Links between atoms can be enforced with

```
link <string atomname> <string atomname>
```

For example, to link atom SG in segment 20 with atom FE in segment 55, use:

```
link 20:_SG 55:FE
```

The format of the sequence file is given in Table 30.8. In addition to the list of segments this file also includes links between non-standard segments or other non-standard links. These links are generated based on distances found between atoms on the pdb file. When atoms are involved in such non-standard links that have not been identified in the fragment of segment files as a non-chain link atom, the prepare module will ignore these links and report them as skipped. If one or more of these links are required, the user has to include them with explicit link directives in the sequence file, making them forced links. Alternatively, these links can be made forced-links by changing `link` into `LINK` in the sequence file.

```
fraction { <integer imol> }
```

Directive `fraction` can be used to separate solute molecules into fractions for which energies will be separately reported during molecular dynamics simulations. The listed molecules will be the last molecule in a fraction. Up to 10 molecules may be specified in this directive.

```
counter <integer num> <string ion>
```

Directive `counter` adds `num` counter ions of type `ion` to the sequence file. Up to 10 `counter` directives may appear in the input block.

26.4 Topology file generation

```
new_top [ new_seq ]
```

Keyword `new_top` is used to force the generation of a new topology file. An existing topology file for the system in the current directory will be overwritten. If keyword `new_seq` is also specified, an existing sequence file will also be overwritten with a newly generated file.

```
amber | charmm
```

The prepare module generates force field specific fragment, segment and topology files. The force field may be explicitly specified in the prepare input block by specifying its name. Currently **AMBER** and **CHARMM** are the supported force fields. A default force field may be specified in the file $HOME/.nwchemrc.

```
standard <string dir_s>
extensions <string dir_x>
user <string dir_u>
temporary <string dir_t>
```

The user can explicitly specify the directories where force field specific databases can be found. These include force field standards, extensions, user preferences and temporary database files.
Defaults for the directories where database files reside may be specified in the file $HOME/.nwchemrc for each of the supported force fields. Fragment, segment and sequence files generated by the **prepare** module are written in the temporary directory. When not specified, the current directory will be used. Topology and restart files are always created in the current directory.

The following directives control the modifications of a topology file. These directives are executed in the order in which they appear in the **prepare** input deck. The topology modifying commands are not stored on the run-time database and are, therefor, not persistent.

```
modify atom <string atomname> [set <integer mset> | initial | final] \
( type <string atomtyp> |  charge <real atomcharge> |  \
  polar <real atompolar> | dummy | self | quantum )
```

These `modify` commands change the atom type, partial atomic charge, atomic polarizability, specify a dummy, self-interaction and quantum atom, respectively. If `mset` is specified, the modification will only apply to the specified set, which has to be 1, 2 or 3. If not specified, the modification will be applied to all three sets. The `atomnam` should be specified as `<integer isgm>:<string name>`, where `isgm` is the segment number, and `name` is the atom name. A leading blank in an atom name should be substituted with an underscore. The modify commands may be combined. For example, the following directive changes for the specified atom the charge and atom type in set 2 and specifies the atom to be a dummy in set 3.

```
modify atom 12:_C1 set 2 charge 0.12 type CA set 3 dummy
```

With the following directives modifications can be made for entire segments.

```
modify segment <integer isgm> [set <integer mset> | initial | final] \
( dummy | self | quantum )
```

Modifications to bonded interaction parameters can be made with the following modify commands.

```
modify ( bond <string atomtyp> <string atomtyp> |  \
 angle <string atomtyp> <string atomtyp> <string atomtyp> |        \
   torsion <string atomtyp> <string atomtyp> <string atomtyp>        \
 <string atomtyp> [ multiplicity <integer multip> ] |      \
 plane <string atomtyp> <string atomtyp> <string atomtyp>          \
 <string atomtyp> ) [set <integer mset> | initial | final] \
 <real value> <real forcon>
```

where `atomtyp` and `mset` are defined as above, `multip` is the torsion ultiplicity for which the modification is to be applied, `value` is the reference bond, angle, torsion angle of out-of-plane angle value respectively, and `forcon` is the force constant for bond, angle, torsion angle of out-of-plane angle. When `multip` or `mset` are not defined the modification will be applied to all multiplicities and sets, respectively, for the identified bonded interaction.

After modifying atoms to quantum atoms the bonded interactions in which only quantum atoms are involved are removed from the bonded lists using

```
update lists
```

Error messages resulting from parameters not being defined for bonded interaction in which only quantum atoms are involved are ignored using

```
ignore
```

26.5 Appending to an existing topology file

```
noe <integer isgm1> <string atom1> <integer isgm2> <string atom3> \
  <real dist1> <real dist2> <real forc1> <real forc2> <real forc3>
```

This directive specifies a distance restraint.

```
select <integer isel> { <string atom1> }
```

Directive `select` specifies a group of atoms used in the definition of potential of mean force potentials.

```
pmf ( zalign | xyplane ) <integer isel> <real forcon1> <real forcon2>
pmf distance <integer isel> <integer jsel> \
             <real dist1> <real dist2> <real forcon1> <real forcon2>
pmf angle <integer isel> <integer jsel> <integer ksel> \
             <real angle1> <real angle2> <real forcon1> <real forcon2>
pmf torsion <integer isel> <integer jsel> <integer ksel> <integer lsel> \
             <real angle1> <real angle2> <real forcon1> <real forcon2>
```

Directive `pmf` specifies a potential of mean force potential in terms of the specified atom selection. Option `zalign` specifies the atoms in the selection to be restrained to a line parallel to the z-axis. Option `xyplane` specifies the atoms in the selection to be restrained to a plane perpendicular to the z-axis. Options `distance`, `angle` and `torsion`, are defined in terms of the center of geometry of the specified atom selections.

26.6 Generating a restart file

```
new_rst
```

Keyword `new_rst` will cause an existing restart file to be overwritten with a new file.

The follwing directives control the manipulation of restart files, and are executed in the order in which they appear in the **prepare** input deck.

```
solvent name <string*3 slvnam default ''HOH''> \
        model <string slvmdl default ''spce''>
```

The solvent keyword can be used to specify the three letter solvent name as expected on the PDB formatted file, and the name of the solvent model for which solvent coordinates will be used.

```
solvate   [ < real rshell default 1.2 > ] \
        ( [ cube <real edge> ] |  \
```

```
          [ box <real xedge> [ <real xedge> [ <real xedge> ]]] | \
          [ sphere <real radius> ] |
          [ troct <real edge> ])
```

Solvation can be specified to be in a cubic box with specified edge, rectangular box with specified edges, or in a sphere with specified radius. Solvation in a cube or rectangular box will automatically also set periodic boundary conditions. Solvation in a sphere will only allow simulations without periodic boundary conditions. The size of the cubic and rectangular boxes will be expanded by a length specified by the expand variable. If no shape is specified, solvation will be done for a cubic box with an edge that leaves rshell nm between any solute atom and a periodic image of any solute atom after the solute has been centered. An explicit `write` is not needed to write the restart file. The `solvate` will write out a file `sys_calc`.rst.

```
touch <real touch default 0.23>
```

The variable `touch` specifies the minimum distance between a solvent and solute atom for which a solvent molecule will be accepted for solvation.

```
expand <real xpndw default 0.1>
```

The variable `xpndw` specifies the size in nm with which the simulation volume will be increased after solvation.

```
read [rst | rst_old | pdb] <string filename>
write [rst | [solute [<integer nsolvent>]] (pdb | xyz)] <string filename>
```

These directives read and write the file `filename` in the specified format. The `solute` option instructs to write out the coordinates for solute and all, or if specified the first `nsolvent`, crystal solvent molecules only. If no format is specified, it will be derived from the extension of the filename. Recognized extensions are rst, rst_old (read only), pdb, xyz (write only) and pov (write only). Reading and then writing the same restart file will cause the sub-block size information to be lost. If this information needs to be retained a shell copy command needs to be used.

```
scale <real scale default -1.0>
```

This directive scales the volume and coordinates written to povray files. A negative value of scale (default) scales the coordinates to lie in [-1:1].

```
cpk [<real cpk default 1.0>]
```

This directive causes povray files to contain cpk model output. The optional value is used to scale the atomic radii. A neagtive value of cpk resets the rendering to stick.

```
center | centerx | centery | centerz
```

These directives center the solute center of geometry at the origin, in the y-z plane, in the x-z plane or in the x-y plane, respectively.

```
orient
```

This directive orients the solute principal axes.

```
translate [atom | segment | molecule] \
 <integer itran> <integer itran> <real xtran(3)>
```

This directive translates solute atoms in the indicated range by xtran, without checking for bad contacts in the resulting structure.

```
remove solvent [inside | outside] [x <real xmin> <real xmax>] \
[y <real ymin> <real ymax>] [z <real zmin> <real zmax>]
```

This directive removes solvent molecules inside or outside the specified coordinate range.

```
periodic
```

This directive enables periodic boundary conditions.

```
vacuo
```

This directive disables periodic boundary conditions.

```
grid <integer mgrid default 24> <real rgrid default 0.2>
```

This directive specifies the grid size of trial counter-ion positions and minimum distance between an atom in the system and a counter-ion.

```
fix ( atoms | segments ) ( beyond | within ) <real rfix> <string atmfix>
```

The `fix` keyword may be used to specify that the identified atoms should remain fixed during any operation.

```
box <real xsize> <real ysize>  <real zsize>
```

The `box` directive resets the box size.

```
align <string atomi> <string atomj> <string atomk>
```

The `align` directive orients the system such that `atomi` and `atomj` are on the z-axis, and `atomk` in the x=y plane.

```
repeat [chains | molecules | fractions ] \
 <integer nx> <integer ny> <integer nz> [<real dist>] [<real zdist>]
```

The `repeat` directive causes a subsequent `write pdb` directive to write out multiple copies of the system, with `nx` copies in the x, `ny` copies in the y, and `nz` copies in the z-direction, with a minimum distance of `dist` between any pair of atoms from different copies. If `nz` is -2, an inverted copy is placed in the z direction, with a separation of `zdist` nm. If `dist` is negative, the box dimensions will be used. For systems with solvent, this directive should be used with a negative `dist`. Optional keywords `chains`, `molecules` and `fractions` specify to write each repeating solute unit as a chain, to repeat each solute molecule, or each solute fraction separately.

Chapter 27

Molecular dynamics

27.1 Introduction

27.1.1 Spacial decomposition

The molecular dynamics module of **NWChem** uses a distribution of data based on a spacial decomposition of the molecular system, offering an efficient parallel implementation in terms of both memory requirements and communication costs, especially for simulations of large molecular systems.

Inter-processor communication using the global array tools and the design of a data structure allowing distribution based on spacial decomposition are the key elements in taking advantage of the distribution of memory requirements and computational work with minimal communication.

In the spacial decomposition approach, the physical simulation volume is divided into rectangular boxes, each of which is assigned to a processor. Depending on the conditions of the calculation and the number of available processors, each processor contains one or more of these spacially grouped boxes. The most important aspects of this decomposition are the dependence of the box sizes and communication cost on the number of processors and the shape of the boxes, the frequent reassignment of atoms to boxes leading to a fluctuating number of atoms per box, and the locality of communication which is the main reason for the efficiency of this approach for very large molecular systems.

To improve efficiency, molecular systems are broken up into separately treated solvent and solute parts. Solvent molecules are assigned to the domains according to their center of geometry and are always owned by a one node. This avoids solvent–solvent bonded interactions crossing node boundaries. Solute molecules are broken up into segments, with each segment assigned to a processor based on its center of geometry. This limits the number of solute bonded interactions that cross node boundaries. The processor to which a particular box is assigned is responsible for the calculation of all interactions between atoms within that box. For the calculation of forces and energies in which atoms in boxes assigned to different processors are involved, data are exchanged between processors. The number of neighboring boxes is determined by the size and shape of the boxes and the range of interaction. The data exchange that takes place every simulation time step represents the main communication requirements. Consequently, one of the main efforts is to design algorithms and data structures to minimize the cost of this communication. However, for very large molecular systems, memory requirements also need to be taken into account.

To compromise between these requirements exchange of data is performed in successive point to point communications rather than using the shift algorithm which reduces the number of communication calls for the same amount of communicated data.

For inhomogeneous systems, the computational load of evaluating atomic interactions will generally differ between box pairs. This will lead to load imbalance between processors. Two algorithms have been implemented that allow for dynamically balancing the workload of each processor. One method is the dynamic resizing of boxes such that boxes gradually become smaller on the busiest node, thereby reducing the computational load of that node. Disadvantages of this method are that the efficiency depends on the solute distribution in the simulation volume and the redistribution of work depends on the number of nodes which could lead to results that depend on the number of nodes used. The second method is based on the dynamic redistribution of intra-node box-box interactions. This method represents a more coarse load balancing scheme, but does not have the disadvantages of the box resizing algorithm. For most molecular systems the box pair redistribution is the more efficient and preferred method.

The description of a molecular system consists of static and dynamic information. The static information does not change during a simulation and includes items such as connectivity, excluded and third neighbor lists, equilibrium values and force constants for all bonded and non-bonded interactions. The static information is called the topology of the molecular system, and is kept on a separate topology file. The dynamic information includes coordinates and velocities for all atoms in the molecular system, and is kept in a so-called restart file.

27.1.2 Topology

The static information about a molecular system that is needed for a molecular simulation is provided to the simulation module in a topology file. Items in this file include, among many other things, a list of atoms, their non-bonded parameters for van der Waals and electrostatic interactions, and the complete connectivity in terms of bonds, angles and dihedrals.

In molecular systems, a distinction is made between *solvent* and *solute*, which are treated separately. A solvent molecule is defined only once in the topology file, even though many solvent molecules usually are included in the actual molecular system. In the current implementation only one solvent can be defined. Everything that is not solvent in the molecular system is solute. Each solute atom in the system must be explicitly defined in the topology.

Molecules are defined in terms of one or more *segment*s. Typically, repetitive parts of a molecule are each defined as a single segment, such as the amino acid residues in a protein. Segments can be quite complicated to define and are, therefore, collected in a set of database files. The definition of a molecular system in terms of segments is a *sequence*.

Topology files are created using the **prepare** module.

27.1.3 Files

File names used have the form `$system$_$calc$.$ext$`, with exception of the topology file (Section 27.1.2), which is named `$system$.top`. Anything that refers to the definition of the chemical system can be used for `$system$`, as long as no periods or underlines are used. The identifier `$calc$` can be anything that refers to the type of calculation to be performed for the system with the topology defined. This file naming convention allows for the creation of a single topology file `$system$.top` that can be used for a number of different calculations, each identified with a different `$calc$`. For example, if `crown.top` is the name of the topology file for a crown ether, `crown_em`, `crown_md`, `crown_ti` could be used with appropriate extensions for the filenames for energy minimization, molecular dynamics simulation and multi-configuration thermodynamic integration, respectively. All of these calculations would use the same topology file `crown.top`.

The extensions `<ext>` identify the kind of information on a file, and are pre-determined.

dbg	debug file
frg	fragment file
gib	free energy data file
mri	free energy multiple run input file
mro	free energy multiple run output file
nw	**NWChem** input file
nwout	**NWChem** output file
out	molecular dynamics output file
pdb	PDB formatted coordinate file
prp	property file
qrs	quenched restart file, resulting from an energy minimization
rst	restart file, used to start or restart a simulation
seq	sequence file, describing the system in segments
sgm	segment file, describing segments
syn	synchronization time file
tst	test file
tim	timing analysis file
top	topology file, contains the static description of a system
trj	trajectory file

Table 27.1: List of file extensions for nwchem chemical system files.

27.1.4 Databases

Database file used by the **prepare** module are found in directories with name `$ffield$_$level$`, where `$ffield$` is any of the supported force fields (Section 27.1.5). The source of the data is identified by `$level$`, and can be

level	Description	Availability
s	original published data	public
x	additional published data	public
u	user preferred data	private
t	user defined temporary data	private

Typically, only the level **s** and **x** databases are publicly available. The user is responsible for the private level **u** and **t** database files. When the **prepare** module scans the databases, the priority is **t**>**u**>**x**>**s**>.

The extension `<ext>` defines the type of database file within each database directory.

frg	fragments
par	parameters
seq	sequences
sgm	segments

Table 27.2: List of database file extensions.

The paths of the different database directories should be defined in a file `.nwchemrc` in a user's home directory, and provides the user the option to select which database files are scanned.

27.1.5 Force fields

Force fields recognized are

Keyword	Force field
`amber`	AMBER95
`charmm`	CHARMM

27.2 Format of fragment files

Fragment files contain the basic information needed to specify all interactions that need to be considered in a molecular simulation. The format of the fragment files is described in Table 30.1. Normally these files are created by the **prepare** module. Manual editing is needed when, for example, the **prepare** module could not complete atom typing, or when modified charges are required.

27.3 Creating segment files

The **prepare** module is used to generate segment files from corresponding fragment files. A segment file contains all information for the calculation of bonded and non-bonded interactions for a given chemical system using a specific force field.

Which atoms form a fragment is specified in the coordinate file, currently only in PDB format. the restriction is that bonded interactions may only involve atoms on at most two segments does no longer exist as of **NWChem** release 3.2.1. The segment entries define three sets of parameters for each interaction.

Free energy perturbations can be performed using set 1 for the generation of the ensemble while using sets 2 and/or 3 as perturbations. Free energy multiconfiguration thermodynamic integration and multistep thermodynamic perturbation calculations are performed by gradually changing the interactions in the system from parameter set 2 to parameter set 3. These modifications can be edited into the segment files manually, or introduced directly into the topology file using the `modify` commands in the input for the **prepare** module.

The format of a segment is described in Tables 30.2–30.7.

27.4 Creating sequence files

A sequence file describes a molecular system in terms of segments. This file is generated by the **prepare** module for the molecular system provided on a PDB-formatted coordinate file. The file format is given in Table 30.8

27.5 Creating topology files

The topology (Section 27.1.2) describes all static information that describes a molecular system. This includes the connectivity in terms of bond-stretching, angle-bending and torsional interactions, as well as the non-bonded van der Waals and Coulombic interactions.

The topology of a molecular system is generated by the **prepare** module from the sequence in terms of segments as specified on the PDB file. For each unique segment specified in this file the segment database directories are searched for the segment definition. For segments not found in one of the database directories a segment definition is generated in the temporary directory if a fragment file was found. If a fragment file could not be found, it is generated by the **prepare** module base on what is found on the PDB file.

When all segments are found or created, the parameter substitutions are performed, using force field parameters taken from the parameter databases. After all lists have been generated the topology is written to a local topology file `$system$.top`.

27.6 Creating restart files

Restart files contain all dynamical information about a molecular system and are created by the **prepare** module if a topology file is available. The **prepare** module will automatically generate coordinates for hydrogen atoms and monatomic counter ions not found on the PDB formatted coordinate file, if no fragment or segment files were generated using that PDB file.

The **prepare** module has a number of other optional input command, including solvation.

27.7 Molecular simulations

The type of molecular dynamics simulation is specified by the **NWChem** task directive.

```
task md [ energy | optimize | dynamics | thermodynamics ]
```

where the theory keyword `md` specifies use of the molecular dynamics module, and the operation keyword is one of

- `energy` for single configuration energy evaluation
- `optimize` for energy minimization
- `dynamics` for molecular dynamics simulations and single step thermodynamic perturbation free energy molecular dynamics simulations
- `thermodynamics` for combined multi-configuration thermodynamic integration and multiple step thermodynamic perturbation free energy molecular dynamics simulations.

27.8 System specification

The chemical system for a calculation is specified in the topology and restart files. These files should be created using the utilities **nwtop** and **nwrst** before a simulation can be performed. The names of these files are determined from the **NWChem** `start` directive. There is no default. If the `rtdb` name is given as `system_calc`, the topology file used is `system.top`, while all other files are named `system_calc.ext`.

27.9 Parameter set

```
set <integer iset>
```

specifies the use of parameter set <iset> for the molecular dynamics simulation. The topology file contains three separate parameters sets that can be used. The default for <iset> is 1.

```
pset <integer isetp1> [<integer isetp2>]
```

specifies the parameter sets to be used as perturbation potentials in single step thermodynamic perturbation free energy evaluations, where <isetp1> specifies the first perturbation parameter set and <isetp2> specifies the second perturbation parameter set. Legal values for <isetp1> are 2 and 3. Legal value for <isetp2> is 3, in which case <isetp1> can only be 2. If specified, <iset> is automatically set to 1.

27.10 Energy minimization algorithms

The energy minimization of the system as found in the restart file is performed with the following directives. If both are specified, steepest descent energy minimization precedes conjugate gradient minimization.

```
sd <integer msdit> [init <real dx0sd>] [min <real dxsdmx>] \
                         [max <real dxmsd>]
```

specifies the variables for steepest descent energy minimizations, where <msdit> is the maximum number of steepest descent steps taken, for which the default is 100, <dx0sd> is the initial step size in nm for which the default is 0.001, <dxsdmx> is the threshold for the step size in nm for which the default is 0.0001, and <dxmsd> is the maximum allowed step size in nm for which the default is 0.05.

```
cg <integer mcgit> [init <real dx0cg>] [min <real dxcgmx>] \
                         [cy <integer ncgcy>]
```

specifies the variables for conjugate gradient energy minimizations, where <mcgit> is the maximum number of conjugate gradient steps taken, for which the default is 100, <dx0cg> is the initial search interval size in nm for which the default is 0.001, <dxcgmx> is the threshold for the step size in nm for which the default is 0.0001, and <ncgcy> is the number of conjugate gradient steps after which the gradient history is discarded for which the default is 10. If conjugate gradient energy minimization is preceded by steepest descent energy minimization, the search interval is set to twice the final step of the steepest descent energy minimization.

27.11 Multi-configuration thermodynamic integration

The following keywords control free energy difference simulations. Multi-configuration thermodynamic integrations are always combined with multiple step thermodynamic perturbations.

```
(forward | reverse) [[<integer mrun> of] <integer maxlam>]
```

specifies the direction and number of integration steps in free energy evaluations, with forward being the default direction. <mrun> is the number of ensembles that will be generated in this calculation, and <maxlam> is the total number of ensembles to complete the thermodynamic integration. The default value for <maxlam> is 21. The default value of <mrun> is the value of <maxlam>.

`error <real edacq>`

specifies the maximum allowed statistical error in each generated ensemble, where `<edacq>` is the maximum error allowed in the ensemble average derivative of the Hamiltonian with respect to λ with a default of 5.0 kJ mol^{-1}.

`drift <real ddacq>`

specifies the maximum allowed drift in the free energy result, where `<ddacq>` is the maximum drift allowed in the ensemble average derivative of the Hamiltonian with respect to $\lambda$with a default of 5.0 kJ mol^{-1}ps^{-1}.

`factor <real fdacq>`

specifies the maximum allowed change in ensemble size where `<fdacq>` is the minimum size of an ensemble relative to the previous ensemble in the calculation with a default value of 0.75.

`decomp`

specifies that a free energy decomposition is to be carried out. Since free energy contributions are path dependent, results from a decomposition analysis can not be unambiguously interpreted, and the default is not to perform this decomposition.

`sss [delta <real delta>]`

specifies that atomic non-bonded interactions describe a dummy atom in either the initial or final state of the thermodynamic calculation will be calculated using separation-shifted scaling, where `<delta>` is the separation-shifted scaling factor with a default of 0.075 nm^2. This scaling method prevents problems associated with singularities in the interaction potentials.

`new | renew | extend`

specifies the initial conditions for thermodynamic calculations. `new` indicates that this is an initial mcti calculation, which is the default. `renew` instructs to obtain the initial conditions for each λ from the **mro**-file from a previous mcti calculation, which has to be renamed to an **mri**-file. The keyword `extend` will extend a previous mcti calculation from the data read from an **mri**-file.

27.12 Time and integration algorithm directives

Following directives control the integration of the equations of motion.

`leapfrog | leapfrog_bc`

specifies the integration algorithm, where `leapfrog` specifies the default leap frog integration, and `leapfrog_bc` specifies the Brown-Clarke leap frog integrator.

`[finish | resume]`

if specified, directs to finish or extend the simulation for which the restart file is read.

`guided`

specifies the use of the guided molecular dynamics simulation technique. The current implementation is still under development.

`equil <integer mequi>`

specifies the number of equilibration steps `<mequi>`, with a default of 100.

```
data <integer mdacq> [over <integer ldacq>]]
```

specifies the number of data gathering steps <mdacq> with a default of 500. In multi-configuration thermodynamic integrations <mequi> and <mdacq> are for each of the ensembles, and variable <ldacq> specifies the minimum number of data gathering steps in each ensemble. In regular molecular dynamics simulations <ldacq> is not used. The default value for <ldacq> is the value of <mdacq>.

```
time <real stime>
```

specifies the initial time <stime> of a molecular simulation in ps, with a default of 0.0.

```
step <real tstep>
```

specifies the time step <tstep> in ps, with 0.001 as the default value.

27.13 Ensemble selection

Following directives control the ensemble type.

```
isotherm [<real tmpext>] [trelax <real tmprlx> [<real tmsrlx>]]
```

specifies a constant temperature ensemble using Berendsen's thermostat, where <tmpext> is the external temperature with a default of 298.15 K, and <tmprlx> and <tmsrlx> are temperature relaxation times in ps with a default of 0.1. If only <tmprlx> is given the complete system is coupled to the heat bath with relaxation time <tmprlx>. If both relaxation times are supplied, solvent and solute are independently coupled to the heat bath with relaxation times <tmprlx> and <tmsrlx>, respectively.

```
isobar [<real prsext>] [trelax <real prsrlx> ] \
          [compress <real compr>] [anisotropic] [noz]
```

specifies a constant pressure ensemble using Berendsen's piston, where <prsext> is the external pressure with a default of 1.025 10^5 Pa, <prsrlx> is the pressure relaxation time in ps with a default of 0.5, and <compr> is the system compressibility in m^2N^{-1} with a default of 4.53E-10.

27.14 Velocity reassignments

Velocities can be periodically reassigned to reflect a certain temperature.

```
vreass <integer nfgaus> <real tgauss>
```

specifies that velocities will be reassigned every <nfgaus> molecular dynamics steps, reflecting a temperature of <tgauss> K. The default is not to reassign velocities, i.e. <nfgaus> is 0.

27.15 Cutoff radii

Cutoff radii can be specified for short range and long range interactions.

```
cutoff [short] <real rshort> [long <real rlong>] \
          [qmmm <real rqmmm>]
```

specifies the short range cutoff radius <rshort>, and the long range cutoff radius <rlong> in nm. If the long range cutoff radius is larger than the short range cutoff radius the twin range method will be used, in which short range forces and energies are evaluated every molecular dynamics step, and long range forces and energies with a frequency of `<nflong>` molecular dynamics steps. Keyword `qmmm` specifies the radius of the zone around quantum atoms defining the QM/MM bare charges. The default value for `<rshort>`, `<rlong>` and `<rqmmm>` is 0.9 nm.

27.16 Polarization

First order and self consistent electronic polarization models have been implemented.

```
polar (first | scf [[<integer mpolit>] <real ptol>])
```

specifies the use of polarization potentials, where the keyword `first` specifies the first order polarization model, and `scf` specifies the self consistent polarization field model, iteratively determined with a maximum of `<mpolit>` iterations to within a tolerance of `<ptol>` D in the generated induced dipoles. The default is not to use polarization models.

27.17 External electrostatic field

```
field <real xfield> [freq <real xffreq>] [vector <real xfvect(1:3)>]
```

specifies an external electrostatic field, where `<xfield>` is the field strength, `<xffreq>` is the frequency in MHz and `<xfvect>` is the external field vector.

27.18 Constraints

Constraints are satisfied using the SHAKE coordinate resetting procedure.

```
shake [<integer mshitw> [<integer mshits>]]  \
         [<real tlwsha> [<real tlssha>]]
```

specifies the use of SHAKE constraints, where `<mshitw>` is the maximum number of solvent SHAKE iterations, and `<mshits>` is the maximum number of solute SHAKE iterations. If only `<mshitw>` is specified, the value will also be used for `<mshits>`. The default maximum number of iterations is 100 for both. `<tlwsha>` is the solvent SHAKE tolerance in nm, and `<tlssha>` is the solute SHAKE tolerance in nm. If only `<tlwsha>` is specified, the value given will also be used for `<tlssha>`. The default tolerance is 0.001 nm for both.

```
noshake (solvent | solute)
```

disables SHAKE and treats the bonded interaction according to the force field.

27.19 Long range interaction corrections

Long range electrostatic interactions are implemented using the smooth particle mesh Ewald technique, for neutral periodic cubic systems in the constant volume ensemble, using pair interaction potentials. Particle-mesh Ewald long range interactions can only be used in molecular dynamics simulations using effective pair potentials, and not in free energy simulations, QMD or QM/MM simulations.

```
pme [grid <integer ng>] [alpha <real ealpha>] \
        [order <integer morder>] [fft <integer imfft>]
```

specifies the use of smooth particle-mesh Ewald long range interaction treatment, where **ng** is the number of grid points per dimension, **ealpha** is the Ewald coefficient in nm^{-1}, with a default that leads to a tolerance of 10^{-4} at the short range cutoff radius, and **morder** is order of the Cardinal B-spline interpolation which must be an even number and at least 4 (default value). A platform specific 3D fast Fourier transform is used, if available, when **imfft** is set to 2.

27.20 Fixing coordinates

Solvent or solute may be fixed using the following keywords.

```
( fix | free ) (solvent [<integer idfirst> [<integer idlast>]] | \
     solute [<integer idfirst> [<integer idlast>]] [ heavy | {<string atomname>}] | \
     permanent )
```

For solvent the molecule numbers **idfirst** and **idlast**may be specified to be the first and last molecule to which the directive applies. If omitted, the directive applies to all molecules. For solute, the segment numbers **idfirst** and **idlast**may be specified to be the first and last segment to which the directive applies. If omitted, the directive applies to all segments. In addition, the keyword **heavy** may be specified to apply to all non hydrogen atoms in the solute, or a set of atom names may be specified in which a wildcard character **?** may be used.

27.21 Autocorrelation function

For the evaluation of the statistical error of multi-configuration thermodynamic integration free energy results a correlated data analysis is carried out, involving the calculation of the autocorrelation function of the derivative of the Hamiltonian with respect to the control variable λ.

```
auto <integer lacf> [fit <integer nfit>] [weight <real weight>]
```

controls the calculation of the autocorrelation, where **<lacf>** is the length of the autocorrelation function, with a default of 1000, **<nfit>** is the number of functions used in the fit of the autocorrelation function, with a default of 15, and **<weight>** is the weight factor for the autocorrelation function, with a default value of 0.0.

27.22 Print options

Keywords that control print to the output file, with extension **out**. Print directives may be combined to a single directive.

```
print topol [nonbond] [solvent] [solute]
```

specifies printing the topology information, where **nonbond** refers to the non-bonded interaction parameters, **solvent** to the solvent bonded parameters, and **solute** to the solute bonded parameters. If only **topol** is specified, all topology information will be printed to the output file.

`print step <integer nfoutp> [extra] [energy]`

specifies the frequency `nfoutp` of printing molecular dynamics step information to the output file. If the keyword `extra` is specified additional energetic data are printed for solvent and solute separately. If the keyword `energy` is specified, information is printed for all bonded solute interactions. The default for `nfoutp` is 0. For molecular dynamics simulations this frequency is in time steps, and for multi-configuration thermodynamic integration in λ-steps.

`print stat <integer nfstat>`

specifies the frequency `<nfstat>` of printing statistical information of properties that are calculated during the simulation. For molecular dynamics simulation this frequency is in time steps, for multi-configuration thermodynamic integration in λ-steps.

27.23 Periodic updates

Following keywords control periodic events during a molecular dynamics or thermodynamic integration simulation. Update directives may be combined to a single directive.

`update pairs <integer nfpair>`

specifies the frequency `<nfpair>` in molecular dynamics steps of updating the pair lists. The default for the frequency is 1. In addition, pair lists are also updated after each step in which recording of the restart or trajectory files is performed. Updating the pair lists includes the redistribution of atoms that changed domain and load balancing, if specified.

`update long <integer nflong>`

specifies the frequency `<nflong>` in molecular dynamics steps of updating the long range forces. The default frequency is 1. The distinction of short range and long range forces is only made if the long range cutoff radius was specified to be larger than the short range cutoff radius. Updating the long range forces is also done in every molecular dynamics step in which the pair lists are regenerated.

`update center <integer nfcntr> [zonly] [fraction <integer idscb(1:5)>]`

specifies the frequency `<nfcntr>` in molecular dynamics steps in which the center of geometry of the solute(s) is translated to the center of the simulation volume. Optional keyword `zonly` can be used to specify that centering will take place in the z-direction only. The solute fractions determining the solutes that will be centered are specified by the keyword `fraction` and the vector `<idscb>`, with a maximum of 5 entries. This translation is implemented such that it has no effect on any aspect of the simulation. The default is not to center, i.e. nfcntr is 0. The default fraction used to center solute is 1.

`update motion <integer nfslow>`

specifies the frequency `<nfslow>` in molecular dynamics steps of removing the center of mass motion.

`update analysis <integer nfanal>`

specifies the frequency `<nfanal>` in molecular dynamics steps of invoking the analysis module.

`update rdf <integer nfrdf> [range <real rrdf>] [bins <integer ngl>]`

specifies the frequency `<nfrdf>` in molecular dynamics steps of calculating contributions to the radial distribution functions. The default is 0. The range of the radial distribution functions is given by `<rrdf>` in nm, with a default of the short range cutoff radius. Note that radial distribution functions are not evaluated beyond the short range cutoff radius. The number of bins in each radial distribution function is given by `<ngl>`, with a default of 1000. If radial distribution function are to be calculated, a **rdi** files needs to be available in which the contributions are specified as follows.

Card	Format	Description
I-1	i	Type, 1=solvent-solvent, 2=solvent-solute, 3-solute-solute
I-2	i	Number of the rdf for this contribution
I-3	i	First atom number
I-4	i	Second atom number

27.24 Recording

The following keywords control recording data to file. Record directives may be combined to a single directive.

`record rest <integer nfrest> [keep]`

specifies the frequency `<nfrest>` in molecular dynamics steps of rewriting the restart file, with extension `rst`. For multi-configuration thermodynamic integration simulations the frequency is in steps in λ. The default is not to record. The restart file is used to start or restart simulations. The keyword `keep` causes all restart files written to be kept on disk, rather than to be overwritten.

`record coord <integer nfcoor>`

specifies the frequency `<nfcoor>` in molecular dynamics steps of writing coordinates to the trajectory file. This directive redefines previous `record coord`, `record wcoor` and `record scoor` directives. The default is not to record.

`record wcoor <integer nfwcoo>`

specifies the frequency `<nfcoor>` in molecular dynamics steps of writing solvent coordinates to the trajectory file. This keyword takes precedent over `record coord`. This directive redefines previous `record coord`, `record wcoor` and `record scoor` directives. The default is not to record.

`record scoor <integer nfscoo>`

specifies the frequency `<nfscoo>` in molecular dynamics steps of writing solute coordinates to the trajectory file. This keyword takes precedent over `record coord`. This directive redefines previous `record coord`, `record wcoor` and `record scoor` directives. The default is not to record.

`record veloc <integer nfvelo>`

specifies the frequency `<nfvelo>` in molecular dynamics steps of writing velocitiesto the trajectory file. This directive redefines previous `record veloc`, `record wvelo` and `record svelo` directives. The default is not to record.

`record wvelo <integer nfwvel>`

specifies the frequency `<nfvelo>` in molecular dynamics steps of writing solvent velocitiesto the trajectory file. This keyword takes precedent over `record veloc`. This directive redefines previous `record veloc`, `record wvelo` and `record svelo` directives. The default is not to record.

`record svelo <integer nfsvel>`

specifies the frequency `<nfsvel>` in molecular dynamics steps of writing solute velocities to the trajectory file. This keyword takes precedent over `record veloc`. This directive redefines previous `record veloc`, `record wvelo` and `record svelo` directives. The default is not to record.

`record prop <integer nfprop>`

specifies the frequency `<nfprop>` in molecular dynamics steps of writing information to the property file, with extension `prp`. For multi-configuration thermodynamic integration simulations the frequency is in steps in λ. The default is not to record.

`record free <integer nffree>`

specifies the frequency `<nffree>` in multi-configuration thermodynamic integration steps to record data to the free energy data file, with extension `gib`. The default is 1, i.e. to record at every λ.

`record sync <integer nfsync>`

specifies the frequency `<nfsync>` in molecular dynamics steps of writing information to the synchronization file, with extension `syn`. The default is not to record. The information written is the simulation time, the wall clock time of the previous MD step, the wall clock time of the previous force evaluation, the total synchronization time, the largest synchronization time and the node on which the largest synchronization time was found. The recording of synchronization times is part of the load balancing algorithm. Since load balancing is only performed when pair-lists are updated, the frequency `<nfsync>` is correlated with the frequency of pair-list updates `<nfpair>`. This directive is only needed for analysis of the load balancing performance. For normal use this directive is not used.

`record times <integer nftime>`

specifies the frequency `<nfsync>` in molecular dynamics steps of writing information to the timings file, with extension `tim`. The default is not to record. The information written is wall clock time used by each of the processors for the different components in the force evaluation. This directive is only needed for analysis of the wall clock time distribution. For normal use this directive is not used.

27.25 Program control options

```
load [reset] ( none | size [<real factld>] | pairs |
        (pairs [<integer ldpair>] size [<real factld>]) )
```

determines the type of dynamic load balancing performed, where the default is `none`. Load balancing option `size` is resizing boxes on a node, and `pairs` redistributes the box-box interactions over nodes. Keyword `reset` will reset the load balancing read from the restart file. The level of box resizing can be influenced with $factld$. The boxes on the busiest node are resized with a factor

$$\left(1 - factld * \frac{\frac{T_{sync}}{n_p} - t_{sync}^{min}}{t_{wall}}\right)^{\frac{1}{3}} \tag{27.1}$$

where T_{sync} is the accumulated synchronization time of all nodes, n_p is the total number of nodes, t_{sync}^{min} is the synchronization time of the busiest node, and t_{wall} is the wall clock time of the molecular dynamics step. For the combined load balancing, `ldpair` is the number of successive pair redistribution load balancing steps in which the accumulated synchronization time increases, before a resizing load balancing step will be attempted. Load balancing is only performed in molecular dynamics steps in which the pair-list is updated. The default is not to use load balancing. In most cases, effective load balancing is obtained with

```
load pairs 10 size 0.75
```

`nodes <integer npx> <integer npy> <integer npz>`

specifies the distribution of the available nodes over the three Cartesian dimensions. The default distribution is chosen such that, `<npx>`$*$`<npy>`$*$`<npz>`=`<np>` and `<npx>` $<=$ `<npy>` $<=$ `<npz>`, where `<npx>`, `<npy>` and `<npz>` are the nodes in the x, y and z dimension respectively, and `<np>` is the number of nodes allocated for the calculation. Where more than one combination of `<npx>`, `<npy>` and `<npz>`

are possible, the combination is chosen with the minimum value of `<npx>`+`<npy>`+`<npz>`. To change the default setting the following optional input option is provided.

`boxes <integer nbx> <integer nby> <integer nbz>`

specifies the distribution of boxes, where `<nbx>`, `<nby>` and `<nbz>` are the number of boxes in x, y and z direction, respectively. The molecular system is decomposed into boxes that form the smallest unit for communication of atomic data between nodes. The size of the boxes is per default set to the short-range cutoff radius. If long-range cutoff radii are used the box size is set to half the long-range cutoff radius if it is larger than the short-range cutoff. If the number of boxes in a dimension is less than the number of processors in that dimension, the number of boxes is set to the number of processors.

`extra <integer madbox>`

sets the number of additional boxes for which memory is allocated. In rare events the amount of memory set aside per node is insufficient to hold all atomic coordinates assigned to that node. This leads to execution which aborts with the message that `mwm` or `msa` is too small. Jobs may be restarted with additional space allocated by where `<madbox>` is the number of additional boxes that are allocated on each node. The default for `<madbox>` is 6. In some cases `<madbox>` can be reduced to 4 if memory usage is a concern. Values of 2 or less will almost certainly result in memory shortage.

`mwm <integer mwmreq>`

sets the maximum number of solvent molecules `<mwmreq>` per node, allowing increased memory to be allocated for solvent molecules. This option can be used if execution aborted because `mwm` was too small.

`msa <integer msareq>`

sets the maximum number of solute atoms `<msareq>` per node, allowing increased memory to be allocated for solute atoms. This option can be used if execution aborted because `msa` was too small.

`mbb <integer mbbreq>`

sets the maximum number of sub-box pairs `<mbbreq>` per node, allowing increased memory to be allocated for the sub-box pair lists. This option can be used if execution aborted because `mbbl` was too small.

`memory <integer memlim>`

sets a limit `<memlim>` in kB on the allocated amount of memory used by the molecular dynamics module. Per default all available memory is allocated. Use of this command is required for QM/MM simulations only.

Chapter 28

Analysis

The **analysis** module is used to analyze molecular trajectories generated by the **NWChem** molecular dynamics module, or partial charges generated by the **NWChem** electrostatic potential fit module. This module should not de run in parallel mode.

Directives for the **analysis** module are read from an input deck,

```
analysis
 ...
end
```

The analysis is performed as post-analysis of trajectory files through using the task directive

```
task analysis
```

or

```
task analyze
```

28.1 Reference coordinates

Most analyses require a set of reference coordinates. These coordinates are read from a **NWChem** restart file by the directive,

```
reference <string filename>
```

where filename is the name of an existing restart file. This input directive is required.

28.2 File specification

The trajectory file(s) to be analyzed are specified with

```
file <string filename> [<integer firstfile> <integer lastfile>]
```

where filename is an existing trj trajectory file. If firstfile and lastfile are specified, the specified filename needs to have a ? wild card character that will be substituted by the 3-character integer number from firstfile to lastfile, and the analysis will be performed on the series of files. For example,

```
file tr_md?.trj 3 6
```

will instruct the analysis to be performed on files *tr_md003.trj*, *tr_md004.trj*, *tr_md005.trj* and *tr_md006.trj*.

From the specified files the subset of frames to be analyzed is specified by

```
frames [<integer firstframe default 1>] <integer lastframe> \
       [<integer frequency default 1>]
```

For example, to analyze the first 100 frames from the specified trajectory files, use

```
frames 100
```

To analyze every 10-th frame between frames 200 and 400 recorded on the specified trajectory files, use

```
frames 200 400 10
```

To set hydrogen bond criteria:

```
hbond [distance [[<real rhbmin default 0.0>] <real rhbmin>]] \
      [donorangle [<real hbdmin> [ <real hbdmax default pi>]]] \
      [acceptorangle [<real hbamin> [ <real hbamax default pi>]]]
```

28.3 Selection

Analyses can be applied to a selection of solute atoms. The selection is determined by

```
select [ <integer isgm> [ <integer jsgm> ]] [ { <string atom> } ]
```

where isgm is the first segment number, jsgm is the last segment number in the selection, and {atom} is the set of atom names selected from the specified residues. By default all solute atoms are selected.

For example, all protein backbone atoms are selected by

```
select _N _CA _C
```

To select the backbone atoms in residues 20 to 80 only, use

```
select 20 80 _N _CA _C
```

This selection is reset to apply to all atoms after each file directive.

Solvent molecules within `range` nm from any selected solute atom are selected by

```
select solvent <real range>
```

After solvent selection, the solute atom selection is reset to being all selected.

Some analysis are performed on groups of atoms. These groups of atoms are define by

```
define <integer igroup> [<real rsel>] [<integer isel> [<integer jsel>]] \\
 [solvent] { <string atom> }
```

The atom string in this definitions takes the form

```
[<integer segment>:]<string atomname>
```

In the atomname a question mark may be used as a wildcard character.

28.4 Coordinate analysis

To analyze the root mean square deviation from the specified reference coordinates:

```
rmsd
```

To analyze protein ϕ-ψ and backbone hydrogen bonding:

```
ramachandran
```

To define a bond:

```
bond <integer ibond> <string atomi> <string atomj>
```

To define an angle:

```
angle <integer iangle> <string atomi> <string atomj> <string atomk>
```

To define a torsion:

```
torsion <integer itorsion> <string atomi> <string atomj> \
                       <string atomk> <string atoml>
```

The atom string in these definitions takes the form

```
<integer segment>:<string atomname> | w<integer molecule>:<string atomname>
```

for solute and solvent atom specification, respectively.

To define charge distribution in z-direction:

```
charge_distribution <integer bins>
```

Analyses on pairs of atoms in predefined groups are specified by

```
groups [<integer igroup> [<integer jgroup>]] [periodic [<integer ipbc default 3>]] \
       <string function> [<real value1> [<real value2>]] [<string filename>]
```

where *igroup* and *jgroup* are groups of atoms defined with a `define` directive. Keyword `periodic` specifies that periodic boundary conditions need to be applied in *ipbc* dimensions. The type of analysis is define by *function*, $value1$ and $value2$. If *filename* is specified, the analysis is applied to the reference coordinates and written to the specified file. If no filename is given, the analysis is applied to the specified trajectory and performed as part of the `scan` directive. Implemented analyses defined by `<string function> [<real value1> [<real value2>]]` include `distances` to calculate all atomic distances between atoms in the specified groups that lie between $value1$ and $value2$.

Coordinate histograms are specified by

```
histogram <integer idef> [<integer length>] zcoordinate <string filename>
```

where *idef* is the atom group definition number, *length* is the size of the histogram, `zcoordinate` is the currently only histogram option, and *filename* is the filname to which the histogram is written.

To perform the coordinate analysis:

```
scan [ super ] <string filename>
```

which will create, depending on the specified analysis options files filename.rms and filename.ana. After the scan directive previously defined coordinate analysis options are all reset. Optional keyword `super` specifies that frames read from the trajectory file(s) are superimposed to the reference structure before the analysis is performed.

28.5 Essential dynamics analysis

Essential dynamics analysis is performed by

```
essential
```

This can be followed by one or more

```
project <integer vector> <string filename>
```

to project the trajectory onto the specified vector. This will create files filename with extensions frm or trj, val, vec, _min.pdb and _max.pdb, with the projected trajectory, the projection value, the eigenvector, and the minimum and maximum projection structure.

For example, an essential dynamics analysis with projection onto the first vector generating files firstvec.{trj, val, vec, _min.pdb, _max.pdb} is generated by

```
essential
project 1 firstvec
```

28.6 Trajectory format conversion

To write a single frame in PDB or XYZ format, use

```
write  [super] [solute] [<integer number default 1>] <string filename>
```

To copy the selected frames from the specified trejctory file(s), onto a new file, use

```
copy  [solute] <string filename>
```

To superimpose the selected atoms for each specified frame to the reference coordinates before copying onto a new file, use

```
super [solute] <string filename>
```

The format of the new file is determined from the extension, which can be one of

amb	**AMBER** formatted trajectory file (obsolete)
arc	**DISCOVER** archive file
bam	**AMBER** unformatted trajectory file
crd	**AMBER** formatted trajectory file
dcd	**CHARMM** formatted trajectory file
esp	**gOpenMol** formatted electrostatic potential files
frm	**ecce** frames file (obsolete)
pov	**povray** input files
trj	**NWChem** trajectory file

If no extension is specified, a trj formatted file will be written.

A special tag can be added to frm and pov formatted files using

```
label <integer itag> <string tag>  [ <real rval default 1.0> ] \\
    [ <integer iatag> [ <integer jatag default iatag> ] [ <real rtag default 0.0> ] ]
    [ <string anam> ]
```

where tag number *itag* is set to the string *tag* for all atoms anam within a distance *rtag* from segments *iatag* through *jatag*. A question mark can be used in anam as a wild card character.

Atom rendering is specified using

```
render ( cpk | stick )  [ <real rval default 1.0> ] \\
    [ <integer iatag> [ <integer jatag default iatag> ] [ <real rtag default 0.0> ] ]
    [ <string anam> ]
```

for all atoms anam within a distance *rtag* from segments *iatag* through *jatag*, and a scaling factor of *rval*. A question mark can be used in anam as a wild card character.

Atom color is specified using

```
color ( <string color> | atom ) \\
    [ <integer iatag> [ <integer jatag default iatag> ] [ <real rtag default 0.0> ] ]
    [ <string anam> ]
```

for all atoms anam within a distance *rtag* from segments *iatag* through *jatag*. A question mark can be used in anam as a wild card character.

For example, to display all carbon atoms in segments 34 through 45 in green and rendered cpk in povray files can be specified with

```
render cpk 34 45 _C??
color green 34 45 _C??
```

Coordinates written to a pov file can be scaled using

```
scale <real factor>
```

A zero or negative scaling factor will scale the coordinates to lie within [-1,1] in all dimensions.

The cpk rendering in povray files can be scaled by

```
cpk <real factor default 1.0>
```

The stick rendering in povray files can be scaled by

```
stick <real factor default 1.0>
```

The initial sequence number of esp related files is defined by

```
index <integer index default 1>
```

28.7 Electrostatic potentials

A file in plt format of the electrostatic potential resulting from partial charges generated by the ESP module is generated by the command

```
esp  [ <integer spacing default 10> ] \
     [ <real rcut default 1.0> ] [periodic [<integer iper default 3>]] \
     [ <string xfile> [ <string pltfile> ] ]
```

The input coordinates are taken from the xyzq file that can be generated from a rst by the prepare module. Parameter spacing specifies the number of gridpoints per nm, rcut specifies extent of the charge grid beyond the molecule. Periodic boundaries will be used if `periodic` is specified. If `iper` is set to 2, periodic boundary conditions are applied in x and y dimensions only. If `periodic` is specified, a negative value of `rcut` will extend the grid in the periodic dimensions by abs(`rcut`), otherwise this value will be ignored in the periodic dimensions. The resulting plt formatted file pltfile can be viewed with the gOpenMol program. The resulting electrostatic potential grid is in units of kJ $\mathrm{mol}^{-1}\mathrm{e}^{-1}$. If no files are specified, only the parameters are set. This analysis applies to solute(s) only.

Chapter 29

Combined quantum and molecular mechanics

Combined or hybrid Quantum Mechanics and Molecular Mechanics (QM/MM) is a simulation methodology that is about 15 years old but in all the literature there are cautions that calibration computations must be done to validate the model for each particular chemical system studied. This is not a black box style computation and the NWChem users are advised that without calibration QM/MM may not give the appropriate results[1].

The QM/MM module in NWChem is driven by the molecular dynamics module. This module currently works for any QM method that has analytic gradients[2]. The input for this requires the definition of chemical system via the same interface that is used by the MD module (c.f. Section 27). The extensions to this interface include the definition of "Quantum" atoms and "Link" where appropriate. The QM information must be present in the traditional NWChem input deck except for the geometry[3]. The geometrical information will be constructed automatically by nwmd. For dynamics and free energy simulations the input is again identical to that for nwmd with limitations on the kinds of simulations that can be done.

The QM/MM module is invoked with the task directive where the "theory" is QMMM. The recognized operations on the QM/MM theory directive are energy, optimize, and dynamics.

```
TASK QMMM (energy | optimize | dynamics)
```

Tasks `gradient`, `saddle`, `frequencies` and `thermodynamics` are currently not available in the QM/MM mode.

The QM/MM input consists of the standard NWChem input block:

```
QMMM
  ...
END
```

[1]c.f., Singh and Kollman, J. Comp. Chem. **7**, 718 (1986); M. J. Field, P. A. Bash and M. Karplus, J. Comp. Chem. **11**, 700, (1990); J. Gao, "Methods and Applications of Combined Quantum Mechanical and Molecular Mechanical Potentials." In *Reviews in Computational Chemistry*; K. B. Lipkowitz, D. B. Boyd, Eds.; VCH Publishers: New York; Vol. 7, pp 119-185 (1995); and M. A. Thompson and G. K. Schenter, J. Phys. Chem **99** 6374 (1995)

[2]The QM/MM method will work with numerical gradients available in NWChem, but it is expected that the performance will not allow any substantive simulations

[3]Any geometry information in the traditional form will be ignored

The `QMMM` has the following the additional sub-directive that the user may specify for the particular simulation. These options currently are:

29.1 EATOMS

There is one compound input directive that must exist for the QM/MM simulation to proceed. This sets the relative zero of energy for the QM component of the system. It is not incorrect to leave this value as zero but the energetics of the QM system will likely over shadow the MM component of the system. Properties based on energy fluctuations of the system will be overly sensitive to the energy of the QM component of the system. The zero of energy for the MM system is by definition of most parameterized force fields the separated atom energy. The zero of energy for QM systems by definition of most QM methods is the vacuum. The *a priori* determination of the separated atom energy for a particular QM method is not well defined and thus leads to a number of assumptions or guess work depending upon the particular QM method being utilized. Therefor, the determination of the QM separated atom energy ("eatoms") is left to the user. The input takes the form:

```
EATOMS <real eatoms>
```

There is no default for this and the input **must** be present for a QM/MM simulation.

All other parameters that control the QM/MM simulation are set via the input to nwmd (see chapter 27).

Chapter 30

File formats

Card	Format	Description
I-1-1	a1	$ to identify the start of a fragment
I-1-2	a10	name of the fragment, the tenth character
		N: identifies beginning of a chain
		C: identifies end of a chain
		blank: identifies chain fragment
		M: identifies an integral molecule
I-2-1	i5	number of atoms in the fragment
For each atom one deck II		
II-1-1	i5	atom sequence number
II-1-2	a6	atom name
II-1-3	a5	atom type
II-1-4	a1	dynamics type
		: normal
		`D` : dummy atom
		`S` : solute interactions only
		`Q` : quantum atom
		other : intramolecular solute interactions only
II-1-5	i5	link number
		0: no link
		1: first atom in chain
		2: second atom in chain
		3 and up: other links
II-1-6	i5	environment type
		0: no special identifier
		1: planar, using improper torsion
		2: tetrahedral, using improper torsion
		3: tetrahedral, using improper torsion
		4: atom in aromatic ring
II-1-7	i5	charge group
II-1-8	i5	polarization group
II-1-9	f12.6	atomic partial charge
II-1-10	f12.6	atomic polarizability
Any number of cards in deck III to specify complete connectivity		
III-1-1	16i5	connectivity, duplication allowed

Table 30.1: The format of fragment files.

Deck	Format	Description
I-1-1	a1	$ to identify the start of a segment
I-1-2	a10	name of the segment, the tenth character
		N: identifies beginning of a chain
		C: identifies end of a chain
		blank: identifies chain fragment
		M: identifies an integral molecule
I-2-1	i5	number of atoms in the segment
I-2-2	i5	number of bonds in the segment
I-2-3	i5	number of angles in the segment
I-2-4	i5	number of proper dihedrals in the segment
I-2-5	i5	number of improper dihedrals in the segment

Table 30.2: Segment file format, table 1 of 6.

Deck	Format	Description
For each atom one deck II		
II-1-1	i5	atom sequence number
II-1-2	a6	atom name
II-1-3	a5	atom type, generic set 1
II-1-4	a1	dynamics type
		: normal
		`D` : dummy atom
		`S` : solute interactions only
		`Q` : quantum atom
		other : intramolecular solute interactions only
II-1-4	a5	atom type, generic set 2
II-1-5	a1	dynamics type
		: normal
		`D` : dummy atom
		`S` : solute interactions only
		`Q` : quantum atom
		other : intramolecular solute interactions only
II-1-6	a5	atom type, generic set 3
II-1-7	a1	dynamics type
		: normal
		`D` : dummy atom
		`S` : solute interactions only
		`Q` : quantum atom
		other : intramolecular solute interactions only
II-1-8	i5	charge group
II-1-9	i5	polarization group
II-1-10	i5	link number
II-1-11	i5	environment type
		0: no special identifier
		1: planar, using improper torsion
		2: tetrahedral, using improper torsion
		3: tetrahedral, using improper torsion
		4: atom in aromatic ring
II-2-1	f12.6	atomic partial charge in e, set 1
II-2-2	f12.6	atomic polarizability/$4\pi\epsilon_o$ in nm^3, set 1
II-2-3	f12.6	atomic partial charge in e, set 2
II-2-4	f12.6	atomic polarizability/$4\pi\epsilon_o$ in nm^3, set 2
II-2-5	f12.6	atomic partial charge in e, set 3
II-2-6	f12.6	atomic polarizability/$4\pi\epsilon_o$ in nm^3, set 3

Table 30.3: Segment file format, table 2 of 6.

Deck	Format	Description
For each bond a deck III		
III-1-1	i5	bond sequence number
III-1-2	i5	bond atom i
III-1-3	i5	bond atom j
III-1-4	i5	bond type
		0: harmonic
		1: constrained bond
III-1-5	i5	bond parameter origin
		0: from database, next card ignored
		1: from next card
III-2-1	f12.6	bond length in nm, set 1
III-2-2	e12.5	bond force constant in kJ nm^2 mol^{-1}, set 1
III-2-3	f12.6	bond length in nm, set 2
III-2-4	e12.5	bond force constant in kJ nm^2 mol^{-1}, set 2
III-2-5	f12.6	bond length in nm, set 3
III-2-6	e12.5	bond force constant in kJ nm^2 mol^{-1}, set 3

Table 30.4: Segment file format, table 3 of 6.

Deck	Format	Description
For each angle a deck IV		
IV-1-1	i5	angle sequence number
IV-1-2	i5	angle atom i
IV-1-3	i5	angle atom j
IV-1-4	i5	angle atom k
IV-1-5	i5	angle type
		0: harmonic
IV-1-6	i5	angle parameter origin
		0: from database, next card ignored
		1: from next card
IV-2-1	f10.6	angle in radians, set 1
IV-2-2	e12.5	angle force constant in kJ mol^{-1}, set 1
IV-2-3	f10.6	angle in radians, set 2
IV-2-4	e12.5	angle force constant in kJ mol^{-1}, set 2
IV-2-5	f10.6	angle in radians, set 3
IV-2-6	e12.5	angle force constant in kJ mol^{-1}, set 3

Table 30.5: Segment file format, table 4 of 6.

Deck	Format	Description
For each proper dihedral a deck V		
V-1-1	i5	proper dihedral sequence number
V-1-2	i5	proper dihedral atom i
V-1-3	i5	proper dihedral atom j
V-1-4	i5	proper dihedral atom k
V-1-5	i5	proper dihedral atom l
V-1-6	i5	proper dihedral type
		0: $C\cos(m\phi - \delta)$
V-1-7	i5	proper dihedral parameter origin
		0: from database, next card ignored
		1: from next card
V-2-1	i3	multiplicity, set 1
V-2-2	f10.6	proper dihedral in radians, set 1
V-2-3	e12.5	proper dihedral force constant in kJ mol^{-1}, set 1
V-2-4	i3	multiplicity, set 2
V-2-5	f10.6	proper dihedral in radians, set 2
V-2-6	e12.5	proper dihedral force constant in kJ mol^{-1}, set 2
V-2-7	i3	multiplicity, set 3
V-2-8	f10.6	proper dihedral in radians, set 3
V-2-9	e12.5	proper dihedral force constant in kJ mol^{-1}, set 3

Table 30.6: Segment file format, table 5 of 6.

Deck	Format	Description
For each improper dihedral a deck VI		
VI-1-1	i5	improper dihedral sequence number
VI-1-2	i5	improper dihedral atom i
VI-1-3	i5	improper dihedral atom j
VI-1-4	i5	improper dihedral atom k
VI-1-5	i5	improper dihedral atom l
VI-1-6	i5	improper dihedral type
		0: harmonic
VI-1-7	i5	improper dihedral parameter origin
		0: from database, next card ignored
		1: from next card
VI-2-1	3x,f10.6	improper dihedral in radians, set 1
VI-2-2	e12.5	improper dihedral force constant in kJ mol^{-1}, set 1
VI-2-3	3x,f10.6	improper dihedral in radians, set 2
VI-2-4	e12.5	improper dihedral force constant in kJ mol^{-1}, set 2
VI-2-5	3x,f10.6	improper dihedral in radians, set 3
VI-2-6	e12.5	improper dihedral force constant in kJ mol^{-1}, set 3

Table 30.7: Segment file format, table 6 of 6.

Card	Format	Description
I-1-1	a1	$ to identify the start of a sequence
I-1-2	a10	name of the sequence
Any number of cards 1 and 2 in deck II to specify the system		
II-1-1	i5	segment number
II-1-2	a10	segment name, last character will be determined from chain
II-2-1	a	`break` to identify a break in the molecule chain
II-2-1	a	`molecule` to identify the end of a solute molecule
II-2-1	a	`fraction` to identify the end of a solute fraction
II-2-1	a5	`link` to specify a link
II-2-2	i5	segment number of first link atom
II-2-3	a4	name of first link atom
II-2-4	i5	segment number of second link atom
II-2-5	a4	name of second link atom
II-2-1	a	`solvent` to identify solvent definition on next card
II-2-1	a	`stop` to identify the end of the sequence
II-2-1	a6	`repeat` to repeat next *ncard* cards *ncount* times
II-2-2	i5	number of cards to repeat (*ncards*)
II-2-3	i5	number of times to repeat cards (*ncount*)
Any number of cards in deck II to specify the system		

Table 30.8: Sequence file format.

Card	Format	Description
I-1-1	a6	keyword `header`
I-2-1	i10	number of atoms per solvent molecule
I-2-2	i10	number of solute atoms
I-2-3	i10	number of solute bonds
I-2-4	i10	number of solvent bonds
For each atoms per solvent molecule one card I-3		
I-3-1	a5	solvent name
I-3-2	a5	atom name
I-3-3	6x,i10	solvent atom counter
For each solute atom one card I-4		
I-4-1	a5	segment name
I-4-2	a5	atom name
I-4-3	i6	segment number
I-4-4	i10	solute atom counter
For each solvent bond one card I-5		
I-5-1	i8	atom index i for bond between i and j
I-5-2	i8	atom index j for bond between i and j
For each solute bond one card I-6		
I-6-1	i8	atom index i for bond between i and j
I-6-2	i8	atom index j for bond between i and j
For each frame one deck II		
II-1-1	a5	keyword `frame`
II-2-1	f12.6	time of frame in ps
II-2-2	f12.6	temperature of frame in K
II-2-3	e12.5	pressure of frame in Pa
II-3-1	f12.6	box dimension x
II-3-2	36x,f12.6	box dimension y
II-3-3	36x,f12.6	box dimension z
II-4-1	l1	logical lxw for solvent coordinates
II-4-2	l1	logical lvw for solvent velocities
II-4-3	l1	logical lxs for solute coordinates
II-4-4	l1	logical lvs for solute velocities
II-4-5	i10	number of solvent molecules
II-4-6	i10	number of solvent atoms
II-4-7	i10	number of solute atoms
For each solvent molecule one card II-5 for each atom		
II-5-1	3f8.3	solvent atom coordinates, if lxw or lvw
II-5-4	3f8.3	solvent atom velocities, if lvw
For each solute atom one card II-6 for each atom		
II-6-1	3f8.3	solute atom coordinates, if lxs or lvs
II-6-4	3f8.3	solute atom velocities, if lvs

Table 30.9: Trajectory file format.

Card	Format	Description
For each step in λ one deck I		
I-1-1	i7	number $nderiv$ of data summed in derivative decomposition array $deriv$
I-1-2	i7	length $ndata$ of total derivative array drf
I-1-3	f12.6	current value of λ
I-1-4	f12.6	step size of λ
I-2-1	4e12.12	derivative decomposition array $deriv(1:24)$
I-3-1	4e12.12	total derivative array $dfr(1:nda)$
I-4-1	i10	size of ensemble at current λ
I-4-2	e20.12	average free energy derivative at current λ
I-4-3	e20.12	average exponent reverse perturbation energy at current λ
I-4-4	e20.12	average exponent forward perturbation energy at current λ

Table 30.10: Free energy file format.

Card	Format	Description
For each analyzed time step one card I-1		
I-1-1	f12.6	time in ps
I-1-2	f12.6	total rms deviation of the selected atoms before superimposition
I-1-3	f12.6	total rms deviation of the selected atoms after superimposition
II-1-1	a8	keyword `analysis`
For each solute atom one card II-2		
II-2-1	a5	segment name
II-2-2	a5	atom name
II-2-3	i6	segment number
II-2-4	i10	atom number
II-2-5	i5	selected if 1
II-2-6	f12.6	average atom rms deviation after superimposition
III-1-1	a8	keyword `analysis`
For each solute segment one card III-2		
III-2-1	a5	segment name
III-2-2	i6	segment number
III-2-3	f12.6	average segment rms deviation after superimposition

Table 30.11: Root mean square deviation file format.

Card	Format	Description
I-1-1	i7	number $nprop$ of recorded properties
I-1-2	1x,2a10	date and time
For each of the $nprop$ properties one card I-2		
I-2-1	a50	description of recorded property
For each recorded step one deck II		
II-1-1	4(1pe12.5)	value of property

Table 30.12: Property file format.

Chapter 31

Pseudopotential Plane-Wave DFT (PSPW)

A pseudopotential plane-wave (PSPW) module, which can perform Car-Parrinello simulations, is being implemented into the NWChem program package. This module complements the capabilities of the more traditional Gaussian function based approaches by including code which allows for the calculation of density functional theory total energies and forces with the technology based on plane-wave basis sets and pseudopotentials. Consistent with NWChem's philosophy this module is able to run on a variety of architectures, including parallel supercomputers.

The advantage of a PSPW method is that it has been shown to have an accuracy close to chemical accuracy for many applications, yet is still fast enough to treat systems containing hundreds of atoms. Another significant advantage is its ability to simulate dynamics on a ground state potential surface directly at run-time. This method's efficiency and accuracy make it a desirable first principles method of simulation in the study of complex molecular, liquid, and solid state systems. Applications for this first principles method include the calculation of free energies, search for global minima, explicit simulation of solvated molecules, and simulations of complex vibrational modes that cannot be described within the harmonic approximation.

Section 31.1 describes the tasks contained within the PSPW module. The RTDB entries and datafiles used by the PSPW module are described in section 31.2. Car-Parrinello output data files are described in section 31.2.9, and the minimization and Car-Parrinello algorithms are described in sections 31.3-31.4. Examples of how to setup and run a steepest descent simulation and a Car-Parrinello simulation are presented in sections 31.5-31.6. Finally in section 31.7 the capabilities and limitations of the PSPW module are discussed.

If you are a first time user of this module it is recommended that you skip the next five sections and proceed directly to the tutorials in sections 31.5-31.6.

31.1 PSPW Tasks

All input to the PSPW Tasks is contained within the compound PSPW block,

```
PSPW
   ...
END
```

To perform an actual calculation a TASK PSPW directive is used. The TASK cg_pspw directive may also be used (see below). The format for the TASK PSPW directive is the TASK directive followed by the PSPW string, and after that an <operation> string is required. The TASK PSPW directive for PSPW calculations is of the following form:

```
TASK PSPW [steepest_descent           ||
           conjugate_gradient         ||
           Car-Parrinello             ||
           psp_formatter              ||
           wavefunction_initializer   ||
           v_wavefunction_initializer ||
           wavefunction_expander      ||
           psp_generator              ||
           (no default)]
```

Currently available tasks are listed. Note that unlike most NWChem modules, the PSPW module does not contain an energy operation. This means that there is no default operation and hence an operation must always be specified.

PSPW tasks are of two types. The first type of task is used to setup or change needed data files for a PSPW simulation. Tasks of this type are psp_formatter, wavefunction_initializer, v_wavefunction_initializer, wavefunction_expander, and psp_generator. The second type of task is used to actually run a PSPW simulation. Tasks of this type are steepest_descent, and Car-Parrinello. The following subsections describe the input to these tasks.

The PSPW module is also interfaced to driver, stepper, and freqency task operations through the cg_pspw theory directive, see section 5.10.1. The PSPW conjugate_gradient sub-module is used in this interface. Specifically the following task operations are interafaced using the cg_pspw theory directive.

```
TASK cg_pspw energy      #equivalent to TASK PSPW conjugate_gradient
TASK cg_pspw gradient
TASK cg_pspw optimize
TASK cg_pspw saddle
TASK cg_pspw freqencies
TASK cg_pspw vib
```

31.1.1 STEEPEST_DESCENT

The steepest_descent task is used to optimize the one-electron orbitals with respect to the total energy. In addition it can also be used to optimize geometries. This method is meant to be used for coarse optimization of the one-electron orbitals. A detailed description of the this method is described in section 31.3.1

Input to the steepest_descent simulation is contained within the steepest_descent sub-block.

```
PSPW
  ...
  STEEPEST_DESCENT
     ...
  END
  ...
END
```

To run a steepest_descent calculation the following directive is used:

```
TASK PSPW steepest_descent
```

The steepest_descent sub-block contains a great deal of input, including pointers to data, as well as parameter input. Listed below is the format of a STEEPEST_DESCENT sub-block.

```
PSPW
...
   STEEPEST_DESCENT
      CELL_NAME: <string cell_name>
      [GEOMETRY_OPTIMIZE]
      (FORMATTED_FILENAME: <string formatted_name> \
       ...)
      INPUT_WAVEFUNCTION_FILENAME:  <string input_wavefunctions  default input_movecs>
      OUTPUT_WAVEFUNCTION_FILENAME: <string output_wavefunctions default input_movecs>
      FAKE_MASS: <real fake_mass default 400000.0>
      TIME_STEP: <real time_step default 5.8>
      LOOP: <integer inner_iteration outer_iteration default 10 1>
      TOLERANCES: <real tole tolc tolr default 1.0d-9 1.0d-9 1.0d-4>
      ENERGY_CUTOFF:       <real ecut default (see input desciption)>
      WAVEFUNCTION_CUTOFF: <real wcut default (see input description)>
      EWALD_NCUT: <integer ncut default 1>
      EWALD_RCUT: <real rcut default (see input description)>
      EXCHANGE_CORRELATION: (Vosko || PBE96  default Vosko)
      [MULLIKEN]

   END
...

END
```

The following list describes the input for the STEEPEST_DESCENT sub-block.

- <cell_name> - name that points to the the simulation_cell named <cell_name>. See section 31.2.3.
- GEOMETRY_OPTIMIZE - optional keyword which if specified turns on geometry optimization.
- <formatted_name> - name that points to a formatted_pseudopotential file. A file must be specified for each kind of ion in the simulation.
- <input_wavefunctions> - name that points to a file containing one-electron orbitals
- <output_wavefunctions> - name that will point to file containing the one-electron orbitals at the end of the run.
- <fake_mass> - value for the electronic fake mass (μ).
- <time_step> - value for the time step (Δt).
- <inner_iteration> - number of iterations between the printing out of energies and tolerances
- <outer_iteration> - number of outer iterations

- <tole> - value for the energy tolerance.
- <tolc> - value for the one-electron orbital tolerance.
- <tolr> - value for the ion position tolerance.
- <ecut> - value for the cutoff energy used to define the density. Default is set to be the maximum value that will fit within the simulation_cell <cell_name>.
- <wcut> - value for the cutoff energy used to define the one-electron orbitals. Default is set to be the maximum value that will fit within the simulation_cell <cell_name>.
- <ncut> - value for the number of unit cells to sum over (in each direction) for the real space part of the Ewald summation. Note Ewald summation is only used if the simulation_cell is periodic.
- <rcut> - value for the cutoff radius used in the Ewald summation. Note Ewald summation is only used if the simulation_cell is periodic.
  Default set to be $\frac{MIN(|\vec{a_i}|)}{\pi}, i = 1, 2, 3$.
- (Vosko || PBE96) - Choose between Vosko et al's LDA parameterization or the Perdew, Burke, and Erzherhoff GGA functional.
- MULLIKEN - optional keyword which if specified causes a Mulliken anaysis to be performed at the end of the simulation. For this option to work angular momentum weights for each kind of atom have to be entered into the RTDB using the PSPW ANALYSIS sub-block (see section 31.2.4.

31.1.2 CONJUGATE_GRADIENT

The conjugate_gradient task is used to optimize the one-electron orbitals with respect to the total energy. This method should be used for finer optimization. It is better to use the steepest_descent task when intitially optimizing the one-electron orbitals. A detailed description of the this method is described in section 31.3.2.

Input to the conjugate_gradient simulation is contained within the conjugate_gradient sub-block.

```
PSPW
  ...
  CONJUGATE_GRADIENT
     ...
  END
  ...
END
```

To run a conjugate_gradient calculation the following directive is used:

```
TASK PSPW conjugate_gradient
```

The conjugate_gradient sub-block contains a great deal of input, including pointers to data, as well as parameter input. Listed below is the format of a conjugate_gradient sub-block.

```
PSPW
...
   CONJUGATE_GRADIENT
      CELL_NAME: <string cell_name>
```

```
      (FORMATTED_FILENAME: <string formatted_name> \
       ...)
      INPUT_WAVEFUNCTION_FILENAME:  <string input_wavefunctions  default input_movecs>
      OUTPUT_WAVEFUNCTION_FILENAME: <string output_wavefunctions default input_movecs>
      [FAKE_MASS: <real fake_mass default 400000.0>]
      [TIME_STEP: <real time_step default 5.8>]
      LOOP: <integer inner_iteration outer_iteration default 10 1>
      TOLERANCES: <real tole tolc default 1.0e-9 1.0e-9>
      ENERGY_CUTOFF:       <real ecut default (see input description)>
      WAVEFUNCTION_CUTOFF: <real wcut default (see input description)>
      EWALD_NCUT: <integer ncut default 1>]
      EWALD_RCUT: <real rcut default (see input description)>
      EXCHANGE_CORRELATION: (Vosko || PBE96  default Vosko)
      [MULLIKEN]

   END
...

END
```

The following list describes the input for the CONJUGATE_GRADIENT sub-block.

- <cell_name> - name that points to the the simulation_cell named <cell_name>. See section 31.2.3.
- <formatted_name> - name that points to a formatted_pseudopotential file. A file must be specified for each kind of ion in the simulation.
- <input_wavefunctions> - name that points to a file containing one-electron orbitals
- <output_wavefunctions> - name that will point to file containing the one-electron orbitals at the end of the run.
- <fake_mass> - value for the electronic fake mass (μ). This parameter is not presently used in a conjugate gradient simulation
- <time_step> - value for the time step (Δt). This parameter is not presently used in a conjugate gradient simulation.
- <inner_iteration> - number of iterations between the printing out of energies and tolerances
- <outer_iteration> - number of outer iterations
- <tole> - value for the energy tolerance.
- <tolc> - value for the one-electron orbital tolerance.
- <ecut> - value for the cutoff energy used to define the density. Default is set to be the maximum value that will fit within the simulation_cell <cell_name>.
- <wcut> - value for the cutoff energy used to define the one-electron orbitals. Default is set to be the maximum value that will fix within the simulation_cell <cell_name>.
- <ncut> - value for the number of unit cells to sum over (in each direction) for the real space part of the Ewald summation. Note Ewald summation is only used if the simulation_cell is periodic.

- <rcut> - value for the cutoff radius used in the Ewald summation. Note Ewald summation is only used if the simulation cell is periodic.
  Default set to be $\frac{MIN(|\vec{a_i}|)}{\pi}, i = 1, 2, 3$.
- (Vosko || PBE96) - Choose between Vosko et al's LDA parameterization or the Perdew, Burke, and Erzherhoff GGA functional.
- MULLIKEN - optional keyword which if specified causes a Mulliken anaysis to be performed at the end of the simulation. For this option to work angular momentum weights for each kind of atom have to be entered into the RTDB using the PSPW ANALYSIS sub-block (see section 31.2.4.

31.1.3 Car-Parrinello

The Car-Parrinello task is used to perform ab initio molecular dynamics using the scheme developed by Car and Parrinello. In this unified ab initio molecular dynamics scheme the motion of the ion cores is coupled to a fictitious motion for the Kohn-Sham orbitals of density functional theory. Constant energy or constant temperature simulations can be performed. A detailed description of this method is described in section 31.4.

Input to the Car-Parrinello simulation is contained within the Car-Parrinello sub-block.

```
PSPW
  ...
  Car-Parrinello
     ...
  END
  ...
END
```

To run a Car-Parrinello calculation the following directive is used:

```
TASK PSPW Car-Parrinello
```

The Car-Parrinello sub-block contains a great deal of input, including pointers to data, as well as parameter input. Listed below is the format of a Car-Parrinello sub-block.

```
PSPW
...
   Car-Parrinello
      CELL_NAME: <string cell_name>
      (FORMATTED_FILENAME: <string formatted_name> \
       ...)
      INPUT_WAVEFUNCTION_FILENAME:    <string input_wavefunctions    default input_movecs>
      OUTPUT_WAVEFUNCTION_FILENAME:   <string output_wavefunctions   default input_movecs>
      INPUT_V_WAVEFUNCTION_FILENAME:  <string input_v_wavefunctions  default input_vmovecs>
      OUTPUT_V_WAVEFUNCTION_FILENAME: <string output_v_wavefunctions default input_vmovecs>
      FAKE_MASS: <real fake_mass default default 1000.0>
      TIME_STEP: <real time_step default 5.0>
      LOOP: <integer inner_iteration outer_iteration default 10 1>
      SCALING: <real scale_c scale_r default 1.0 1.0>
      ENERGY_CUTOFF:       <real ecut default (see input description)>
```

```
      WAVEFUNCTION_CUTOFF: <real wcut default (see input description)>
      EWALD_NCUT: <integer ncut default 1>
      EWALD_RCUT: <real rcut    default (see input description)>
      EXCHANGE_CORRELATION: (Vosko || PBE96  default Vosko)
      [Nose-Hoover: <real Period_electron Temperature_electrion Period_ion Temperature_ion
                          default 100.0 298.15 100.0 298.15>]
      XYZ_FILENAME: <string xyz_filename default XYZ>
      EMOTION_FILENAME: <string emotion_filename default EMOTION>
      HMOTION_FILENAME: <string hmotion_filename default HMOTION>
      OMOTION_FILENAME: <string omotion_filename default OMOTION>
      EIGMOTION_FILENAME: <string eigmotion_filename default EIGMOTION>
      ION_MOTION_FILENAME: <string ion_motion_filename default MOTION>

   END
...

END
```

The following list describes the input for the Car-Parrinello sub-block.

- <cell_name> - name that points to the the simulation_cell named <cell_name>. See section 31.2.3.
- <formatted_name> - name that points to a formatted_pseudopotential file. A file must be specified for each kind of ion in the simulation.
- <input_wavefunctions> - name that points to a file containing one-electron orbitals
- <output_wavefunctions> - name that will point to file containing the one-electron orbitals at the end of the run.
- <input_v_wavefunctions> - name that points to a file containing one-electron orbital velocities.
- <output_v_wavefunctions> - name that will point to file containing the one-electron orbital velocities at the end of the run.
- <fake_mass> - value for the electronic fake mass (μ).
- <time_step> - value for the Verlet integration time step (Δt).
- <inner_iteration> - number of iterations between the printing out of energies.
- <outer_iteration> - number of outer iterations
- <scale_c> - value for the initial velocity scaling of the one-electron orbital velocities.
- <scale_r> - value for the initial velocity scaling of the ion velocities.
- <ecut> - value for the cutoff energy used to define the density. Default is set to be the maximum value that will fit within the simulation_cell <cell_name>.
- <wcut> - value for the cutoff energy used to define the one-electron orbitals. Default is set to be the maximum value that will fit within the simulation_cell <cell_name>.
- <ncut> - value for the number of unit cells to sum over (in each direction) for the real space part of the Ewald summation. Note Ewald summation is only used if the simulation_cell is periodic.

- <rcut> - value for the cutoff radius used in the Ewald summation. Note Ewald summation is only used if the simulation_cell is periodic.
  Default set to be $\frac{MIN(|\vec{a_i}|)}{\pi}, i = 1, 2, 3$.
- (Vosko || PBE96) - Choose between Vosko et al's LDA parameterization or the Perdew, Burke, and Erzherhoff GGA functional.
- Nose-Hoover: - optional subblock which if specified causes the simulation to perform Nose-Hoover dynamics. If this subblock is not specified the simulation performs constant energy dyanmics. See section 31.4.2 for a description of the parameters.
  - <Period_electron> $\equiv P_{electron}$ - estimated period for fictitious electron thermostat.
  - <Temperature_electron> $\equiv T_{electron}$ - temperature for fictitious electron motion
  - <Period_ion> $\equiv P_{ion}$ - estimated period for ionic thermostat
  - <Temperature_ion> $\equiv T_{ion}$ - temperature for ion motion
- <xyz_filename> - name that points to the XYZ motion file generated
- <emotion_filename> - name that points to the emotion motion file generated. See section 31.2.9 for a description of the datafile.
- <hmotion_filename> - name that points to the hmotion motion file generated. See section 31.2.9 for a description of the datafile.
- <eigmotion_filename> - name that points to the eigmotion motion file generated. See section 31.2.9 for a description of the datafile.
- <ion_motion_filename> - name that points to the ion_motion motion file generated. See section 31.2.9 for a description of the datafile.
- MULLIKEN - optional keyword which if specified causes an omotion motion file to be created. For this option to work angular momentum weights for each kind of atom have to be entered into the RTDB using the PSPW ANALYSIS sub-block (see section 31.2.4.
- <omotion_filename> - name that points to the omotion motion file generated. See section 31.2.9 for a description of the datafile.

31.1.4 PSP_FORMATTER

The psp_formatter task takes a non-separable pseudopotential defined in one-dimension real-space in a one-dimensional psp datafile and does two things to it. First it puts it into the semi-local form suggested by Kleinman and Bylander and then it expands in in a periodic Fourier series defined by the simulation cell in the RTDB.

Input to the PSP_FORMATTER task is contained within the PSP_FORMATTER sub-block.

```
PSPW
  ...
  PSP_FORMATTER
     ...
  END
  ...
END
```

To run a PSP_FORMATTER calculation the following directive is used:

```
TASK PSPW PSP_FORMATTER
```

Listed below is the format of a PSP_FORMATTER sub-block.

```
PSPW
...
   PSP_FORMATTER
      CELL_NAME:          <string cell_name>
      PSP_FILENAME:       <string psp_name>
      FORMATTED_FILENAME: <string formatted_name>
      LOCP: (s||p||d||f||g default (see input description))
      LMAX: (s||p||d||f||g default (see input description))
   end
...
END
```

The following list describes the input for the PSP_FORMATTER sub-block.

- <cell_name> - name that points to the simulation_cell named <cell_name>. See section 31.2.3.
- <psp_name> - name that points to a one-dimensional pseudopotential datafile.
- <formatted_name> - name that points to a formatted_pseudopotential datafile.
- LMAX (s||p||d||f||g) - used to specify the maximum number of angular potentials to use. The default is set to the maximum angular momentum in <psp_name>.
- LOCP (s||p||d||f||g) - used to specify which angular potential is to used as the local potential. The default is is set to LMAX.

31.1.5 WAVEFUNCTION_INTITIALIZER

The wavefunction_initializer task is used to generate an initial wavefunction datafile. Input to the WAVEFUNCTION_INITIALIZER task is contained within the WAVEFUNCTION_INITIALIZER sub-block.

```
PSPW
  ...
  WAVEFUNCTION_INITIALIZER
     ...
  END
  ...
END
```

To run a WAVEFUNCTION_INITIALIZER calculation the following directive is used:

```
TASK PSPW WAVEFUNCTION_INITIALIZER
```

Listed below is the format of a WAVEFUNCTION_INITIALIZER sub-block.

```
PSPW
...
   WAVEFUNCTION_INITIALIZER
     CELL_NAME: <string cell_name>
     WAVEFUNCTION_FILENAME: <string wavefunction_name default input_movecs>
     (RESTRICTED||UNRESTRICTED)
     if (RESTRICTED)
        RESTRICTED_ELECTRONS: <integer restricted electrons>
     if (UNRESTRICTED)
        UP_ELECTRONS: <integer up_electrons>
        DOWN_ELECTRONS: <integer down_electrons>
   END
...
END
```

The following list describes the input for the WAVEFUNCTION_INITIALIZER sub-block.

- <cell_name> - name that points to the simulation_cell named <cell_name>. See section 31.2.3.
- <wavefunction_name> - name that will point to a wavefunction file.
- RESTRICTED - keyword specifying that the calculation is restricted.
- UNRESTRICTED - keyword specifying that the calculation is unrestricted.
- <restricted_electrons> - number of restricted electrons. Not used if an UNRESTRICTED calculation.
- <up_electrons> - number of spin-up electrons. Not used if a RESTRICTED calculation.
- <down_electrons> - number of spin-down electrons. Not used if a RESTRICTED calculation.

Old Style Input (version 3.3) to WAVEFUNCTION_INTITIALIZER

For backwards compatibility, the input to the WAVEFUNCTION_INITIALIZER sub-block can also be of the form

```
PSPW
...
   WAVEFUNCTION_INITIALIZER
     CELL_NAME: <string cell_name>
     WAVEFUNCTION_FILENAME: <string wavefunction_name default input_movecs>
     (RESTRICTED||UNRESTRICTED)

     [UP_FILLING: <integer up_filling>
        [0 0 0   0]
        {<integer kx ky kz> (-2||-1||1||2)}]
     [DOWN_FILLING: <integer down_filling>
        [0 0 0   0]
        {<integer kx ky kz> (-2||-1||1||2)}]
   END
...
END
```

where

- <cell_name> - name that points to the simulation_cell named <cell_name>. See section 31.2.3.
- <wavefunction_name> - name that will point to a wavefunction file.
- RESTRICTED - keyword specifying that the calculation is restricted.
- UNRESTRICTED - keyword specifying that the calculation is unrestricted.
- <up_filling> - number of restricted molecular orbitals if RESTRICTED and number of spin-up molecular orbitals if UNRESTRICTED.
- <down_filling> - number of spin-down moleclar orbitals if UNRESTRICTED. Not used if a RESTRICTED calculation.
- <kx ky kz> - specifies which planewave is to be filled.

The values for the planewave $(-2||-1||1||2)$ are used to represent whether the specified planewave is a cosine or a sine function, in addition random noise can be added to these base functions. That is $+1$ represents a cosine function, and -1 represents a sine function. The $+2$ and -2 values are used to represent a cosine function with random components added and a sine function with random components added respectively.

31.1.6 V_WAVEFUNCTION_INITIALIZER

The v_wavefunction_initializer task is used to generate an initial velocity wavefunction datafile. Input to the V_WAVEFUNCTION_INITIALIZER task is contained within the V_WAVEFUNCTION_INITIALIZER sub-block.

```
PSPW
  ...
  V_WAVEFUNCTION_INITIALIZER
     ...
  END
  ...
END
```

To run a V_WAVEFUNCTION_INITIALIZER calculation the following directive is used:

```
TASK PSPW WAVEFUNCTION_INITIALIZER
```

Listed below is the format of a V_WAVEFUNCTION_INITIALIZER sub-block.

```
PSPW
...
   V_WAVEFUNCTION_INITIALIZER
     V_WAVEFUNCTION_FILENAME: <string v_wavefunction_name default input_vmovecs>
     CELL_NAME: <string cell_name>
     (RESTRICTED||UNRESTRICTED)
     UP_FILLING: <integer up_filling>
     DOWN_FILLING: <integer down_filling>
   END
...
END
```

The following list describes the input for the V_WAVEFUNCTION_INITIALIZER sub-block.

- <cell_name> - name that points to the simulation_cell named <cell_name>. See section 31.2.3.
- <wavefunction_name> - name that will point to a velocity wavefunction file.
- RESTRICTED - keyword specifying that the calculation is restricted.
- UNRESTRICTED - keyword specifying that the calculation is unrestricted.
- <up_filling> - number of restricted velocity molecular orbitals if RESTRICTED and number of spin-up velocity molecular orbitals if UNRESTRICTED.
- <down_filling> - number of spin-down velocity moleclar orbitals if UNRESTRICTED. Not used if a RESTRICTED calculation.

31.1.7 WAVEFUNCTION_EXPANDER

The v_wavefunction_initializer task is used to convert a new wavefunction file that spans a larger grid space from an old wavefunction file. Input to the WAVEFUNCTION_EXPANDER task is contained within the WAVEFUNCTION_EXPANDER sub-block.

```
PSPW
  ...
  WAVEFUNCTION_EXPANDER
     ...
  END
  ...
END
```

To run a WAVEFUNCTION_EXPANDER calculation the following directive is used:

```
TASK PSPW WAVEFUNCTION_EXPANDER
```

Listed below is the format of a WAVEFUNCTION_EXPANDER sub-block.

```
PSPW
...
   WAVEFUNCTION_EXPANDER
     OLD_WAVEFUNCTION_FILENAME: <string old_wavefunction_name default input_movecs>
     NEW_WAVEFUNCTION_FILENAME: <string new_wavefunction_name default input_movecs>
     NEW_NGRID: <integer na1 na2 na3>

   END
...
END
```

The following list describes the input for the WAVEFUNCTION_EXPANDER sub-block.

- <old_wavefunction_name> - name that points to a wavefunction file.
- <new_wavefunction_name> - name that will point to a wavefunction file.
- <na1 na2 na3> - number of grid points in each dimension for the new wavefunction file.

31.1.8 PSP_GENERATOR

A one-dimensional pseudopotential code has been integrated into NWChem. This code allows the user to modify and develop pseudopotentials. Currently, only the Hamann and Troullier-Martins norm-conserving pseudopotentials can be generated. This file can then be used by the pseudopotential_formatter task to generate a formatted pseudopotential file. Input to the PSP_GENERATOR task is contained within the PSP_GENERATOR sub-block.

```
PSPW
  ...
  PSP_GENERATOR
     ...
  END
  ...
END
```

To run a PSP_GENERATOR calculation the following directive is used:

```
TASK PSPW PSP_GENERATOR
```

Listed below is the format of a PSP_GENERATOR sub-block.

```
PSPW
...
   PSP_GENERATOR
      PSEUDOPOTENTIAL_FILENAME: <string psp_name>
      ELEMENT: <string element>
      CHARGE: <real charge>
      MASS_NUMBER: <real mass_number>
      ATOMIC_FILLING: <integer ncore nvalence>
      ( (1||2||...) (s||p||d||f||...) <real filling> \
         ...)

      [CUTOFF: <integer lmax>
         ( (s||p||d||f||g) <real rcut>\
         ...)
      ]
      PSEUDOPOTENTIAL_TYPE: (TROULLIER-MARTINS || HAMANN default HAMANN)
      SOLVER_TYPE: (PAULI || SCRHODINGER default PAULI)
      EXCHANGE_TYPE: (dirac || PBE96 default DIRAC)
      CORRELATION_TYPE: (VOSKO || PBE96 default VOSKO)
      [SEMICORE_RADIUS: <real rcore>]

   end
...
END
```

The following list describes the input for the PSP_GENERATOR sub-block.

- <psp_name> - name that points to a.

- <element> - Atomic symbol.
- <charge> - charge of the atom
- <mass> - mass number for the atom
- <ncore> - number of core states
- <nvalence> - number of valence states.
- ATOMIC_FILLING:.....(see below)
- <filling> - occupation of atomic state
- CUTOFF:....(see below)
- <rcore> - value for the semicore radius (see below)

ATOMIC_FILLING Block

This required block is used to define the reference atom which is used to define the pseudopotential. After the ATOMIC_FILLING: <ncore> <nvalence> line The core states are listed (one per line), and then the valence states are listed (one per line). Each state contains two integer and a value. The first integer specifies the radial quantum number, n, The second integer specifies the angular momentum quantum number, l, and the third value specifies the occupation of the state.

For example to define a pseudopotential for the Neon atom in the $1s^2 2s^2 2p^6$ state could have the block

```
ATOMIC_FILLING: 1 2
       1  s  2.0   #core state    - 1s^2
       2  s  2.0   #valence state - 2s^2
       2  p  6.0   #valence state - 2p^6
```

for a pseudopotential with a $2s$ and $2p$ valence electrons or the block

```
ATOMIC_FILLING: 3 0
       1  s  2.0    #core state
       2  s  2.0    #core state
       2  p  6.0    #core state
```

could be used for a pseudopotential with no valence electrons.

CUTOFF Block

This optional block specifies the cutoff distances used to match the all-electron atom to the pseudopotential atom. For Hamann pseudopotentials $r_{cut}(l)$ defines the distance where the all-electron potential is matched to the pseudopotential, and for Troullier-Martins pseudopotentials $r_{cut}(l)$ defines the distance where the all-electron orbital is matched to the pseudowavefunctions. Thus the definition of the radii depends on the type of pseudopotential. The cutoff radii used in Hamann pseudopotentials will be smaller than the cufoff raddi used in Troullier-Martins pseudopotentials.

For example to define a softened Hamann pseudopotential for Carbon would be

```
ATOMIC_FILLING: 1 2
  1  s  2.0
  2  s  2.0
  2  p  2.0
CUTOFF: 2
  s  0.8
  p  0.85
  d  0.85
```

while a similarly softened Troullier-Marting pseudopotential for Carbon would be

```
ATOMIC_FILLING: 1 2
  1  s  2.0
  2  s  2.0
  2  p  2.0
CUTOFF: 2
  s  1.200
  p  1.275
  d  1.275
```

SEMICORE_RADIUS Option

Specifying the SEMICORE_RADIUS option turns on the semicore correction approximation proposed by Louie et al (S.G. Louie, S. Froyen, and M.L. Cohen, Phys. Rev. B, **26**, 1738, (1982)). This approximation is known to dramatically improve results for systems containing alkali and transition metal atoms.

The implmentation in the PSPW module defines the semi-core denisty, $\rho_{semicore}$ in terms of the core density, ρ_{core}, by using the sixth-order polynomial

$$\rho_{semicore}(r) = \begin{cases} \rho_{core} & \text{if } r \geq r_{semicore} \\ c_0 + c_3 r^3 + c_4 r^4 + c_5 r^5 + c_6 r^6 & \text{if } r < r_{semicore} \end{cases} \tag{31.1}$$

This expansion was suggested by Fuchs and Scheffler (M. Fuchs, and M. Scheffler, Comp. Phys. Comm.,**119**,67 (1999)), and is better behaved for taking derivatives (i.e. calculating ionic forces) than the expansion suggested by Louie et al.

31.2 PSPW RTDB Entries and DataFiles

Input to the PSPW module is contained in both the RTDB and datafiles. The RTDB is used to store input that the user will need to directly specify. Input of this kind includes ion positions, ion velocities, and simulation cell parameters. The datafiles are used to store input, such the one-electron orbitals, one-electron orbital velocities, formatted pseudopotentials, and one-dimensional pseudopotentials, that the user will in most cases run a program to generate.

31.2.1 Ion Positions

The positions of the ions are stored in the default geometry structure in the RTDB and must be specified using the GEOMETRY directive.

31.2.2 Ion Velocities

The velocities of the ions are stored in the default geometry structure in the RTDB, and must be specified using the GEOMETRY directive.

31.2.3 Simulation Cell

Simulation cells are stored in the RTDB. To enter a simulation cell into the RTDB the user defines a simulation_cell sub-block within the PSPW block. Listed below is the format of a simulation_cell sub-block.

```
PSPW
...
   SIMULATION_CELL
      CELL_NAME: <string name>
      BOUNDRY_CONDITIONS: (periodic || aperiodic)
      LATTICE_VECTORS:
        <real a1.x a1.y a1.z>
        <real a2.x a2.y a2.z>
        <real a3.x a3.y a3.z>
      NGRID: <integer na1 na2 na3>
   END
...
END
```

Basically, the user needs to enter the dimensions, gridding and boundry conditions of the simulation cell. The following list describes the input in detail.

- <name> - user-supplied name for the simulation block.
- periodic - keyword specifying that the simulation cell has periodic boundary conditions.
- aperiodic - keyword specifying that the simulation cell has free-space boundary conditions.
- <a1.x a1.y a1.z> - user-supplied values for the first lattice vector
- <a2.x a2.y a2.z> - user-supplied values for the second lattice vector
- <a3.x a3.y a3.z> - user-supplied values for the third lattice vector
- <na1 na2 na3> - user-supplied values for discretization along lattice vector directions.

31.2.4 Analysis: Mulliken RTDB data

To perform Mulliken analysis information is needed from one-dimensional pseudopotential files. In-order to facilitate the transfer of this information to the simulation the ANALYSIS sub-block is used to exract the necessary information and put it into the RTDB.

```
PSPW
...
   ANALYSIS
      (psp_filename: <string psp_name> \
```

```
      ...)
   END
...
END
```

Basically, the user needs to enter each pseudopotential used in the simulation.

- <psp_name> - name that ponts to a one-dimensional pseudopotential file.

31.2.5 Wavefunction Datafile

The one-electron orbitals are stored in a wavefunction datafile. This is a binary file and cannot be directly edited. This datafile is used by steepest_descent and Car-Parrinello tasks and can be generated using the wavefunction_initializer or wavefunction_expander tasks.

31.2.6 Velocity Wavefunction Datafile

The one-electron orbital velocities are stored in a velocity wavefunction datafile. This is a binary file and cannot be directly edited. This datafile is used by the Car-Parrinello task and can be generated using the v_wavefunction_initializer task.

31.2.7 Formatted Pseudopotential Datafile

The pseudopotentials in Kleinman-Bylander form expanded on a simulation cell (3d grid) are stored in a formatted pseudopotential datafile. This is a binary file and cannot be directly edited. This datafile is used by steepest_descent and Car-Parrinello tasks and can be generated using the pseudpotential_formatter task.

31.2.8 One-Dimensional Pseudopotential Datafile

The one-dimensional pseudopotentials are stored in a one-dimensional pseudopotential file. This is an ascii file and can be directly edited with a text editor. However, the user will usually use the psp_generator task to generate this datafile.

The data stored in the one-dimensional pseudopotential file is

```
character*2 element       :: element name
integer     charge        :: valence charge of ion
real        mass          :: mass of ion
integer     lmax          :: maximum angular component
real        rcut(lmax)    :: cutoff radii used to define pseudopotentials
integer     nr            :: number of points in the radial grid
real        dr            :: linear spacing of the radial grid
real        r(nr)         :: one-dimensional radial grid
real        Vpsp(nr,lmax) :: one-dimensional pseudopotentials
real        psi(nr,lmax)  :: one-dimensional pseudowavefunctions
real        r_semicore        :: semicore radius
real        rho_semicore(nr)  :: semicore density
```

and the format of it is:

```
[line 1:     ] element
[line 2:     ] charge mass lmax
[line 3:     ] (rcut(l), l=1,lmax)
[line 4:     ] nr dr
[line 5:     ]    r(1)  (Vpsp(1,l),  l=1,lmax)
[line 6:     ] ....
[line nr+4:  ] r(nr) (Vpsp(nr,l), l=1,lmax)
[line nr+5:  ] r(1)  (psi(1,l), l=1,lmax)
[line nr+6:  ] ....
[line 2*nr+4:] r(nr) (psi(nr,l), l=1,lmax)
[line 2*nr+5:] r_semicore
if (r_semicore read) then
[line 2*nr+6:] r(1)  rho_semicore(1)
[line 2*nr+7:] ....
[line 3*nr+5:] r(nr) rho_semicore(nr)
end if
```

31.2.9 PSPW Car-Parrinello Output Datafiles

XYZ motion file

Data file that stores ion positions and velocities as a function of time in XYZ format.

```
[line 1:          ]  n_ion
[line 2:          ]
do ii=1,n_ion
[line 2+ii:       ] atom_name(ii), x(ii),y(ii),z(ii),vx(ii),vy(ii),vz(ii)
end do
[line n_ion+3     ] n_nion

do ii=1,n_ion
[line n_ion+3+ii: ] atom_name(ii), x(ii),y(ii),z(ii), vx(ii),vy(ii),vz(ii)
end do
[line 2*n_ion+4:  ]  ....
```

ION_MOTION motion file

Datafile that stores ion positions and velocities as a function of time

```
[line 1:          ]  it_out, n_ion, omega
[line 2:          ]  time
do ii=1,n_ion
[line 2+ii:       ] x(ii),y(ii),z(ii), vx(ii),vy(ii),vz(ii)
end do
[line n_ion+3     ] time
do
do ii=1,n_ion
```

```
[line n_ion+3+ii: ] x(ii),y(ii),z(ii), vx(ii),vy(ii),vz(ii)
end do
[line 2*n_ion+4:  ]  ....
```

EMOTION motion file

Datafile that store energies as a function of time

```
[line 1:          ]  time, E1,E2,E3,E4,E5,E6,E7,E8, (E9,E10, if Nose-Hoover)
[line 2:          ] ...
```

HMOTION motion file

Datafile that stores the rotation matrix as a function of time.

```
[line 1:          ] time
[line 2:          ] ms,ne(ms),ne(ms)
do i=1,ne(ms)
[line 2+i:        ] (hml(i,j), j=1,ne(ms)
end do
[line 3+ne(ms):   ] time
[line 4+ne(ms):   ] ....
```

EIGMOTION motion file

Datafile that stores the eigenvalues for the one-electron orbitals as a function of time.

```
[line 1:          ]  time, (eig(i), i=1,number_orbitals)
[line 2:          ] ...
```

OMOTION motion file

Datafile that stores a reduced representation of the one-electron orbitals. To be used with a molecular orbital viewer that will be ported to NWChem in the near future.

31.3 Minimizing the DFT Energy Functional

In this section the algorithms for steepest_descent and conjugate_gradient methods that are used to minimize the DFT energy functional are presented. First is the the steepest descent method without line minimization. This method is slow, but doesn't require line minimization. Second is the recent method developed by Edelman, Arias, and Smith. These researchers have developed methods for performing line minimizations on the constrained space used in the DFT energy functional (A. Edelman, T. Arias, and S.T. Smith, Siam J. Matrix Anal. Appl., **20**,303, (1998)).

31.3.1 Steepest Descent Equations

To minimize the DFT energy functional with respect to $\{\psi_{i,\sigma}\}$ we define the the auxiliary functional, F, which takes into account the orthonormality constraints.

$$
\begin{aligned}
F\left(\{\psi_{i,\sigma}\},\{\vec{R}_I\}\right) &= E\left(\{\psi_{i,\sigma}\},\{\vec{R}_I\}\right) \\
&+ \sum_{ij,\sigma}\left(\int d\vec{r}\ \psi^*_{i,\sigma}(\vec{r})\psi_{j,\sigma}(\vec{r})\Lambda_{ji,\sigma} - \delta_{ij,\sigma}\right)
\end{aligned} \tag{31.2}
$$

The steepest descent minimization procedure without line minimization is then

$$
\psi^{t+\Delta t}_{i,\sigma} \leftarrow \psi^t_{i,\sigma} - \alpha\left[\frac{\delta F}{\delta\psi^*_{i,\sigma}}\right]_t = \psi^t_{i,\sigma} - \alpha\left[\frac{\delta E}{\delta\psi^*_{i,\sigma}} + \sum_j \psi_{j,\sigma}\Lambda_{ji,\sigma}\right]_t \tag{31.3}
$$

where $\alpha = \frac{\Delta t}{\sqrt{\mu}}$ is a positive numerical parameter. In this minimization procedure we have to know the variational derivative $\frac{\delta E}{\delta\psi^*_{i,\sigma}}$ and the matrix $\Lambda_{ij,\sigma}$. The variational derivative $\frac{\delta E}{\delta\psi^*_{i,\sigma}}$ can be analytically found and is

$$
\begin{aligned}
\frac{\delta E}{\delta\psi^*_{i,\sigma}} &= -\frac{1}{2}\nabla^2\psi_{i,\sigma}(\vec{r}) \\
&+ \int d\vec{r'}\,W_{ext}(\vec{r},\vec{r'})\psi_{i,\sigma}(\vec{r'}) \\
&+ \int d\vec{r'}\,\frac{n(\vec{r'})}{|\vec{r}-\vec{r'}|}\psi_{i,\sigma}(\vec{r}) \\
&+ \mu^\sigma_{xc}(\vec{r})\psi_{i,\sigma}(\vec{r}) \\
&\equiv \hat{H}\psi_{i,\sigma}
\end{aligned} \tag{31.4}
$$

The ion positions can also be minimized using a fixed step minimization procedure.

$$
\vec{R}^{t+\Delta t}_I \leftarrow \vec{R}^t_I - \frac{(\Delta t)}{\sqrt{M_I}}\frac{\partial E}{\partial\vec{R}_I} \tag{31.5}
$$

To find the matrix $\Lambda_{ij,\sigma}$ we impose the orthonormality constraint on $\psi^{t+\Delta t}_{i,\sigma}$ to obtain the matrix Riccatti equation (to simplify the following equations we define the following symbol $\bar{\psi}^t_{i,\sigma} = \left(\frac{\delta E}{\delta\psi^*_{i,\sigma}}\right)_t$)

$$
\begin{aligned}
I &= <\psi^{t+\Delta t}_{i,\sigma}|\psi^{t+\Delta t}_{j,\sigma}> \\
&= \left(<\psi^t_{i,\sigma}| - \alpha\left[<\bar{\psi}^t_{i,\sigma}| + \sum_k \Lambda^*_{ik,\sigma} <\psi^t_{k,\sigma}|\right]\right) \\
&\quad \left(|\psi^t_{j,\sigma}> -\alpha\left[|\bar{\psi}^t_{j,\sigma}> + \sum_l |\psi^t_{l,\sigma}> \Lambda_{lj,\sigma}\right]\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \; < \psi^t_{i,\sigma} | \psi^t_{j,\sigma} > \\
&- \; \alpha \begin{bmatrix} < \psi^t_{i,\sigma} | \bar{\psi}^t_{j,\sigma} > & + < \bar{\psi}^t_{i,\sigma} | \psi^t_{j,\sigma} > \\ + \sum_k \Lambda^*_{ik,\sigma} < \psi^t_{k,\sigma} | \psi^t_{j,\sigma} > & + \sum_l < \psi^t_{i,\sigma} | \psi^t_{l,\sigma} > \Lambda_{lj,\sigma} \end{bmatrix} \\
&+ \; \alpha^2 \left[< \bar{\psi}^t_{i,\sigma} | \bar{\psi}^t_{j,\sigma} > + \sum_k \Lambda^*_{ik,\sigma} < \psi^t_{k,\sigma} | \bar{\psi}^t_{j,\sigma} > + \sum_l < \bar{\psi}^t_{i,\sigma} | \psi^t_{l,\sigma} > \Lambda_{lj,\sigma} \right] \\
&+ \; \alpha^2 \left[\sum_k \sum_l \Lambda^*_{ik,\sigma} < \psi^t_{k,\sigma} | \psi^t_{l,\sigma} > \Lambda_{lj,\sigma} \right] \\
&= \; A + \alpha X B + \alpha B^\dagger X^\dagger + \alpha^2 X C X^\dagger
\end{aligned} \tag{31.6}
$$

where $X_{ij,\sigma} = \Lambda^*_{ij,\sigma}$ and the matrices A, B, and C are given by

$$A_{ij,\sigma} = \int d\vec{r} \left[\psi^t_{i,\sigma}(\vec{r}) - \alpha \left(\frac{\delta E}{\delta \psi^*_{i,\sigma}} \right)_t \right]^* \left[\psi^t_{j,\sigma}(\vec{r}) - \alpha \left(\frac{\delta E}{\delta \psi^*_{j,\sigma}} \right)_t \right] \tag{31.7}$$

$$B_{ij,\sigma} = \int d\vec{r} \left[\psi^t_{i,\sigma}(\vec{r}) \right]^* \left[\psi^t_{j,\sigma}(\vec{r}) - \alpha \left(\frac{\delta E}{\delta \psi^*_{j,\sigma}} \right)_t \right] \tag{31.8}$$

$$C_{ij,\sigma} = \int d\vec{r} \left[\psi^t_{i,\sigma}(\vec{r}) \right]^* \left[\psi^t_{j,\sigma}(\vec{r}) \right] \tag{31.9}$$

To solve the Riccatti equation an iterative solution is setup by rewriting Eq. 31.6 in the following form.

$$\alpha^2 X C X^\dagger + \alpha X (B - I) + \alpha (B^\dagger - I) X^\dagger + \alpha (X + X^\dagger) = I - A \tag{31.10}$$

Since X is by definition Hermitian, $X = X^\dagger$, and

$$
\begin{aligned}
A &\approx I + O(\alpha) \\
B &\approx I + O(\alpha) \qquad (31.11)
\end{aligned}
$$

$$(31.12)$$

we can solve for X iteratively.

$$
\begin{aligned}
X_{(n)} \leftarrow \; & \frac{1}{2\alpha}(I - A) + \\
& \frac{1}{2} \left(X_{(n-1)}(I - B) + (I - B^\dagger) X^\dagger_{(n-1)} - \alpha X_{(n-1)} C X^\dagger_{(n-1)} \right) \qquad (31.13) \\
X_{(0)} \leftarrow \; & \left(\frac{I - A}{2\alpha} \right) \qquad (31.14)
\end{aligned}
$$

31.3.2 Conjugate Gradient with Curvature: Grassmann Manifold

Minimizing the DFT energy functional with a non-linear conjugate gradient algorithm requires an algorithm for minimizing along a search direction or line. However, the equation for a line (or search direction) in the DFT minimization problem is complicated by the fact that the energy functional is subject to orthonormality constraints. Furthermore, the orthonormality constraints complicate the conjugate gradient's algorithm for generating the current search direction by using a linear combination of the current gradient and the previous search direction, since the previous search direction must be parallel transported to the current search point.

These orthonormality constraint problems have been investigated and solved for by Edelman *et al* (A. Edelman, T. Arias, and S.T. Smith, Siam J. Matrix Anal. Appl., **20**,303, (1998)). They recognized that the orthonormal Kohn-Sham orbitals $\{\psi_{i,\sigma}\}$ belong to a quotient space called a Grassmann manifold. Using the properties of the Grassmann manifold they developed an equation for a line (or geodesic) which conserves orthonormality, as well as a conjugate gradient procedure for generating the current search direction on this constrained space.

Following Edelman *et al* we write $\{\psi_{i,\sigma}(\vec{r})\}$ in terms of the orthonormal basis $\{\phi_j(\vec{r})\}$.

$$\psi_{i,\sigma}(\vec{r}) = \sum_{j}^{N_{basis}} \psi_{i,\sigma}(\phi_j)\phi_j(\vec{r}) \tag{31.15}$$

Which allows us to rewrite the DFT energy functional as a function, $E(Y_\uparrow, Y_\downarrow)$, of two tall and skinny N_{basis}-by-N_σ matrices which are

$$Y_\uparrow = \begin{bmatrix} \psi_{1,\uparrow}(\phi_1) & \psi_{2,\uparrow}(\phi_1) & \cdots & \psi_{N_\uparrow,\uparrow}(\phi_1) \\ \psi_{1,\uparrow}(\phi_2) & \psi_{2,\uparrow}(\phi_2) & \cdots & \psi_{N_\uparrow,\uparrow}(\phi_2) \\ \psi_{1,\uparrow}(\phi_3) & \psi_{2,\uparrow}(\phi_3) & \cdots & \psi_{N_\uparrow,\uparrow}(\phi_3) \\ \vdots & \vdots & & \vdots \\ \psi_{1,\uparrow}(\phi_{N_{basis}}) & \psi_{2,\uparrow}(\phi_{N_{basis}}) & \cdots & \psi_{N_\uparrow,\uparrow}(\phi_{N_{basis}}) \end{bmatrix} \tag{31.16}$$

and

$$Y_\downarrow = \begin{bmatrix} \psi_{1,\downarrow}(\phi_1) & \psi_{2,\downarrow}(\phi_1) & \cdots & \psi_{N_\downarrow,\downarrow}(\phi_1) \\ \psi_{1,\downarrow}(\phi_2) & \psi_{2,\downarrow}(\phi_2) & \cdots & \psi_{N_\downarrow,\downarrow}(\phi_2) \\ \psi_{1,\downarrow}(\phi_3) & \psi_{2,\downarrow}(\phi_3) & \cdots & \psi_{N_\downarrow,\downarrow}(\phi_3) \\ \vdots & \vdots & & \vdots \\ \psi_{1,\downarrow}(\phi_{N_{basis}}) & \psi_{2,\uparrow}(\phi_{N_{basis}}) & \cdots & \psi_{N_\downarrow,\downarrow}(\phi_{N_{basis}}) \end{bmatrix} \tag{31.17}$$

The orthonormality constraints make the Y_σ matrices obey $Y_\sigma^\dagger Y_\sigma = I$. Furthermore since the LSDA energy can be written as

$$E(Y_\uparrow, Y_\downarrow) = \sum_\sigma tr\left(Y_\sigma^\dagger(\text{Operator})Y_\sigma\right) \tag{31.18}$$

we are only interested in subspace spanned by Y_σ. This homogeneity property allows us to state that $E(Y_\uparrow, Y_\downarrow) = E(Y_\uparrow Q_\uparrow, Y_\downarrow Q_\downarrow)$ where Q_σ is any N_σ-by-N_σ orthogonal matrix (i.e. Q is any group element of the orthogonal group $O(N_\sigma)$). The constrained surface that each Y_σ spans is known as the Grassmann manifold (i.e. Y_σ is a group element of the quotient space group $\frac{\left(\frac{O(N_{basis})}{O(N_{basis}-N_\sigma)}\right)}{O(N_\sigma)}$ [1]).

In this work Algorithm 31.1 (see below) is the conjugate gradient algorithm that is used to minimize the DFT energy functional $E = E(Y_\uparrow, Y_\downarrow)$. What makes this algorithm different from a standard Polak-Ribière conjugate gradient algorithm is that a line search on a Euclidean space

$$Y_\sigma(t) = Y_\sigma^{(0)} + t * H_\sigma^{(0)}$$

[1] See reference on Lie Groups, e.g. reference (P.J. Olver, *Equivalence, Invariants, and Symmetry*, 1st ed. Cambridge University Press, New York, 1995), for a definition of a quotient space and a definition of the orthogonal group $O(n)$.

is replaced by Eq. 31.19, and the parallel transports on a Euclidean space

$$\tau H_\sigma^{(0)} = H_\sigma^{(0)}$$

$$\tau G_\sigma^{(0)} = G_\sigma^{(0)}$$

are replaced by Eqs. 31.20-31.21.

Algorithm 31.1: Edelman *et al's* Algorithm for Conjugate Gradient Minimization on the Grassmann Manifold

1. *Given* $Y_\sigma^{(0)}$ *such that* $Y_\sigma^{(0)\dagger} Y_\sigma^{(0)}$,
   Compute $G_\sigma^{(0)} = \left[\frac{\delta E}{\delta Y_\sigma}\right]_{Y_\sigma = Y_\sigma^{(0)}} - Y_\sigma^{(0)} Y_\sigma^{(0)\dagger} \left[\frac{\delta E}{\delta Y_\sigma}\right]_{Y_\sigma = Y_\sigma^{(0)}}$
   Set $H_\sigma^{(0)} = -G_\sigma^{(0)}$.
2. *Find the compact singular value decompositions of* $H_\sigma^{(0)} \rightarrow U_\sigma \Sigma_\sigma V_\sigma^\dagger$
3. *Minimize* $E\left(Y_\uparrow(t), Y_\downarrow(t)\right)$ *along the geodesic lines derived by Edelman et al for the Grassmann Manifold*

$$Y_\sigma(t) = Y_\sigma^{(0)} V_\sigma \cos\left(\Sigma_\sigma t\right) V^\dagger + U_\sigma \sin\left(\Sigma_\sigma t\right) V_\sigma^\dagger \tag{31.19}$$

4. *Set* $Y_\sigma^{(1)} = Y_\sigma(t_{min})$ *and*
   compute $G_\sigma^{(1)} = \left[\frac{\delta E}{\delta Y_\sigma}\right]_{Y_\sigma = Y_\sigma^{(1)}} - Y_\sigma^{(1)} Y_\sigma^{(1)\dagger} \left[\frac{\delta E}{\delta Y_\sigma}\right]_{Y_\sigma = Y_\sigma^{(1)}}$
5. *Parallel transport along the geodesics the tangent vectors* $H_\sigma^{(0)}$ *and* $G_\sigma^{(0)}$

$$\tau H_\sigma^{(0)} = \left(-Y_\sigma^{(1)} V_\sigma \sin\left(\Sigma_\sigma t\right) + U_\sigma \cos\left(\Sigma_\sigma t\right)\right) \Sigma_\sigma V_\sigma^\dagger \tag{31.20}$$

$$\tau G_\sigma^{(0)} = G_\sigma^{(0)} \left(Y_\sigma^{(1)} V_\sigma \sin\left(\Sigma_\sigma t\right) + U_\sigma \left(I - \cos\left(\Sigma_\sigma t\right)\right)\right) U_\sigma^\dagger G_\sigma^{(0)} \tag{31.21}$$

6. *Compute the new search direction*

$$H_\sigma^{(1)} = -G_\sigma^{(1)} + \Gamma_\sigma \tau H_\sigma^{(0)}$$

   where

$$\Gamma_\sigma = \frac{tr\left[\left(G_\sigma^{(1)} - \tau G_\sigma^{(0)}\right) G_\sigma^{(1)}\right]}{tr\left[G_\sigma^{(0)} G_\sigma^{(0)}\right]}$$

7. *Set* $Y_\sigma^{(0)} = Y_\sigma^{(1)}$, $G_\sigma^{(0)} = G_\sigma^{(1)}$, *and* $H_\sigma^{(0)} = H_\sigma^{(1)}$
8. *Go to step 2.*

31.4 Car-Parrinello Scheme for Ab Initio Molecular Dynamics

Car and Parrinello developed a unified scheme for doing *ab initio* molecular dynamics by combining the motion of the ion cores and a fictacious motion for the Kohn-Sham orbitals of density-functional theory (R. Car and M. Parrinello, Phys. Rev. Lett. **55**, 2471, (1985)). At the heart of this method they introduced a fictacious kinetic energy functional for the Kohn-Sham orbitals.

$$KE(\{\psi_{i,\sigma}(\vec{r})\}) = \sum_{i,\sigma}^{occ} \int d\vec{r}\; \mu \left|\dot{\psi}_{i,\sigma}(\vec{r})\right|^2 \tag{31.22}$$

Given this kinetic energy the constrained equations of motion are found by taking the first variation of the auxiliary Lagrangian.

$$\begin{aligned} L = & \sum_{i,\sigma}^{occ} \int d\vec{r}\; \mu \left|\dot{\psi}_{i,\sigma}(\vec{r})\right|^2 + \frac{1}{2}\sum_I M_I \left|\dot{\vec{R}}_I\right|^2 - E\left[\{\psi_{i,\sigma}(\vec{r})\}, \left\{\vec{R}_I\right\}\right] \\ & + \sum_{ij,\sigma} \Lambda_{ij,\sigma}\left(\int d\vec{r}\; \psi^*_{i,\sigma}(\vec{r})\psi_{j,\sigma}(\vec{r}) - \delta_{ij,\sigma}\right) \end{aligned} \tag{31.23}$$

Which generates a dynamics for the wavefunctions $\psi_{i,\sigma}(\vec{r})$ and atoms positions $\vec{R}_I$ through the constrained equations of motion:

$$\mu \ddot{\psi}_{i,\sigma}(\vec{r},t) = -\frac{\delta E}{\delta \psi^*_{i,\sigma}(\vec{r},t)} + \sum_j \Lambda_{ij,\sigma}\psi_{j,\sigma}(\vec{r},t) \tag{31.24}$$

$$M_I \ddot{\vec{R}}_I = -\frac{\partial E}{\partial \vec{R}_I} \tag{31.25}$$

where μ is the fictitious mass for the electronic degrees of freedom and M_I are the ionic masses. The adjustable parameter μ is used to describe the relative rate at which the wavefunctions change with time. $\Lambda_{ij,\sigma}$ are the Lagrangian multipliers for the orthonormalization of the single-particle orbitals $\psi_{i,\sigma}(\vec{r})$. They are defined by the orthonormalization constraint conditions and can be rigorously found. However, the equations of motion for the Lagrange multipliers depend on the specific algorithm used to integrate Eqs. 31.24-31.25.

For this method to give ionic motions that are physically meaningful the kinetic energy of the Kohn-Sham orbitals must be relatively small when compared to the kinetic energy of the ions. There are two ways where this criterion can fail. First, the numerical integrations for the Car-Parrinello equations of motion can often lead to large relative values of the kinetic energy of the Kohn-Sham orbitals relative to the kinetic energy of the ions. This kind of failure is easily fixed by requiring a more accurate numerical integration, i.e. use a smaller time step for the numerical integration. Second, during the motion of the system a the ions can be in locations where there is an Kohn-Sham orbital level crossing, i.e. the density-functional energy can have two states that are nearly degenerate. This kind of failure often occurs in the study of chemical reactions. This kind of failure is not easily fixed and requires the use of a more sophisticated density-functional energy that accounts for low-lying excited electronic states.

31.4.1 Verlet Algorithm for Integration

Eqs. 31.24-31.25 integrated using the Verlet algorithm results in

$$\psi_{i,\sigma}^{t+\Delta t} \leftarrow 2\psi_{i,\sigma}^{t} - \psi_{i,\sigma}^{t-\Delta t} + \frac{(\Delta t)^2}{\mu}\left[\frac{\delta E}{\delta \psi_{i,\sigma}^*} + \sum_j \psi_{j,\sigma}\Lambda_{ji,\sigma}\right]_t \tag{31.26}$$

$$\vec{R}_I^{t+\Delta t} \leftarrow 2\vec{R}_I^t - \vec{R}_I^{t-\Delta t} + \frac{(\Delta t)^2}{M_I}\frac{\partial E}{\partial \vec{R}_I} \tag{31.27}$$

In this molecular dynamic procedure we have to know variational derivative $\frac{\delta E}{\delta \psi_{i,\sigma}^*}$ and the matrix $\Lambda_{ij,\sigma}$. The variational derivative $\frac{\delta E}{\delta \psi_{i,\sigma}^*}$ can be analytically found and is

$$\begin{aligned}
\frac{\delta E}{\delta \psi_{i,\sigma}^*} &= -\frac{1}{2}\nabla^2\psi_{i,\sigma}(\vec{r}) \\
&+ \int d\vec{r'}\, W_{ext}(\vec{r},\vec{r'})\psi_{i,\sigma}(\vec{r'}) \\
&+ \int d\vec{r'}\, \frac{n(\vec{r'})}{|\vec{r}-\vec{r'}|}\psi_{i,\sigma}(\vec{r}) \\
&+ \mu_{xc}^{\sigma}(\vec{r})\psi_{i,\sigma}(\vec{r}) \\
&\equiv \hat{H}\psi_{i,\sigma}
\end{aligned} \tag{31.28}$$

To find the matrix $\Lambda_{ij,\sigma}$ we impose the orthonormality constraint on $\psi_{i,\sigma}^{t+\Delta t}$ to obtain a matrix Riccatti equation, and then Riccatti equation is solved by an iterative solution (see section 31.3.1).

31.4.2 Constant Temperature Simulations: Nose-Hoover Thermostats

Nose-Hoover Thermostats for the electrons and ions can also be added to the Car-Parrinello simulation. In this type of simulation thermostats variables x_e and x_R are added to the simulation by adding the auxillary energy functionals to the total energy.

$$ION\_THERMOSTAT(x_R) = \frac{1}{2}Q_R\dot{x_R} + E_{R0}x_R \tag{31.29}$$

$$ELECTRON\_THERMOSTAT(x_e) = \frac{1}{2}Q_e\dot{x_e} + E_{e0}x_e \tag{31.30}$$

In these equations, the average kinetic energy for the ions is

$$E_{R0} = \frac{1}{2}fk_BT \tag{31.31}$$

where f is the number of atomic degrees of freedom, k_B is Boltzmans constant, and T is the desired temperature. Defining the average fictacious kinetic energy of the electrons is not as straighforward. Blöchl and Parrinello (P.E. Blöchl and M. Parrinello, Phys. Rev. B, **45**, 9413, (1992)) have suggested the following formula for determining the average fictacious kinetic energy

$$E_{e0} = 4k_B T \frac{\mu}{M} \sum_i < \psi_i | - \frac{1}{2}\nabla^2 | \psi_i > \tag{31.32}$$

where μ is the fictacious electronic mass, M is average mass of one atom, and $\sum_i < \psi_i | - \frac{1}{2}\nabla^2 | \psi_i >$ is the kinetic energy of the electrons.

Blöchl and Parrinello suggested that the choice of mass parameters, Q_e, and Q_R should be made such that the period of oscillating thermostats should be chosen larger than the typical time scale for the dynamical events of interest but shorter than the simulation time.

$$P_{ion} = 2\pi\sqrt{\frac{Q_R}{4E_{R0}}} \tag{31.33}$$

$$P_{electron} = 2\pi\sqrt{\frac{Q_e}{4E_{e0}}} \tag{31.34}$$

where P_{ion} and $P_{electron}$ are the periods of oscillation for the ionic and ficatious electronic thermostats.

31.5 PSPW Tutorial 1: Minimizing the one-electron orbitals by Running a Steepest Descent and Conjuagate Gradient Simulation in Tandem

In this section we show how to setup and run a steepest descent simulation followed by a conjugate gradient simulation to optimize the one-electron orbitals with respect to the total energy for a NaCl molecule. It is practical to start most simulation in such a tandem fashion because the conjugate gradient optimizer does not work well when the one-electron orbitals are far from minimized.

Before running a steepest_descent or conjugate_gradient simulation several files and RTDB structures must be defined or initialized. Specifically, the user is required to have defined:

1. **ion positions**
2. **simulation cell** in the RTDB
3. **formatted pseudopotential file** for each kind of ion
4. **file containing the one-electron orbitals**
5. **steepest_descent PSPW sub-block**
6. **conjugate_gradient PSPW sub-block**

In the following tutorial we show the input needed to find an energy for an NaCl molecule. In this example default Hamann pseudopotentials are used for Na and Cl, the exchange correlation functional is LSDA, and the cutoff energy is 12 au. This example input deck can be found in the nwchem source tree in the file: nwchem/examples/pspw/NaCl.nw

1. Define the ion positions using the geometry directive.

```
...
geometry units au
Na    2.23 0.0 0.0
Cl   -2.23 0.0 0.0
end
...
```

2. Generate a pseudopotential for Sodium atoms and store it in a one-dimensional pseudopotential datafile called Na.psp. The following input can be used to generate a default Hamann pseudopotential.

```
...
PSPW
   PSP_GENERATOR
      pseudopotential_filename: Na.psp
       element: Na
       charge: 11.0
       mass_number: 23.0
       solver_type: pauli
       pseudopotential_type: hamann
       atomic_filling: 3 1
         1 s 2.0
         2 s 2.0
         2 p 6.0
         3 s 1.0
   END
END
task PSPW PSP_GENERATOR
...
```

3. Use the same proceedure as above generate a pseudopotential for Chlorine.

```
...
PSPW
   PSP_GENERATOR
      pseudopotential_filename: Cl.psp
      element: Cl
      charge: 17.0
      mass_number: 35.0
      solver_type: pauli
      pseudopotential_type: hamann
      atomic_filling: 3 2
         1 s 2.0
         2 s 2.0
         2 p 6.0
         3 s 2.0
         3 p 5.0

   END
END
task PSPW PSP_GENERATOR
...
```

4. Define the simulation cell. The following input defines a simulation cell called "small." This cell is periodic and cubic with a side length of 20.0 au and has 32 grid points in each direction.

```
...
PSPW
   SIMULATION_CELL
      cell_name: small
      boundry_conditions: periodic
      lattice_vectors:
         20.0  0.0  0.0
          0.0 20.0  0.0
          0.0  0.0 20.0
      ngrid: 32 32 32
   END
END
...
```

5. Format the Na pseudopotential onto the "small" simulation cell.

```
...
PSPW
   PSP_FORMATTER
      cell_name: small
      psp_name: Na.psp
      formatted_name: Na.vpp
   END
END
task PSPW PSP_FORMATTER
...
```

6. Format the Cl pseudopotential onto the "small" simulation cell.

```
...
PSPW
   PSP_FORMATTER
      cell_name: small
      psp_name: Cl.psp
      formatted_name: Cl.vpp
   END
END
task PSPW PSP_FORMATTER
...
```

7. Generate an initial guess for the one-electron orbitals.

```
...
PSPW
   WAVEFUNCTION_INITIALIZER
      cell_name: small
      unrestricted
      up_electrons: 4
      down_electrons: 4
      wavefunction_filename: NaCl.small.00.elc
   END
END
task PSPW WAVEFUNCTION_INTITALIZER
...
```

8. Do a coarse optimization of the one-electron orbitals with respect to energy using steepest descent.

```
...
PSPW
   STEEPEST_DESCENT
      cell_name: small
      formatted_filename: Na.vpp
      formatted_filename: Cl.vpp
      input_wavefunction_filename:  NaCl.small.00.elc
      output_wavefunction_filename: NaCl.small.00.elc
      fake_mass: 400000.0d0
      time_step: 51.8d0
      loop: 10 100
      tolerances: 1.0d-4 1.0d-4 1.0d-4
      energy_cutoff:       21.0d0
      wavefunction_cutoff: 21.0d0
   END
END
task PSPW STEEPEST_DESCENT
...
```

9. Do a finer optimization of the one-electron orbitals with respect to energy using conjugate gradient, and perform a Mulliken analysis. Note that an analysis block must be defined.

```
...
PSPW

   CONJUGATE_GRADIENT
      cell_name: small
      formatted_filename: Na.vpp
      formatted_filename: Cl.vpp
      input_wavefunction_filename:  NaCl.small.00.elc
      output_wavefunction_filename: NaCl.small.00.elc
      loop: 25 10
      tolerances: 1.0d-9 1.0d-9
      energy_cutoff:       21.0d0
      wavefunction_cutoff: 21.0d0
      Mulliken
   END
   ANALYSIS
       psp_name: Na.psp
       psp_name: Cl.psp
   END
END
task PSPW CONJUGATE_GRADIENT
...
```

31.6 PSPW Tutorial 2: Running a Car-Parrinello Simulation

In this section we show how to perform a Car-Parrinello molecular dynamic simulation for an NaCl molecule. As with the example in the previous tutorial, this example uses default Hamann pseudopotentials for Na and Cl, LSDA exchange correlation functional, and a cutoff energy of 12 au.

Before running a PSPW Car-Parrinello simulation the system should be on the Born-Oppenheimer surface, i.e. the one-electron orbitals should be minimized with respect to the total energy using steepest descent and conjugate gradient simulations(see section 31.5).

Continuning where we left off in section 31.5 we now continue the simulation to run a MD simulation of an NaCl molecule. This example input deck can be found in the nwchem source tree in the file: nwchem/examples/pspw/NaCl-md.nw

1. Optimize wavefunctions - See previous section
2. Generate an initial guess for the one-electron orbitals velocities.

```
...
PSPW
   V_WAVEFUNCTION_INITIALIZER
      cell_name: small
      unrestricted
      up_filling: 4
      down_filling: 4
      v_wavefunction_filename: NaCl.small.00.velc
   END
END
task PSPW WAVEFUNCTION_INTITALIZER
...
```

3. Run an MD simulation using Car-Parrinello.

```
PSPW
   Car-Parrinello
      cell_name: small
      formatted_filename: Na.vpp
      formatted_filename: Cl.vpp
      input_wavefunction_filename:    NaCl.small.00.elc
      output_wavefunction_filename:   NaCl.small.01.elc
      v_input_wavefunction_filename:  NaCl.00.velc
      v_output_wavefunction_filename: NaCl.01.velc
      fake_mass: 800.0d0
      time_step: 5.0d0
      loop: 10 100
      scaling: 1.0d0 1.0d0
      energy_cutoff:       21.0d0
      wavefunction_cutoff: 21.0d0
   END
END
task PSPW Car-Parrinello
...
```

31.7 PSPW Capabilities and Limitations

- You cannot use more processors than the size of the third dimension (e.g. a 64x64x64 FFT grid can use at most 64 processors).
- The second and third dimensions of the FFT grid must be the same (i.e. the parameters na2 and na3 must be the same for each simulation cell).

31.8 Questions and Difficulties

Questions and encountered problems should be reported to nwchem-support@emsl.pnl.gov or to Eric J. Bylaska, Eric.Bylaska@pnl.gov

Chapter 32

Controlling NWChem with Python

Python (version 1.5.1) programs may be embedded into the NWChem input and used to control the execution of NWChem. Python is a very powerful and widely used scripting language that provides useful things such as variables, conditional branches and loops, and is also readily extended. Example applications include scanning potential energy surfaces, computing properties in a variety of basis sets, optimizing the energy w.r.t. parameters in the basis set, computing polarizabilities with finite field, and simple molecular dynamics.

Look in the NWChem `contrib` directory for useful scripts and examples. Visit the Python web-site http://www.python.org for a full manual and lots of useful code and resources.

32.1 How to input and run a Python program inside NWChem

A Python program is input into NWChem inside a Python compound directive.

```
python [print|noprint]
  ...
end
```

The `END` directive must be flush against the left margin (see the Troubleshooting section for the reason why).

The program is by default printed to standard output when read, but this may be disabled with the `noprint` keyword. Python uses indentation to indicate scope (and the initial level of indentation must be zero), whereas NWChem uses optional indentation only to make the input more readable. For example, in Python, the contents of a loop, or conditionally-executed block of code must be indented further than the surrounding code. Also, Python attaches special meaning to several symbols also used by NWChem. For these reasons, the input inside a `PYTHON` compound directive is read verbatim except that if the first line of the Python program is indented, the same amount of indentation is removed from all subsequent lines. This is so that a program may be indented inside the `PYTHON` input block for improved readability of the NWChem input, while satisfying the constraint that when given to Python the first line has zero indentation.

E.g., the following two sets of input specify the same Python program.

```
python
  print 'Hello'
  print 'Goodbye'
```

```
end

python
print 'Hello'
print 'Goodbye'
end
```

whereas this program is in error since the indentation of the second line is less than that of the first.

```
python
  print 'Hello'
print 'Goodbye'
end
```

The Python program is not executed until the following directive is encountered

```
task python
```

which is to maintain consistency with the behavior of NWChem in general. *The program is executed by all nodes.* This enables the full functionality and speed of NWChem to be accessible from Python, but there are some gotchas

- Print statements and other output will be executed by all nodes so you will get a lot more output than probably desired unless the output is restricted to just one node (by convention node zero).
- The calls to NWChem functions are all collective (i.e., all nodes must execute them). If these calls are not made collectively your program may deadlock (i.e., cease to make progress).
- When writing to the database (`rtdb_put()`) it is the data from node zero that is written.
- NWChem overrides certain default signal handlers so care must be taken when creating processes (see Section 32.3.11).

32.2 NWChem extensions

Since we have little experience using Python, the NWChem-Python interface might change in a non-backwardly compatible fashion as we discover better ways of providing useful functionality. We would appreciate suggestions about useful things that can be added to the NWChem-Python interface. In principle, nearly any Fortran or C routine within NWChem can be extended to Python, but we are also interested in ideas that will enable users to build completely new things. For instance, how about being able to define your own energy functions that can be used with the existing optimizers or dynamics package?

Python has been extended with a module named `"nwchem"` which is automatically imported and contains the following NWChem-specific commands. They all handle NWChem-related errors by raising the exception `"NWChemError"`, which may be handled in the standard Python manner (see Section 32.3.9).

- `input_parse(string)` — invokes the standard NWChem input parser with the data in `string` as input. Note that the usual behavior of NWChem will apply — the parser only reads input up to either end of input or until a `TASK` directive is encountered (the task directive is *not* executed by the parser).
- `task_energy(theory)` — returns the energy as if computed with the NWChem directive `TASK ENERGY <THEORY>`.

- `task_gradient(theory)` — returns a tuple `(energy,gradient)` as if computed with the NWChem directive `TASK GRADIENT <THEORY>`.
- `task_optimize(theory)` — returns a tuple `(energy,gradient)` as if computed with the NWChem directive `TASK OPTIMIZE <THEORY>`. The energy and gradient will be those at the last point in the optimization and consistent with the current geometry in the database.
- `ga_nodeid()` — returns the number of the parallel process.
- `rtdb_print(print_values)` — prints the contents of the RTDB. If `print_values` is 0, only the keys are printed, if it is 1 then the values are also printed.
- `rtdb_put(name, values)` or `rtdb_put(name, values, type)` — puts the values into the database with the given name. In the first form, the type is inferred from the first value, and in the second form the type is specified using the last argument as one of `INT`, `DBL`, `LOGICAL`, or `CHAR`.
- `rtdb_get(name)` — returns the data from the database associated with the given name.

An example below (Section 32.3.10) explains, in lieu of a Python wrapper for the geometry object, how to obtain the Cartesian molecular coordinates directly from the database.

32.3 Examples

Several examples will provide the best explanation of how the extensions are used, and how Python might prove useful.

32.3.1 Hello world

```
python
  print 'Hello world from process ', ga_nodeid()
end

task python
```

This input prints the traditional greeting from each parallel process.

32.3.2 Scanning a basis exponent

```
geometry units au
  O 0 0 0; H 0 1.430 -1.107; H 0 -1.430 -1.107
end

python
  exponent = 0.1
  while (exponent <= 2.01):
     input_parse('''
        basis noprint
           H library 3-21g; O library 3-21g; O d; %f 1.0
        end
```

```
      ''' % (exponent))
      print ' exponent = ', exponent, ' energy = ', task_energy('scf')
      exponent = exponent + 0.1
  end

  print none

  task python
```

This program augments a 3-21g basis for water with a d-function on oxygen and varies the exponent from 0.1 to 2.0 in steps of 0.1, printing the exponent and energy at each step.

The geometry is input as usual, but the basis set input is embedded inside a call to `input_parse()` in the Python program. The standard Python string substitution is used to put the current value of the exponent into the basis set (replacing the `%f`) before being parsed by NWChem. The energy is returned by `task_energy('scf')` and printed out. The `print none` in the NWChem input switches off all NWChem output so all you will see is the output from your Python program.

Note that execution in parallel may produce unwanted output since all process execute the print statement inside the Python program.

Look in the NWChem `contrib` directory for a routine that makes the above task easier.

32.3.3 Scanning a basis exponent revisited.

```
  geometry units au
    O 0 0 0; H 0 1.430 -1.107; H 0 -1.430 -1.107
  end

  print none

  python
    if (ga_nodeid() == 0): plotdata = open("plotdata",'w')

    def energy_at_exponent(exponent):
       input_parse('''
          basis noprint
             H library 3-21g; O library 3-21g; O d; %f 1.0
          end
       ''' % (exponent))
       return task_energy('scf')

    exponent = 0.1
    while exponent <= 2.01:
       energy = energy_at_exponent(exponent)
       if (ga_nodeid() == 0):
          print ' exponent = ', exponent, ' energy = ', energy
          plotdata.write('%f %f\n' % (exponent , energy))
       exponent = exponent + 0.1

    if (ga_nodeid() == 0): plotdata.close()
  end
```

```
task python
```

This input performs exactly the same calculation as the previous one, but uses a slightly more sophisticated Python program, also writes the data out to a file for easy visualization with a package such as `gnuplot`, and protects write statements to prevent duplicate output in a parallel job. The only significant differences are in the Python program. A file called `"plotdata"` is opened, and then a procedure is defined which given an exponent returns the energy. Next comes the main loop that scans the exponent through the desired range and prints the results to standard output and to the file. When the loop is finished the additional output file is closed.

32.3.4 Scanning a geometric variable

```
python
  geometry = '''
    geometry noprint; symmetry d2h
       C 0 0 %f; H 0  0.916 1.224
    end
  '''
  x = 0.6
  while (x < 0.721):
    input_parse(geometry % x)
    energy = task_energy('scf')
    print ' x = %5.2f   energy = %10.6f' % (x, energy)
    x = x + 0.01
end

basis; C library 6-31g*; H library 6-31g*; end

print none

task python
```

This scans the bond length in ethene from 1.2 to 1.44 in steps of 0.2 computing the energy at each geometry. Since it is using D_{2h} symmetry the program actually uses a variable (`x`) that is half the bond length.

Look in the NWChem `contrib` directory for a routine that makes the above task easier.

32.3.5 Scan using the BSSE counterpoise corrected energy

```
basis spherical
  Ne library cc-pvdz; BqNe library Ne cc-pvdz
  He library cc-pvdz; BqHe library He cc-pvdz
end

mp2; tight; freeze core atomic; end

print none
```

```
python noprint
  supermolecule = 'geometry noprint;   Ne 0 0 0;   He 0 0 %f; end\n'
  fragment1     = 'geometry noprint;   Ne 0 0 0; BqHe 0 0 %f; end\n'
  fragment2     = 'geometry noprint; BqNe 0 0 0;   He 0 0 %f; end\n'

  def energy(geometry):
    input_parse(geometry + 'scf; vectors atomic; end\n')
    return task_energy('mp2')

  def bsse_energy(z):
    return energy(supermolecule % z) - \
           energy(fragment1 % z) - \
           energy(fragment2 % z)
  z = 3.3
  while (z < 4.301):
    e = bsse_energy(z)
    if (ga_nodeid() == 0):
      print ' z = %5.2f   energy = %10.7f ' % (z, e)
    z = z + 0.1
end

task python
```

This example scans the He—Ne bond-length from 3.3 to 4.3 and prints out the BSSE counterpoise corrected MP2 energy.

The basis set is specified as usual, noting that we will need functions on ghost centers to do the counterpoise correction. The Python program commences by defining strings containing the geometry of the super-molecule and two fragments, each having one variable to be substituted. Next, a function is defined to compute the energy given a geometry, and then a function is defined to compute the counterpoise corrected energy at a given bond length. Finally, the bond length is scanned and the energy printed. When computing the energy, the atomic guess has to be forced in the SCF since by default it will attempt to use orbitals from the previous calculation which is not appropriate here.

Since the counterpoise corrected energy is a linear combination of other standard energies, it is possible to compute the analytic derivatives term by term. Thus, combining this example and the next could yield the foundation of a BSSE corrected geometry optimization package.

32.3.6 Scan the geometry and compute the energy and gradient

```
basis noprint; H library sto-3g; O library sto-3g; end

python noprint
  print '   y     z     energy                gradient'
  print ' ----- ----- ---------- ------------------------------------'
  y = 1.2
  while y <= 1.61:
     z = 1.0
     while z <= 1.21:
        input_parse('''
```

```
          geometry noprint units atomic
             O 0   0   0
             H 0  %f -%f
             H 0 -%f -%f
          end
       ''' % (y, z, y, z))

       (energy,gradient) = task_gradient('scf')

       print ' %5.2f %5.2f %9.6f' % (y, z, energy),
       i = 0
       while (i < len(gradient)):
          print '%5.2f' % gradient[i],
          i = i + 1
       print ''
       z = z + 0.1
    y = y + 0.1
end

print none

task python
```

This program illustrates evaluating the energy and gradient by calling `task_gradient()`. A water molecule is scanned through several C_{2v} geometries by varying the y and z coordinates of the two hydrogen atoms. At each geometry the coordinates, energy and gradient are printed.

The basis set (sto-3g) is input as usual. The two while loops vary the y and z coordinates. These are then substituted into a geometry which is parsed by NWChem using `input_parse()`. The energy and gradient are then evaluated by calling `task_gradient()` which returns a tuple containing the energy (a scalar) and the gradient (a vector or list). These are printed out exploiting the Python convention that a print statement ending in a comma does not print end-of-line.

32.3.7 Reaction energies varying the basis set

```
mp2; freeze atomic; end

print none

python
  energies = {}
  c2h4 = 'geometry noprint; symmetry d2h; \
          C 0 0 0.672; H 0 0.935 1.238; end\n'
  ch4  = 'geometry noprint; symmetry td; \
          C 0 0 0; H 0.634 0.634 0.634; end\n'
  h2   = 'geometry noprint; H 0 0 0.378; H 0 0 -0.378; end\n'

  def energy(basis, geometry):
    input_parse('''
      basis spherical noprint
        c library %s ; h library %s
```

```
      end
    ''' % (basis, basis))
    input_parse(geometry)
    return task_energy('mp2')

  for basis in ('sto-3g', '6-31g', '6-31g*', 'cc-pvdz', 'cc-pvtz'):
     energies[basis] =   2*energy(basis, ch4) \
                       - 2*energy(basis, h2) - energy(basis, c2h4)
     if (ga_nodeid() == 0): print basis, ' %8.6f' % energies[basis]
end

task python
```

In this example the reaction energy for $2H_2 + C_2H_4 \rightarrow 2CH_4$ is evaluated using MP2 in several basis sets. The geometries are fixed, but could be re-optimized in each basis. To illustrate the useful associative arrays in Python, the reaction energies are put into the associative array `energies` — note its declaration at the top of the program.

32.3.8 Using the database

```
python
  rtdb_put("test_int2", 22)
  rtdb_put("test_int", [22, 10, 3],    INT)
  rtdb_put("test_dbl", [22.9, 12.4, 23.908],  DBL)
  rtdb_put("test_str", "hello", CHAR)
  rtdb_put("test_logic", [0,1,0,1,0,1], LOGICAL)
  rtdb_put("test_logic2", 0, LOGICAL)

  rtdb_print(1)

  print "test_str    = ", rtdb_get("test_str")
  print "test_int    = ", rtdb_get("test_int")
  print "test_in2    = ", rtdb_get("test_int2")
  print "test_dbl    = ", rtdb_get("test_dbl")
  print "test_logic  = ", rtdb_get("test_logic")
  print "test_logic2 = ", rtdb_get("test_logic2")
end

task python
```

This example illustrates how to access the database from Python.

32.3.9 Handling exceptions from NWChem

```
geometry; he 0 0 0; he 0 0 2; end
basis; he library 3-21g; end
scf; maxiter 1; end

python
```

```
    try:
      task_energy('scf')
    except NWChemError, message:
      print 'Error from NWChem ... ', message
  end

  task python
```

The above test program shows how to handle exceptions generated by NWChem by forcing an SCF calculation on He_2 to fail due to insufficient iterations.

If an NWChem command fails it will raise the exception `"NWChemError"` (case sensitive) unless the error was fatal. If the exception is not caught, then it will cause the entire Python program to terminate with an error. This Python program catches the exception, prints out the message, and then continues as if all was well since the exception has been handled.

If your Python program detects an error, raise an unhandled exception. Do not call `exit(1)` since this may circumvent necessary clean-up of the NWChem execution environment.

32.3.10 Accessing geometry information — a temporary hack

In an ideal world the geometry and basis set objects would have full Python wrappers, but until then a back-door solution will have to suffice. We've already seen how to use `input_parse()` to put geometry (and basis) data into NWChem, so it only remains to get the geometry data back after it has been updated by a geometry optimzation or some other operation.

The following Python procedure retrieves the coordinates in the same units as initially input for a geometry of a given name. Its full source is included in the NWChem `contrib` directory.

```
  def geom_get_coords(name):
    try:
      actualname = rtdb_get(name)
    except NWChemError:
      actualname = name
    coords = rtdb_get('geometry:' + actualname + ':coords')
    units  = rtdb_get('geometry:' + actualname + ':user units')
    if (units == 'a.u.'):
      factor = 1.0
    elif (units == 'angstroms'):
      factor = rtdb_get('geometry:'+actualname+':angstrom_to_au')
    else:
      raise NWChemError,'unknown units'
    i = 0
    while (i < len(coords)):
      coords[i] = coords[i] / factor
      i = i + 1
    return coords
```

A geometry (see Section 6) with name `NAME` has its coordinates (in atomic units) stored in the database entry `geometry:NAME:coords`. A minor wrinkle here is that indirection is possible (and used by the optimizers) so that we must first check if `NAME` actually points to another name. In the program this is done in the first `try...except` sequence. With the actual name of the geometry, we can get the coordinates. Any exceptions are passed up to the caller. The rest of the code is just to convert back into the initial input units — only atomic units or Angstrøms are handled in this simple example. Returned is a list of the atomic coordinates in the same units as your initial input.

The routine is used as follows

```
coords = geom_get_coords('geometry')
```

or, if you want better error handling

```
try:
  coords = geom_get_coords('geometry')
except NWChemError,message:
  print 'Coordinates for geometry not found ', message
else:
  print coords
```

This is very dirty and definitely not supported from one release to another, but, browsing the output of `rtdb_print()` at the end of a calculation is a good way to find stuff. To be on safer ground, look in the programmers manual since some of the high-level routines do pass data via the database in a well-defined and supported manner. *Be warned* — you must be very careful if you try to modify data in the database. The input parser does many important things that are not immediately apparent (e.g., ensure the geometry is consistent with the point group, mark the SCF as not converged if the SCF options are changed, ...). Where at all possible your Python program should generate standard NWChem input and pass it to `input_parse()` rather than setting parameters directly in the database.

32.3.11 Scaning a basis exponent yet again — plotting and handling child processes

```
geometry units au
  O 0 0 0; H 0 1.430 -1.107; H 0 -1.430 -1.107
end

print none

python
  import Gnuplot, time, signal

  def energy_at_exponent(exponent):
     input_parse('''
        basis noprint
           H library 3-21g; O library 3-21g; O d; %f 1.0
        end
     ''' % (exponent))
     return task_energy('scf')

  data = []
  exponent = 0.5
  while exponent <= 0.6:
```

```
      energy = energy_at_exponent(exponent)
      print ' exponent = ', exponent, ' energy = ', energy
      data = data + [[exponent,energy]]
      exponent = exponent + 0.02

   if (ga_nodeid() == 0):
      signal.signal(signal.SIGCHLD, signal.SIG_DFL)
      g = Gnuplot.Gnuplot()
      g('set data style linespoints')
      g.plot(data)
      time.sleep(30)  # 30s to look at the plot

end

task python
```

This illustrates how to handle signals from terminating child processes and how to generate simple plots on UNIX systems. The example from Section 32.3.3 is modified so that instead of writing the data to a file for subsequent visualization, it is saved for subsequent visualization with Gnuplot (you'll need both Gnuplot and the corresponding package for Python in your `PYTHONPATH`. Look at http://monsoon.harvard.edu/ mhagger/download).

The issue is that NWChem traps various signals from the O/S that usually indicate bad news in order to provide better error handling and reliable clean-up of shared, parallel resources. One of these signals is `SIGCHLD` which is generated whenever a child process terminates. If you want to create child processes within Python, then the NWChem handler for `SIGCHLD` must be replaced with the default handler. There seems to be no easy way to restore the NWChem handler after the child has completed, but this should have no serious side effect.

32.4 Troubleshooting

Common problems with Python programs inside NWChem.

1. You get the message

   ```
   0:python_input: indentation must be >= that of first line: 4
   ```

   This indicates that NWChem thinks that a line is less indented than the first line. If this is not the case then perhaps there is a tab in your input which NWChem treats as a single space character but appears to you as more spaces. Try running `untabify` in Emacs. The problem could also be the `END` directive that terminates the `PYTHON` compound directive — since Python also has an `end` statement. To avoid confusion the `END` directive for NWChem *must* be at the start of the line.

2. Your program hangs or deadlocks — most likely you have a piece of code that is restricted to executing on a subset of the processors (perhaps just node 0) but is calling (perhaps indirectly) a function that must execute on all nodes.

Chapter 33

Interfaces to Other Programs

NWChem has interfaces to several different packages which are listed below. In general, the NWChem authors work with the authors of the other packages to make sure that the interface works. However, any problems with the interface should be reported to the `nwchem-support@emsl.pnl.gov` e-mail list.

33.1 NBO — Natural Bond Orbital Analysis

```
  NBO
    ...
  END
```

This directive is used to run the NBO package within NWChem. (To only print out an input file for NBO, see Section 24.1.1.) Currently, this analysis module only runs sequentially. This restriction will soon go away.

Inside the NBO block are the typical commands that would be needed for NBO (Please see an NBO user's manual for more information.). The following directive is needed to execute NBO.

```
task nbo
```

As an example:

```
title "Methylamine...rhf/3-21g//Pople-Gordon standard geometry"

start methylamine

echo

memory 8 mw

basis
 C library 3-21g
 N library 3-21g
 H library 3-21g
```

```
end

geometry
 C           .052902     .711852     .000000
 N           .052902    -.758148     .000000
 H          -.974760    1.075185     .000000
 H           .566733    1.075185     .889981
 H           .566733    1.075185    -.889981
 H          -.423217   -1.094815     .824662
 H          -.423217   -1.094815    -.824662
 symmetry c1
end

task SCF energy

nbo
 $nbo cmo $end
end

task nbo
```

Chapter 34

Acknowledgments

This work was supported by funds from the Environmental and Molecular Sciences Laboratory Construction Project at Pacific Northwest National Laboratory. Development of some of the parallel programming tools and algorithms employed by NWChem was performed under the auspices of the High Performance Computing and Communications Program of the Mathematical, Information, and Computational Sciences Division, U.S. Department of Energy. Pacific Northwest National Laboratory is operated by Battelle Memorial Institute for the U.S. Department of Energy under Contract DE-AC06-76RLO 1830.

Appendix A

Standard Basis Sets

Basis sets and effective core potentials were obtained (7/6/2000) from the Extensible Computational Chemistry Environment (ECCE) Basis Set Database, as developed and distributed by the Molecular Science Computing Facility, Environmental and Molecular Sciences Laboratory which is part of the Pacific Northwest National Laboratory, P.O. Box 999, Richland, Washington 99352, USA, and is funded by the U.S. Department of Energy. The Pacific Northwest National Laboratory is a multi-program laboratory operated by Battelle Memorial Institute for the U.S. Department of Energy under contract DE-AC06-76RLO. Contact David Feller (`df_feller@pnl.gov`) or Deborah Gracio (`gracio@pnl.gov`) for further information.

The names in the NWChem library are consistent with those in the ECCE database and thus may include spaces. The standard NWChem input routines require that strings including spaces are enclosed in quotation marks (`"..."`) or that blanks are escaped with a backslash. As a convenience, basis set names may also have the blanks replaced with underscores. Thus, the following all yield the same basis set for oxygen

```
oxygen library "DZP + Diffuse (Dunning)"
oxygen library DZP\ +\ Diffuse\ (Dunning)
oxygen library DZP_+_Diffuse_(Dunning)
```

Case may be ignored when specifying basis set names, but otherwise names should be specified exactly as provided below. A good method is just to cut/paste from the WWW pages since they were generated electronically from the library source.

Errors found in the basis set library of NWChem version 3.3 have been corrected in the current library of NWChem version 4.0. The changes are listed below:

1. Basis Set `"Hay-Wadt VDZ (n+1) ECP"`
   Element: Ag
   Correction of contraction coefficient in first s contraction.

2. Basis Set `"LANL2DZ ECP"`
   Element: Ag
   Correction of contraction coefficient in first s contraction.

3. Basis Set `"CRENBS ECP"`
   Element: All elements in the `"CRENBS ECP"` Family
   Complete revision of basis sets. The original sets were completely uncontracted and some of the first row transition metals were missing all of their d functions. The current sets relflect the minimal contraction of the original papers.

4. ECP "SBKJC VDZ ECP"
   Element: Ce
   Correction of switched integer powers for d-f component.

Relativistic contractions of standard basis sets for use in the Douglas-Kroll and Dyall-modified-Dirac method have also been included in the library. These are identified by tags following the standard basis set name.

For the Douglas-Kroll method the tag `dk` should be specified:

```
oxygen library cc-pVDZ_DK
```

For the Dyall-modified-Dirac (DMD) method three tags should be specified. The first is a tag for the nuclear model, which can be `pt` or `fi` for a point or a finite Gaussian model (see Section6). The second is a tag for the relativistic Hamiltonian: `sf` is for the spin-free modified Dirac Hamiltonian. The third tags the component type, `fw` for the atomic FW transformed large component, `lc` for the large component and `sc` for the small component:

```
oxygen library cc-pvdz_pt_sf_fw
oxygen library cc-pvdz_pt_sf_lc
```

Basis sets which are available with either the DmD or DK contractions are indicated in the list below.

Here is a list of known all-electron non-relativistic, DK and DmD basis sets, effective core potentials with their respective basis sets, and fitting basis sets along with the elements included for each. Additional information about each basis set in the NWChem library can be obtained from the online EMSL Gaussian Basis Set library.

Standard all-electron basis sets:

1. Basis Set "STO-2G" (number of atoms 20)
   H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca
2. Basis Set "STO-3G" (number of atoms 52)
   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
3. Basis Set "STO-6G" (number of atoms 36)
   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
4. Basis Set "STO-3G*" (number of atoms 18)
   H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar
5. Basis Set "3-21G" (number of atoms 55)
   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
6. Basis Set "3-21++G" (number of atoms 18)
   H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar
7. Basis Set "3-21G*" (number of atoms 17)
   H He Li Be B C N O F Ne Na Mg Al Si P S Cl

8. Basis Set `"3-21++G*"` (number of atoms 18)
   H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

9. Basis Set `"3-21GSP"` (number of atoms 18)
   H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

10. Basis Set `"4-22GSP"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

11. Basis Set `"4-31G"` (number of atoms 13)
    H He Li Be B C N O F Ne P S Cl

12. Basis Set `"6-31G"` (number of atoms 30)
    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

13. Basis Set `"6-31++G"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

14. Basis Set `"6-31G*"` (number of atoms 30)
    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

15. Basis Set `"6-31G**"` (number of atoms 30)
    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

16. Basis Set `"6-31+G*"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

17. Basis Set `"6-31++G*"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

18. Basis Set `"6-31++G**"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

19. Basis Set `"6-31G(3df,3pd)"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

20. Basis Set `"6-31G-Blaudeau"` (number of atoms 2)
    K Ca

21. Basis Set `"6-31G*-Blaudeau"` (number of atoms 2)
    K Ca

22. Basis Set `"6-311G"` (number of atoms 24)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar Ga Ge As Se Br Kr

23. Basis Set `"6-311G*"` (number of atoms 24)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar Ga Ge As Se Br Kr

24. Basis Set `"6-311G**"` (number of atoms 24)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar Ga Ge As Se Br Kr

25. Basis Set `"6-311++G**"` (number of atoms 10)
    H He Li Be B C N O F Ne

26. Basis Set `"6-311++G(2d,2p)"` (number of atoms 10)
    H He Li Be B C N O F Ne

27. Basis Set `"6-311G(2df,2pd)"` (number of atoms 10)
    H He Li Be B C N O F Ne

28. Basis Set `"6-311+G*"` (number of atoms 10)
    H He Li Be B C N O F Ne

29. Basis Set `"6-311++G(3df,3pd)"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

30. Basis Set `"MINI (Huzinaga)"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar

31. Basis Set `"MINI (Scaled)"` (number of atoms 20)
    H He Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca

32. Basis Set `"MIDI (Huzinaga)"` (number of atoms 18)
    H He Li Be B C N O F Ne Na Al Si P S Cl Ar K

33. Basis Set `"MIDI!"` (number of atoms 11)
    H C N O F Si P S Cl Br I

34. Basis Set `"SV (Dunning-Hay)"` (number of atoms 9)
    H Li Be B C N O F Ne

35. Basis Set `"SVP (Dunning-Hay)"` (number of atoms 9)
    H Li Be B C N O F Ne

36. Basis Set `"SVP + Diffuse (Dunning-Hay)"` (number of atoms 9)
    H Li Be B C N O F Ne

37. Basis Set `"TZ (Dunning)"` (number of atoms 8)
    H Li B C N O F Ne

38. Basis Set `"Chipman DZP + Diffuse"` (number of atoms 6)
    H B C N O F

39. Basis Set `"DZ (Dunning)"` (number of atoms 12)
    H B C N O F Ne Al Si P S Cl

40. Basis Set `"DZP (Dunning)"` (number of atoms 12)
    H B C N O F Ne Al Si P S Cl

41. Basis Set `"DZP + Diffuse (Dunning)"` (number of atoms 7)
    H B C N O F Ne

42. Basis Set `"cc-pVDZ"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

43. Basis Set `"cc-pVTZ"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

44. Basis Set `"cc-pVQZ"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

45. Basis Set `"cc-pV5Z"` (number of atoms 21)
    H He Li B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

46. Basis Set `"cc-pV6Z"` (number of atoms 14)
H He B C N O F Ne Al Si P S Cl Ar

47. Basis Set `"cc-pCVDZ"` (number of atoms 6)
B C N O F Ne

48. Basis Set `"cc-pCVTZ"` (number of atoms 6)
B C N O F Ne

49. Basis Set `"cc-pCVQZ"` (number of atoms 6)
B C N O F Ne

50. Basis Set `"cc-pCV5Z"` (number of atoms 6)
B C N O F Ne

51. Basis Set `"aug-cc-pVDZ"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

52. Basis Set `"aug-cc-pVTZ"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

53. Basis Set `"aug-cc-pVQZ"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

54. Basis Set `"aug-cc-pV5Z"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

55. Basis Set `"aug-cc-pV6Z"` (number of atoms 14)
H He B C N O F Ne Al Si P S Cl Ar

56. Basis Set `"aug-cc-pCVDZ"` (number of atoms 5)
B C N O F

57. Basis Set `"aug-cc-pCVTZ"` (number of atoms 6)
B C N O F Ne

58. Basis Set `"aug-cc-pCVQZ"` (number of atoms 6)
B C N O F Ne

59. Basis Set `"aug-cc-pCV5Z"` (number of atoms 5)
B C N O F

60. Basis Set `"d-aug-cc-pVDZ"` (number of atoms 8)
H He B C N O F Ne

61. Basis Set `"d-aug-cc-pVTZ"` (number of atoms 8)
H He B C N O F Ne

62. Basis Set `"d-aug-cc-pVQZ"` (number of atoms 8)
H He B C N O F Ne

63. Basis Set `"d-aug-cc-pV5Z"` (number of atoms 8)
H He B C N O F Ne

64. Basis Set `"d-aug-cc-pV6Z"` (number of atoms 5)
H B C N O

65. Basis Set `"GAMESS VTZ"` (number of atoms 8)
    H Be B C N O F Ne

66. Basis Set `"GAMESS PVTZ"` (number of atoms 8)
    H Be B C N O F Ne

67. Basis Set `"Partridge Uncontr. 1"` (number of atoms 34)
    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

68. Basis Set `"Partridge Uncontr. 2"` (number of atoms 34)
    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

69. Basis Set `"Partridge Uncontr. 3"` (number of atoms 28)
    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

70. Basis Set `"Ahlrichs VDZ"` (number of atoms 36)
    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

71. Basis Set `"Ahlrichs pVDZ"` (number of atoms 36)
    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

72. Basis Set `"Ahlrichs VTZ"` (number of atoms 36)
    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

73. Basis Set `"Binning/Curtiss SV"` (number of atoms 6)
    Ga Ge As Se Br Kr

74. Basis Set `"Binning/Curtiss VTZ"` (number of atoms 6)
    Ga Ge As Se Br Kr

75. Basis Set `"Binning/Curtiss SVP"` (number of atoms 6)
    Ga Ge As Se Br Kr

76. Basis Set `"Binning/Curtiss VTZP"` (number of atoms 6)
    Ga Ge As Se Br Kr

77. Basis Set `"McLean/Chandler VTZ"` (number of atoms 8)
    Na Mg Al Si P S Cl Ar

78. Basis Set `"SV + Rydberg (Dunning-Hay)"` (number of atoms 9)
    H Li Be B C N O F Ne

79. Basis Set `"SVP + Rydberg (Dunning-Hay)"` (number of atoms 9)
    H Li Be B C N O F Ne

80. Basis Set `"SVP + Diffuse + Rydberg"` (number of atoms 9)
    H Li Be B C N O F Ne

81. Basis Set `"DZ + Rydberg (Dunning)"` (number of atoms 12)
    H B C N O F Ne Al Si P S Cl

82. Basis Set `"DZP + Rydberg (Dunning)"` (number of atoms 12)
    H B C N O F Ne Al Si P S Cl

83. Basis Set `"DZ + Double Rydberg (Dunning-Hay)"` (number of atoms 12)
    H B C N O F Ne Al Si P S Cl

84. Basis Set `"SV + Double Rydberg (Dunning-Hay)"` (number of atoms 9)
    H Li Be B C N O F Ne

85. Basis Set `"Wachters+f"` (number of atoms 9)
    Sc Ti V Cr Mn Fe Co Ni Cu

86. Basis Set `"Bauschlicher ANO"` (number of atoms 9)
    Sc Ti V Cr Mn Fe Co Ni Cu

87. Basis Set `"NASA Ames ANO"` (number of atoms 12)
    H B C N O F Ne Al P Ti Fe Ni

88. Basis Set `"Sadlej pVTZ"` (number of atoms 18)
    H Li Be C N O F Na Mg Si P S Cl K Ca Br Rb Sr

89. Basis Set `"WTBS"` (number of atoms 84)
    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 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

Resolution of Identity (RI) fitting basis sets:

1. Basis Set `"cc-pVDZ-fit2-1"` (number of atoms 10)
   H He Li Be B C N O F Ne

2. Basis Set `"cc-pVTZ-fit2-1"` (number of atoms 10)
   H He Li Be B C N O F Ne

Density functional specific basis sets:

1. Basis Set `"DZVP (DFT Orbital)"` (number of atoms 54)
   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

2. Basis Set `"DZVP2 (DFT Orbital)"` (number of atoms 25)
   H He Li Be B C N O F Al Si P S Cl Ar Sc Ti V Cr Mn Fe Co Ni Cu Zn

3. Basis Set `"TZVP (DFT Orbital)"` (number of atoms 11)
   H C N O F Al Si P S Cl Ar

Density functional Coulomb and Exchange fitting basis sets:

1. Basis Set `"DGauss A1 DFT Coulomb Fitting"` (number of atoms 54)
   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

2. Basis Set `"DGauss A1 DFT Exchange Fitting"` (number of atoms 54)
   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

3. Basis Set `"DGauss A2 DFT Coulomb Fitting"` (number of atoms 25)
   H He Li Be B C N O F Al Si P S Cl Ar Sc Ti V Cr Mn Fe Co Ni Cu Zn

4. Basis Set `"DGauss A2 DFT Exchange Fitting"` (number of atoms 25)
   H He Li Be B C N O F Al Si P S Cl Ar Sc Ti V Cr Mn Fe Co Ni Cu Zn

5. Basis Set `"DeMon Coulomb Fitting"` (number of atoms 54)
   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

6. Basis Set `"Ahlrichs Coulomb Fitting"` (number of atoms 50)
   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

Effective core potentials and their respective basis sets:

1. Basis Set `"Hay-Wadt MB (n+1) ECP"` (number of atoms 24)
   K Ca Sc Ti V Cr Mn Fe Co Ni Cu Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cs Ba

2. ECP `"Hay-Wadt MB (n+1) ECP"` (number of atoms 24)
   K Ca Sc Ti V Cr Mn Fe Co Ni Cu Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cs Ba

3. Basis Set `"Hay-Wadt VDZ (n+1) ECP"` (number of atoms 24)
   K Ca Sc Ti V Cr Mn Fe Co Ni Cu Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cs Ba

4. ECP `"Hay-Wadt VDZ (n+1) ECP"` (number of atoms 24)
   K Ca Sc Ti V Cr Mn Fe Co Ni Cu Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cs Ba

5. Basis Set `"LANL2DZ ECP"` (number of atoms 67)
   H 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 Hf Ta W Re Os Ir Pt Au U Np Pu

6. ECP `"LANL2DZ ECP"` (number of atoms 57)
   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 Hf Ta W Re Os Ir Pt Au U Np

7. Basis Set `"SBKJC VDZ ECP"` (number of atoms 73)
   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 Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn

8. ECP `"SBKJC VDZ ECP"` (number of atoms 71)
   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 Hf Ta W Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn

9. Basis Set `"CRENBL ECP"` (number of atoms 116)
   H 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 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 Un Uu Ub Ut Uq Up Uh Us Uo

10. ECP `"CRENBL ECP"` (number of atoms 115)
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 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 Un Uu Ub Ut Uq Up Uh Us Uo

11. Basis Set `"CRENBS ECP"` (number of atoms 50)
Sc Ti V Cr Mn Fe Co Ni Cu Zn Y Zr Nb Mo Tc Ru Rh Pd Ag Cd La Hf Ta W Re Os Ir Pt Au Hg Pb Bi Po At Rn Rf Db Sg Bh Hs Mt Un Uu Ub Ut Uq Up Uh Us Uo

12. ECP `"CRENBS ECP"` (number of atoms 50)
Sc Ti V Cr Mn Fe Co Ni Cu Zn Y Zr Nb Mo Tc Ru Rh Pd Ag Cd La Hf Ta W Re Os Ir Pt Au Hg Pb Bi Po At Rn Rf Db Sg Bh Hs Mt Un Uu Ub Ut Uq Up Uh Us Uo

13. Basis Set `"Stuttgart RLC ECP"` (number of atoms 57)
Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca Zn Ga Ge As Se Br Kr Rb Sr In Sn Sb Te I Xe Cs Ba Hg Tl Pb Bi Po At Rn Ac Th Pa U Np Pu Am Cm Bk Cf Es Fm Md No Lr

14. ECP `"Stuttgart RLC ECP"` (number of atoms 57)
Li Be B C N O F Ne Na Mg Al Si P S Cl Ar K Ca Zn Ga Ge As Se Br Kr Rb Sr In Sn Sb Te I Xe Cs Ba Hg Tl Pb Bi Po At Rn Ac Th Pa U Np Pu Am Cm Bk Cf Es Fm Md No Lr

15. Basis Set `"Stuttgart RSC ECP"` (number of atoms 64)
K Ca Sc Ti V Cr Mn Fe Co Ni Cu Zn Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd Cs Ba Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb Hf Ta W Re Os Ir Pt Au Hg Ac Th Pa U Np Pu Am Cm Bk Cf Es Fm Md No Lr Db

16. ECP `"Stuttgart RSC ECP"` (number of atoms 64)
K Ca Sc Ti V Cr Mn Fe Co Ni Cu Zn Rb Sr Y Zr Nb Mo Tc Ru Rh Pd Ag Cd Cs Ba Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb Hf Ta W Re Os Ir Pt Au Hg Ac Th Pa U Np Pu Am Cm Bk Cf Es Fm Md No Lr Db

Douglas-Kroll (DK) all-electron basis sets:

1. Basis Set `"cc-pVDZ DK"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

2. Basis Set `"cc-pVTZ DK"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

3. Basis Set `"cc-pVQZ DK"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

4. Basis Set `"cc-pV5Z DK"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

Dyall's Modified Dirac (DmD) all-electron basis sets:

1. Basis Set `"cc-pvdz fi sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

2. Basis Set `"cc-pvdz fi sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

3. Basis Set `"cc-pvdz fi sf sc"` (number of atoms 20)
   H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

4. Basis Set `"cc-pvdz pt sf fw"` (number of atoms 20)
   H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

5. Basis Set `"cc-pvdz pt sf lc"` (number of atoms 20)
   H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

6. Basis Set `"cc-pvdz pt sf sc"` (number of atoms 20)
   H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

7. Basis Set `"cc-pvtz fi sf fw"` (number of atoms 20)
   H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

8. Basis Set `"cc-pvtz fi sf lc"` (number of atoms 20)
   H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

9. Basis Set `"cc-pvtz fi sf sc"` (number of atoms 20)
   H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

10. Basis Set `"cc-pvtz pt sf fw"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

11. Basis Set `"cc-pvtz pt sf lc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

12. Basis Set `"cc-pvtz pt sf sc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

13. Basis Set `"cc-pvqz fi sf fw"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

14. Basis Set `"cc-pvqz fi sf lc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

15. Basis Set `"cc-pvqz fi sf sc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

16. Basis Set `"cc-pvqz pt sf fw"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

17. Basis Set `"cc-pvqz pt sf lc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

18. Basis Set `"cc-pvqz pt sf sc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

19. Basis Set `"cc-pv5z fi sf fw"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

20. Basis Set `"cc-pv5z fi sf lc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

21. Basis Set `"cc-pv5z fi sf sc"` (number of atoms 20)
    H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

22. Basis Set `"cc-pv5z pt sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

23. Basis Set `"cc-pv5z pt sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

24. Basis Set `"cc-pv5z pt sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

25. Basis Set `"aug-cc-pvdz fi sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

26. Basis Set `"aug-cc-pvdz fi sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

27. Basis Set `"aug-cc-pvdz fi sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

28. Basis Set `"aug-cc-pvdz pt sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

29. Basis Set `"aug-cc-pvdz pt sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

30. Basis Set `"aug-cc-pvdz pt sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

31. Basis Set `"aug-cc-pvtz fi sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

32. Basis Set `"aug-cc-pvtz fi sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

33. Basis Set `"aug-cc-pvtz fi sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

34. Basis Set `"aug-cc-pvtz pt sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

35. Basis Set `"aug-cc-pvtz pt sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

36. Basis Set `"aug-cc-pvtz pt sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

37. Basis Set `"aug-cc-pvqz fi sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

38. Basis Set `"aug-cc-pvqz fi sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

39. Basis Set `"aug-cc-pvqz fi sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

40. Basis Set `"aug-cc-pvqz pt sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

41. Basis Set `"aug-cc-pvqz pt sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

42. Basis Set `"aug-cc-pvqz pt sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

43. Basis Set `"aug-cc-pv5z fi sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

44. Basis Set `"aug-cc-pv5z fi sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

45. Basis Set `"aug-cc-pv5z fi sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

46. Basis Set `"aug-cc-pv5z pt sf fw"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

47. Basis Set `"aug-cc-pv5z pt sf lc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

48. Basis Set `"aug-cc-pv5z pt sf sc"` (number of atoms 20)
H He B C N O F Ne Al Si P S Cl Ar Ga Ge As Se Br Kr

49. Basis Set `"dyall relpvdz fi sf fw"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

50. Basis Set `"dyall relpvdz fi sf lc"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

51. Basis Set `"dyall relpvdz fi sf sc"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

52. Basis Set `"dyall relrcpvdz fi sf fw"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

53. Basis Set `"dyall relrcpvdz fi sf lc"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

54. Basis Set `"dyall relrcpvdz fi sf sc"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

55. Basis Set `"dyall relpcpvdz fi sf fw"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

56. Basis Set `"dyall relpcpvdz fi sf lc"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

57. Basis Set `"dyall relpcpvdz fi sf sc"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

58. Basis Set `"dyall relapvdz fi sf fw"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

59. Basis Set `"dyall relapvdz fi sf lc"` (number of atoms 18)
Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn

60. Basis Set `"dyall relapvdz fi sf sc"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
61. Basis Set `"dyall relrcapvdz fi sf fw"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
62. Basis Set `"dyall relrcapvdz fi sf lc"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
63. Basis Set `"dyall relrcapvdz fi sf sc"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
64. Basis Set `"dyall relpcapvdz fi sf fw"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
65. Basis Set `"dyall relpcapvdz fi sf lc"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
66. Basis Set `"dyall relpcapvdz fi sf sc"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
67. Basis Set `"dyall reldz aug"` (number of atoms 15)
    Ga Ge As Se Br In Sn Sb Te I Tl Pb Bi Po At
68. Basis Set `"dyall nrpvdz fi"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
69. Basis Set `"dyall nrpcpvdz fi"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
70. Basis Set `"dyall nrrcpvdz fi"` (number of atoms 18)
    Ga Ge As Se Br Kr In Sn Sb Te I Xe Tl Pb Bi Po At Rn
71. Basis Set `"dyall nrapvdz fi"` (number of atoms 15)
    Ga Ge As Se Br In Sn Sb Te I Tl Pb Bi Po At
72. Basis Set `"dyall nrrcapvdz fi"` (number of atoms 15)
    Ga Ge As Se Br In Sn Sb Te I Tl Pb Bi Po At
73. Basis Set `"dyall nrpcapvdz fi"` (number of atoms 15)
    Ga Ge As Se Br In Sn Sb Te I Tl Pb Bi Po At

Appendix B

Sample input files

B.1 Water SCF calculation and geometry optimization in a 6-31g basis

The input file in section 2 performs a geometry optimization in a single task. A single point SCF energy calculation is performed and then restarted to perform the optimization (both could of course be performed in a single task).

B.1.1 Job 1. Single point SCF energy

```
start h2o
title "Water in 6-31g basis set"

geometry units au
  O      0.00000000    0.00000000    0.00000000
  H      0.00000000    1.43042809   -1.10715266
  H      0.00000000   -1.43042809   -1.10715266
end
basis
  H library 6-31g
  O library 6-31g
end
task scf
```

The final energy should be -75.983998.

B.1.2 Job 2. Restarting and perform a geometry optimization

```
restart h2o
title "Water geometry optimization"

task scf optimize
```

There is no need to specify anything that has not changed from the previous input deck, though it will do no harm to repeat it.

B.2 Compute the polarizability of Ne using finite field

B.2.1 Job 1. Compute the atomic energy

```
start ne
title "Neon"
geometry; ne 0 0 0; end
basis spherical
  ne library aug-cc-pvdz
end
scf; thresh 1e-10; end
task scf
```

The final energy should be -128.496350.

B.2.2 Job 2. Compute the energy with applied field

An external field may be simulated with point charges. The charges here apply a field of magnitude 0.01 atomic units to the atom at the origin. Since the basis functions have not been reordered by the additional centers we can also restart from the previous vectors, which is the default for a restart job.

```
restart ne
title "Neon in electric field"
geometry units atomic
  bq1 0 0 100 charge 50
  ne  0 0 0
  bq2 0 0 -100 charge -50
end
task scf
```

The final energy should be -128.496441, which together with the previous field-free result yields an estimate for the polarizability of 1.83 atomic units. Note that by default NWChem does not include the interaction between the two point charges in the total energy (section 6).

B.3 SCF energy of H_2CO using ECPs for C and O

The following will compute the SCF energy for formaldehyde with ECPs on the Carbon and Oxygen centers.

```
title "formaldehyde ECP deck"

start ecpchho

geometry units au
```

```
  C         0.000000  0.000000 -1.025176
  O         0.000000  0.000000  1.280289
  H         0.000000  1.767475 -2.045628
  H         0.000000 -1.767475 -2.045628
end

basis
  C  SP
   0.1675097360D+02 -0.7812840500D-01  0.3088908800D-01
   0.2888377460D+01 -0.3741108860D+00  0.2645728130D+00
   0.6904575040D+00  0.1229059640D+01  0.8225024920D+00
  C  SP
   0.1813976910D+00  0.1000000000D+01  0.1000000000D+01
  C  D
   0.8000000000D+00  0.1000000000D+01
  C  F
   0.1000000000D+01  0.1000000000D+01
  O  SP
   0.1842936330D+02 -0.1218775590D+00  0.5975796600D-01
   0.4047420810D+01 -0.1962142380D+00  0.3267825930D+00
   0.1093836980D+01  0.1156987900D+01  0.7484058930D+00
  O  SP
   0.2906290230D+00  0.1000000000D+01  0.1000000000D+01
  O  D
   0.8000000000D+00  0.1000000000D+01
  O  F
   0.1100000000D+01  0.1000000000D+01
  H  S
   0.1873113696D+02  0.3349460434D-01
   0.2825394365D+01  0.2347269535D+00
   0.6401216923D+00  0.8137573262D+00
  H  S    1 1.00
   0.1612777588D+00  0.1000000000D+01
end

ecp
  C nelec 2
  C ul
        1       80.0000000       -1.60000000
        1       30.0000000       -0.40000000
        2        0.5498205       -0.03990210
  C s
        0        0.7374760        0.63810832
        0      135.2354832       11.00916230
        2        8.5605569       20.13797020
  C p
        2       10.6863587       -3.24684280
        2       23.4979897        0.78505765
  O nelec 2
  O ul
        1       80.0000000       -1.60000000
```

```
      1      30.0000000      -0.40000000
      2       1.0953760      -0.06623814
  O s
      0       0.9212952       0.39552179
      0      28.6481971       2.51654843
      2       9.3033500      17.04478500
  O p
      2      52.3427019      27.97790770
      2      30.7220233     -16.49630500
end

scf
  vectors input hcore
  maxiter 20
end

task scf
```

This should produce the following output:

```
       Final RHF  results
       ------------------

         Total SCF energy =    -22.507927218024
      One electron energy =    -71.508730162974
      Two electron energy =     31.201960019808
 Nuclear repulsion energy =     17.798842925142
```

B.4 MP2 optimization and CCSD(T) on nitrogen

The following performs an MP2 geometry optimization followed by a CCSD(T) energy evaluation at the converged geometry. A Dunning correlation-consistent triple-zeta basis is used. The default of Cartesian basis functions must be overridden using the keyword `spherical` on the `BASIS` directive. The 1*s* core orbitals are frozen in both the MP2 and coupled-cluster calculations (note that these must separately specified). The final MP2 energy is -109.383276, and the CCSD(T) energy is -109.399662.

```
start n2

geometry
  symmetry d2h
  n 0 0 0.542
end

basis spherical
  n library cc-pvtz
end

mp2
  freeze core
```

```
end

task mp2 optimize

ccsd
  freeze core
end

task ccsd(t)
```

Appendix C

Examples of geometries using symmetry

Below are examples of the use of the `SYMMETRY` directive in the compound `GEOMETRY` directive (Section 6). The z axis is always the primary rotation axis. When in doubt about which axes and planes are used for the group elements, the keyword `print` may be added to the `SYMMETRY` directive to obtain this information.

C.1 C_s methanol

The σ_h plane is the xy plane.

```
geometry units angstroms
  C    0.11931097    -0.66334875     0.00000000
  H    1.20599017    -0.87824237     0.00000000
  H   -0.32267592    -1.15740001     0.89812652
  O   -0.01716588     0.78143468     0.00000000
  H   -1.04379735     0.88169812     0.00000000
end
```

C.2 C_{2v} water

The z axis is the C_2 axis and the σ_v may be either the xz or the yz planes.

```
geometry units au
  O     0.00000000    0.00000000    0.00000000
  H     0.00000000    1.43042809   -1.10715266
  symmetry group c2v
end
```

C.3 D_{2h} acetylene

Although acetylene has symmetry $D_{\infty h}$ the subgroup D_{2h} includes all operations that interchange equivalent atoms which is what determines how much speedup you gain from using symmetry in building a Fock matrix.

The C_2 axes are the x, y, and z axes. The σ planes are the xy, xz and yz planes. Generally, the unique atoms are placed to use the z as the primary rotational axis and use the xz or yz planes as the σ plane.

```
geometry units au
  symmetry group d2h
  C      0.000000000    0.000000000   -1.115108538
  H      0.000000000    0.000000000   -3.106737425
end
```

C.4 D_{2h} ethylene

The C_2 axes are the x, y, and z axes. The σ planes are the xy, xz and yz planes. Generally, the unique atoms are placed to use the z as the primary rotational axis and use the xz or yz planes as the σ plane.

```
geometry units angstroms
  C 0 0 0.659250
  H 0 0.916366 1.224352
  symmetry d2h
end
```

C.5 T_d methane

For ease of use, the primary C_3 axis should be the x=y=z axis. The 3 C_2 axes are the x, y, and z.

```
geometry units au
  c   0.0000000      0.0000000      0.0000000
  h   1.1828637      1.1828637      1.1828637
  symmetry group Td
end
```

C.6 I_h buckminsterfullerene

One of the C_5 axes is the z axis and the point of inversion is the origin.

```
geometry units angstroms # Bonds = 1.4445, 1.3945
  symmetry group Ih
  c   -1.2287651   0.0   3.3143121
end
```

C.7 S_4 porphyrin

The S_4 and C_2 rotation axis is the z axis. The reflection plane for the S_4 operation is the xy plane.

```
geometry units angstroms
   symmetry group s4

   fe                0.000  0.000  0.000
   h                 2.242  6.496 -3.320
   h                 1.542  4.304 -2.811
   c                 1.947  6.284 -2.433
   c                 1.568  4.987 -2.084
   h                 2.252  8.213 -1.695
   c                 1.993  7.278 -1.458
   h                 5.474 -1.041 -1.143
   c                 1.234  4.676 -0.765
   h                 7.738 -1.714 -0.606
   c                 0.857  3.276 -0.417
   h                 1.380 -4.889 -0.413
   c                 1.875  2.341 -0.234
   h                 3.629  3.659 -0.234
   c                 0.493 -2.964 -0.229
   c                 1.551 -3.933 -0.221
   c                 5.678 -1.273 -0.198
   c                 1.656  6.974 -0.144
   c                 3.261  2.696 -0.100
   n                 1.702  0.990 -0.035
end
```

C.8 D_{3h} iron penta-carbonyl

The C_3 axis is the z axis. The σ_h plane is the xy plane. One of the perpendicular C_2 axes is the x=y axis. One of the σ_v planes is the plane containing the x=y axis and the z axis. (The other axes and planes are generated by the C_3 operation.)

```
geometry units au
  symmetry group d3h

  fe        0.0         0.0         0.0

  c         0.0         0.0         3.414358
  o         0.0         0.0         5.591323

  c         2.4417087   2.4417087   0.0
  o         3.9810552   3.9810552   0.0
end
```

C.9 D_{3d} sodium crown ether

The C_3 axis is the z axis. The point of inversion is the origin. One of the perpendicular C_2 axes is the x=y axis. One of the σ_d planes is the plane containing the -x=y axis and the z axis.

Note that the oxygen atom is rotated in the x-y plane 15 degrees away from the y-axis so that it lies in a mirror plane. There is a total of six atoms generated from the unique oxygen, in contrast to twelve from each of the carbon and hydrogen atoms.

```
geometry units au
  symmetry D3d

 NA     .0000000000    .0000000000   .0000000000
 O     1.3384771885   4.9952647969   .1544089284
 H     6.7342048019  -0.6723850379  2.6581562148
 C     6.7599180056  -0.4844977035   .6136583870
 H     8.6497577017   0.0709194071   .0345361934

end
```

C.10 C_{3v} ammonia

The C_3 axis is the z axis. One of the σ_v planes is the plane containing the x=y axis and the z axis.

```
geometry units angstroms
  N 0     0     -0.055
  H 0.665 0.665 -0.481
  symmetry c3v
end
```

C.11 D_{6h} benzene

The C_6 axis is the z axis. The point of inversion is the origin. One of the 6 perpendicular C_2' axes is the x=y axis. (-x=y works as a C_2'' axis.) The σ_h plane is the xy plane. The σ_d planes contain the C_2'' axis and the z axis. The σ_v planes contain the C_2' axis and the z axis.

```
geometry units au
  C 1.855 1.855 0
  H 3.289 3.289 0
  symmetry D6h
end
```

C.12 C_{3h} BO_3H_3

The C_3 axis is the z axis. The σ_h plane is the xy plane.

```
geometry units au
  b 0 0 0
  o 2.27238285 1.19464491 0.00000000
  h 2.10895420 2.97347707 0.00000000
 symmetry C3h
end
```

C.13 D_{5d} ferrocene

The C_5 axis is the z axis. The center of inversion is the origin. One of the perpendicular C_2 axes is the x axis. One of the σ_d planes is the yz plane.

```
geometry units angstroms
  symmetry d5d

  fe 0 0     0
  c  0 1.194 1.789
  h  0 2.256 1.789
end
```

C.14 C_{4v} SF_5Cl

The C_4 axis is the z axis. The σ_v planes are the yz and the xz planes. The σ_d planes are: 1) the plane containing the x=y axis and the z axis and 2) the plane containing the -x=y axis and the z axis.

```
geometry units au
  S  0.00000000 0.00000000 -0.14917600
  Cl 0.00000000 0.00000000  4.03279700
  F  3.13694200 0.00000000 -0.15321800
  F  0.00000000 0.00000000 -3.27074500

  symmetry C4v
end
```

C.15 C_{2h} trans-dichloroethylene

The C_2 axis is the z axis. The origin is the inversion center. The σ_h plane is the xy plane.

```
geometry units angstroms
  C  0.65051239 -0.08305064 0
  Cl 1.75249381  1.30491767 0
  H  1.14820954 -1.04789741 0
  symmetry C2h
end
```

C.16 D_{2d} CH_2CCH_2

The C_2 axis is the z axis (z is also the S_4 axis). The x and y axes are the perpendicular C_2's. The σ_d planes are: 1) the plane containing the x=y axis and the z axis and 2) the plane containing the -x=y axis and the z axis.

```
geometry units angstroms
  symmetry d2d
  c 0     0     0
  c 0     0     1.300
  h 0.656 0.656 1.857
end
```

C.17 D_{5h} cyclopentadiene anion

The C_5 axis is the z axis (z is also the S_5 axis). The y axis is one of the perpendicular C_2 axes. The σ_h plane is the xy plane and one of the σ_d planes is the yz plane.

```
charge -1
geometry units angstroms
  symmetry d5h
  c 0 1.1853 0
  h 0 2.2654 0
end
```

C.18 D_{4h} gold tetrachloride

The C_4 axis is the z axis (z is also the S_4 axis). The C_2' axes are the x and y axes and the C_2'' axes are the x=y axis and the x=-y axis. The inversion center is the origin. The σ_h plane is the xy plane. The σ_v planes are the xz and yz planes and the σ_d planes are 1) the plane containing the x=-y axis and the z axis and 2) the plane containing the x=y axis and the z axis.

```
geometry units au
  Au 0 0     0
  Cl 0 4.033 0
  symmetry D4h
end
```

Appendix D

Running NWChem

The command required to invoke NWChem is machine dependent, whereas most of the NWChem input is machine independent[1] .

D.1 Sequential execution

To run NWChem sequentially on nearly all UNIX-based platforms simply use the command `nwchem` and provide the name of the input file as an argument (See section 2.1 for more information). This does assume that either `nwchem` is in your path or you have set an alias of `nwchem` to point to the appropriate executable.

Output is to standard output, standard error and Fortran unit 6 (usually the same as standard output). Files are created by default in the current directory, though this may be overridden in the input (section 5.2).

Generally, one will run a job with the following command:

```
nwchem input.nw >& input.out &
```

D.2 Parallel execution on UNIX-based parallel machines including workstation clusters using TCGMSG

These platforms require the use of the TCGMSG[2] `parallel` command and thus also require the definition of a process-group (or procgroup) file. The process-group file describes how many processes to start, what program to run, which machines to use, which directories to work in, and under which userid to run the processes. By convention the process-group file has a `.p` suffix.

The process-group file is read to end-of-file. The character `#` (hash or pound sign) is used to indicate a comment which continues to the next new-line character. Each line describes a cluster of processes and consists of the following whitespace separated fields:

```
userid hostname nslave executable workdir
```

[1] Machine dependence within the input arises from file names, machine specific resources, and differing services provided by the operating system.

[2] Where required TCGMSG is automatically built with NWChem.

- `userid` – The user-name on the machine that will be executing the process.
- `hostname` – The hostname of the machine to execute this process. If it is the same machine on which parallel was invoked the name must match the value returned by the command hostname. If a remote machine it must allow remote execution from this machine (see man pages for rlogin, rsh).
- `nslave` – The total number of copies of this process to be executing on the specified machine. Only "clusters" of identical processes specified in this fashion can use shared memory to communicate. If no shared memory is supported on machine `<hostname>` then only the value one (1) is valid.
- `executable` – Full path name on the host `<hostname>` of the image to execute. If `<hostname>` is the local machine then a local path will suffice.
- `workdir` – Full path name on the host `<hostname>` of the directory to work in. Processes execute a chdir() to this directory before returning from pbegin(). If specified as a "." then remote processes will use the login directory on that machine and local processes (relative to where parallel was invoked) will use the current directory of parallel.

For example, if your file `"nwchem.p"` contained the following

```
d3g681 pc 4 /msrc/apps/bin/nwchem /scr22/rjh
```

then 4 processes running NWChem would be started on the machine `pc` running as user `d3g681` in directory `"/scr22/rjh"`. To actually run this simply type:

```
parallel nwchem big_molecule.nw
```

N.B. : The first process specified (process zero) is the only process that

- opens and reads the input file, and
- opens and reads/updates the database.

Thus, if your file systems are physically distributed (e.g., most workstation clusters) you must ensure that process zero can correctly resolve the paths for the input and database files.

N.B. In releases of NWChem prior to 3.3 additional processes had to be created on workstation clusters to support remote access to shared memory. This is no longer the case. The TCGMSG process group file now just needs to refer to processes running NWChem.

D.3 Parallel execution on UNIX-based parallel machines including workstation clusters using MPI

To run with MPI, `parallel` should not be used. The way we usually run nwchem under MPI are the following

- using mpirun:

  ```
  mpirun -np 8 $NWCHEM_TOP/bin/$NWCHEM_TARGET/nwchem input.nw
  ```

- If you have all nodes connected via shared memory and you have installed the ch_shmem version of MPICH, you can do

  ```
  $NWCHEM_TOP/bin/$NWCHEM_TARGET/nwchem -np 8 h2o.nw
  ```

D.4 Parallel execution on MPPs

All of these machines require use of different commands in order to gain exclusive access to computational resources.

D.5 IBM SP

If using POE (IBM's Parallel Operating Environment) interactively, simply create the list of nodes to use in the file "host.list" in the current directory and invoke NWChem with

```
  nwchem <input_file> -procs <n>
```

where n is the number of processes to use. Process 0 will run on the first node in "host.list" and must have access to the input and other necessary files. Very significant performance gains may be had by setting the following environment variables before running NWChem (or setting them using POE command line options).

- setenv MP_EUILIB us — dedicated user space communication over the switch (the default is IP over the switch which is much slower).
- setenv MP_CSS_INTERRUPT yes — enable interrupts when a message arrives (the default is to poll which significantly slows down global array accesses).

In addition, if the IBM is running PSSP version 3.1, or later

- setenv MP_MSG_API lapi, or
- setenv MP_MSG_API mpi,lapi (if using both GA and MPI)

For batch execution, we recommend use of the llnw command which is installed in /usr/local/bin on the EMSL/PNNL IBM SP. If you are not running on that system, the llnw script may be found in the NWChem distribution directory contrib/loadleveler. Interactive help may be obtained with the command llnw -help. Otherwise, the very simplest job to run NWChem in batch using Load Leveller is something like this

```
#!/bin/csh -x
# @ job_type         =    parallel
# @ class            =    small
# @ network.lapi     = css0,not_shared,US
# @ input            =    /dev/null
# @ output           =    <OUTPUT_FILE_NAME>
# @ error            =    <ERROUT_FILE_NAME>
# @ environment      =    COPY_ALL; MP_PULSE=0; MP_SINGLE_THREAD=yes; MP_WAIT_MODE=yield; restart=no
# @ min_processors   =    7
# @ max_processors   =    7
# @ cpu_limit        =    1:00:00
# @ wall_clock_limit =    1:00:00
# @ queue
#
```

```
cd /scratch

nwchem <INPUT_FILE_NAME>
```

Substitute `<OUTPUT_FILE_NAME>`, `<ERROUT_FILE_NAME>` and `<INPUT_FILE_NAME>` with the *full* path of the appropriate files. Also, if you are using an SP with more than one processor per node, you will need to substitute

```
# @ network.lapi     = css0,shared,US
# @ node             = NNODE
# @ tasks_per_node   = NTASK
```

for the lines

```
# @ network.lapi     = css0,not_shared,US
# @ min_processors   =    7
# @ max_processors   =    7
```

where `NNODE` is the number of physical nodes to be used and `NTASK` is the number of tasks per node.

These files and the NWChem executable must be in a file system accessible to all processes. Put the above into a file (e.g., `"test.job"`) and submit it with the command

```
  llsubmit test.job
```

It will run a 7 processor, 1 hour job in the queue `small`. It should be apparent how to change these values.

Note that on many IBM SPs, including that at EMSL, the local scratch disks are wiped clean at the beginning of each job and therefore persistent files should be stored elsewhere. PIOFS is recommended for files larger than a few MB.

D.6 Cray T3E

```
  mpprun -n <npes> $NWCHEM_TOP/bin/$NWCHEM_TARGET/nwchem <input_file>
```

where `npes` is the number of processors and `input_file` is the name of your input file.

D.7 Linux

If running in parallel across multiple machines you should consider applying this patch to your kernel to boost the performance of TCP/IP

- http://www.icase.edu/coral/LinuxTCP.html

D.8 Alpha systems with Quadrics switch

```
prun -n <npes> $NWCHEM_TOP/bin/$NWCHEM_TARGET/nwchem <input_file>
```

where `npes` is the number of processors and `input_file` is the name of your input file.

D.9 Windows 98 and NT

```
$NWCHEM_TOP/bin/win32/nw32 <input_file>
```

where and `input_file` is the name of your input file. If you use WMPI, uou must have a file named `nw32.pg` in the `$NWCHEM_TOP/bin/win32` directory; the file must only contains the following single line

```
local 0
```

D.10 Tested Platforms and O/S versions

- IBM SP with P2SC nodes, AIX 4.2.1, and PSSP 2.3
- IBM SP with silver nodes (SMP nodes with two 604e processors), AIX 4.3.2, and PSSP 3.1.
- IBM RS6000 workstation, AIX 3.2 and 4.1.
- Cray T3E, 2.0.4.61 UNICOSMK
- SGI R8000/10000, IRIX 6.2, 6.5
- SGI R4000, IRIX 5.3
- SUN workstations, SunOS 4.1.3 and Solaris 5.5, 5.6
- Compaq DEC alpha workstion
  - (600 MHz EV6), Digital UNIX V4.0E Rev. 1091, DEC C V5.8-009, Digital Fortran V5.2
  - (667 MHz EV6), Digital UNIX V4.0F Rev. 1229, DEC C V5.9-005, Digital Fortran V5.2
- Linux. Since there are at least 8 popular distributions of the Linux operating system and numerous others in existence, including downloading everything and building your own Linux OS, it is impossible to test all possible versions of Linux with NWChem. NWChem Release 3.3 has been tested on Slackware 3.4, 3.5, 4.0, 7.0 RedHat 5.1, 5.2, and 6.0, 6.1, 6.2 Mandrake based on RedHat 6.0, RedHat 6.0 for the Power PC Macintosh, and Black Lab 1.2 or Yellow Dog 1.2 Linux for Power PC Macintosh. These all use the EGCS compilers at different levels. Those distributed from Slackware are somewhat different than those distributed from RedHat but the code is configured to run on all of them.
- Linux on Alpha cpus has been tested with RedHat 6.1 and 6.2. Usage of the Compaq Fortran and C compilers is necessary for compiling.
- HP 9000/800 workstations with HPUX B.11.00. f90 must be used for compilation.
- FreeBSD. Just use the Linux target and NWChem will build for FreeBSD version 4.0 and the pre-release 5.0 versions of the Operating System.
- Intel x86 with Windows 98 and NT has been tested with Compaq Visual Fortran 6.0 and 6.1 with WMPI 1.3 or NT-MPICH. NT-MPICH is available from http://www-unix.mcs.anl.gov/~ashton/mpich.nt/