq, 143
objective, 143
prepare_temp_directory, 136
PrintOptionDict, 144
printcool_table, 137
qfnm, 144
qmatoms, 144
qmboltz_wts, 144
read_reference_data, 137
read_topology, 138
refresh_temp_directory, 139
respterm, 144
rundir, 144
save_vmvals, 144
set_option, 139
sget, 139
stage, 140
submit_jobs, 140
tempdir, 144
topfnm, 144
topology_flag, 144
tq_err, 145
tq_err_pct, 145
tq_ref, 145
traj, 145
trajfnm, 145
use_nft, 145
verbose_options, 145
w_energy, 145
w_force, 145
w_netforce, 145
w_resp, 145
w_torque, 145
whamboltz, 145
whamboltz_wts, 145
wq_complete, 140
xct, 146
forcebalance::gmxio::ITP_Reader
__init__, 338
adict, 339
AtomTypes, 339
atomnames, 339
atomtype_to_mass, 339
atomtypes, 340
build_pid, 338
feed, 338
itype, 340
ln, 340
mol, 340
molatom, 340
Molecules, 340
nbtype, 340
pdict, 340
sec, 340
Split, 339
suffix, 340
Whites, 339
forcebalance::gmxio::Interaction_GMX
__init__, 299
Diel_B, 307
Dielectric, 307
divisor, 307
e_err, 307
e_err_pct, 307
emm, 307
eqm, 307
FF, 307
gct, 307
get, 299
get_G, 299
get_H, 300
get_X, 301
hct, 308
indicate, 302
interaction_driver, 302
interaction_driver_all, 302
label, 308
link_from_tempdir, 303
ns, 308
objective, 308
prefactor, 308
prepare_temp_directory, 303
PrintOptionDict, 308
printcool_table, 303
qfnm, 308
read_reference_data, 304
refresh_temp_directory, 304
rundir, 308
select1, 308
select2, 308
set_option, 305
sget, 305
stage, 306
submit_jobs, 306
tempdir, 309
topfnm, 309
traj, 309
trajfnm, 309

verbose_options, 309
weight, 309
wq_complete, 306
xct, 309
forcebalance::gmxio::Liquid_GMX
__init__, 368
do_self_pol, 376
engine, 376
extra_output, 377
FF, 377
gas_fnm, 377
gct, 377
get, 368
get_G, 369
get_H, 370
get_X, 371
hct, 377
indicate, 372
Labels, 377
last_traj, 377
link_from_tempdir, 372
liquid_conf, 377
liquid_fnm, 377
liquid_traj, 377
MBarEnergy, 377
npt_simulation, 372
nptfiles, 377
nptpfx, 378
nptsfx, 378
objective_term, 372
PhasePoints, 378
polarization_correction, 373
prepare_temp_directory, 373
PrintOptionDict, 378
printcool_table, 373
read_data, 374
RefData, 378
refresh_temp_directory, 374
rundir, 378
SavedMVal, 378
SavedTraj, 378
set_option, 374
sget, 375
stage, 375
submit_jobs, 376
tempdir, 378
verbose_options, 378
w_alpha, 378
w_cp, 379
w_eps0, 379
w_hvap, 379
w_kappa, 379
w_rho, 379
wq_complete, 376
xct, 379
forcebalance::gmxqpio
get_monomer_properties, 35
logger, 35
forcebalance::gmxqpio::Monomer_QTPIE
__init__, 484
calc_moments, 492
FF, 492
gct, 492
get, 485
get_G, 485
get_H, 486
get_X, 487
hct, 492
indicate, 488
link_from_tempdir, 488
objective, 492
prepare_temp_directory, 488
PrintOptionDict, 492
printcool_table, 489
ref_moments, 493
refresh_temp_directory, 489
rundir, 493
set_option, 490
sget, 490
stage, 491
submit_jobs, 491
tempdir, 493
unpack_moments, 491
verbose_options, 493
weights, 493
wq_complete, 492
xct, 493
forcebalance::interaction
logger, 36
forcebalance::interaction::Interaction
__init__, 286
divisor, 293
e_err, 293
e_err_pct, 293
emm, 293
eqm, 293
FF, 293
gct, 293
get, 286
get_G, 286
get_H, 287
get_X, 288
hct, 293
indicate, 289
label, 294
link_from_tempdir, 289
ns, 294
objective, 294

prefactor, 294
prepare_temp_directory, 289
PrintOptionDict, 294
printcool_table, 289
qfnm, 294
read_reference_data, 290
refresh_temp_directory, 290
rundir, 294
select1, 294
select2, 294
set_option, 291
sget, 291
stage, 292
submit_jobs, 292
tempdir, 294
traj, 295
verbose_options, 295
weight, 295
wq_complete, 292
xct, 295
forcebalance::leastsq
CHECK_BASIS, 36
CheckBasis, 36
LAST_MVALS, 36
LastMvals, 36
logger, 36
forcebalance::leastsq::LeastSquares
__init__, 343
call_derivatives, 350
D, 350
FF, 350
gct, 350
get, 344
get_G, 344
get_H, 345
get_X, 346
hct, 350
indicate, 347
link_from_tempdir, 347
MAQ, 351
objective, 351
PrintOptionDict, 351
printcool_table, 347
refresh_temp_directory, 348
rundir, 351
set_option, 348
sget, 348
stage, 349
submit_jobs, 350
tempdir, 351
verbose_options, 351
wq_complete, 350
xct, 351
forcebalance::liquid
logger, 37
weight_info, 37
forcebalance::liquid::Liquid
__init__, 354
do_self_pol, 362
extra_output, 362
FF, 362
gct, 362
get, 354
get_G, 355
get_H, 356
get_X, 357
hct, 362
indicate, 358
Labels, 363
last_traj, 363
link_from_tempdir, 358
MBarEnergy, 363
npt_simulation, 358
nptfiles, 363
nptpfx, 363
nptsfx, 363
objective_term, 358
PhasePoints, 363
polarization_correction, 359
PrintOptionDict, 363
printcool_table, 359
read_data, 359
RefData, 363
refresh_temp_directory, 359
rundir, 363
SavedMVal, 363
SavedTraj, 363
set_option, 360
sget, 360
stage, 361
submit_jobs, 361
tempdir, 364
verbose_options, 364
w_alpha, 364
w_cp, 364
w_eps0, 364
w_hvap, 364
w_kappa, 364
w_rho, 364
wq_complete, 362
xct, 364
forcebalance::mol2io
mol2_pdict, 39
forcebalance::mol2io::Mol2_Reader
__init__, 429
adict, 430
atom, 430
AtomTypes, 430

build_pid, 429
feed, 429
itype, 430
ln, 430
molatom, 430
Molecules, 431
pdict, 431
Split, 430
suffix, 431
Whites, 430
forcebalance::molecule
add_strip_to_mat, 40
AlignToDensity, 40
AlignToMoments, 41
Alive, 47
AllVariableNames, 47
AtomVariableNames, 47
bohrang, 47
both, 41
Box, 47
BuildLatticeFromLengthsAngles, 41
BuildLatticeFromVectors, 41
ComputeOverlap, 42
diff, 42
either, 42
Elements, 47
EulerMatrix, 42
even_list, 42
format_gro_box, 43
format_gro_coord, 43
format_xyz_coord, 44
format_xyzgen_coord, 44
FrameVariableNames, 48
get_rotate_translate, 44
getElement, 45
grouper, 45
is_charmm_coord, 45
is_gro_box, 45
is_gro_coord, 46
isfloat, 46
isint, 46
main, 46
MetaVariableNames, 48
nodematch, 47
PeriodicTable, 48
pvec, 47
QuantumVariableNames, 48
radian, 48
Radii, 48
splitter, 49
forcebalance::molecule::Molecule
__add__, 436
__delitem__, 436
__getattr__, 437
__getitem__, 437
__iadd__, 438
__init__, 436
__iter__, 438
__len__, 439
__setattr__, 439
add_quantum, 439
add_virtual_site, 440
align, 440
align_by_moments, 440
align_center, 440
all_pairwise_rmsd, 440
append, 441
atom_select, 441
atom_stack, 441
boxes, 454
build_topology, 441
center_of_mass, 442
comms, 454
Data, 454
edit_qcrems, 442
fout, 454
Funnel, 454
load_frames, 442
measure_dihedrals, 442
molecules, 454
openmm_boxes, 443
openmm_positions, 443
positive_resid, 455
radius_of_gyration, 443
Read_Tab, 455
read_arc, 443
read_charmm, 444
read_com, 444
read_dcd, 445
read_gro, 445
read_mdcrd, 446
read_mol2, 446
read_pdb, 447
read_qcesp, 447
read_qcin, 447
read_qcout, 448
read_qdata, 448
read_xyz, 448
replace_peratom, 449
replace_peratom_conditional, 449
require, 449
require_boxes, 449
require_resid, 450
require_resname, 450
resid, 455
resname, 455
split, 450
topology, 455

write, 451
Write_Tab, 455
write_arc, 451
write_dcd, 451
write_gro, 452
write_mdcrd, 452
write_molproq, 452
write_pdb, 452
write_qcin, 453
write_qdata, 453
write_xyz, 454
forcebalance::moments
logger, 49
forcebalance::moments::Moments
__init__, 460
calc_moments, 467
denoms, 467
FF, 467
gct, 467
get, 460
get_G, 460
get_H, 461
get_X, 462
hct, 467
indicate, 463
link_from_tempdir, 463
mfnm, 468
na, 468
objective, 468
prepare_temp_directory, 463
PrintOptionDict, 468
printcool_table, 464
read_reference_data, 464
ref_eigvals, 468
ref_eigvecs, 468
ref_moments, 468
refresh_temp_directory, 464
rundir, 468
set_option, 465
sget, 465
stage, 466
submit_jobs, 466
tempdir, 468
unpack_moments, 466
verbose_options, 468
wq_complete, 467
xct, 469
forcebalance::nifty
allsplit, 52
bohrang, 66
col, 52
commadash, 53
concurrent_map, 53
CopyFile, 53
createWorkQueue, 53
encode, 54
eqcgmx, 66
flat, 54
floatornan, 54
fqcgmx, 66
get_least_squares, 55
getWQIds, 56
getWorkQueue, 55
GoInto, 56
invert_svd, 56
isdecimal, 56
isfloat, 57
isint, 57
kb, 66
Leave, 58
link_dir_contents, 58
LinkFile, 58
logger, 66
lp_dump, 58
lp_load, 58
MissingFileInspection, 59
multiopen, 59
orthogonalize, 59
pmat2d, 60
printcool, 60
printcool_dictionary, 61
pvec1d, 61
queue_up, 61
queue_up_src_dest, 62
remove_if_exists, 62
row, 62
segments, 63
specific_dct, 66
specific_lst, 66
statisticalInefficiency, 63
uncommadash, 64
warn_once, 64
warn_press_key, 64
which, 65
wq_wait, 65
wq_wait1, 65
XMLFILE, 67
forcebalance::nifty::Pickler_LP
__init__, 518
forcebalance::nifty::Unpickler_LP
__init__, 574
forcebalance::objective
Implemented_Targets, 67
Letters, 68
logger, 68
forcebalance::objective::Objective
__init__, 495
FF, 496

Full, 496
Indicate, 496
ObjDict, 497
ObjDict_Last, 497
Penalty, 497
PrintOptionDict, 497
set_option, 496
Target_Terms, 496
Targets, 497
verbose_options, 497
WTot, 497
forcebalance::objective::Penalty
__init__, 514
a, 516
b, 516
compute, 514
FF, 516
FUSE, 514
FUSE_BARRIER, 515
FUSE_L0, 515
fadd, 516
fmul, 516
HYP, 515
L2_norm, 516
Pen_Names, 517
Pen_Tab, 517
ptyp, 517
spacings, 517
forcebalance::openmmio
CopyAmoebaAngleParameters, 69
CopyAmoebaBondParameters, 69
CopyAmoebaInPlaneAngleParameters, 69
CopyAmoebaMultipoleParameters, 70
CopyAmoebaOutOfPlaneBendParameters, 70
CopyAmoebaVdwParameters, 70
CopyHarmonicAngleParameters, 70
CopyHarmonicBondParameters, 70
CopyNonbondedParameters, 71
CopyPeriodicTorsionParameters, 71
CopySystemParameters, 71
do_nothing, 72
get_dipole, 72
logger, 73
MTSVVVRIntegrator, 72
pdict, 73
ResetVirtualSites, 73
suffix_dict, 73
UpdateSimulationParameters, 73
forcebalance::openmmio::AbInitio_OpenMM
__init__, 170
AtomLists, 181
build_invdist, 170
compute_netforce_torque, 170
e_err, 181
e_err_pct, 181
e_ref, 181
emd0, 181
energy_force_driver_all, 170
energy_force_driver_all_external_, 171
energy_force_driver_all_internal_, 171
energy_force_transformer, 171
energy_force_transformer_all, 171
eqm, 181
esp_err, 181
espval, 181
espxyz, 181
f_err, 182
f_err_pct, 182
f_ref, 182
FF, 182
fitatoms, 182
force, 182
force_map, 182
fqm, 182
fref, 182
gct, 182
get, 172
get_G, 173
get_H, 174
get_X, 175
get_energy_force_, 172
get_resp_, 175
hct, 182
indicate, 176
invdists, 182
link_from_tempdir, 176
nesp, 183
new_vsites, 183
nf_err, 183
nf_err_pct, 183
nf_ref, 183
nftqm, 183
nnf, 183
nparticles, 183
ns, 183
ntq, 183
objective, 183
platform, 183
prepare_temp_directory, 176
PrintOptionDict, 184
printcool_table, 177
qfnm, 184
qmatoms, 184
qmboltz_wts, 184
read_reference_data, 177
read_topology, 178
refresh_temp_directory, 178
respterm, 184

rundir, 184
save_vmvals, 184
set_option, 179
sget, 179
simulation, 184
stage, 180
submit_jobs, 180
tempdir, 184
topology_flag, 185
tq_err, 185
tq_err_pct, 185
tq_ref, 185
traj, 185
trajfnm, 185
use_nft, 185
verbose_options, 185
w_energy, 185
w_force, 185
w_netforce, 185
w_resp, 185
w_torque, 185
whamboltz, 186
whamboltz_wts, 186
wq_complete, 180
xct, 186
xyz_omms, 186
forcebalance::openmmio::Interaction_OpenMM
__init__, 313
divisor, 320
e_err, 320
e_err_pct, 320
emm, 320
energy_driver_all, 313
eqm, 320
FF, 320
gct, 320
get, 313
get_G, 313
get_H, 314
get_X, 315
hct, 320
indicate, 316
interaction_driver_all, 316
label, 321
link_from_tempdir, 316
ns, 321
objective, 321
platform, 321
prefactor, 321
prepare_temp_directory, 316
PrintOptionDict, 321
printcool_table, 316
qfnm, 321
read_reference_data, 317
refresh_temp_directory, 317
rundir, 321
select1, 321
select2, 322
set_option, 318
sget, 318
simulations, 322
stage, 319
submit_jobs, 319
tempdir, 322
traj, 322
trajfnm, 322
verbose_options, 322
weight, 322
wq_complete, 319
xct, 322
forcebalance::openmmio::Liquid_OpenMM
__init__, 383
do_self_pol, 391
engine, 391
extra_output, 392
FF, 392
gas_fnm, 392
gct, 392
get, 383
get_G, 384
get_H, 385
get_X, 386
hct, 392
indicate, 387
Labels, 392
last_traj, 392
link_from_tempdir, 387
liquid_conf, 392
liquid_fnm, 392
liquid_traj, 392
MBarEnergy, 392
mpdb, 392
msim, 393
npt_simulation, 387
nptfiles, 393
nptpfx, 393
nptsfx, 393
objective_term, 387
PhasePoints, 393
platform, 393
polarization_correction, 388
prepare_temp_directory, 388
PrintOptionDict, 393
printcool_table, 388
read_data, 389
RefData, 393
refresh_temp_directory, 389
rundir, 393

SavedMVal, 393
SavedTraj, 393
set_option, 389
sget, 389
stage, 390
submit_jobs, 391
tempdir, 393
verbose_options, 394
w_alpha, 394
w_cp, 394
w_eps0, 394
w_hvap, 394
w_kappa, 394
w_rho, 394
wq_complete, 391
xct, 394
forcebalance::openmmio::OpenMM_Reader
__init__, 499
adict, 501
AtomTypes, 501
build_pid, 499, 500
feed, 500
itype, 501
ln, 501
molatom, 501
Molecules, 501
pdict, 501
Split, 500
suffix, 501
Whites, 500
forcebalance::optimizer
Counter, 74
GOODSTEP, 75
GoodStep, 74
ITERATION_NUMBER, 75
logger, 75
forcebalance::optimizer::Optimizer
__init__, 504
Anneal, 505
BFGS, 505
bhyp, 511
chk, 511
ConjugateGradient, 505
dx, 511
excision, 511
FDCheckG, 505
FDCheckH, 506
FF, 512
GeneticAlgorithm, 506
Grad, 512
Gradient, 506
H, 512
Hess, 512
Hessian, 506
MainOptimizer, 507
mvals0, 512
NewtonRaphson, 507
np, 512
Objective, 512
OptTab, 512
Penalty, 513
Powell, 508
PrintOptionDict, 513
readchk, 508
Run, 508
Scan_Values, 508
ScanMVals, 509
ScanPVals, 509
ScipyOptimizer, 509
set_option, 510
Simplex, 510
SinglePoint, 510
step, 511
Val, 513
verbose_options, 513
writechk, 511
forcebalance::output::CleanFileHandler
emit, 237
forcebalance::output::CleanStreamHandler
__init__, 238
emit, 239
forcebalance::output::ForceBalanceLogger
__init__, 266
addHandler, 266
defaultHandler, 266
removeHandler, 266
forcebalance::output::RawFileHandler
emit, 523
forcebalance::output::RawStreamHandler
__init__, 525
emit, 525
forcebalance::parser
bkwd, 79
gen_opts_defaults, 79
gen_opts_types, 79
logger, 79
mainsections, 80
ParsTab, 80
parse_inputs, 77
printsection, 78
read_internals, 78
read_mvals, 78
read_priors, 78
read_pvals, 79
subdict, 80
tgt_opts_defaults, 80
tgt_opts_types, 80
forcebalance::psi4io

logger, 81
forcebalance::psi4io::GBS_Reader
__init__, 272
adict, 274
amom, 274
AtomTypes, 274
basis_number, 274
build_pid, 272
contraction_number, 274
destroy, 274
element, 274
feed, 273
isdata, 274
itype, 274
last_amom, 274
ln, 274
molatom, 274
Molecules, 275
pdict, 275
Split, 273
suffix, 275
Whites, 273
forcebalance::psi4io::Grid_Reader
__init__, 281
adict, 282
AtomTypes, 282
build_pid, 281
element, 282
feed, 281
isdata, 282
itype, 283
ln, 283
molatom, 283
Molecules, 283
pdict, 283
point, 283
radii, 283
Split, 282
suffix, 283
Whites, 282
forcebalance::psi4io::RDVR3_Psi4
__init__, 528
callderivs, 535
driver, 528
elements, 535
FF, 535
factor, 535
gct, 535
get, 528
get_G, 529
get_H, 530
get_X, 531
gradd, 536
hct, 536
hdiagd, 536
indicate, 532
link_from_tempdir, 532
molecules, 536
objd, 536
objective, 536
objfiles, 536
objvals, 536
PrintOptionDict, 536
printcool_table, 532
refresh_temp_directory, 533
rundir, 536
set_option, 533
sget, 534
stage, 534
submit_jobs, 535
tdir, 536
tempdir, 536
verbose_options, 537
wq_complete, 535
xct, 537
forcebalance::psi4io::THCDF_Psi4
__init__, 560
call_derivatives, 567
D, 567
DATfnm, 567
DF_Energy, 567
driver, 560
Elements, 567
FF, 568
GBSfnm, 568
gct, 568
get, 560
get_G, 561
get_H, 561
get_X, 562
hct, 568
indicate, 563
link_from_tempdir, 563
MAQ, 568
MP2_Energy, 568
Molecules, 568
objective, 568
prepare_temp_directory, 563
PrintOptionDict, 568
printcool_table, 564
refresh_temp_directory, 564
rundir, 568
set_option, 565
sget, 565
stage, 566
submit_jobs, 566
tempdir, 569
throw_outs, 569

verbose_options, 569
wq_complete, 566
write_nested_destroy, 567
xct, 569
forcebalance::qchemio
logger, 83
ndtypes, 83
pdict, 83
QChem_Dielectric_Energy, 82
forcebalance::qchemio::QCIn_Reader
__init__, 520
adict, 521
atom, 521
AtomTypes, 521
build_pid, 520
cnum, 521
feed, 521
itype, 522
In, 522
molatom, 522
Molecules, 522
pdict, 522
sec, 522
shell, 522
snum, 522
Split, 521
suffix, 522
Whites, 521
forcebalance::target
logger, 83
forcebalance::target::RemoteTarget
__init__, 539
FF, 545
gct, 545
get, 539
get_G, 539
get_H, 540
get_X, 541
hct, 545
indicate, 542
link_from_tempdir, 542
PrintOptionDict, 545
printcool_table, 542
r_options, 545
r_tgt_opts, 545
refresh_temp_directory, 543
remote_indicate, 545
rundir, 546
set_option, 543
sget, 543
stage, 544
submit_jobs, 544
tempdir, 546
verbose_options, 546
wq_complete, 545
xct, 546
forcebalance::target::Target
__init__, 549
FF, 556
gct, 556
get, 550
get_G, 550
get_H, 551
get_X, 552
hct, 556
link_from_tempdir, 553
PrintOptionDict, 556
printcool_table, 553
refresh_temp_directory, 554
rundir, 556
set_option, 554
sget, 554
stage, 555
submit_jobs, 555
tempdir, 556
verbose_options, 556
wq_complete, 555
xct, 557
forcebalance::tinkerio
logger, 85
modify_key, 85
pdict, 85
write_key_with_prm, 85
forcebalance::tinkerio::AbInitio_TINKER
__init__, 191
all_at_once, 202
AtomLists, 202
build_invdist, 191
compute_netforce_torque, 191
e_err, 202
e_err_pct, 202
e_ref, 202
emd0, 202
energy_driver_all, 192
energy_force_driver, 192
energy_force_driver_all, 192
energy_force_transformer, 192
energy_force_transformer_all, 193
eqm, 202
esp_err, 202
espval, 202
espxyz, 203
f_err, 203
f_err_pct, 203
f_ref, 203
FF, 203
fitatoms, 203
force, 203

force_map, 203
fqm, 203
fref, 203
gct, 203
get, 193
get_G, 195
get_H, 195
get_X, 196
get_energy_force_, 193
get_resp_, 196
hct, 203
indicate, 197
invdists, 204
link_from_tempdir, 197
nesp, 204
new_vsites, 204
nf_err, 204
nf_err_pct, 204
nf_ref, 204
nftqm, 204
nnf, 204
nparticles, 204
ns, 204
ntq, 204
objective, 204
prepare_temp_directory, 197
PrintOptionDict, 205
printcool_table, 198
qfnm, 205
qmatoms, 205
qmboltz_wts, 205
read_reference_data, 198
read_topology, 199
refresh_temp_directory, 199
respterm, 205
rundir, 205
save_vmvals, 205
set_option, 200
sget, 200
stage, 201
submit_jobs, 201
tempdir, 205
topology_flag, 205
tq_err, 205
tq_err_pct, 206
tq_ref, 206
traj, 206
trajfnm, 206
use_nft, 206
verbose_options, 206
w_energy, 206
w_force, 206
w_netforce, 206
w_resp, 206
w_torque, 206
whamboltz, 206
whamboltz_wts, 206
wq_complete, 201
xct, 207
forcebalance::tinkerio::BindingEnergy_TINKER
__init__, 228
energy_part, 234
FF, 235
gct, 235
get, 229
get_G, 229
get_H, 229
get_X, 230
hct, 235
indicate, 231
inter_opts, 235
link_from_tempdir, 231
objective, 235
optprog, 235
prepare_temp_directory, 231
PrintDict, 235
PrintOptionDict, 235
printcool_table, 231
RMSDDict, 235
refresh_temp_directory, 232
rmsd_part, 235
rundir, 235
set_option, 232
sget, 232
stage, 233
submit_jobs, 234
system_driver, 234
tempdir, 235
verbose_options, 236
wq_complete, 234
xct, 236
forcebalance::tinkerio::Interaction_TINKER
__init__, 326
divisor, 333
e_err, 333
e_err_pct, 333
emm, 333
energy_driver_all, 326
eqm, 333
FF, 333
gct, 333
get, 326
get_G, 326
get_H, 327
get_X, 328
hct, 333
indicate, 329
interaction_driver_all, 329

label, 334
link_from_tempdir, 329
ns, 334
objective, 334
prefactor, 334
prepare_temp_directory, 329
PrintOptionDict, 334
printcool_table, 329
qfnm, 334
read_reference_data, 330
refresh_temp_directory, 330
rundir, 334
select1, 334
select2, 334
set_option, 331
sget, 331
stage, 332
submit_jobs, 332
tempdir, 334
traj, 335
trajfnm, 335
verbose_options, 335
weight, 335
wq_complete, 332
xct, 335
forcebalance::tinkerio::Liquid_TINKER
__init__, 398
do_self_pol, 406
DynDict, 406
DynDict_New, 406
extra_output, 406
FF, 406
gct, 406
get, 398
get_G, 399
get_H, 400
get_X, 401
hct, 407
indicate, 402
Labels, 407
last_traj, 407
link_from_tempdir, 402
MBarEnergy, 407
npt_simulation, 402
nptfiles, 407
nptpfx, 407
nptsfx, 407
objective_term, 402
PhasePoints, 407
polarization_correction, 403
prepare_temp_directory, 403
PrintOptionDict, 407
printcool_table, 403
read_data, 404
RefData, 407
refresh_temp_directory, 404
rundir, 407
SavedMVal, 407
SavedTraj, 408
set_option, 404
sget, 404
stage, 405
submit_jobs, 405
tempdir, 408
verbose_options, 408
w_alpha, 408
w_cp, 408
w_eps0, 408
w_hvap, 408
w_kappa, 408
w_rho, 408
wq_complete, 406
xct, 408
forcebalance::tinkerio::Moments_TINKER
__init__, 472
calc_moments, 479
denoms, 480
FF, 480
gct, 480
get, 472
get_G, 472
get_H, 473
get_X, 474
hct, 480
indicate, 475
link_from_tempdir, 475
mfnm, 480
moments_driver, 475
na, 480
objective, 480
prepare_temp_directory, 476
PrintOptionDict, 480
printcool_table, 476
read_reference_data, 477
ref_eigvals, 480
ref_eigvecs, 480
ref_moments, 480
refresh_temp_directory, 477
rundir, 480
set_option, 477
sget, 477
stage, 478
submit_jobs, 479
tempdir, 481
unpack_moments, 479
verbose_options, 481
wq_complete, 479
xct, 481

forcebalance::tinkerio::Tinker_Reader
__init__, 571
adict, 573
atom, 573
AtomTypes, 573
build_pid, 571
feed, 571
itype, 573
ln, 573
molatom, 573
Molecules, 573
pdict, 573
Split, 572
suffix, 573
Whites, 572
forcebalance::tinkerio::Vibration_TINKER
__init__, 587
prepare_temp_directory, 588
vibration_driver, 588
forcebalance::vibration
logger, 86
forcebalance::vibration::Vibration
__init__, 577
calc_eigvals, 585
FF, 585
gct, 585
get, 578
get_G, 578
get_H, 579
get_X, 580
hct, 585
indicate, 581
link_from_tempdir, 581
na, 585
objective, 585
prepare_temp_directory, 581
PrintOptionDict, 585
printcool_table, 581
read_reference_data, 582
ref_eigvals, 585
ref_eigvecs, 585
refresh_temp_directory, 582
rundir, 585
set_option, 583
sget, 583
stage, 584
submit_jobs, 584
tempdir, 586
verbose_options, 586
vfnm, 586
wq_complete, 584
xct, 586
forcefield.py, 593
format_gro_box
forcebalance::molecule, 43
format_gro_coord
forcebalance::molecule, 43
format_xyz_coord
forcebalance::molecule, 44
format_xyzgen_coord
forcebalance::molecule, 44
fout
forcebalance::molecule::Molecule, 454
fqcgmx
forcebalance::nifty, 66
fqm
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
FrameVariableNames
forcebalance::molecule, 48
frcmod_pdict
forcebalance::amberio, 16
fref
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 203
Full
forcebalance::objective::Objective, 496
Funnel
forcebalance::molecule::Molecule, 454

GBSfnm
forcebalance::psi4io::THCDF_Psi4, 568
GOODSTEP
forcebalance::optimizer, 75
gas_fnm
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::openmmio::Liquid_OpenMM, 392
gct
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::binding::BindingEnergy, 224
forcebalance::counterpoise::Counterpoise, 249
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::gmxio::Interaction_GMX, 307
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::gmxqpio::Monomer_QTPIE, 492
forcebalance::interaction::Interaction, 293
forcebalance::leastsq::LeastSquares, 350
forcebalance::liquid::Liquid, 362

forcebalance::moments::Moments, 467
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::openmmio::Liquid_OpenMM, 392
forcebalance::psi4io::RDVR3_Psi4, 535
forcebalance::psi4io::THCDF_Psi4, 568
forcebalance::target::RemoteTarget, 545
forcebalance::target::Target, 556
forcebalance::tinkerio::AbInitio_TINKER, 203
forcebalance::tinkerio::BindingEnergy_TINKER, 235
forcebalance::tinkerio::Interaction_TINKER, 333
forcebalance::tinkerio::Liquid_TINKER, 406
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585

gen_opts_defaults
forcebalance::parser, 79

gen_opts_types
forcebalance::parser, 79

generate_vsite_positions
forcebalance::gmxio::AbInitio_GMX, 132

GeneticAlgorithm
forcebalance::optimizer::Optimizer, 506

get
forcebalance::abinitio::AbInitio, 92
forcebalance::abinitio_internal::AbInitio_Internal, 152
forcebalance::amberio::AbInitio_AMBER, 112
forcebalance::binding::BindingEnergy, 218
forcebalance::counterpoise::Counterpoise, 242
forcebalance::gmxio::AbInitio_GMX, 132
forcebalance::gmxio::Interaction_GMX, 299
forcebalance::gmxio::Liquid_GMX, 368
forcebalance::gmxqpio::Monomer_QTPIE, 485
forcebalance::interaction::Interaction, 286
forcebalance::leastsq::LeastSquares, 344
forcebalance::liquid::Liquid, 354
forcebalance::moments::Moments, 460
forcebalance::openmmio::AbInitio_OpenMM, 172
forcebalance::openmmio::Interaction_OpenMM, 313
forcebalance::openmmio::Liquid_OpenMM, 383
forcebalance::psi4io::RDVR3_Psi4, 528
forcebalance::psi4io::THCDF_Psi4, 560
forcebalance::target::RemoteTarget, 539
forcebalance::target::Target, 550
forcebalance::tinkerio::AbInitio_TINKER, 193
forcebalance::tinkerio::BindingEnergy_TINKER, 229
forcebalance::tinkerio::Interaction_TINKER, 326
forcebalance::tinkerio::Liquid_TINKER, 398
forcebalance::tinkerio::Moments_TINKER, 472
forcebalance::vibration::Vibration, 578

get_G
forcebalance::abinitio::AbInitio, 94
forcebalance::abinitio_internal::AbInitio_Internal, 154
forcebalance::amberio::AbInitio_AMBER, 113
forcebalance::binding::BindingEnergy, 218
forcebalance::counterpoise::Counterpoise, 242
forcebalance::gmxio::AbInitio_GMX, 134
forcebalance::gmxio::Interaction_GMX, 299
forcebalance::gmxio::Liquid_GMX, 369
forcebalance::gmxqpio::Monomer_QTPIE, 485
forcebalance::interaction::Interaction, 286
forcebalance::leastsq::LeastSquares, 344
forcebalance::liquid::Liquid, 355
forcebalance::moments::Moments, 460
forcebalance::openmmio::AbInitio_OpenMM, 173
forcebalance::openmmio::Interaction_OpenMM, 313
forcebalance::openmmio::Liquid_OpenMM, 384
forcebalance::psi4io::RDVR3_Psi4, 529
forcebalance::psi4io::THCDF_Psi4, 561
forcebalance::target::RemoteTarget, 539
forcebalance::target::Target, 550
forcebalance::tinkerio::AbInitio_TINKER, 195
forcebalance::tinkerio::BindingEnergy_TINKER, 229
forcebalance::tinkerio::Interaction_TINKER, 326
forcebalance::tinkerio::Liquid_TINKER, 399
forcebalance::tinkerio::Moments_TINKER, 472
forcebalance::vibration::Vibration, 578

get_H
forcebalance::abinitio::AbInitio, 94
forcebalance::abinitio_internal::AbInitio_Internal, 155
forcebalance::amberio::AbInitio_AMBER, 114
forcebalance::binding::BindingEnergy, 219
forcebalance::counterpoise::Counterpoise, 243
forcebalance::gmxio::AbInitio_GMX, 135
forcebalance::gmxio::Interaction_GMX, 300
forcebalance::gmxio::Liquid_GMX, 370
forcebalance::gmxqpio::Monomer_QTPIE, 486
forcebalance::interaction::Interaction, 287
forcebalance::leastsq::LeastSquares, 345
forcebalance::liquid::Liquid, 356
forcebalance::moments::Moments, 461
forcebalance::openmmio::AbInitio_OpenMM, 174
forcebalance::openmmio::Interaction_OpenMM, 314
forcebalance::openmmio::Liquid_OpenMM, 385
forcebalance::psi4io::RDVR3_Psi4, 530
forcebalance::psi4io::THCDF_Psi4, 561
forcebalance::target::RemoteTarget, 540
forcebalance::target::Target, 551
forcebalance::tinkerio::AbInitio_TINKER, 195
forcebalance::tinkerio::BindingEnergy_TINKER, 229
forcebalance::tinkerio::Interaction_TINKER, 327
forcebalance::tinkerio::Liquid_TINKER, 400
forcebalance::tinkerio::Moments_TINKER, 473
forcebalance::vibration::Vibration, 579

get_X
forcebalance::abinitio::AbInitio, 95
forcebalance::abinitio_internal::AbInitio_Internal, 155
forcebalance::amberio::AbInitio_AMBER, 115
forcebalance::binding::BindingEnergy, 220

forcebalance::counterpoise::Counterpoise, 244
forcebalance::gmxio::AbInitio_GMX, 135
forcebalance::gmxio::Interaction_GMX, 301
forcebalance::gmxio::Liquid_GMX, 371
forcebalance::gmxqpio::Monomer_QTPIE, 487
forcebalance::interaction::Interaction, 288
forcebalance::leastsq::LeastSquares, 346
forcebalance::liquid::Liquid, 357
forcebalance::moments::Moments, 462
forcebalance::openmmio::AbInitio_OpenMM, 175
forcebalance::openmmio::Interaction_OpenMM, 315
forcebalance::openmmio::Liquid_OpenMM, 386
forcebalance::psi4io::RDVR3_Psi4, 531
forcebalance::psi4io::THCDF_Psi4, 562
forcebalance::target::RemoteTarget, 541
forcebalance::target::Target, 552
forcebalance::tinkerio::AbInitio_TINKER, 196
forcebalance::tinkerio::BindingEnergy_TINKER, 230
forcebalance::tinkerio::Interaction_TINKER, 328
forcebalance::tinkerio::Liquid_TINKER, 401
forcebalance::tinkerio::Moments_TINKER, 474
forcebalance::vibration::Vibration, 580

get_atom
forcebalance::Mol2::mol2, 410

get_bonded_atoms
forcebalance::Mol2::mol2, 411

get_dipole
forcebalance::openmmio, 72

get_energy_force_
forcebalance::abinitio::AbInitio, 92
forcebalance::abinitio_internal::AbInitio_Internal, 153
forcebalance::amberio::AbInitio_AMBER, 112
forcebalance::gmxio::AbInitio_GMX, 133
forcebalance::openmmio::AbInitio_OpenMM, 172
forcebalance::tinkerio::AbInitio_TINKER, 193

get_least_squares
forcebalance::nifty, 55

get_monomer_properties
forcebalance::gmxqpio, 35

get_resp_
forcebalance::abinitio::AbInitio, 95
forcebalance::abinitio_internal::AbInitio_Internal, 155
forcebalance::amberio::AbInitio_AMBER, 115
forcebalance::gmxio::AbInitio_GMX, 135
forcebalance::openmmio::AbInitio_OpenMM, 175
forcebalance::tinkerio::AbInitio_TINKER, 196

get_rotate_translate
forcebalance::molecule, 44

getElement
forcebalance::molecule, 45

getWQIds
forcebalance::nifty, 56

getWorkQueue
forcebalance::nifty, 55

gmxio.py, 594

gmxqpio.py, 595

GoInto
forcebalance::nifty, 56

GoodStep
forcebalance::optimizer, 74

Grad
forcebalance::optimizer::Optimizer, 512

gradd
forcebalance::psi4io::RDVR3_Psi4, 536

Gradient
forcebalance::optimizer::Optimizer, 506

grouper
forcebalance::molecule, 45

H
forcebalance::optimizer::Optimizer, 512

HYP
forcebalance::objective::Penalty, 515

hct
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 122
forcebalance::binding::BindingEnergy, 224
forcebalance::counterpoise::Counterpoise, 249
forcebalance::gmxio::AbInitio_GMX, 142
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::gmxqpio::Monomer_QTPIE, 492
forcebalance::interaction::Interaction, 293
forcebalance::leastsq::LeastSquares, 350
forcebalance::liquid::Liquid, 362
forcebalance::moments::Moments, 467
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::openmmio::Interaction_OpenMM, 320
forcebalance::openmmio::Liquid_OpenMM, 392
forcebalance::psi4io::RDVR3_Psi4, 536
forcebalance::psi4io::THCDF_Psi4, 568
forcebalance::target::RemoteTarget, 545
forcebalance::target::Target, 556
forcebalance::tinkerio::AbInitio_TINKER, 203
forcebalance::tinkerio::BindingEnergy_TINKER, 235
forcebalance::tinkerio::Interaction_TINKER, 333
forcebalance::tinkerio::Liquid_TINKER, 407
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585

hdiagd
forcebalance::psi4io::RDVR3_Psi4, 536

Hess
forcebalance::optimizer::Optimizer, 512

Hessian
forcebalance::optimizer::Optimizer, 506

ITERATION_NUMBER

forcebalance::optimizer, 75
Implemented_Targets
forcebalance::objective, 67
in_fd
forcebalance::finite_difference, 27
Indicate
forcebalance::objective::Objective, 496
indicate
forcebalance::abinitio::AbInitio, 96
forcebalance::abinitio_internal::AbInitio_Internal, 156
forcebalance::amberio::AbInitio_AMBER, 116
forcebalance::binding::BindingEnergy, 221
forcebalance::gmxio::AbInitio_GMX, 136
forcebalance::gmxio::Interaction_GMX, 302
forcebalance::gmxio::Liquid_GMX, 372
forcebalance::gmxqpio::Monomer_QTPIE, 488
forcebalance::interaction::Interaction, 289
forcebalance::leastsq::LeastSquares, 347
forcebalance::liquid::Liquid, 358
forcebalance::moments::Moments, 463
forcebalance::openmmio::AbInitio_OpenMM, 176
forcebalance::openmmio::Interaction_OpenMM, 316
forcebalance::openmmio::Liquid_OpenMM, 387
forcebalance::psi4io::RDVR3_Psi4, 532
forcebalance::psi4io::THCDF_Psi4, 563
forcebalance::target::RemoteTarget, 542
forcebalance::tinkerio::AbInitio_TINKER, 197
forcebalance::tinkerio::BindingEnergy_TINKER, 231
forcebalance::tinkerio::Interaction_TINKER, 329
forcebalance::tinkerio::Liquid_TINKER, 402
forcebalance::tinkerio::Moments_TINKER, 475
forcebalance::vibration::Vibration, 581
inter_opts
forcebalance::binding::BindingEnergy, 224
forcebalance::tinkerio::BindingEnergy_TINKER, 235
interaction.py, 595
interaction_driver
forcebalance::gmxio::Interaction_GMX, 302
interaction_driver_all
forcebalance::gmxio::Interaction_GMX, 302
forcebalance::openmmio::Interaction_OpenMM, 316
forcebalance::tinkerio::Interaction_TINKER, 329
invdists
forcebalance::abinitio::AbInitio, 102
forcebalance::abinitio_internal::AbInitio_Internal, 162
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 182
forcebalance::tinkerio::AbInitio_TINKER, 204
invert_svd
forcebalance::nifty, 56
is_charmm_coord
forcebalance::molecule, 45
is_gro_box
forcebalance::molecule, 45
is_gro_coord
forcebalance::molecule, 46
is_mol2_atom
forcebalance::amberio, 15
isdata
forcebalance::psi4io::GBS_Reader, 274
forcebalance::psi4io::Grid_Reader, 282
isdecimal
forcebalance::nifty, 56
isfloat
forcebalance::molecule, 46
forcebalance::nifty, 57
isint
forcebalance::molecule, 46
forcebalance::nifty, 57
itype
forcebalance::amberio::FrcMod_Reader, 269
forcebalance::amberio::Mol2_Reader, 427
forcebalance::BaseReader, 215
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 340
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 274
forcebalance::psi4io::Grid_Reader, 283
forcebalance::qchemio::QCIn_Reader, 522
forcebalance::tinkerio::Tinker_Reader, 573

kb
forcebalance::nifty, 66

L
forcebalance::chemistry, 19
L2_norm
forcebalance::objective::Penalty, 516
LAST_MVALS
forcebalance::leastsq, 36
label
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::interaction::Interaction, 294
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::tinkerio::Interaction_TINKER, 334
Labels
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 392
forcebalance::tinkerio::Liquid_TINKER, 407
last_amom
forcebalance::psi4io::GBS_Reader, 274
last_traj
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 392

forcebalance::tinkerio::Liquid_TINKER, 407
LastMvals
forcebalance::leastsq, 36
leastsq.py, 596
Leave
forcebalance::nifty, 58
Letters
forcebalance::objective, 68
line
forcebalance::chemistry, 19
linedestroy_save
forcebalance::forcefield::FF, 262
linedestroy_this
forcebalance::forcefield::FF, 262
link_dir_contents
forcebalance::nifty, 58
link_from_tempdir
forcebalance::abinitio::AbInitio, 96
forcebalance::abinitio_internal::AbInitio_Internal, 156
forcebalance::amberio::AbInitio_AMBER, 116
forcebalance::binding::BindingEnergy, 221
forcebalance::counterpoise::Counterpoise, 245
forcebalance::gmxio::AbInitio_GMX, 136
forcebalance::gmxio::Interaction_GMX, 303
forcebalance::gmxio::Liquid_GMX, 372
forcebalance::gmxqpio::Monomer_QTPIE, 488
forcebalance::interaction::Interaction, 289
forcebalance::leastsq::LeastSquares, 347
forcebalance::liquid::Liquid, 358
forcebalance::moments::Moments, 463
forcebalance::openmmio::AbInitio_OpenMM, 176
forcebalance::openmmio::Interaction_OpenMM, 316
forcebalance::openmmio::Liquid_OpenMM, 387
forcebalance::psi4io::RDVR3_Psi4, 532
forcebalance::psi4io::THCDF_Psi4, 563
forcebalance::target::RemoteTarget, 542
forcebalance::target::Target, 553
forcebalance::tinkerio::AbInitio_TINKER, 197
forcebalance::tinkerio::BindingEnergy_TINKER, 231
forcebalance::tinkerio::Interaction_TINKER, 329
forcebalance::tinkerio::Liquid_TINKER, 402
forcebalance::tinkerio::Moments_TINKER, 475
forcebalance::vibration::Vibration, 581
LinkFile
forcebalance::nifty, 58
liquid.py, 596
liquid_conf
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::openmmio::Liquid_OpenMM, 392
liquid_fnm
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::openmmio::Liquid_OpenMM, 392
liquid_traj
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::openmmio::Liquid_OpenMM, 392
list_map
forcebalance::forcefield::FF, 259
ln
forcebalance::amberio::FrcMod_Reader, 270
forcebalance::amberio::Mol2_Reader, 427
forcebalance::BaseReader, 215
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 340
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 274
forcebalance::psi4io::Grid_Reader, 283
forcebalance::qchemio::QCIn_Reader, 522
forcebalance::tinkerio::Tinker_Reader, 573
load_cp
forcebalance::counterpoise::Counterpoise, 245
load_frames
forcebalance::molecule::Molecule, 442
loadxyz
forcebalance::counterpoise::Counterpoise, 245
logger
forcebalance::abinitio, 14
forcebalance::amberio, 16
forcebalance::binding, 17
forcebalance::counterpoise, 22
forcebalance::finite_difference, 27
forcebalance::forcefield, 30
forcebalance::gmxio, 34
forcebalance::gmxqpio, 35
forcebalance::interaction, 36
forcebalance::leastsq, 36
forcebalance::liquid, 37
forcebalance::moments, 49
forcebalance::nifty, 66
forcebalance::objective, 68
forcebalance::openmmio, 73
forcebalance::optimizer, 75
forcebalance::parser, 79
forcebalance::psi4io, 81
forcebalance::qchemio, 83
forcebalance::target, 83
forcebalance::tinkerio, 85
forcebalance::vibration, 86
LookupByMass
forcebalance::chemistry, 18
lp_dump
forcebalance::nifty, 58
lp_load
forcebalance::nifty, 58

MAQ
forcebalance::leastsq::LeastSquares, 351
forcebalance::psi4io::THCDF_Psi4, 568

MBarEnergy
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 392
forcebalance::tinkerio::Liquid_TINKER, 407
MP2_Energy
forcebalance::psi4io::THCDF_Psi4, 568
MTSVVVRIntegrator
forcebalance::openmmio, 72
main
forcebalance::molecule, 46
MainOptimizer
forcebalance::optimizer::Optimizer, 507
mainsections
forcebalance::parser, 80
make
forcebalance::forcefield::FF, 259
make_redirect
forcebalance::forcefield::FF, 260
map
forcebalance::forcefield::FF, 262
measure_dihedrals
forcebalance::molecule::Molecule, 442
MetaVariableNames
forcebalance::molecule, 48
mfnm
forcebalance::moments::Moments, 468
forcebalance::tinkerio::Moments_TINKER, 480
MissingFileInspection
forcebalance::nifty, 59
mktransmat
forcebalance::forcefield::FF, 260
modify_key
forcebalance::tinkerio, 85
mol
forcebalance::amberio::Mol2_Reader, 427
forcebalance::gmxio::ITP_Reader, 340
Mol2.py, 597
mol2_pdict
forcebalance::amberio, 16
forcebalance::mol2io, 39
mol2io.py, 597
mol_name
forcebalance::Mol2::mol2, 415
mol_type
forcebalance::Mol2::mol2, 415
molatom
forcebalance::amberio::FrcMod_Reader, 270
forcebalance::amberio::Mol2_Reader, 427
forcebalance::BaseReader, 215
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 340
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 274
forcebalance::psi4io::Grid_Reader, 283
forcebalance::qchemio::QCIn_Reader, 522
forcebalance::tinkerio::Tinker_Reader, 573
molecule.py, 597
Molecules
forcebalance::amberio::FrcMod_Reader, 270
forcebalance::amberio::Mol2_Reader, 427
forcebalance::BaseReader, 215
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 340
forcebalance::mol2io::Mol2_Reader, 431
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 275
forcebalance::psi4io::Grid_Reader, 283
forcebalance::psi4io::THCDF_Psi4, 568
forcebalance::qchemio::QCIn_Reader, 522
forcebalance::tinkerio::Tinker_Reader, 573
molecules
forcebalance::molecule::Molecule, 454
forcebalance::psi4io::RDVR3_Psi4, 536
moments.py, 599
moments_driver
forcebalance::tinkerio::Moments_TINKER, 475
mpdb
forcebalance::openmmio::Liquid_OpenMM, 392
msim
forcebalance::openmmio::Liquid_OpenMM, 393
multiopen
forcebalance::nifty, 59
mvals0
forcebalance::optimizer::Optimizer, 512

na
forcebalance::counterpoise::Counterpoise, 249
forcebalance::moments::Moments, 468
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585
nbtype
forcebalance::gmxio::ITP_Reader, 340
ndtypes
forcebalance::custom_io, 23
forcebalance::qchemio, 83
nesp
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
new_vsites
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123

forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
NewtonRaphson
forcebalance::optimizer::Optimizer, 507
nf_err
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
nf_err_pct
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
nf_ref
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
nftqm
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
nftypes
forcebalance::gmxio, 34
nifty.py, 599
nnf
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
nodematch
forcebalance::molecule, 47
np
forcebalance::forcefield::FF, 262
forcebalance::optimizer::Optimizer, 512
nparticles
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
npt_simulation
forcebalance::gmxio::Liquid_GMX, 372
forcebalance::liquid::Liquid, 358
forcebalance::openmmio::Liquid_OpenMM, 387
forcebalance::tinkerio::Liquid_TINKER, 402
nptfiles
forcebalance::gmxio::Liquid_GMX, 377
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::tinkerio::Liquid_TINKER, 407
nptpfx
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::tinkerio::Liquid_TINKER, 407
nptsfx
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::tinkerio::Liquid_TINKER, 407
ns
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::counterpoise::Counterpoise, 249
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::interaction::Interaction, 294
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::tinkerio::AbInitio_TINKER, 204
forcebalance::tinkerio::Interaction_TINKER, 334
ntq
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::tinkerio::AbInitio_TINKER, 204
num_atoms
forcebalance::Mol2::mol2, 415
num_bonds
forcebalance::Mol2::mol2, 415
num_compounds
forcebalance::Mol2::mol2_set, 432
num_feat
forcebalance::Mol2::mol2, 415
num_sets
forcebalance::Mol2::mol2, 415
num_subst
forcebalance::Mol2::mol2, 415

ObjDict

forcebalance::objective::Objective, 497
ObjDict_Last
forcebalance::objective::Objective, 497
objd
forcebalance::psi4io::RDVR3_Psi4, 536
Objective
forcebalance::optimizer::Optimizer, 512
objective
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 123
forcebalance::binding::BindingEnergy, 225
forcebalance::gmxio::AbInitio_GMX, 143
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::gmxqpio::Monomer_QTPIE, 492
forcebalance::interaction::Interaction, 294
forcebalance::leastsq::LeastSquares, 351
forcebalance::moments::Moments, 468
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::psi4io::RDVR3_Psi4, 536
forcebalance::psi4io::THCDF_Psi4, 568
forcebalance::tinkerio::AbInitio_TINKER, 204
forcebalance::tinkerio::BindingEnergy_TINKER, 235
forcebalance::tinkerio::Interaction_TINKER, 334
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585
objective.py, 601
objective_term
forcebalance::gmxio::Liquid_GMX, 372
forcebalance::liquid::Liquid, 358
forcebalance::openmmio::Liquid_OpenMM, 387
forcebalance::tinkerio::Liquid_TINKER, 402
objfiles
forcebalance::psi4io::RDVR3_Psi4, 536
objvals
forcebalance::psi4io::RDVR3_Psi4, 536
openmm_boxes
forcebalance::molecule::Molecule, 443
openmm_positions
forcebalance::molecule::Molecule, 443
openmmio.py, 602
openmmxml
forcebalance::forcefield::FF, 262
OptTab
forcebalance::optimizer::Optimizer, 512
optimizer.py, 603
optprog
forcebalance::tinkerio::BindingEnergy_TINKER, 235
origin_atom_id
forcebalance::Mol2::mol2_bond, 423
orthogonalize
forcebalance::nifty, 59
out
forcebalance::Mol2::mol2, 411
output.py, 603

PT.py, 605
parmdestroy_save
forcebalance::forcefield::FF, 262
parmdestroy_this
forcebalance::forcefield::FF, 263
ParsTab
forcebalance::parser, 80
parse
forcebalance::Mol2::mol2, 411
forcebalance::Mol2::mol2_atom, 417
forcebalance::Mol2::mol2_bond, 421
forcebalance::Mol2::mol2_set, 432
parse_atomtype_line
forcebalance::gmxio, 32
parse_inputs
forcebalance::parser, 77
parse_interactions
forcebalance::binding, 17
parser.py, 604
patoms
forcebalance::forcefield::FF, 263
pdict
forcebalance::amberio::FrcMod_Reader, 270
forcebalance::amberio::Mol2_Reader, 427
forcebalance::BaseReader, 215
forcebalance::custom_io, 23
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio, 34
forcebalance::gmxio::ITP_Reader, 340
forcebalance::mol2io::Mol2_Reader, 431
forcebalance::openmmio, 73
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 275
forcebalance::psi4io::Grid_Reader, 283
forcebalance::qchemio, 83
forcebalance::qchemio::QCIn_Reader, 522
forcebalance::tinkerio, 85
forcebalance::tinkerio::Tinker_Reader, 573
Pen_Names
forcebalance::objective::Penalty, 517
Pen_Tab
forcebalance::objective::Penalty, 517
Penalty
forcebalance::objective::Objective, 497
forcebalance::optimizer::Optimizer, 513
PeriodicTable
forcebalance::chemistry, 19
forcebalance::molecule, 48
forcebalance::PT, 81
pfields
forcebalance::forcefield::FF, 263

pftypes
forcebalance::gmxio, 34
PhasePoints
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::tinkerio::Liquid_TINKER, 407
platform
forcebalance::openmmio::AbInitio_OpenMM, 183
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::openmmio::Liquid_OpenMM, 393
plist
forcebalance::forcefield::FF, 263
pmat2d
forcebalance::nifty, 60
point
forcebalance::psi4io::Grid_Reader, 283
polarization_correction
forcebalance::gmxio::Liquid_GMX, 373
forcebalance::liquid::Liquid, 359
forcebalance::openmmio::Liquid_OpenMM, 388
forcebalance::tinkerio::Liquid_TINKER, 403
positive_resid
forcebalance::molecule::Molecule, 455
Powell
forcebalance::optimizer::Optimizer, 508
prefactor
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::interaction::Interaction, 294
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::tinkerio::Interaction_TINKER, 334
prepare_temp_directory
forcebalance::abinitio::AbInitio, 96
forcebalance::abinitio_internal::AbInitio_Internal, 156
forcebalance::amberio::AbInitio_AMBER, 116
forcebalance::gmxio::AbInitio_GMX, 136
forcebalance::gmxio::Interaction_GMX, 303
forcebalance::gmxio::Liquid_GMX, 373
forcebalance::gmxqpio::Monomer_QTPIE, 488
forcebalance::interaction::Interaction, 289
forcebalance::moments::Moments, 463
forcebalance::openmmio::AbInitio_OpenMM, 176
forcebalance::openmmio::Interaction_OpenMM, 316
forcebalance::openmmio::Liquid_OpenMM, 388
forcebalance::psi4io::THCDF_Psi4, 563
forcebalance::tinkerio::AbInitio_TINKER, 197
forcebalance::tinkerio::BindingEnergy_TINKER, 231
forcebalance::tinkerio::Interaction_TINKER, 329
forcebalance::tinkerio::Liquid_TINKER, 403
forcebalance::tinkerio::Moments_TINKER, 476
forcebalance::tinkerio::Vibration_TINKER, 588
forcebalance::vibration::Vibration, 581
print_map
forcebalance::forcefield::FF, 260
PrintDict
forcebalance::binding::BindingEnergy, 225
forcebalance::tinkerio::BindingEnergy_TINKER, 235
PrintOptionDict
forcebalance::abinitio::AbInitio, 103
forcebalance::abinitio_internal::AbInitio_Internal, 163
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::BaseClass, 211
forcebalance::binding::BindingEnergy, 225
forcebalance::counterpoise::Counterpoise, 249
forcebalance::forcefield::FF, 263
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::gmxqpio::Monomer_QTPIE, 492
forcebalance::interaction::Interaction, 294
forcebalance::leastsq::LeastSquares, 351
forcebalance::liquid::Liquid, 363
forcebalance::moments::Moments, 468
forcebalance::objective::Objective, 497
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::optimizer::Optimizer, 513
forcebalance::psi4io::RDVR3_Psi4, 536
forcebalance::psi4io::THCDF_Psi4, 568
forcebalance::target::RemoteTarget, 545
forcebalance::target::Target, 556
forcebalance::tinkerio::AbInitio_TINKER, 205
forcebalance::tinkerio::BindingEnergy_TINKER, 235
forcebalance::tinkerio::Interaction_TINKER, 334
forcebalance::tinkerio::Liquid_TINKER, 407
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585
printcool
forcebalance::nifty, 60
printcool_dictionary
forcebalance::nifty, 61
printcool_table
forcebalance::abinitio::AbInitio, 96
forcebalance::abinitio_internal::AbInitio_Internal, 156
forcebalance::amberio::AbInitio_AMBER, 117
forcebalance::binding::BindingEnergy, 221
forcebalance::counterpoise::Counterpoise, 245
forcebalance::gmxio::AbInitio_GMX, 137
forcebalance::gmxio::Interaction_GMX, 303
forcebalance::gmxio::Liquid_GMX, 373
forcebalance::gmxqpio::Monomer_QTPIE, 489
forcebalance::interaction::Interaction, 289
forcebalance::leastsq::LeastSquares, 347
forcebalance::liquid::Liquid, 359
forcebalance::moments::Moments, 464
forcebalance::openmmio::AbInitio_OpenMM, 177
forcebalance::openmmio::Interaction_OpenMM, 316

forcebalance::openmmio::Liquid_OpenMM, 388
forcebalance::psi4io::RDVR3_Psi4, 532
forcebalance::psi4io::THCDF_Psi4, 564
forcebalance::target::RemoteTarget, 542
forcebalance::target::Target, 553
forcebalance::tinkerio::AbInitio_TINKER, 198
forcebalance::tinkerio::BindingEnergy_TINKER, 231
forcebalance::tinkerio::Interaction_TINKER, 329
forcebalance::tinkerio::Liquid_TINKER, 403
forcebalance::tinkerio::Moments_TINKER, 476
forcebalance::vibration::Vibration, 581
printsection
forcebalance::parser, 78
psi4io.py, 605
ptyp
forcebalance::objective::Penalty, 517
pvals0
forcebalance::forcefield::FF, 263
pvec
forcebalance::molecule, 47
pvec1d
forcebalance::nifty, 61

QChem_Dielectric_Energy
forcebalance::qchemio, 82
qchemio.py, 605
qfnm
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::interaction::Interaction, 294
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::tinkerio::AbInitio_TINKER, 205
forcebalance::tinkerio::Interaction_TINKER, 334
qid
forcebalance::forcefield::FF, 263
qid2
forcebalance::forcefield::FF, 263
qmap
forcebalance::forcefield::FF, 263
qmatoms
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::tinkerio::AbInitio_TINKER, 205
qmboltz_wts
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::tinkerio::AbInitio_TINKER, 205
QuantumVariableNames
forcebalance::molecule, 48
queue_up
forcebalance::nifty, 61
queue_up_src_dest
forcebalance::nifty, 62

r_options
forcebalance::target::RemoteTarget, 545
r_tgt_opts
forcebalance::target::RemoteTarget, 545
RMSDDict
forcebalance::binding::BindingEnergy, 225
forcebalance::tinkerio::BindingEnergy_TINKER, 235
radian
forcebalance::molecule, 48
Radii
forcebalance::chemistry, 19
forcebalance::molecule, 48
radii
forcebalance::psi4io::Grid_Reader, 283
radius_of_gyration
forcebalance::molecule::Molecule, 443
Read_Tab
forcebalance::molecule::Molecule, 455
read_arc
forcebalance::molecule::Molecule, 443
read_charmm
forcebalance::molecule::Molecule, 444
read_com
forcebalance::molecule::Molecule, 444
read_data
forcebalance::gmxio::Liquid_GMX, 374
forcebalance::liquid::Liquid, 359
forcebalance::openmmio::Liquid_OpenMM, 389
forcebalance::tinkerio::Liquid_TINKER, 404
read_dcd
forcebalance::molecule::Molecule, 445
read_gro
forcebalance::molecule::Molecule, 445
read_internals
forcebalance::parser, 78
read_mdcrd
forcebalance::molecule::Molecule, 446
read_mol2
forcebalance::molecule::Molecule, 446
read_mvals
forcebalance::parser, 78
read_pdb
forcebalance::molecule::Molecule, 447
read_priors

forcebalance::parser, 78
read_pvals
forcebalance::parser, 79
read_qcesp
forcebalance::molecule::Molecule, 447
read_qcin
forcebalance::molecule::Molecule, 447
read_qcout
forcebalance::molecule::Molecule, 448
read_qdata
forcebalance::molecule::Molecule, 448
read_reference_data
forcebalance::abinitio::AbInitio, 97
forcebalance::abinitio_internal::AbInitio_Internal, 157
forcebalance::amberio::AbInitio_AMBER, 117
forcebalance::gmxio::AbInitio_GMX, 137
forcebalance::gmxio::Interaction_GMX, 304
forcebalance::interaction::Interaction, 290
forcebalance::moments::Moments, 464
forcebalance::openmmio::AbInitio_OpenMM, 177
forcebalance::openmmio::Interaction_OpenMM, 317
forcebalance::tinkerio::AbInitio_TINKER, 198
forcebalance::tinkerio::Interaction_TINKER, 330
forcebalance::tinkerio::Moments_TINKER, 477
forcebalance::vibration::Vibration, 582
read_topology
forcebalance::abinitio::AbInitio, 98
forcebalance::abinitio_internal::AbInitio_Internal, 158
forcebalance::amberio::AbInitio_AMBER, 118
forcebalance::gmxio::AbInitio_GMX, 138
forcebalance::openmmio::AbInitio_OpenMM, 178
forcebalance::tinkerio::AbInitio_TINKER, 199
read_xyz
forcebalance::molecule::Molecule, 448
readchk
forcebalance::optimizer::Optimizer, 508
Readers
forcebalance::forcefield::FF, 263
redirect
forcebalance::forcefield::FF, 263
ref_eigvals
forcebalance::moments::Moments, 468
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585
ref_eigvecs
forcebalance::moments::Moments, 468
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585
ref_moments
forcebalance::gmxqpio::Monomer_QTPIE, 493
forcebalance::moments::Moments, 468
forcebalance::tinkerio::Moments_TINKER, 480
RefData
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::tinkerio::Liquid_TINKER, 407
refresh_temp_directory
forcebalance::abinitio::AbInitio, 98
forcebalance::abinitio_internal::AbInitio_Internal, 158
forcebalance::amberio::AbInitio_AMBER, 118
forcebalance::binding::BindingEnergy, 222
forcebalance::counterpoise::Counterpoise, 246
forcebalance::gmxio::AbInitio_GMX, 139
forcebalance::gmxio::Interaction_GMX, 304
forcebalance::gmxio::Liquid_GMX, 374
forcebalance::gmxqpio::Monomer_QTPIE, 489
forcebalance::interaction::Interaction, 290
forcebalance::leastsq::LeastSquares, 348
forcebalance::liquid::Liquid, 359
forcebalance::moments::Moments, 464
forcebalance::openmmio::AbInitio_OpenMM, 178
forcebalance::openmmio::Interaction_OpenMM, 317
forcebalance::openmmio::Liquid_OpenMM, 389
forcebalance::psi4io::RDVR3_Psi4, 533
forcebalance::psi4io::THCDF_Psi4, 564
forcebalance::target::RemoteTarget, 543
forcebalance::target::Target, 554
forcebalance::tinkerio::AbInitio_TINKER, 199
forcebalance::tinkerio::BindingEnergy_TINKER, 232
forcebalance::tinkerio::Interaction_TINKER, 330
forcebalance::tinkerio::Liquid_TINKER, 404
forcebalance::tinkerio::Moments_TINKER, 477
forcebalance::vibration::Vibration, 582
remote_indicate
forcebalance::target::RemoteTarget, 545
remove_if_exists
forcebalance::nifty, 62
removeHandler
forcebalance::output::ForceBalanceLogger, 266
replace_peratom
forcebalance::molecule::Molecule, 449
replace_peratom_conditional
forcebalance::molecule::Molecule, 449
require
forcebalance::molecule::Molecule, 449
require_boxes
forcebalance::molecule::Molecule, 449
require_resid
forcebalance::molecule::Molecule, 450
require_resname
forcebalance::molecule::Molecule, 450
ResetVirtualSites
forcebalance::openmmio, 73
resid
forcebalance::molecule::Molecule, 455
residue_distances
forcebalance::contact, 20

resname
forcebalance::molecule::Molecule, 455
respterm
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::tinkerio::AbInitio_TINKER, 205
rm_gmx_baks
forcebalance::gmxio, 33
rmsd_part
forcebalance::binding::BindingEnergy, 225
forcebalance::tinkerio::BindingEnergy_TINKER, 235
row
forcebalance::nifty, 62
rs
forcebalance::forcefield::FF, 264
rs_override
forcebalance::forcefield, 29
rsmake
forcebalance::forcefield::FF, 261
Run
forcebalance::optimizer::Optimizer, 508
rundir
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::binding::BindingEnergy, 225
forcebalance::counterpoise::Counterpoise, 249
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::gmxqpio::Monomer_QTPIE, 493
forcebalance::interaction::Interaction, 294
forcebalance::leastsq::LeastSquares, 351
forcebalance::liquid::Liquid, 363
forcebalance::moments::Moments, 468
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::psi4io::RDVR3_Psi4, 536
forcebalance::psi4io::THCDF_Psi4, 568
forcebalance::target::RemoteTarget, 546
forcebalance::target::Target, 556
forcebalance::tinkerio::AbInitio_TINKER, 205
forcebalance::tinkerio::BindingEnergy_TINKER, 235
forcebalance::tinkerio::Interaction_TINKER, 334
forcebalance::tinkerio::Liquid_TINKER, 407
forcebalance::tinkerio::Moments_TINKER, 480
forcebalance::vibration::Vibration, 585

save_vmvals
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::tinkerio::AbInitio_TINKER, 205
SavedMVal
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::tinkerio::Liquid_TINKER, 407
SavedTraj
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::liquid::Liquid, 363
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::tinkerio::Liquid_TINKER, 408
Scan_Values
forcebalance::optimizer::Optimizer, 508
ScanMVals
forcebalance::optimizer::Optimizer, 509
ScanPVals
forcebalance::optimizer::Optimizer, 509
ScipyOptimizer
forcebalance::optimizer::Optimizer, 509
sec
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 340
forcebalance::qchemio::QCIn_Reader, 522
section
forcebalance::amberio::Mol2_Reader, 427
segments
forcebalance::nifty, 63
select1
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::interaction::Interaction, 294
forcebalance::openmmio::Interaction_OpenMM, 321
forcebalance::tinkerio::Interaction_TINKER, 334
select2
forcebalance::gmxio::Interaction_GMX, 308
forcebalance::interaction::Interaction, 294
forcebalance::openmmio::Interaction_OpenMM, 322
forcebalance::tinkerio::Interaction_TINKER, 334
set_atom_id
forcebalance::Mol2::mol2_atom, 417
set_atom_name
forcebalance::Mol2::mol2_atom, 417
set_atom_type
forcebalance::Mol2::mol2_atom, 417
set_bond_id
forcebalance::Mol2::mol2_bond, 422
set_bond_type
forcebalance::Mol2::mol2_bond, 422
set_charge
forcebalance::Mol2::mol2_atom, 418
set_charge_type

forcebalance::Mol2::mol2, 411
set_crds
forcebalance::Mol2::mol2_atom, 418
set_donnor_acceptor_atoms
forcebalance::Mol2::mol2, 412
set_mol_name
forcebalance::Mol2::mol2, 412
set_mol_type
forcebalance::Mol2::mol2, 412
set_num_atoms
forcebalance::Mol2::mol2, 413
set_num_bonds
forcebalance::Mol2::mol2, 413
set_num_feat
forcebalance::Mol2::mol2, 413
set_num_sets
forcebalance::Mol2::mol2, 414
set_num_subst
forcebalance::Mol2::mol2, 414
set_option
forcebalance::abinitio::AbInitio, 99
forcebalance::abinitio_internal::AbInitio_Internal, 159
forcebalance::amberio::AbInitio_AMBER, 119
forcebalance::BaseClass, 211
forcebalance::binding::BindingEnergy, 222
forcebalance::counterpoise::Counterpoise, 247
forcebalance::forcefield::FF, 261
forcebalance::gmxio::AbInitio_GMX, 139
forcebalance::gmxio::Interaction_GMX, 305
forcebalance::gmxio::Liquid_GMX, 374
forcebalance::gmxqpio::Monomer_QTPIE, 490
forcebalance::interaction::Interaction, 291
forcebalance::leastsq::LeastSquares, 348
forcebalance::liquid::Liquid, 360
forcebalance::moments::Moments, 465
forcebalance::objective::Objective, 496
forcebalance::openmmio::AbInitio_OpenMM, 179
forcebalance::openmmio::Interaction_OpenMM, 318
forcebalance::openmmio::Liquid_OpenMM, 389
forcebalance::optimizer::Optimizer, 510
forcebalance::psi4io::RDVR3_Psi4, 533
forcebalance::psi4io::THCDF_Psi4, 565
forcebalance::target::RemoteTarget, 543
forcebalance::target::Target, 554
forcebalance::tinkerio::AbInitio_TINKER, 200
forcebalance::tinkerio::BindingEnergy_TINKER, 232
forcebalance::tinkerio::Interaction_TINKER, 331
forcebalance::tinkerio::Liquid_TINKER, 404
forcebalance::tinkerio::Moments_TINKER, 477
forcebalance::vibration::Vibration, 583
set_origin_atom_id
forcebalance::Mol2::mol2_bond, 422
set_status_bit
forcebalance::Mol2::mol2_atom, 418
forcebalance::Mol2::mol2_bond, 422
set_subst_id
forcebalance::Mol2::mol2_atom, 418
set_subst_name
forcebalance::Mol2::mol2_atom, 419
set_target_atom_id
forcebalance::Mol2::mol2_bond, 423
sget
forcebalance::abinitio::AbInitio, 99
forcebalance::abinitio_internal::AbInitio_Internal, 159
forcebalance::amberio::AbInitio_AMBER, 119
forcebalance::binding::BindingEnergy, 222
forcebalance::counterpoise::Counterpoise, 247
forcebalance::gmxio::AbInitio_GMX, 139
forcebalance::gmxio::Interaction_GMX, 305
forcebalance::gmxio::Liquid_GMX, 375
forcebalance::gmxqpio::Monomer_QTPIE, 490
forcebalance::interaction::Interaction, 291
forcebalance::leastsq::LeastSquares, 348
forcebalance::liquid::Liquid, 360
forcebalance::moments::Moments, 465
forcebalance::openmmio::AbInitio_OpenMM, 179
forcebalance::openmmio::Interaction_OpenMM, 318
forcebalance::openmmio::Liquid_OpenMM, 389
forcebalance::psi4io::RDVR3_Psi4, 534
forcebalance::psi4io::THCDF_Psi4, 565
forcebalance::target::RemoteTarget, 543
forcebalance::target::Target, 554
forcebalance::tinkerio::AbInitio_TINKER, 200
forcebalance::tinkerio::BindingEnergy_TINKER, 232
forcebalance::tinkerio::Interaction_TINKER, 331
forcebalance::tinkerio::Liquid_TINKER, 404
forcebalance::tinkerio::Moments_TINKER, 477
forcebalance::vibration::Vibration, 583
shell
forcebalance::qchemio::QCIn_Reader, 522
Simplex
forcebalance::optimizer::Optimizer, 510
simulation
forcebalance::openmmio::AbInitio_OpenMM, 184
simulations
forcebalance::openmmio::Interaction_OpenMM, 322
SinglePoint
forcebalance::optimizer::Optimizer, 510
snum
forcebalance::qchemio::QCIn_Reader, 522
spacings
forcebalance::objective::Penalty, 517
specific_dct
forcebalance::nifty, 66
specific_lst
forcebalance::nifty, 66
Split
forcebalance::amberio::FrcMod_Reader, 269

forcebalance::amberio::Mol2_Reader, 426
forcebalance::BaseReader, 214
forcebalance::custom_io::Gen_Reader, 277
forcebalance::gmxio::ITP_Reader, 339
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::openmmio::OpenMM_Reader, 500
forcebalance::psi4io::GBS_Reader, 273
forcebalance::psi4io::Grid_Reader, 282
forcebalance::qchemio::QCIn_Reader, 521
forcebalance::tinkerio::Tinker_Reader, 572
split
forcebalance::molecule::Molecule, 450
splitter
forcebalance::molecule, 49
stage
forcebalance::abinitio::AbInitio, 100
forcebalance::abinitio_internal::AbInitio_Internal, 160
forcebalance::amberio::AbInitio_AMBER, 120
forcebalance::binding::BindingEnergy, 223
forcebalance::counterpoise::Counterpoise, 248
forcebalance::gmxio::AbInitio_GMX, 140
forcebalance::gmxio::Interaction_GMX, 306
forcebalance::gmxio::Liquid_GMX, 375
forcebalance::gmxqpio::Monomer_QTPIE, 491
forcebalance::interaction::Interaction, 292
forcebalance::leastsq::LeastSquares, 349
forcebalance::liquid::Liquid, 361
forcebalance::moments::Moments, 466
forcebalance::openmmio::AbInitio_OpenMM, 180
forcebalance::openmmio::Interaction_OpenMM, 319
forcebalance::openmmio::Liquid_OpenMM, 390
forcebalance::psi4io::RDVR3_Psi4, 534
forcebalance::psi4io::THCDF_Psi4, 566
forcebalance::target::RemoteTarget, 544
forcebalance::target::Target, 555
forcebalance::tinkerio::AbInitio_TINKER, 201
forcebalance::tinkerio::BindingEnergy_TINKER, 233
forcebalance::tinkerio::Interaction_TINKER, 332
forcebalance::tinkerio::Liquid_TINKER, 405
forcebalance::tinkerio::Moments_TINKER, 478
forcebalance::vibration::Vibration, 584
statisticalInefficiency
forcebalance::nifty, 63
status_bit
forcebalance::Mol2::mol2_atom, 419
forcebalance::Mol2::mol2_bond, 423
step
forcebalance::optimizer::Optimizer, 511
subdict
forcebalance::parser, 80
submit_jobs
forcebalance::abinitio::AbInitio, 100
forcebalance::abinitio_internal::AbInitio_Internal, 160
forcebalance::amberio::AbInitio_AMBER, 120
forcebalance::binding::BindingEnergy, 224
forcebalance::counterpoise::Counterpoise, 248
forcebalance::gmxio::AbInitio_GMX, 140
forcebalance::gmxio::Interaction_GMX, 306
forcebalance::gmxio::Liquid_GMX, 376
forcebalance::gmxqpio::Monomer_QTPIE, 491
forcebalance::interaction::Interaction, 292
forcebalance::leastsq::LeastSquares, 350
forcebalance::liquid::Liquid, 361
forcebalance::moments::Moments, 466
forcebalance::openmmio::AbInitio_OpenMM, 180
forcebalance::openmmio::Interaction_OpenMM, 319
forcebalance::openmmio::Liquid_OpenMM, 391
forcebalance::psi4io::RDVR3_Psi4, 535
forcebalance::psi4io::THCDF_Psi4, 566
forcebalance::target::RemoteTarget, 544
forcebalance::target::Target, 555
forcebalance::tinkerio::AbInitio_TINKER, 201
forcebalance::tinkerio::BindingEnergy_TINKER, 234
forcebalance::tinkerio::Interaction_TINKER, 332
forcebalance::tinkerio::Liquid_TINKER, 405
forcebalance::tinkerio::Moments_TINKER, 479
forcebalance::vibration::Vibration, 584
subst_id
forcebalance::Mol2::mol2_atom, 420
subst_name
forcebalance::Mol2::mol2_atom, 420
suffix
forcebalance::amberio::FrcMod_Reader, 270
forcebalance::amberio::Mol2_Reader, 427
forcebalance::BaseReader, 215
forcebalance::custom_io::Gen_Reader, 278
forcebalance::gmxio::ITP_Reader, 340
forcebalance::mol2io::Mol2_Reader, 431
forcebalance::openmmio::OpenMM_Reader, 501
forcebalance::psi4io::GBS_Reader, 275
forcebalance::psi4io::Grid_Reader, 283
forcebalance::qchemio::QCIn_Reader, 522
forcebalance::tinkerio::Tinker_Reader, 573
suffix_dict
forcebalance::openmmio, 73
system_driver
forcebalance::tinkerio::BindingEnergy_TINKER, 234

target.py, 606
Target_Terms
forcebalance::objective::Objective, 496
target_atom_id
forcebalance::Mol2::mol2_bond, 423
Targets
forcebalance::objective::Objective, 497
tdir
forcebalance::psi4io::RDVR3_Psi4, 536
tempdir

forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::binding::BindingEnergy, 225
forcebalance::counterpoise::Counterpoise, 250
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::gmxio::Interaction_GMX, 309
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::gmxqpio::Monomer_QTPIE, 493
forcebalance::interaction::Interaction, 294
forcebalance::leastsq::LeastSquares, 351
forcebalance::liquid::Liquid, 364
forcebalance::moments::Moments, 468
forcebalance::openmmio::AbInitio_OpenMM, 184
forcebalance::openmmio::Interaction_OpenMM, 322
forcebalance::openmmio::Liquid_OpenMM, 393
forcebalance::psi4io::RDVR3_Psi4, 536
forcebalance::psi4io::THCDF_Psi4, 569
forcebalance::target::RemoteTarget, 546
forcebalance::target::Target, 556
forcebalance::tinkerio::AbInitio_TINKER, 205
forcebalance::tinkerio::BindingEnergy_TINKER, 235
forcebalance::tinkerio::Interaction_TINKER, 334
forcebalance::tinkerio::Liquid_TINKER, 408
forcebalance::tinkerio::Moments_TINKER, 481
forcebalance::vibration::Vibration, 586

tgt_opts_defaults
forcebalance::parser, 80

tgt_opts_types
forcebalance::parser, 80

throw_outs
forcebalance::psi4io::THCDF_Psi4, 569

tinkerio.py, 606

tinkerprm
forcebalance::forcefield::FF, 264

tm
forcebalance::forcefield::FF, 264

tmI
forcebalance::forcefield::FF, 264

topfnm
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::gmxio::Interaction_GMX, 309

topology
forcebalance::molecule::Molecule, 455

topology_flag
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::gmxio::AbInitio_GMX, 144
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 205

tq_err
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 124
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 205

tq_err_pct
forcebalance::abinitio::AbInitio, 104
forcebalance::abinitio_internal::AbInitio_Internal, 164
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206

tq_ref
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206

traj
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::gmxio::Interaction_GMX, 309
forcebalance::interaction::Interaction, 295
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::openmmio::Interaction_OpenMM, 322
forcebalance::tinkerio::AbInitio_TINKER, 206
forcebalance::tinkerio::Interaction_TINKER, 335

trajfnm
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::gmxio::Interaction_GMX, 309
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::openmmio::Interaction_OpenMM, 322
forcebalance::tinkerio::AbInitio_TINKER, 206
forcebalance::tinkerio::Interaction_TINKER, 335

uncommadash
forcebalance::nifty, 64

unpack_moments
forcebalance::gmxqpio::Monomer_QTPIE, 491
forcebalance::moments::Moments, 466
forcebalance::tinkerio::Moments_TINKER, 479

UpdateSimulationParameters
forcebalance::openmmio, 73

use_nft
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206

Val
forcebalance::optimizer::Optimizer, 513
verbose_options
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::BaseClass, 211
forcebalance::binding::BindingEnergy, 225
forcebalance::counterpoise::Counterpoise, 250
forcebalance::forcefield::FF, 264
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::gmxio::Interaction_GMX, 309
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::gmxqpio::Monomer_QTPIE, 493
forcebalance::interaction::Interaction, 295
forcebalance::leastsq::LeastSquares, 351
forcebalance::liquid::Liquid, 364
forcebalance::moments::Moments, 468
forcebalance::objective::Objective, 497
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::openmmio::Interaction_OpenMM, 322
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::optimizer::Optimizer, 513
forcebalance::psi4io::RDVR3_Psi4, 537
forcebalance::psi4io::THCDF_Psi4, 569
forcebalance::target::RemoteTarget, 546
forcebalance::target::Target, 556
forcebalance::tinkerio::AbInitio_TINKER, 206
forcebalance::tinkerio::BindingEnergy_TINKER, 236
forcebalance::tinkerio::Interaction_TINKER, 335
forcebalance::tinkerio::Liquid_TINKER, 408
forcebalance::tinkerio::Moments_TINKER, 481
forcebalance::vibration::Vibration, 586
vfnm
forcebalance::vibration::Vibration, 586
vibration.py, 607
vibration_driver
forcebalance::tinkerio::Vibration_TINKER, 588

w_alpha
forcebalance::gmxio::Liquid_GMX, 378
forcebalance::liquid::Liquid, 364
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::tinkerio::Liquid_TINKER, 408
w_cp
forcebalance::gmxio::Liquid_GMX, 379
forcebalance::liquid::Liquid, 364
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::tinkerio::Liquid_TINKER, 408
w_energy
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206
w_eps0
forcebalance::gmxio::Liquid_GMX, 379
forcebalance::liquid::Liquid, 364
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::tinkerio::Liquid_TINKER, 408
w_force
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206
w_hvap
forcebalance::gmxio::Liquid_GMX, 379
forcebalance::liquid::Liquid, 364
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::tinkerio::Liquid_TINKER, 408
w_kappa
forcebalance::gmxio::Liquid_GMX, 379
forcebalance::liquid::Liquid, 364
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::tinkerio::Liquid_TINKER, 408
w_netforce
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206
w_resp
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206
w_rho
forcebalance::gmxio::Liquid_GMX, 379
forcebalance::liquid::Liquid, 364
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::tinkerio::Liquid_TINKER, 408
w_torque
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 185
forcebalance::tinkerio::AbInitio_TINKER, 206
WORK_QUEUE
forcebalance, 12
WQIDS
forcebalance, 12

WTot
forcebalance::objective::Objective, 497
warn_once
forcebalance::nifty, 64
warn_press_key
forcebalance::nifty, 64
weight
forcebalance::gmxio::Interaction_GMX, 309
forcebalance::interaction::Interaction, 295
forcebalance::openmmio::Interaction_OpenMM, 322
forcebalance::tinkerio::Interaction_TINKER, 335
weight_info
forcebalance::liquid, 37
weighted_variance
forcebalance::abinitio, 13
weighted_variance2
forcebalance::abinitio, 14
weights
forcebalance::gmxqpio::Monomer_QTPIE, 493
whamboltz
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 186
forcebalance::tinkerio::AbInitio_TINKER, 206
whamboltz_wts
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 165
forcebalance::amberio::AbInitio_AMBER, 125
forcebalance::gmxio::AbInitio_GMX, 145
forcebalance::openmmio::AbInitio_OpenMM, 186
forcebalance::tinkerio::AbInitio_TINKER, 206
which
forcebalance::nifty, 65
Whites
forcebalance::amberio::FrcMod_Reader, 269
forcebalance::amberio::Mol2_Reader, 426
forcebalance::BaseReader, 214
forcebalance::custom_io::Gen_Reader, 277
forcebalance::gmxio::ITP_Reader, 339
forcebalance::mol2io::Mol2_Reader, 430
forcebalance::openmmio::OpenMM_Reader, 500
forcebalance::psi4io::GBS_Reader, 273
forcebalance::psi4io::Grid_Reader, 282
forcebalance::qchemio::QCIn_Reader, 521
forcebalance::tinkerio::Tinker_Reader, 572
wq_complete
forcebalance::abinitio::AbInitio, 100
forcebalance::abinitio_internal::AbInitio_Internal, 160
forcebalance::amberio::AbInitio_AMBER, 120
forcebalance::binding::BindingEnergy, 224
forcebalance::counterpoise::Counterpoise, 248
forcebalance::gmxio::AbInitio_GMX, 140
forcebalance::gmxio::Interaction_GMX, 306
forcebalance::gmxio::Liquid_GMX, 376
forcebalance::gmxqpio::Monomer_QTPIE, 492
forcebalance::interaction::Interaction, 292
forcebalance::leastsq::LeastSquares, 350
forcebalance::liquid::Liquid, 362
forcebalance::moments::Moments, 467
forcebalance::openmmio::AbInitio_OpenMM, 180
forcebalance::openmmio::Interaction_OpenMM, 319
forcebalance::openmmio::Liquid_OpenMM, 391
forcebalance::psi4io::RDVR3_Psi4, 535
forcebalance::psi4io::THCDF_Psi4, 566
forcebalance::target::RemoteTarget, 545
forcebalance::target::Target, 555
forcebalance::tinkerio::AbInitio_TINKER, 201
forcebalance::tinkerio::BindingEnergy_TINKER, 234
forcebalance::tinkerio::Interaction_TINKER, 332
forcebalance::tinkerio::Liquid_TINKER, 406
forcebalance::tinkerio::Moments_TINKER, 479
forcebalance::vibration::Vibration, 584
wq_wait
forcebalance::nifty, 65
wq_wait1
forcebalance::nifty, 65
write
forcebalance::molecule::Molecule, 451
Write_Tab
forcebalance::molecule::Molecule, 455
write_arc
forcebalance::molecule::Molecule, 451
write_dcd
forcebalance::molecule::Molecule, 451
write_gro
forcebalance::molecule::Molecule, 452
write_key_with_prm
forcebalance::tinkerio, 85
write_mdcrd
forcebalance::molecule::Molecule, 452
write_molproq
forcebalance::molecule::Molecule, 452
write_nested_destroy
forcebalance::psi4io::THCDF_Psi4, 567
write_pdb
forcebalance::molecule::Molecule, 452
write_qcin
forcebalance::molecule::Molecule, 453
write_qdata
forcebalance::molecule::Molecule, 453
write_xyz
forcebalance::molecule::Molecule, 454
writechk
forcebalance::optimizer::Optimizer, 511

x

forcebalance::Mol2::mol2_atom, 420
XMLFILE
forcebalance::nifty, 67
xct
forcebalance::abinitio::AbInitio, 105
forcebalance::abinitio_internal::AbInitio_Internal, 166
forcebalance::amberio::AbInitio_AMBER, 126
forcebalance::binding::BindingEnergy, 225
forcebalance::counterpoise::Counterpoise, 250
forcebalance::gmxio::AbInitio_GMX, 146
forcebalance::gmxio::Interaction_GMX, 309
forcebalance::gmxio::Liquid_GMX, 379
forcebalance::gmxqpio::Monomer_QTPIE, 493
forcebalance::interaction::Interaction, 295
forcebalance::leastsq::LeastSquares, 351
forcebalance::liquid::Liquid, 364
forcebalance::moments::Moments, 469
forcebalance::openmmio::AbInitio_OpenMM, 186
forcebalance::openmmio::Interaction_OpenMM, 322
forcebalance::openmmio::Liquid_OpenMM, 394
forcebalance::psi4io::RDVR3_Psi4, 537
forcebalance::psi4io::THCDF_Psi4, 569
forcebalance::target::RemoteTarget, 546
forcebalance::target::Target, 557
forcebalance::tinkerio::AbInitio_TINKER, 207
forcebalance::tinkerio::BindingEnergy_TINKER, 236
forcebalance::tinkerio::Interaction_TINKER, 335
forcebalance::tinkerio::Liquid_TINKER, 408
forcebalance::tinkerio::Moments_TINKER, 481
forcebalance::vibration::Vibration, 586
xyz_omms
forcebalance::openmmio::AbInitio_OpenMM, 186
xyzs
forcebalance::counterpoise::Counterpoise, 250

y
forcebalance::Mol2::mol2_atom, 420

z
forcebalance::Mol2::mol2_atom, 420