(continued from previous page)

```
        command=os.getenv("ORCA_COMMAND"),
        deleteJobFiles=False, header="bp86 def2-svp qmmmopt/pdynamo",
        job="chignolin", run=True)

# Assign the models to the system.
system.DefineMMModel(mmmodel)
system.DefineQCModel(
        qcmodel, qcSelection=pCore.Selection([35, 36, 37, 34, 40, 41]))
system.DefineNBModel(nbmodel)
system.electronicState = pMolecule.ElectronicState(
        charge=-1, multiplicity=1)

# Print a summary and calculate the energy.
system.Summary()
system.Energy()
```

After the execution of the above Python program, a series of files are going to be created `chignolin.inp`, `chignolin.pc`, `chignolin.lj` and ORCA is going to be called. The generated ORCA input file is listed below.

```
! bp86 def2-svp qmmmopt/pdynamo
% geom
    constraints
        {C 0 C}
        {C 1 C}
        end
    end

% pointcharges "chignolin.pc"
% ljcoefficients "chignolin.lj"
* xyz -1 1
H                  -1.0637532468        1.1350324675        2.4244220779
C                  -0.5230000000        0.6870000000        3.2490000000
C                   0.4180000000        1.7240000000        3.8660000000
O                  -0.0690000000        2.7620000000        4.2830000000
O                   1.6090000000        1.4630000000        3.9110000000
H                  -1.2240000000        0.3460000000        3.9970000000
H                   0.0550000000       -0.1510000000        2.8890000000
*
```

There are few points that have to be raised here. Because the keyword `qmmm/pdynamo` was specified in the `header` variable, the pDynamo library will automatically add the `constraint` block in the ORCA input, which will freeze the link atoms and the QM atoms to which they are bound. It will also generate the `chignolin.lj` file containing all the Lennard-Jones parameters. The important parts of this file, which is somewhat different than the one generated by Gromacs, are listed next.

```
# number of atoms  combination rule
138 0
#      x            y            z          sigma       epsilon   id
   -6.778000    -1.424000     4.200000     3.250000     0.711280   -1
   -6.878000    -0.708000     2.896000     3.500000     0.276144   -1
   -5.557000    -0.840000     2.138000     3.750000     0.439320   -1
...
    0.433000     0.826000     0.502000     2.960000     0.878640   -1
   -0.523000     0.687000     3.249000     3.500000     0.276144    1
    0.418000     1.724000     3.866000     3.750000     0.439320    2
   -0.069000     2.762000     4.283000     2.960000     0.878640    3
    1.609000     1.463000     3.911000     2.960000     0.878640    4
   -2.259000    -0.588000     1.846000     0.000000     0.000000   -1
   -1.795000     2.207000     2.427000     2.500000     0.125520   -1
   -1.224000     0.346000     3.997000     2.500000     0.125520    5
```

(continues on next page)

(continued from previous page)

```
    0.055000    -0.151000     2.889000     2.500000     0.125520    6
   -0.311000     2.922000     0.557000     3.250000     0.711280   -1
...
   -1.387000    -2.946000     5.106000     2.500000     0.125520   -1
# number of special pairs
22
#         atom1         atom2    factor
          34            32     0.000000
          35            39     0.500000
          40            31     0.000000
          41            30     0.500000
          41            32     0.500000
          36            31     0.500000
          40            32     0.500000
          40            39     0.500000
          34            31     0.000000
          35            30     0.500000
          34            11     0.500000
          34            38     0.500000
          41            31     0.000000
          37            31     0.500000
          34            33     0.500000
          34            39     0.000000
          40            30     0.500000
          41            39     0.500000
          34            30     0.000000
          35            31     0.000000
          34            42     0.500000
          35            32     0.500000
```

The second number on the first line refers to the type of combination rule used to calculate the Lennard-Jones interaction. It is 0 if a geometric average is used (OPLS force field), or 1 for the Lorentz-Berthelot rules (AMBER force field). The `id` on the last column is -1 for MM atoms and is equal to the atom number for the QM atoms. In this case the hydrogen link atom is atom 0. The last block of the file is composed of atom pairs and a special factor by which their Lennard-Jones interaction is scaled. In general this factor is equal to 1, but for atoms one or two bonds apart is zero, while for atoms three bonds apart depends on the type of force field, and in this case is 0.5.

After successful completion of the ORCA optimization run, the information will be relayed back the pDynamo library, which will report the total QM/MM energy of the system. At this point the type QM/MM of calculation is limited only by the capabilities of the pDynamo library, which are quite extensive.

6.4.3 ORCA and NAMD

Since version 2.12, NAMD is able to perform hybrid QM/MM calculations. A more detailed explanation of all available key words, setting up the calculation and information on tutorials and on the upcoming graphic interface to VMD are available on the NAMD website.

Similar to other calculations with NAMD, the QM/MM is using a pdb file to control the active regions. An example is shown below, where the sidechain of a histidine protonated at Nϵ is chosen to be the QM region. Either the occupancy column or the b-factor column of the file are used to indicate which atom are included in a QM area and which are treated by the forcefield. In the other column, atoms which are connecting the QM area and the MM part are indicated similarly. To clarify which column is used for which purpose, the keywords `qmColumn` and `qmBondColumn` have to be defined in the NAMD input.

```
...
ATOM   1737  CA  HSE P 117      14.762  47.946  31.597  1.00  0.00      PROT C
ATOM   1738  HA  HSE P 117      14.751  47.579  32.616  0.00  0.00      PROT H
ATOM   1739  CB  HSE P 117      14.129  49.300  31.501  1.00  1.00      PROT C
ATOM   1740  HB1 HSE P 117      14.407  49.738  30.518  0.00  1.00      PROT H
```

(continues on next page)

(continued from previous page)

```
ATOM   1741  HB2 HSE P 117      13.024  49.194  31.509  0.00  1.00       PROT H
ATOM   1742  ND1 HSE P 117      13.899  51.381  32.779  0.00  1.00       PROT N
ATOM   1743  CG  HSE P 117      14.572  50.261  32.582  0.00  1.00       PROT C
ATOM   1744  CE1 HSE P 117      14.615  52.043  33.669  0.00  1.00       PROT C
ATOM   1745  HE1 HSE P 117      14.356  53.029  34.064  0.00  1.00       PROT H
ATOM   1746  NE2 HSE P 117      15.678  51.318  33.982  0.00  1.00       PROT N
ATOM   1747  HE2 HSE P 117      16.369  51.641  34.627  0.00  1.00       PROT H
ATOM   1748  CD2 HSE P 117      15.706  50.183  33.335  0.00  1.00       PROT C
ATOM   1749  HD2 HSE P 117      16.451  49.401  33.388  0.00  1.00       PROT H
ATOM   1750  C   HSE P 117      13.916  47.000  30.775  0.00  0.00       PROT C
ATOM   1751  O   HSE P 117      12.965  46.452  31.334  0.00  0.00       PROT O
...
```

NOTES:

- If one wants to include more than one QM region, integers bigger than 1 can be used to define the different regions.
- Charge groups cannot be split when selecting QM and MM region. The reason is that non-integer partial charges may occur if a charge group is split. Since the QM partial charges are updated in every QM iteration, this may lead to a change in the total charge of the system over the course of the MD simulation.
- The occupancy and b-factor columns are used for several declarations in NAMD. If two of these come together in one simulation, the keyword `qmParamPDB` is used to define which pdb file contains the information about QM atoms and bonds.
- To simplify the selection of QM atoms and writing the pdb file a set of scripts is planned to be included in future releases of NAMD.

To run the calculation, the keyword `qmForces on` must be set. To select ORCA `qmSoftware "orca"` must be specified and the path to the executables must be given to `qmExecPath`, as well as a directory where the calculation is carried out (`qmBaseDir`). To pass the method and specifications from NAMD to ORCA `qmConfigLine` is used. These lines will be copied to the beginning of the input file and can contain both simple input as well as block input. To ensure the calculation of the gradient, the `engrad` keyword should be used.

The geometry of the QM region including the selected links as well as the MM point charges are copied to the ORCA inputfile automatically. Multiplicity and charge can be defined using `qmMult` and `qmCharge`, although the latter can be determined automatically by NAMD using the MM parameters. It should be noted at this point that NAMD is capable to handle more than one QM region per QM/MM calculation. Therefore for each region, charge and multiplicity are expected. In the case of only one QM region, the input looks like the following:

```
qmMult          "1 1"
qmCharge        "1 0"
```

Currently, two charge modes are available: Mulliken and CHELPG. They have to be specified in the NAMD input using `QMChargeMode` and in the `qmConfigLine`, respectively. Different embedding schemes, point charge schemes and switching functions are available, which will be not further discussed here. Another useful tool worth mentioning is the possibility to call secondary executables before the first or after each QM software execution using `QMPrepProc` or `QMSecProc`, respectively. Both are called with the complete path and name to the QM input file, allowing e.g. storage of values during an QM/MM-MD.

It is strongly emphasized that at this points both programs are constantly developed further. For the latest information, either the ORCA forum or the NAMD website should be consulted.

6.5 Fast Multipole Method

The Fast Multipole Method (FMM) algorithm was proposed in the 1980s[797], to reduce the computation time for two-centre interactions in large systems, by moving from a quadratic relationship (O(N^2)) to a quasi-linear relationship (O($Nlog(N)$)) of the computation time with the number N of particles (atoms, point charges, etc.). This algorithm is particularly useful for long-range interactions that are difficult to ignore, such as the Coulombic interaction (1/r) which is fundamental to any system but grows quadratically with the number of centers.

The FMM is used in ORCA for accelerating QMMM calculations in the scope of electrostatic embedding: FMM-QMMM (cf. *Embedding Types*).

6.5.1 The Octree hierarchy

There exists a lot of detailed and pedagogical literature on the subject (we recommend for instance [801] and [802]). In the following we will only describe the main parameters.

The idea of the algorithm is to divide space to handle differently short- and long-range interactions. The latter will be approximated (cf. *Fast Multipole Method*). To do so, the whole system is placed in a cubic box which is iteratively divided in 8 children boxes (forming levels L: $L = 0 \rightarrow L = L_{max}$) so as to form a structure called an Octree (see Fig. Fig. 6.4).

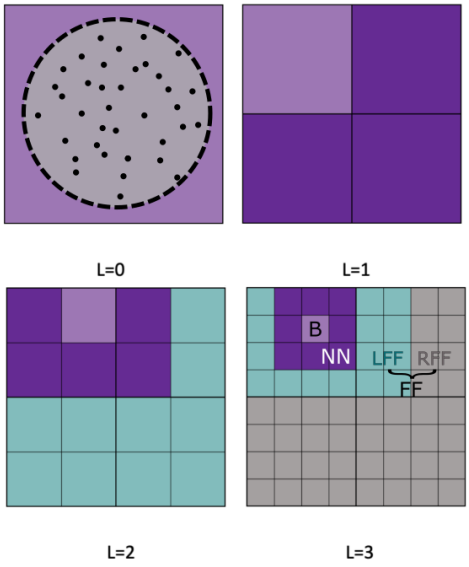

Fig. 6.4: Division of the system into boxes, schematic representation in 2D.

At the deepest level, $L = L_{max}$, for every box, we can define a layer of nearest neighbours boxes (NN) which will be responsible for the Near-Field (NF, i.e.short-range). The electrostatic potential due to the rest of the boxes is define as the Far-Field (FF, i.e. long-range).

$$\forall B \ , \ \mathbf{V}_B = \mathbf{V}_B^{\mathbf{NF}} + \mathbf{V}_B^{\mathbf{FF}}$$

As visible on Figure Fig. 6.4, one can divide the FF area between a Local Far Field (LFF, green) area and a Remote Far Field area (RFF, grey), so that a recursive scheme between levels appears:

$$\begin{aligned} \forall B \ \ \mathbf{V}_B^{\mathbf{FF}} &= \mathbf{V}_B^{\mathbf{LFF}} + \mathbf{V}_B^{\mathbf{RFF}} \\ &= \mathbf{V}_B^{\mathbf{LFF}} + \mathbf{W}^T_{Parent(B)\rightarrow B} \mathbf{V}^{\mathbf{FF}}_{Parent(B)} \end{aligned}$$

The LFF boxes are the boxes in the NF of the parent but not in the NF of the children, that represents a maximum of 189 boxes. The RFF potential is due to boxes which actually represent the FF area of the parent box of B. This

means that one needs to calculate only the LFF term at every level, the rest of the potential will be inherited from the parent box.

The number of levels, L_{max}, can be setup in the input directly (FMMQMMM_Levels), or will be deduced from the provided box dimension (FMMQMMM_BoxDimInp, dimension of the box at L_{max}). The second option (default) is recommended, with a box dimension around 9.0 Bohr (FMMQMMMM_BoxDimInp 9.0). If FMMQMMM_DoBoxDimOpt option is turned to TRUE (default), the box dimension will be reduced as much as possible while keeping the same value of L_{max}, this to ensure the algorithm works optimally regarding both accuracy and efficiency.

```
%method
  DOFMMQMMM              True  #turn ON the use of the FMM
  FMMQMMM_BoxDimInp       9.0  #Higher boundary to set up the dimension (in Bohr!)
  FMMQMMM_DoBoxDimOpt    True  #Optimize the box dimension
end
```

The higher L_{max}, the bigger the number of boxes: at $L = 2$ one has 64 boxes only but for $L = 6$ the code generates more than 200,000 boxes (8^6). For $L_{max} \geq 7$, more than 2 millions boxes are generated, this can start leading to memory issues, so that the user should ensure enough memory is available (cf. *Global Memory Use*).

6.5.2 Approximation of the Far Field interactions

In a space of origin O(0,0,0), let P ($\mathbf{r}_P$) be the center of a charge distribution (e.g. a Gaussian overlap or a distribution of point charges). Let's consider a point A in the vicinity of P, such that A = (q_A; $\mathbf{r}_A$), with q_A the charge of a point charge or $q_A = -1$ if we consider a Gaussian overlap. In that case, the charge would indeed be the one of the electron situated at a position $\mathbf{r}_{AP} = \mathbf{r}_A - \mathbf{r}_P$ from the center of the Gaussian overlap. Similarly, let Q ($\mathbf{r}_Q$) be the center of a charge distribution and B, a point in the vicinity of Q, such that B = (q_B; $\mathbf{r}_B$). We note $\mathbf{r}_{QP} = \mathbf{r}_Q - \mathbf{r}_P$.

The Coulomb interaction between A and B can be expressed as:

$$U_{AB} = f\Big(\frac{1}{|\mathbf{r}_a - \mathbf{r}_b|}\Big),$$

The name of the algorithm comes from the way the FF interactions are evaluated. It is done through a mutlipole expansion of the $\frac{1}{|\mathbf{r}_a - \mathbf{r}_b|}$ term, leading to the following expression:

$$\frac{1}{|\mathbf{r}_A - \mathbf{r}_B|} = \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\sum_{j=0}^{\infty}\sum_{k=-j}^{j}(-1)^j R_{l,m}(\mathbf{r}_{AP}) I_{l+j,m+k}(\mathbf{r}_{QP}) R_{j,k}(\mathbf{r}_{BQ}),$$

This series converges for well separated centers only, that is why a NF layer is required. $R_{l,m}$ and $I_{l,m}$ are regular and irregular solid scaled harmonics [803]:

$$R_{l,m}(\mathbf{r}) = \frac{1}{\sqrt{(l-m)!(l+m)!}}\mathbf{r}^l\sqrt{\frac{4\pi}{2l+1}}Y_{l,m}(\mathbf{r})$$
$$I_{l,m}(\mathbf{r}) = \sqrt{(l-m)!(l+m)!}\frac{1}{\mathbf{r}^{l+1}}\sqrt{\frac{4\pi}{2l+1}}Y_{l,m}(\mathbf{r})$$

$Y_{l,m}$ are spherical harmonics of degree l and order m. For details on the derivation of this equation see [801].

We can now introduce the multipole expansion centered in P ($\mathbf{r}_P$), of a charge distribution $\mathbf{Q}(\mathbf{r}, P)$, as:

$$\mathbf{Q}(\mathbf{r}, P) = f\Big(\sum_{l=0}^{\infty}\sum_{m=-l}^{l} R_{l,m}(\mathbf{r} - \mathbf{r}_P)\Big)$$

In practise:

$$\mathbf{Q}(\mathbf{r}, P) \approx f\Big(\sum_{l=0}^{MAM}\sum_{m=-l}^{l} R_{l,m}(\mathbf{r} - \mathbf{r}_P)\Big)$$

Q is a vector, with elements $Q_{l,m}$ defined by the pair of values (l,m). In the case of a point charge A (q_A,**r**) we have:

$$Q^A_{l,m}(\mathbf{r}, P) = q_A \times R_{l,m}(\mathbf{r} - \mathbf{r}_P),$$

while for a Gaussian overlap distribution $\langle\mu|\nu\rangle$ it becomes:

$$\begin{aligned} Q^{\mu\nu}_{l,m}(\mathbf{r}, P) &= \langle\mu|\, R_{l,m}(\mathbf{r} - \mathbf{r}_P)\,|\nu\rangle \\ &= \int \mu(\mathbf{r'}) R_{l,m}(\mathbf{r} - \mathbf{r}_P)\nu(\mathbf{r'})d\mathbf{r'} \end{aligned}$$

So that we can approximate the electrostatic interaction between a Gaussian overlap $\langle\mu|\nu\rangle$ with center in $\mathbf{r}_B$ and a point charge A (q_A,$\mathbf{r}_A$) in the FF, through the following expansion:

$$\frac{\int q_A \mu(\mathbf{r})\nu(\mathbf{r})d\mathbf{r}}{|\mathbf{r}_A - \mathbf{r}_B|} \approx \mathbf{Q}^{\mu\nu}(\mathbf{r}_b, P')\mathbf{I}(\mathbf{r}_{PP'})\mathbf{Q}^A(\mathbf{r}_A, P),$$

with P and P' the respective center of the multipole expansions, and I the interaction matrix.

The truncation parameter of the expansion, MAM, is the Maximum Angular Momentum of the underlying solid scaled harmonics. In our setup, it is the same value for the expansion of the point charges in the embedding or the Gaussian overlaps in the QM part. It can be setup through the keyword FMMQMMM_MAM. We recommend FMMQMMM_MAM = 20 to ensure a good accuracy, in the order of 0.0001-0,0001 Ha. With a lower value of MAM=15, one can usually expect a mHa (0.63 kcal.mol^{-1}) precision. However it is system dependent, so that we recommend you to start with MAM=20 and see how it behaves if you decrease it to a value of 15. The available values for MAM are in the range 1-25.

```
%method
  DOFMMQMMM              True  #turn ON the use of the FMM
  FMMQMMM_MAM              20   #Default Maximum Angular Momentum
end
```

6.5.3 The "Very" Fast Multipole Method (VFMM)

Due to the recursive scheme between levels, the FF felt in the center of a box is built by only calculating interactions with the LFF boxes at every level. The evaluation of V^{LFF} becomes the bottleneck of the algorithm. To accelerate it, an option has been implemented in which different MAM truncation parameters will be used for all the (>189) boxes in the LFF area. This option FMMQMMMM_DoVFMM is turned ON by default, but should be switched OFF whenever the MAM is smaller than 15, that would impact to much the accuracy without improving much the efficiency.

```
%method
  DOFMMQMMM              True  #turn ON the use of the FMM
  FMMQMMM_DoVFMM         True  #Use the Very Fast Multipole Method
end
```

6.5.4 Recommended input

Whenever there is an electrostatic embedding, for systems containing more than 10,000 point charges (ECM) or MM atoms (QMMM), it is recommended to turn on the FMM in order to accelerate the calculation of the electrostatic potential. However, when using heavy parallelization with more than 24 processors, the impact of enabling the Fast Multipole Method (FMM) may be negligible for small embedding size.

Caution

FMM-QMMM does not replace the QMMM keyword.

The two elements one can play on are the truncation parameter of the expansion, MAM, and the box dimension at the deepest level (L_{max}). Recommended/default parameters can be called in the keyword line directly using FMM-QMMM or by setting the parameters accordingly in the %method block:

```
#keyword line
! FMM-QMMM

#OR through method block

%method
  DOFMMQMMM              True  #turn ON the use of the FMM
  USEFMMQMMM             True  #turn ON the use of the FMM
  ###parameters:
  FMMQMMM_DoVFMM         True  #Use the Very Fast Multipole Method
  FMMQMMM_MAM              20  #Maximum Angular Momentum
  FMMQMMM_Levels            0  #Number of levels set to 0, will be set␣
↪automatically by box dim
  FMMQMMM_BoxDimInp       9.0  #Higher boundary to set up the dimension (in Bohr!)
  FMMQMMM_DoBoxDimOpt    True  #Optimize the box dimension
end
```

The parameters used by the algorithm are printed in the output by default, we recomend you to check them in your first calculations:

```
----------------------
SHARK INTEGRAL PACKAGE
----------------------

[...]

Use FMM for one center integrals with PCs

- - - - - - - FMM PARAMETERS- - - - - - -
num PCs               75894
VFMM                   USED
MAM                    20
Input box dim         9.000
Refined box dim       5.197
Tree depth/levels       6
- - - - - - - - - - - - - - - - - - - - -
```

This can be turned off by modifying the relative printing options.

```
%method
 FMMQMMM_Printing 1 # remove FMM PARMETERS printing, default value is  2
end
```

6.5.5 Some examples

Call the recommanded FMM parameters through the keyword line:

```
!QMMM wB97X-D3BJ RIJCOSX FMM-QMMM

%qmmm
 ORCAFFFilename "file.prms"
 Embedding Electrostatic
 ChargeAlteration CS
 QMAtoms {2016:2026} end
end
```

(continues on next page)

(continued from previous page)

```
*pdbfile 0 1 file.pdb
```

Change the MAM from 20 (value when one uses the keyword line) to 15, and use 5 levels:

```
!QMMM wB97X-D3BJ RIJCOSX

%method
  DOFMMQMMM             True  #turn ON the use of the FMM
  USEFMMQMMM            True  #turn ON the use of the FMM
  ###parameters:
  FMMQMMM_DoVFMM        True  #Use the Very Fast Multipole Method
  FMMQMMM_MAM             15  #Maximum Angular Momentum
  FMMQMMM_Levels           5  #number of levels
end

%qmmm
 ORCAFFFilename "file.prms"
 Embedding Electrostatic
 ChargeAlteration CS
 QMAtoms {2016:2026} end
end


*pdbfile 0 1 file.pdb
```

Specify the box dimension without optimizing it:

```
!QMMM wB97X-D3BJ RIJCOSX

%method
  DOFMMQMMM             True  #turn ON the use of the FMM
  USEFMMQMMM            True  #turn ON the use of the FMM
  ###parameters:
  FMMQMMM_DoVFMM        True  #Use the Very Fast Multipole Method
  FMMQMMM_MAM             15  #Maximum Angular Momentum
  FMMQMMM_Levels           0  #number of levels
  FMMQMMM_BoxDimInp      9.0  #Higher boundary to set up the dimension (in Bohr!)
  FMMQMMM_DoBoxDimOpt   False #Does not optimize the box dimension
end

%qmmm
 ORCAFFFilename "file.prms"
 Embedding Electrostatic
 ChargeAlteration CS
 QMAtoms {2016:2026} end
end


*pdbfile 0 1 file.pdb
```

Specifying the box dimension and ask to optimize it:

```
!QMMM wB97X-D3BJ RIJCOSX

%method
  DOFMMQMMM             True  #turn ON the use of the FMM
  USEFMMQMMM            True  #turn ON the use of the FMM
  ###parameters:
  FMMQMMM_DoVFMM        True  #Use the Very Fast Multipole Method
  FMMQMMM_MAM             15  #Maximum Angular Momentum
```

(continues on next page)

(continued from previous page)

```
  FMMQMMM_Levels           0  #number of levels
  FMMQMMM_BoxDimInp      9.0  #Higher boundary to set up the dimension (in Bohr!)
  FMMQMMM_DoBoxDimOpt   True  #Optimize the box dimension
end

%qmmm
 ORCAFFFilename "file.prms"
 Embedding Electrostatic
 ChargeAlteration CS
 QMAtoms {2016:2026} end
end


*pdbfile 0 1 file.pdb
```

CHAPTER

SEVEN

MOLECULAR DYNAMICS

7.1 *Ab initio* Molecular Dynamics

A few years ago, we included an *ab initio* molecular dynamics (AIMD) module into ORCA.[1] As a plethora of different electron structure methods with analytical gradients is already implemented, all these methods are now available also for MD simulations, offering a wide range of accuracy/performance trade-offs.

Despite the relatively short history inside of the ORCA package, the MD module has grown considerably over the last years. A few features found in other MD codes are still missing. In future releases, many more new features and methods will (hopefully) be added to this part of the program. We will always do our best to keep a strict backward compatibility, such that the sample inputs from this section will remain valid in all future releases.

For some more information as well as input examples for the ORCA MD module, please visit

https://brehm-research.de/orcamd

7.1.1 Changes in ORCA 5.0

- Added a `Metadynamics` module with many features and options:
  - Can perform one-dimensional and two-dimensional Metadynamics simulations [804] to explore free energy profiles along reaction coordinates, called collective variables ("Colvars").
  - `Colvars` can be distances *(including projections onto vectors and into planes)*, angles, dihedrals, and coordination numbers [805]. The latter allows, *e. g.*, to accurately compute pK_A values of weak acids [806, 807].
  - For all Colvars, groups of atoms *(e. g., centers of mass)* can be used instead of single atoms.
  - Metadynamics simulations can be easily restarted and split over multiple runs.
  - Ability to run well-tempered Metadynamics [808] for a smoothly converging free energy profile.
  - Ability to run extended Langrangian Metadynamics [805], where a virtual particle on the bias profile is coupled to the real system via a spring. The virtual particle can be thermostated.
- Added two modern and powerful `thermostats` *(both available as global and massive)*:
  - The widely used Nosé–Hoover chain thermostat (NHC) with high-order Yoshida integrator [809, 810]; allows for a very accurate sampling of the canonical ensemble.
  - The stochastical "Canonical Sampling through Velocity Rescaling" (CSVR) thermostat [811] which has become quite popular recently.
- Can define harmonic and Gaussian `restraints` for all Colvars *(distance, angle, dihedral, coordination number)*. This allows for umbrella sampling [812, 813], among other methods. Can also define one-sided restraints which act as lower or upper wall.

[1] Strictly speaking, these simulations are Born–Oppenheimer molecular dynamics simulations (BOMD), because they approximately solve the time-independent Schrödinger equation to compute gradients and then move the atoms according to these gradients.

- Can now print the instantaneous and average force on `constraints` and `restraints`; this allows for thermodynamic integration [813].
- The target value for `constraints` and `restraints` can now be a ramp, so that it can linearly change during the simulation.
- Can now keep the system's center of mass fixed during MD runs.
- Can now print population analyses, orbital energies, and .engrad files in every MD step if requested.

7.1.2 Changes in ORCA 4.2 (Aug 2019)

- Added a Cartesian minimization command to the MD module, based on L-BFGS and simulated annealing. Works for large systems ($> 10\,000$) atoms and also with constraints. Offers a flag to only optimize hydrogen atom positions *(for crystal structure refinement)*. See `Minimize` command.
- The MD module can now write trajectories in DCD file format *(in addition to the already implemented XYZ and PDB formats)*, see `Dump` command.
- The `thermostat` is now able to apply temperature ramps during simulation runs.
- Added more flexibility to region definition *(can now add/remove atoms to/from existing regions)*. Renamed the `Define_Region` command to `Manage_Region`.
- Added two new constraint types which keeps centers of mass fixed or keep complete groups of atoms rigid, see `Constraint` command.
- Ability to store the GBW file every n-th step during MD runs *(e.g. for plotting orbitals along the trajectory)*, see `Dump` command.
- Can now set limit for maximum displacement of any atom in a MD step, which can stabilize dynamics with poor initial structures. Runs can be cleanly aborted by "`touch EXIT`". See `Run` command.
- Better handling/reporting of non-converged SCF during MD runs.
- Fixed an issue which slowed down molecular dynamics after many steps.
- Stefan Grimme's xTB method can now be used in the MD module, allowing fast simulations of large systems.

7.1.3 Changes in ORCA 4.1 (Dec 2018)

- Molecular dynamics simulation can now use Cartesian, distance, angle, and dihedral angle constraints. These are managed with the `Constraint` command.
- The MD module now features cells of several geometries (cube, orthorhombic, parallelepiped, sphere, ellipsoid), which can help to keep the system inside of a well-defined volume. The cells have repulsive harmonic walls.
- The cells can be defined as elastic, such that their size adapts to the system. This enables to run simulations under constant pressure.
- Trajectories can now be written in XYZ and PDB file format.
- A restart file is written in each simulation step. With this file, simulations can be restarted to seamlessly continue (useful for batch runs or if the job crashed). Restart is handled via the `Restart` command; see below.
- Introduced regions (*i. e.*, subsets of atoms), which can be individually defined. Regions can be used to thermostat different parts of the system to different temperatures (*e. g.*, cold solute in hot solvent), or to write subset trajectories of selected atoms.
- The energy drift of the simulation is now displayed in every step (in units of Kelvin per atom). Large energy drift can be caused by poor SCF convergence, or by a time step length chosen too large.
- Physical units in the MD input are now connected to their numeric values via underscore, such as `350_pm`. A whitespace between value and unit is no longer acceptable. This slightly breaks backward compatibility – sorry.

- Fixed a bug in the time integration of the equations of motion, which compromised energy conservation.
- Fixed crashes for semiempirics and if ECPs were employed. You can now run MD simulations with methods such as PM3 and with ECPs.

7.1.4 Input Format

The molecular dynamics module is activated by specifying "MD" in the simple input line. The actual MD input which describes the simulation follows in the "%md" section at some later position in the input file. The contents of this section will subsequently be referred to as "MD input".

```
! MD BLYP D3 def2-SVP
%md
  Timestep 0.5_fs # This is a comment
  Initvel 350_K
  Thermostat NHC 350_K Timecon 10.0_fs
  Dump Position Stride 1 Filename "trajectory.xyz"
  Run 200
end
* xyz 0 1
O         -4.54021        0.78439         0.09307
H         -3.64059        0.38224        -0.01432
H         -4.63463        1.39665        -0.67880
*
```

Please note that the MD input is not processed by ORCA's main parser, but by a dedicated parser in the MD module. Therefore, the MD input is not required to obey the general ORCA syntax rules. The syntax will be described in the following.

In contrast to general ORCA input, the MD input is not based on keywords, but on *commands*, which are executed consecutively on a line-by-line basis starting at the top (like, *e. g.*, in a shell script). This means that identical commands with different arguments may be given, coming into effect when the interpreter reaches the corresponding line. This enables to perform multiple simulations (*e. g.*, pre-equilibration and production run) within a single input file:

```
%md
  Timestep 1.0_fs
  Run 200
  Timestep 2.0_fs
  Run 500
end
```

Work is already under way to add variable definitions, loops, and conditional branching to the MD input.[2] This will enable even larger flexibility (*e. g.*, to run a simulation until a certain quantity has converged). The MD input is written in the SANscript language ("Scientific Algorithm Notation Script"), which is under development. A first glimpse can be found at

https://brehm-research.de/sanscript

As in standard ORCA input, **comments** in the MD input are initiated by a "#" sign and span to the end of the current line. Commands can be started both at the beginning of a line and after a command. The only place where a "#" is **not** treated as start of a comment is inside of a string literal (*e. g.*, in file names).

```
%md
  # Comment
  Timestep 0.5_fs # Comment
  Dump Position Filename "trajectory#1.xyz"
end
```

Some more MD input syntax rules:

[2] Technically speaking, ORCA will then be a Turing-complete script interpreter, such that *any* computational problem can be solved with ORCA :-)

- The MD input is generally **not** case-sensitive. The only exception are file names on platforms with case-sensitive file systems (such as GNU Linux).
- Empty lines are allowed.
- Commands and options are separated by space or tabulator characters. Any combination of these characters may be used as separator.
- Both DOS and UNIX line break style is acceptable.

Commands

As already noted above, the central item of the MD input is a command. Each input line contains (at most) one command, and these commands are executed in the given order. A command typically takes one or more arguments, which are given behind the command name, separated by whitespaces, tabulator characters, or commas (optional). The order of the arguments for a command is fixed (see command list in section *Command List*. Commands may have optional arguments, which are always specified at the end of the argument list, after the last non-optional argument. If there exist multiple optional arguments for a command, not all of them need to be specified; however, they need to be specified in the correct order and without gaps:

```
%md
  Command Arg1 Arg2 Arg3                  # fine
  Command Arg1, Arg2, Arg3                # fine
  Command Arg1 Arg2 Arg3 Optarg1          # fine
  Command Arg1 Arg2 Arg3 Optarg1 Optarg2  # fine
  Command Arg1 Arg2 Arg3 Optarg2          # will not work
end
```

Apart from arguments and optional arguments, commands can also have *modifiers*. These can be considered as "sub-commands", which modify a given command, and may possess their own argument lists. Modifiers generally follow after all non-optional and optional arguments, and they may **not** possess optional arguments on their own. If a command has multiple modifiers, the order in which they are given is not important.

In the following input example, "`Mod1`" and "`Mod2`" are modifiers of "`Command`". "`Mod1`" takes one argument, "`Mod2`" does not take arguments:

```
%md
  Command Arg1                            # fine
  Command Arg1 Optarg1                    # fine
  Command Arg1 Mod1 Modarg1 Mod2          # fine
  Command Arg1 Mod2                       # fine
  Command Arg1 Mod2 Mod1 Modarg1          # fine
  Command Arg1 Optarg1 Mod1 Modarg1 Mod2  # fine
end
```

To make this abstract definition a little more illustrative, please consider again one line from the input sample at the beginning of this section:

```
%md
  Dump Position Stride 1 Filename "trajectory.xyz"
end
```

Here, "`Dump`" is the command, which takes one non-optional argument to specify which quantity shall be dumped – in this case, "`Position`". The "`Dump`" command has two modifiers, namely "`Stride`" and "`Filename`". The former takes one integer argument, the latter a string argument. Swapping the two modifiers (together with their respective arguments, of course) would not change the behavior.

Separating Arguments

As shown above, the arguments which are passed to a command do not need to be separated by commas. However, it is allowed (and recommended) to still use commas. First, it can increase the readability of the input file. Secondly, there exist a few ambiguous cases in which commas (or parentheses) should be used to clarify the intended meaning. One of these cases is the arithmetic minus operator. It can either be used as binary operator (subtracting one number from another), or as unary operator (returning the negative of a number). By default, the minus operator will be considered as binary operator (if possible).

Consider the case in which you want to pass two integer arguments "`10`" and "`-10`" to a command. Without commas (or parentheses), the minus is mistreated as binary operator, and only one argument will be passed to the command:

```
Command 10 -10    #  Pitfall: treated as "Command (10 - 10)", i.e., "Command 0"
Command 10, -10   #  Two arguments, as intended
Command 10 (-10)  #  Also works
```

Physical Units

In many cases, it is required to specify quantities which bear a physical unit in an input file (*e. g.*, temperature, time step lengths, …). For many quantities, there are different units in widespread use, which always leads to some confusion (just consider the "kcal vs kJ" case). ORCA handles this problem by defining default units for each quantity and requiring that all quantities are given in their default unit. ORCA's default units are the atomic units, which are heavily used in the quantum chemistry community, but not so much in the molecular dynamics community. As an *ab initio* molecular dynamics module exists in the small overlap region of both communities, some "unit conflicts" might arise. To prevent those from the beginning, it is allowed to specify units of personal choice within ORCA's MD input.

Luckily, this is as simple and convenient as it sounds. The parser of the MD module checks if a unit is given after a numeric constant, and automatically converts the constant to the internal default unit. If no explicit unit is given, the default unit is assumed. Please note that the default units within the MD module are not necessarily atomic units (see table below). Units are connected to the preceding numerical value by an underscore:

```
%md
  Timestep  1.0_fs
  Timestep 41.3_au # identical
  Timestep  1.0    # identical, as default time unit in MD module is fs
end
```

In the following, all units which are currently known to the MD module's parser are listed, sorted by physical quantities. The default unit for each quantity is printed in **`bold letters`**. Additive constant and factor are applied to convert a unit into the default unit. The additive constant is applied before the factor. A "−" means that the constant/factor is not applied. More units will be probably added in the future.

Unit Symbol	Additive Constant	Factor
— Length Units —		
Angstrom	–	–
A	–	–
Bohr	–	0.5291
pm	–	0.01
nm	–	10.0
— Time Units —		
fs	–	–
ps	–	1000
au	–	0.02419
— Temperature Units —		
Kelvin	–	–
K	–	–
Celsius	273.15	–
C	273.15	–
— Angle Units —		
Deg	–	–
Rad	–	$180/\pi$

7.1.5 Discussion of Features

Restarting Simulations

Ab initio molecular dynamics simulation are computationally expensive, and will typically run for a long time even in the case of medium-sized systems. Often, it is desirable to perform such a simulation as a combination of multiple short runs (*e. g.*, if the queuing system of the cluster imposes a maximum job time). The ORCA MD module writes a restart file in each simulation step, which allows for the seamless continuation of simulations. This restart file has the name "`basename.mdrestart`", where `basename` is the project's base name. To load an existing restart file, use the *`Restart`* command *(see command list below)*.

In the first run of a planned sequence of runs, no restart file exists yet. for this case, the *`Restart`* command offers the `IfExists` modifier. The restart file is only loaded if it exists. If not, the restart is simply skipped, and no error is thrown. By using this modifier, you can have the *`Restart`* command already in place in the first run of a sequence (where no restart file exists in the beginning), and do not need to modify the input after the first run has finished.

Concerning the *`Dump`* command, it is good to know that trajectory files are appended (*not* overwritten) by default. If you ever want to overwrite an existing trajectory file by a *`Dump`* command, use the `Replace` modifier.

Please note that **only** the positions, velocities, thermostat internal state *(only for NHC)*, Metadynamics hills, and time step counters are restarted when executing a *`Restart`* command. All other properties (thermostats, regions, trajectory dumps, constraints, cells, etc.) are **not** restarted. They should all remain in the input file, as executed in the first run of a sequence. Just add the *`Restart`* command after all other relevant commands have been executed, directly before the *`Run`* command.

To conclude this discussion, a short example is given. If the MD input file

```
%md
  Timestep 0.5_fs
  Initvel 300_K
  Thermostat NHC 300_K Timecon 10.0_fs
  Dump Position Stride 1 Filename "trajectory.xyz"
  Restart IfExists
```

(continues on next page)

(continued from previous page)

```
  Run 100
end
```

is subsequently executed ten times (without any modification), the resulting trajectory file will be identical *(apart from numerical noise)* to that obtained if the following input is executed once:

```
%md
  Timestep 0.5_fs
  Initvel 300_K
  Thermostat NHC 300_K Timecon 10.0_fs
  Dump Position Stride 1 Filename "trajectory.xyz"
  Run 1000
end
```

Regions

In the ORCA MD module, **regions** can be defined. This concept does *not* refer to regions in space, but rather to subsets of atoms in the system. A region is nothing more than a list of atoms. Regions may overlap, *i. e.*, atoms can be part of more than one region at a time. The atoms which are part of a certain region remain the same until the region is manually re-defined, *i. e.*, regions are fixed and do not adapt to any changes in the system. There exist a few pre-defined regions which have a name. User-defined regions, in contrast, only carry an integer identifier. The following regions are pre-defined in any case:

- `all`: Contains all atoms of the system. This is the default if no region is specified in some commands, so by default, these commands will always act on the whole system.
- `active`: This region contains all movable ("non-frozen") atoms. By default, it is identical to the `all` region. Atoms inside of this region are updated by the time integration in a molecular dynamics run, displaced in a minimization, and are considered for computing the kinetic energy.
- `inactive`: This region contains all atoms which are *not* part of the `active` region. These atoms are "frozen"; they are ignored by the time integration / minimization, and also not considered for the computation of the kinetic energy. They simply remain on their initial positions. This is in principle identical to applying Cartesian constraints to the atoms; however, it is much faster. As constraints have to be solved iteratively *(see below)*, Cartesian constraints become quite computationally demanding if applied to thousands of atoms.

From these three pre-defined regions, only the `active` region can be manually modified. Changes in the composition of the `active` region automatically modify the `inactive` region. The `all` region obviously cannot be changed.

In case of a QM/MM simulation, the following four additional regions can be used:

- `qm`: This is the "quantum mechanics" region – it contains all atoms which are treated by the electron structure method.
- `mm`: This is the "molecular mechanics" region – it contains all atoms which are treated by a force-field approach. It exactly contains those atoms which are not part of the `qm` region.
- `active_qm`: Contains exactly those atoms which are part of both the `qm` and the `active` regions.
- `active_mm`: Contains exactly those atoms which are part of both the `mm` and the `active` regions.

These regions can **not** be modified in the MD input. The MD module just reads the region definitions from the QM/MM module, but is not able to make any changes here.

Regions can be useful for many purposes. For example, a "realistic" wall of atoms can be built around the system by defining the `active` region such that it only contains the non-wall atoms. The wall atoms will then be frozen. Apart from that, trajectories of regions can be written to disk, only containing the "interesting" part of a simulation. Furthermore, velocity initialization can be applied to regions, enabling to start a simulation in which different sets of atoms possess different initial temperatures. Thermostats can be attached to regions to keep different sets of atoms at different temperatures during the whole simulation. This allows for sophisticated simulation setups (cold solute in hot solvent, temperature gradient through the system, etc).

Regions are defined or modified by the `Manage_Region` command. Many other commands take regions as optional arguments. Please see the command list below.

Metadynamics

Metadynamics is a powerful tool to analyze free energy profiles of reactions and other processes *(solvation, aggregation, conformer change, dissociation)* based on molecular dynamics simulations. It has been developed by Laio and Parrinello in 2002 [804]. In principle, the frequency of observing a certain process in MD simulations is directly related to the free energy barrier of the process. However, many interesting processes *(such as chemical reactions)* possess such a high free energy barrier that they will never occur on the time scales typical for AIMD simulations (100 ps). To increase the frequency at which such processes happen, so-called rare event sampling methods can be employed. Metadynamics is one among those. It works by building up a bias potential as a sum of Gaussian hills, so that free energy minima are slowly filled up and the system is gradually pushed away from its resting points.

Please note that there is also a method with the same name for exploring conformation space that has been published by Grimme in 2019 [814]. It is in principle based on the original "Parrinello" Metadynamics, but with several modifications and extensions. The ORCA MD module contains the original Parrinello variant of Metadynamics [804], together with several extensions such as well-tempered Metadynamics [808] and extended Lagrangian Metadynamics [805]. The Grimme method for conformer search will probably be implemented in the future.

In Metadynamics, one has to define one or more "collective variables" (Colvars) along which the free energy profile of the system will be sampled. A Colvar is in principle nothing more than a continuous function of all atom positions which returns a real number. A simple example of a Colvar is the distance between two atoms, which could be used to explore the free energy profile of a bond formation or cleavage. In the ORCA MD module, Colvars can be defined via the `Manage_Colvar` command. Available Colvar types are distances *(including projections onto lines or into planes)*, angles, dihedral angles, and coordination numbers [805]. The latter allows, *e. g.*, to accurately compute pK_A values of weak acids in solvent [806, 807]. For the distances, angles, and dihedral angles, atom groups instead of single atoms can be specified, so that, *e. g.*, the distance between the centers of mass of two molecules can be defined as a Colvar.

Based on one or two Colvars (ORCA supports one-dimensional and two-dimensional Metadynamics), a Metadynamics simulation can be set up. There are many parameters to choose, which are described in the section of the `Metadynamics` command. After all parameters have been set, the actual simulation is simply started via the `Run` command. It is also possible to restart Metadynamics simulations so that they can be split into multiple successive runs; see the `Restart` command. A full example for a two-dimensional well-tempered extended Lagrangian Metadynamics simulation can be found on *below*.

Note that Metadynamics simulations typically require very much computational time *(at least several 10 000 MD steps for a roughly converged result, depending on the Colvar choice)*. So this is by no means a method to "shortly try out". However, there are no cheaper methods for predicting free energy profiles *(apart from very simple approximations such as the harmonic oscillator)*, and the predictive power of computing free energy profiles comes at a price.

7.1.6 Command List

In Table 7.1, an alphabetical list of all commands currently known to the MD module is given. The description of each command starts with a small box which contains the command's name and a table of arguments and modifiers. The last-but-one column in the table specifies the type of each argument. Possible types are "Integer", "Real", "String", and "Keyword". In the latter case, the last column contains a list of allowed keyword values in { `braces` }. If the type is "Real" and is a physical quantity with unit, the quantity is given in the last column in [square brackets]. Each such box is followed by a textual description of the corresponding command.

Table 7.1: Overview of commands in the `%md` block

Command	Description
`Cell`	Defines and modifies cells
`Constraint`	Manages constraints
`Dump`	Controls trajectory output
`Initvel`	Randomly initializes atom velocities
`Manage_Colvar`	Manages collective variables ("Colvars")
`Manage_Region`	Manages regions
`Metadynamics`	Sets parameters for Metadynamics runs
`Minimize`	Performs a Cartesian energy minimization
`PrintLevel`	Controls the output verbosity
`Randomize`	Sets the random seed
`Restart`	Restarts a simulation to seamlessly continue
`Restraint`	Manages restraints on Colvars
`Run`	Performs a molecular dynamics run
`SCFLog`	Controls the ORCA log file output
`Screendump`	Prints current MD state to screen
`Thermostat`	Manages thermostats
`Timestep`	Sets the integrator time step Δt

`Cell`

Manadory Arguments:		-		
Optional Arguments:		-		
Modifiers:	`Cube`	...	...	*see text*
	`Rect`	...	...	*see text*
	`Rhomb`	...	...	*see text*
	`Sphere`	...	...	*see text*
	`Ellipsoid`	...	...	*see text*
	`None`	—	—	—
	`Spring`	k	Real	*see text*
	`Elastic`	t_{avg}	Real	[time]
		c_{response}	Real	*see text*
	`Anisotropic`	—	—	—
	`Pressure`	...	...	*see text*
	`Fixed`	—	—	—

Defines a harmonic repulsive wall around the system *(the wall is "soft" with a spring constant and atoms can slightly penetrate; "hard" repulsive walls are not supported)*. This helps to keep the molecules inside of a well-defined volume, or to keep a constant pressure in the system. In the latter case, the cell can be defined as elastic, such that it exerts a well-defined pressure *(see below)*. Please note that ORCA does not feature periodic boundary conditions, and therefore, all cells are non-periodic (just repulsive walls). There are several cell geometries available *(only one type of cell can be active at a time)*:

- **`Cube`**: Defines a cubic cell. If two real values p_1 and p_2 are specified as coordinates, the cell ranges from (p_1, p_1, p_1) to (p_2, p_2, p_2). If only one real value p is supplied, the cell ranges from $(-\frac{p}{2}, -\frac{p}{2}, -\frac{p}{2})$ to $(\frac{p}{2}, \frac{p}{2}, \frac{p}{2})$, *i. e.* it is centered at the origin with edge length p.
- **`Rect`**: Defines an orthorhombic cell. Six real values x_1, y_1, z_1, x_2, y_2, and z_2 have to be specified as coordinates *(in this order)*. The cell will range from (x_1, y_1, z_1) to (x_2, y_2, z_2).
- **`Rhomb`**: Defines a parallelepiped-shaped cell (also termed as rhomboid sometimes). You have to specify twelve real values in total. The first three define one corner point p of the cell, and the remaining nine define three cell vectors v_1, v_2, and v_3, each given as Cartesian vector components. The cell is then defined as the set of points $\{p + c_1 v_1 + c_2 v_2 + c_3 v_3 \mid 0 \leq c_1, c_2, c_3 \leq 1\}$ The vectors v_1, v_2, and v_3 do not need to be orthogonal to each other, but they may not all lie within one plane (cell volume would be zero).

- **`Sphere`**: Defines a spherical cell. You need to specify four real values c_x, c_y, c_z, and r. The cell will then be defined as a sphere around the central point $\left(c_x, c_y, c_z\right)$ with radius r.
- **`Ellipsoid`**: Defines an ellipsoid-shaped cell. As first three arguments, you have to specify three real values c_x, c_y, c_z, which define the center of the ellipsoid to be $\left(c_x, c_y, c_z\right)$. As fourth argument, a keyword has to follow, which may either be "`XYZ`" or "`Vectors`". In the "`XYZ`" case, three more real values r_x, r_y, and r_z have to be specified, which define the partial radii of the ellipsoid along the X, Y, and Z coordinate axes. If instead "`Vectors`" was given, nine more real values v_x^1, v_y^1, v_z^1, v_x^2, v_y^2, v_z^2, v_x^3, v_y^3, v_z^3 have to follow after the keyword. These values define three vectors $v^1 := \left(v_x^1, v_y^1, v_z^1\right)$, $v^2 := \left(v_x^2, v_y^2, v_z^2\right)$, and $v^3 := \left(v_x^3, v_y^3, v_z^3\right)$, which are the principal axes of the ellipsoid. These vectors have to be strictly orthogonal to each other. The length of each vector defines the partial radius of the ellipsoid along the corresponding principal axis.

All cell types define a harmonic potential $E_{\text{cell}}\left(r\right) := k \cdot r^2$ which acts on all atoms in the system **outside of the cell**, where r is the closest distance from the atom's center to the defined cell surface. Atoms whose center is inside of the cell or directly on the cell surface do not experience any repulsive force. Following from the definition, the force which acts on an atom outside of the cell is always parallel to the normal vector of the cell surface at the point which is closest to the atom center. This is trivial in case of cubic, rectangular, rhombic, and spherical cells, but not so trivial for ellipsoid-shaped cells.

The spring constant k in the above equation (*i. e.*, the "steepness" of the wall) can be specified by the "`Spring`" modifier, which expects one real value as argument. The spring constant has to be specified in the unit kJ mol^{-1} Å^{-2}, other units cannot be specified here. The default value is 10 kJ mol^{-1} Å^{-2}. Larger spring constants reduce the penetration depth of atoms into the wall, but may require shorter integration time steps to ensure energy conservation. If jumps in the total energy occur, try to use a smaller spring constant (*e. g.*, the default value).

The command "`Cell None`" disables any previously defined cell.

If you want to perform simulations under constant pressure, you can define an elastic cell. Then, ORCA accumulates the force which the cell exerts on the atoms in each time step, and divides this total force by the cell surface area to obtain a pressure. As this momentarily pressure heavily fluctuated, a running average is used to smooth this quantity. If the averaged pressure is larger than the external pressure which was specified, the cell will slightly grow; if it was smaller, the cell will slightly shrink. In the beginning of a simulation, the cell size will not vary until at least the running average history depth of steps have been performed.

An elastic cell is enabled by using the "`Elastic`" modifier after the cell geometry definition. Subsequently, two real values t_{avg} and c_{response} are required. While t_{avg} defines the length of the running average to smooth the pressure *(in units of physical time,* ***not*** *time steps)*, the c_{response} constant controls how fast the cell size will change at most. More specific, c_{response} is the fraction of the cell volume growth per time step if the ratio of averaged and external pressure would be infinite, and at the same time the fraction of the cell volume reduction per step if the aforementioned ratio is zero. Put into mathematical form, the cell volume change per time step is

$$V_{\text{new}} := \begin{cases} V_{\text{old}} \cdot \frac{c_{\text{response}} \cdot \frac{\langle p \rangle}{p_{\text{ext}}} + 1}{c_{\text{response}} + 1} & \text{if } \frac{\langle p \rangle}{p_{\text{ext}}} \leq 1, \\ V_{\text{old}} \cdot \frac{\left(c_{\text{response}} + 1\right) \cdot \frac{\langle p \rangle}{p_{\text{ext}}}}{\frac{\langle p \rangle}{p_{\text{ext}}} + c_{\text{response}}} & \text{if } \frac{\langle p \rangle}{p_{\text{ext}}} > 1, \end{cases} \tag{7.1}$$

where $\langle p \rangle$ represents the averaged pressure the system exerts on the walls, and p_{ext} is the specified external pressure. Good starting points are $t_{\text{avg}} = 100\,\text{fs}$ and $c_{\text{response}} = 0.001$. Please note that larger values of c_{response} or smaller values of t_{avg} may lead to uncontrolled fluctuations of the cell size. An already defined fixed cell can be switched to elastic by the command "`Cell Elastic ...`" (the dots represent the two real arguments).

By default, the size change of an elastic cell due to pressure is performed *isotropically*, *i. e.*, the cell is scaled as a whole, and exactly retains its aspect ratio. By specifying the "`Anisotropic`" modifier after switching on an elastic cell, the cell pressure is broken down into individual components, and the size of the cell is allowed to change independently in the individual directions. This, of course, only makes sense for the cell geometries `Rect`, `Rhomb`, and `Ellipsoid`. An already defined isotropic cell can be switched to anisotropic by simply executing "`Cell Anisotropic`".

In case of an elastic cell, the external pressure is defined by the modifier "`Pressure`", which expects either one or three real values as arguments. If one argument is given, this is the isotropic external pressure. If three arguments are supplied, these are the components of the pressure in X, Y, and Z direction *(in case of orthorhombic cells)* or along the direction of the three specified vectors *(in case of parallelepiped-shaped and ellipsoid-shaped cells)*. This allows for anisotropic external pressure (probably only useful for solid state computations). Both the pressure and the pressure