(continued from previous page)

```
 3 H    -0.500000    0.870000    1.740000
 4 H    -0.500000   -0.870000    1.740000
 5 H     1.000000    0.000000    1.740000
----------------------------------------


----------------------------------------
 Match: 1, Assigned Fragment: 2
----------------------------------------
  Name:  CH3 Method: Ext_lib
  Natoms: 4 Charge: 0 Mult: 1

 2 C     1.300000    0.000000   -0.460000
 6 H     1.300000    0.000000   -1.530000
 7 H     1.800000    0.870000   -0.100000
 8 H     1.800000   -0.870000   -0.100000
----------------------------------------
```

Note

- `XZYFRAGLIB` allows the inclusion of up to 10 files. Each file may contain multiple fragment definitions; however, each definition must consist of a single, connected molecule—unconnected structures within a single definition are not allowed.
- The format of `Mylib.xyz` follows the standard XYZ file structure, but optionally supports three identifiers: `CHARGE`, `MULT`, and `NAME`, with `NAME` expected to appear last. These identifiers are printed when a fragment is recognized, though they are not currently used in any calculations.
- Subsequent modifications to fragments by other methods (see *Fusebyatoms* and *Extend* below) do not affect the fragment's CHARGE or MULT values in the identifiers.

Important

- ORCA's VF2 subgraph isomorphism algorithm is based solely on atomic connectivity; it does not consider stereochemistry or bond orders.
- Fragment assignment by `FragProc Extlib` follows the order of the files listed in `XZYFRAGLIB` and the order of fragment definitions within each file. Since an atom is excluded from subsequent matching once it has been assigned to a fragment, the order in which fragments are defined in the library is critically important. For example, in the previous *case*, if CH3 is defined before CH3O in `Mylib.xyz`, both CH3 groups in the *input file* will be matched first. This prevents CH3O from being recognized later, even if it is present in the system.
- Atom assignment to fragments follows the order in which atoms appear in the geometry. In the *previous example*, the CH3O fragment could be assigned using either the C(1)-O or C(2)-O bond. However, since C(1) appears first in the geometry, it is selected for the CH3O fragment.

Automatic Fragmentation: Extend

Many fragments in the internal libraries represent incomplete molecular structures. Therefore, in some cases, it is necessary to add additional hydrogen atoms—or, in certain situations, a hydroxyl group—to complete the system. The `FragProc Extend` addresses this by identifying oxygen atoms bonded to carbon atoms in previously assigned fragments, as well as hydrogen atoms bonded to carbon, nitrogen, or oxygen. It then extends the corresponding fragments to include these atoms.

Consider the example below, which consists of a zwitterionic glycine molecule fragmented using an external library that defines a C–C–N fragment. In this case, `FragProc Extend` can be applied to add the missing oxygen and hydrogen atoms, thereby completing the molecular structure.

```
%frag
  PrintLevel 3
  FragProc Extlib, Extend
  XZYFRAGLIB "Mylib.xyz"
end

*xyz 0 1
 N     0.000000     0.000000     0.000000
 C     0.000000     0.000000     1.460000
 C     1.403962     0.000000     2.015868
 O     1.837767     0.962613     2.627089
 O     2.161436    -0.947142     1.883370
 H     0.423874    -0.764689    -0.525339
 H    -0.538292    -0.884247     1.831954
 H    -0.538292     0.884247     1.831954
 H    -0.970478     0.016940    -0.313504
 H     0.499909     0.831989    -0.313504
*
```

Where `Mylib.xyz` in this case is:

```
    3
Name CCN
 C     0.000000     0.000000     0.000000
 C     0.000000     0.000000     1.500000
 N     1.376503     0.000000    -0.486661
```

The result is an initial assignment of the C–C–N fragment, followed by an extension of the fragment to include two oxygen atoms and five hydrogen atoms.

```
----------------------------------------
 Match: 1, Assigned Fragment: 1
----------------------------------------
  Name:  CCN Method: Ext_lib
  Natoms: 3 Charge: 0 Mult: 1

 0 N     0.000000    0.000000    0.000000
 1 C     0.000000    0.000000    1.460000
 2 C     1.403962    0.000000    2.015868
----------------------------------------

================================================
Tfragmentator: Extending Fragments
================================================
 Extending Fragment 0 with O  (3)
 Extending Fragment 0 with O  (4)
 Extending Fragment 0 with H  (6)
 Extending Fragment 0 with H  (7)
 Extending Fragment 0 with H  (5)
 Extending Fragment 0 with H  (8)
```

(continues on next page)

(continued from previous page)

```
Extending Fragment 0 with H  (9)
```

Important

- `FragProc Extend` attempts to extend any previously assigned fragment.
- `FragProc Extend` does not modify the `Charge` and `Mult` identifiers of fragments.

Automatic Fragmentation: Fusebyatoms

When multiple fragmentation procedures are used in combination, it may be necessary to merge two previously assigned fragments into one. This can be achieved using the `FragProc Fusebyatoms` procedure, which identifies atom pairs that should belong to the same fragment, as specified in `FuseAtomPairs`.

In the example below, the objective is to fragment a propane molecule into a CH3 group and a CH3–CH2 fragment. One way to achieve this is by first applying `FragProc FunctionalGroups`, which fragments the molecule into two CH3 groups and one CH2 group. The CH2 group can then be merged with one of the CH3 groups. This is done by specifying the carbon atoms of the CH2 and one CH3 group in the `FuseAtomPairs` directive: `FuseAtomPairs {0 1} end`.

```
%frag
        printlevel 3
        FragProc FunctionalGroups, Fusebyatoms
        FuseAtomPairs {0 1} end
end

*xyz 0 1
 C     0.000000     1.270000    -0.260000
 C     0.000000    -0.000000     0.580000
 C     0.000000    -1.270000    -0.260000
 H    -0.890000     1.320000    -0.910000
 H     0.890000     1.320000    -0.910000
 H     0.000000     2.180000     0.370000
 H     0.880000    -0.000000     1.250000
 H    -0.880000    -0.000000     1.250000
 H     0.890000    -1.320000    -0.910000
 H     0.000000    -2.180000     0.370000
 H    -0.880000    -1.300000    -0.910000
*
```

The output from the fragmentator first reports the initial fragmentation performed by the `Functional_groups` library, followed by a message indicating the subsequent fusion of fragments as specified by the `FuseAtomPairs` directive.

```
================================================
Tfragmentator: Fragmenting by Functional_groups
================================================

-----------------------------------------
 Match: 1, Assigned Fragment: 1
-----------------------------------------
  Name:  CH3 Method: Functional_groups
  Natoms: 4 Charge: 0 Mult: 1

 0 C     0.000000    1.270000   -0.260000
 3 H    -0.890000    1.320000   -0.910000
 4 H     0.890000    1.320000   -0.910000
 5 H     0.000000    2.180000    0.370000
```

(continues on next page)

(continued from previous page)

```
----------------------------------------


----------------------------------------
 Match: 2, Assigned Fragment: 2
----------------------------------------
  Name:  CH3 Method: Functional_groups
  Natoms: 4 Charge: 0 Mult: 1

 2 C     0.000000   -1.270000   -0.260000
 8 H     0.890000   -1.320000   -0.910000
 9 H     0.000000   -2.180000    0.370000
 10 H    -0.880000   -1.300000   -0.910000
----------------------------------------

----------------------------------------
 Match: 1, Assigned Fragment: 3
----------------------------------------
  Name:  CH2 Method: Functional_groups
  Natoms: 3 Charge: 0 Mult: 1

 1 C     0.000000   -0.000000    0.580000
 6 H     0.880000   -0.000000    1.250000
 7 H    -0.880000   -0.000000    1.250000
----------------------------------------

================================================
Tfragmentator: Fusing Fragments
================================================
Fusing Fragment 0 (atom 0) with Fragment 2 (atom 1)
```

Automatic Fragmentation: Delete and advanced fragmentation workflows

Almost all fragmentation schemes have an associated delete procedure, which removes the fragments generated by that specific scheme. This enables the construction of advanced fragmentation workflows, where a procedure can temporarily "protect" certain atoms from being fragmented by subsequent methods. These protected fragments can later be deleted, allowing the atoms to be reprocessed using a different fragmentation approach.

Consider the system *below*, which consists of both a phenylalanine molecule and a benzene molecule. The goal is to fragment the phenylalanine into its backbone, one CH2 group, and the phenyl ring, while fragmenting the benzene into six individual CH fragments.

On one hand, by combining the three fragmentation procedures — `FragProc AABackbone, Extend, AASCFinegrained` — the desired fragmentation of phenylalanine can be achieved. The `AABackbone` and `Extend` options identify the aminoacid backbone, while `AASCFinegrained` separates the CH2 group and the phenyl ring into distinct fragments. Meanwhile, the benzene molecule can be fragmented into six individual CH fragments using `FragProc Extlib`, which relies on an external library defining the CH fragment. However, these two fragmentation schemes are incompatible. When `FragProc AASCFinegrained` is applied to the entire system, it fragments not only the phenylalanine side chain but also breaks the benzene ring into a phenyl group and a hydrogen atom. Conversely, if `FragProc Extlib` is applied first to benzene, it may inadvertently fragment the phenylalanine residue, leading to undesired structural splits.

A viable approach to achieve the desired fragmentation involves first identifying the amino acid using `FragProc Aminoacids`. This procedure assigns all atoms that belong to amino acid fragments, thereby protecting them from reassignment by subsequent fragmentation steps. The benzene molecule can then be fragmented using an external library via `FragProc Extlib`. Finally, the phenylalanine fragment is removed using `FragProc DelAminoacids`, allowing it to be re-fragmented as needed. All these operations are executed by simply concatenating the procedures as follows: `FragProc Aminoacids, Extlib, DelAminoacids, AABackbone, Extend, AASCFinegrained`

```
%frag
  PrintLevel 3
  FragProc Aminoacids, Extlib, DelAminoacids, AABackbone, Extend, AASCFinegrained
  XZYFRAGLIB "Mylib.xyz"
  STOREFRAGS true
end

*xyz 0 1
 O     1.300780     3.093320     1.571260
 C     0.407700     2.826000     0.770390
 O    -0.101890     3.606620    -0.030470
 C    -0.115350     1.396690     0.770390
 N    -1.564350     1.396690     0.770390
 H    -1.900780     1.872730     1.595200
 H     0.242580     0.884560     1.663540
 H    -1.901490     0.444630     0.770390
 H    -1.901016     1.873069    -0.054118
 C     0.436090     0.696200    -0.461750
 H     1.512450     0.863150    -0.502770
 H    -0.018950     1.150700    -1.341790
 C     0.178410    -0.790680    -0.515340
 C    -0.895760    -1.288470    -1.262580
 C    -1.134680    -2.667040    -1.312270
 C    -0.299410    -3.547820    -0.614730
 C     0.774760    -3.050040     0.132500
 C     1.013680    -1.671470     0.182200
 H     1.842330    -1.287460     0.758630
 H     1.419110    -3.729500     0.670600
 H    -0.483720    -4.611290    -0.653070
 H    -1.963330    -3.051050    -1.888700
 H    -1.540110    -0.609010    -1.800680
 C     1.181325    -3.732146    -2.620584
 C     2.016592    -4.612923    -1.923046
 C     3.090772    -4.115131    -1.175824
 C     3.329684    -2.736563    -1.126139
 C     2.494417    -1.855785    -1.823677
 C     1.420237    -2.353577    -2.570900
 H     0.770518    -1.668458    -3.113485
 H     0.422252    -4.288369    -3.129823
 H     1.830753    -5.685253    -1.961693
 H     3.740491    -4.800250    -0.633239
 H     4.165243    -2.349352    -0.544907
 H     2.680256    -0.783455    -1.785030
*
```

Where “Mylib.xyz” in this case is:

```
2
NAME CH
C 0.00 0.00 0.00
H 1.08 0.00 0.00
```

The complete fragmentation sequence follows the same structure as in previous examples. In this case, the message `Deleted 1 fragments` is printed after the fragment assignment performed by `Ext_lib`, indicating that a fragment previously defined by `FragProc Aminoacids` has been successfully removed.

```
================================================
Tfragmentator: Fragmenting by Amino_Acids
================================================

----------------------------------------
 Match: 1, Assigned Fragment: 1
```

(continues on next page)

(continued from previous page)

```
----------------------------------------
  Name:  CPHE Method: Amino_Acids
  Natoms: 21 Charge: -1 Mult: 1

 0 O     1.300780    3.093320    1.571260
 1 C     0.407700    2.826000    0.770390
 2 O    -0.101890    3.606620   -0.030470
 3 C    -0.115350    1.396690    0.770390
 4 N    -1.564350    1.396690    0.770390
 5 H    -1.900780    1.872730    1.595200
 6 H     0.242580    0.884560    1.663540
 9 C     0.436090    0.696200   -0.461750
 10 H     1.512450    0.863150   -0.502770
 11 H    -0.018950    1.150700   -1.341790
 12 C     0.178410   -0.790680   -0.515340
 13 C    -0.895760   -1.288470   -1.262580
 14 C    -1.134680   -2.667040   -1.312270
 15 C    -0.299410   -3.547820   -0.614730
 16 C     0.774760   -3.050040    0.132500
 17 C     1.013680   -1.671470    0.182200
 18 H     1.842330   -1.287460    0.758630
 19 H     1.419110   -3.729500    0.670600
 20 H    -0.483720   -4.611290   -0.653070
 21 H    -1.963330   -3.051050   -1.888700
 22 H    -1.540110   -0.609010   -1.800680
----------------------------------------

================================================
Tfragmentator: Fragmenting by Ext_lib
================================================
****
**** There are 1 Ref structures found in file Mylib.xyz
****

----------------------------------------
 Match: 1, Assigned Fragment: 2
----------------------------------------
  Name:  CH Method: Ext_lib
  Natoms: 2 Charge: 0 Mult: 1

 23 C     1.181325   -3.732146   -2.620584
 30 H     0.422252   -4.288369   -3.129823
----------------------------------------

----------------------------------------
 Match: 2, Assigned Fragment: 3
----------------------------------------
  Name:  CH Method: Ext_lib
  Natoms: 2 Charge: 0 Mult: 1

 24 C     2.016592   -4.612923   -1.923046
 31 H     1.830753   -5.685253   -1.961693
----------------------------------------

...

----------------------------------------
 Match: 6, Assigned Fragment: 7
----------------------------------------
  Name:  CH Method: Ext_lib
  Natoms: 2 Charge: 0 Mult: 1
```

(continues on next page)

(continued from previous page)

```
 28 C     1.420237   -2.353577   -2.570900
 29 H     0.770518   -1.668458   -3.113485
----------------------------------------

================================================
Tfragmentator: Deleting Fragments (Amino_Acids)
================================================
 Deleted 1 fragments
================================================
Tfragmentator: Fragmenting by Backbone
================================================

----------------------------------------
 Match: 1, Assigned Fragment: 7
----------------------------------------
  Name: CO-NH3-CH Method: AA_Backbone
  Natoms: 8 Charge: 0 Mult: 0

 0 O     1.300780    3.093320    1.571260
 1 C     0.407700    2.826000    0.770390
 3 C    -0.115350    1.396690    0.770390
 4 N    -1.564350    1.396690    0.770390
 5 H    -1.900780    1.872730    1.595200
 6 H     0.242580    0.884560    1.663540
 7 H    -1.901490    0.444630    0.770390
 8 H    -1.901016    1.873069   -0.054118
----------------------------------------
================================================
Tfragmentator: Extending Fragments
================================================
 Extending Fragment 6 with O  (2)

================================================
Tfragmentator: Fragmenting by AA_SideChains_FG
================================================

----------------------------------------
 Match: 1, Assigned Fragment: 8
----------------------------------------
  Name: Ph Method: AA_SideChains_FG
  Natoms: 11 Charge: 0 Mult: 1

 12 C     0.178410   -0.790680   -0.515340
 13 C    -0.895760   -1.288470   -1.262580
 14 C    -1.134680   -2.667040   -1.312270
 15 C    -0.299410   -3.547820   -0.614730
 16 C     0.774760   -3.050040    0.132500
 17 C     1.013680   -1.671470    0.182200
 18 H     1.842330   -1.287460    0.758630
 19 H     1.419110   -3.729500    0.670600
 20 H    -0.483720   -4.611290   -0.653070
 21 H    -1.963330   -3.051050   -1.888700
 22 H    -1.540110   -0.609010   -1.800680
----------------------------------------

----------------------------------------
 Match: 1, Assigned Fragment: 9
----------------------------------------
  Name: CH2 Method: AA_SideChains_FG
  Natoms: 3 Charge: 0 Mult: 1
```

(continues on next page)

(continued from previous page)

```
 9 C     0.436090    0.696200   -0.461750
10 H     1.512450    0.863150   -0.502770
11 H    -0.018950    1.150700   -1.341790
----------------------------------------
```

Table Table 2.61 lists the corresponding delete procedure associated with each fragmentation method.

Table 2.61: Simple input keywords for Fragment detection and their corresponding deletion procedure

Fragment detection Keyword	Fragment deletion Keyword
`Extlib`	`DELExtlib`
`Connectivity`	`DELConnectivity`
`Atomic`	`DELAtomic`
`FunctionalGroups`	`DELFunctionalGroups`
`NotAssigned`	
`Backbone`	`DELBackbone`
`SeqBackbone`	`DELSeqBackbone`
`AABackbone`	`DELAABackbone`
`Aminoacids`	`DELAminoacids`
`AASideChains`	`DELAASideChains`
`AASCFineGrained`	`DELAASCFineGrained`
`NABackbone`	`DELNABackbone`
`NABBFineGrained`	`DELNABBFineGrained`
`SEQNABackbone`	`DELSEQNABackbone`
`NucleoticAcid`	`DELNucleoticAcid`
`NASideChains`	`DELNASideChains`
`Solvents`	`DELSolvents`
`Water`	`DELWater`
`Extend`	
`FuseByAtoms`	

2.18.3 Options available in the `%frag` input block

Table Table 2.62 contains a list of the options available in the `%frag` input block.

Table 2.62: List of options in the `%frag` input block

Option	Type	Default	Description
`Printlevel`	Integer	`1`	Verbose output control for automated fragmentation.
`STOREFRAGS`	Boolean	`False`	Stores assigned fragments in a `.fragments.xyz` file.
`DoInterFragBon`	Boolean	`False`	Automatically detects bonds between fragments for CoVaLED analysis.
`XZYFRAGLIB`	String	`None`	Filenames used in `FragProc Extlib`.
`FragProc`	See Table 2.60, and Table 2.61	`ExtLib, Connectivi`	Fragmentation procedures to be applied automatically.
`Usetopology`	Boolean	`False`	Generate main geometry graph based on .prms file.

continues on next page

Table 2.62 – continued from previous page

Option	Type	Default	Description
TopolFile	String	""	Topology file name to been used when Usetopology True
PrintInputFlag	Boolean	True	Writes a %frag block equivalet to current calculation fragments.

2.19 ORCA and Symmetry

For most of its life, ORCA did not take advantage of molecular symmetry. Starting from version 2.8 (released in September 2010), there has been at least limited use. On request (using the simple keyword UseSym for instance, see below), the program detects the point group, orients the molecule, cleans up the coordinates and produces symmetry-adapted molecular orbitals.

Only for geometry cleanup the full point group is taken into account. For all other purposes such as the construction of symmetry-adapted molecular orbitals and or to describe electronic states, only D_{2h} and subgroups are currently supported. Here the use of symmetry helps to control the calculation and the interpretation of the results.

2.19.1 Getting started

Utilization of symmetry is turned on by the simple keyword UseSymmetry (which may be abbreviated by UseSym), or if a %Symmetry (or %Sym) input block is present in the input. ORCA will then automatically determine the point group, reorient and center the molecule to align its symmetry elements with the coordinate system, and replace the input structure by a geometry that corresponds exactly to this point group and which minimizes the sum of square distances between the atoms of both structures.

Any program that attempts to find the point group of an arbitrary atom cluster must be prepared to cope with some amount of numerical noise in the atom coordinates. ORCA by default allows each atom to deviate at most 10^{-4} atomic units from the ideal position that is consistent with the point group being examined. The rationale behind this value is the rounding error that occurs when the user feeds Cartesian coordinates with five significant digits after the decimal point into the program which otherwise represent an exact (symmetry-adapted) geometry. A threshold that is about one order of magnitude higher than the numerical noise in the coordinates is usually very safe.

If the maximum error in the Cartesian coordinates exceeds these 10^{-4} atomic units, the symmetry module in ORCA will fail to recognize the expected point group. The user is strongly advised to always make sure that the detected point group meets their expectations. If the point group reported by the symmetry module appears to be too low, the user may try to increase the detection threshold to 10^{-3} or 10^{-2} Bohr radii using option SymThresh in the %Symmetry input block:

```
%Sym SymThresh 0.01 End
```

A great method to obtain a structure with perfect symmetry avoiding any expensive calculation is to use the simple keywords ! NoIter XYZFile with an appropriate threshold. The structure in the resulting file with the extension .xyz may then be used as input for the actual calculation.

To give an illustrative example, coordinates for staggered ethane have been obtained by geometry optimization *without* using symmetry. If symmetry is turned on, point group C_i is recognized instead of the expected point group D_{3d} due to the remaining numerical noise. To counter this, the detection threshold is increased to 10^{-2} a. u. and a coordinate file with perfect symmetry is produced by the following input:

```
! RHF SVP NoIter XYZfile
%sym SymThresh 1.0e-2 end
*xyz 0 1
  C    -0.002822   -0.005082   -0.001782
  C    -0.723141   -1.252323   -0.511551
  H     0.017157    0.029421    1.100049
```

(continues on next page)

(continued from previous page)

```
  H     1.042121    0.030085   -0.350586
  H    -0.495109    0.917401   -0.350838
  H    -0.743120   -1.286826   -1.613382
  H    -0.230855   -2.174806   -0.162495
  H    -1.768085   -1.287489   -0.162747
*
```

If ORCA fails to find the expected point group even though a value of 10^{-2} atomic units has been selected for `SymThresh`, the user is strongly advised to take a careful look at the structure by means of their favorite visualization tool before increasing this value any further. Look for any obvious distortions or even missing atoms. An especially tricky point may be the orientation of methyl groups or the conformation of floppy side chains. A small rotation about a single bond may be enough to push some atom positions above the limit. If the conformational deviations cannot be fixed using a molecular editor or modelling program, a possible alternative may be to pre-optimize the structure without symmetry using a less expensive method like `PB86` and a small basis set like `def2-SVP`. Even several passes of pre-optimization and structure editing may be considered until all symmetry-equivalent side chains are locked in the same conformation so that ORCA finally detects the correct point group.

It is not recommended to run calculations using a value of `SymThresh` which is much too high or much too small since this may result in some really strange behavior of the symmetry module. Consider for instance the following input file which contains a perfectly octahedral geometry of a sulfur hexafluoride molecule. Its coordinates may be easily created by hand by placing the sulfur atom into the origin and two fluorine atoms on each coordinate axis at equal distances r from the origin ($r = 1.56$ Å or approximately 2.95 atomic units). Using a value for `SymThresh` as large as 0.1 Bohr radii works fine in this case, resulting in the correct point group O_h.

```
# Sulfur hexafluoride (SF6), point group Oh.
! BP86 def2-SVP
%Sym SymThresh 0.1 End
* xyz 0 1
S    0.00    0.00    0.00
F    1.56    0.00    0.00
F   -1.56    0.00    0.00
F    0.00    1.56    0.00
F    0.00   -1.56    0.00
F    0.00    0.00    1.56
F    0.00    0.00   -1.56
*
```

However, if `SymThresh` is increased further to $t = 0.5$ atomic units, the point group detection algorithm breaks down (strange warnings are printed as a consequence) and the reported point group decreases to C_i (in which the center of inversion is the only non-trivial symmetry element). This is because the center of inversion is easy to detect and this is done by one of the early checks. The breakdown of the point group recognition may be explained as follows. During the process of point group detection the symmetry module is of course unaware that the given input geometry is exact. Hence it will be treated as any other input structure. A value of $t = 0.5$ Bohr radii for `SymThresh` means that the unknown exact atom position is located within a sphere of radius $t = 0.5$ atomic units around the input atom position. The input distance $a = \sqrt{2}\,r$ between two adjacent fluorine atoms is approximately $a \approx 2.21$ Å ≈ 4.17 a. u., so their unknown exact distance d may vary in the following interval (see the diagram in Fig. 2.6):

$$d_{\text{min}} = a - 2t = 3.17\,\text{a.u.} \leq d \leq d_{\text{max}} = a + 2t = 5.17\,\text{a.u.}$$

Analogously, the unknown exact distance d' between two opposite fluorine atoms with the input distance $a' = 2r = 5.90$ a. u. is:

$$d'_{\text{min}} = a' - 2t = 4.90\,\text{a.u.} \leq d' \leq d'_{\text{max}} = a' + 2t = 6.90\,\text{a.u.}$$

Since the possible intervals of d and d' overlap (due to $d_{\text{max}} > d'_{\text{min}}$), *all* fifteen F–F distances are considered equal. Since there is no solid with six vertices and fifteen equal inter-vertex distances in three dimensions, the point group detection algorithm fails.

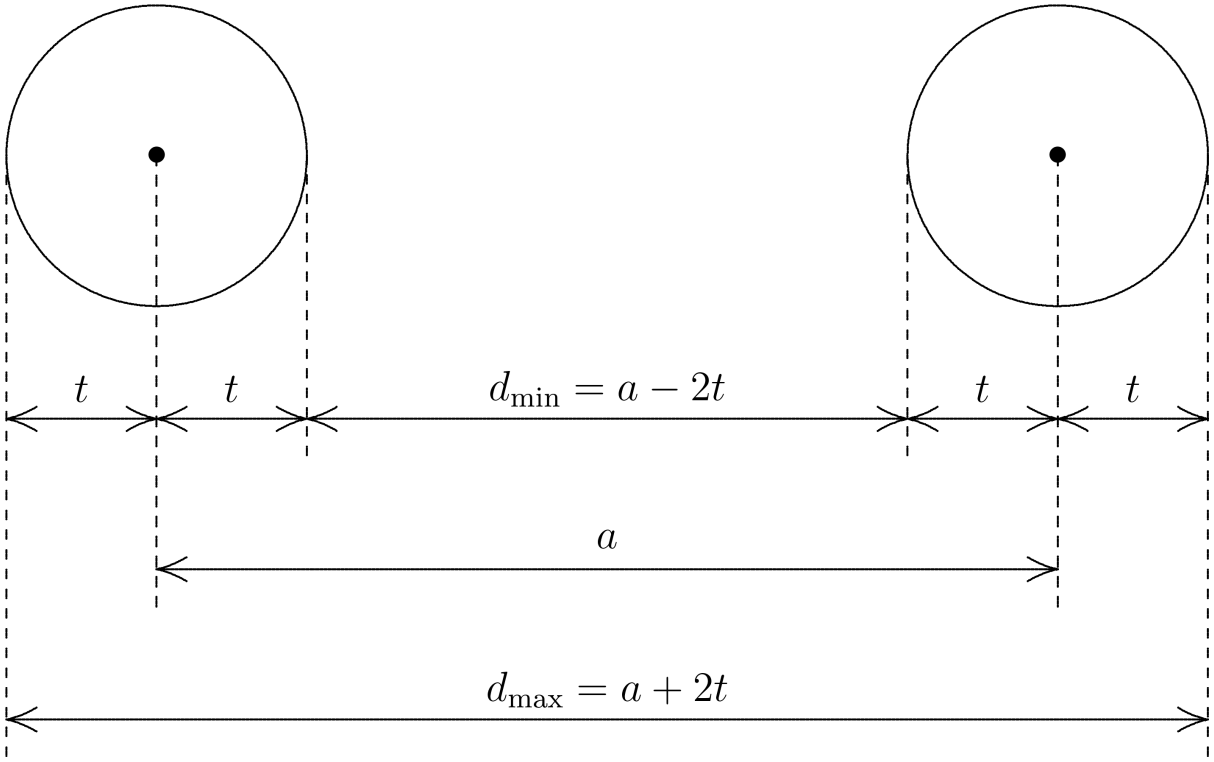

Fig. 2.6: The relation between the value t of `SymThresh`, the distance a of some input atom pair, and the allowed interval $[d_{\min}, d_{\max}]$ for the distance d between the exact atom positions. This interval has the width $d_{\max} - d_{\min} = 4t$.

2.19.2 Geometry optimizations using symmetry

If a geometry optimization is performed with symmetry turned on, ORCA will first determine the point group of the starting structure and replace the geometry that is presumed to contain numerical noise with one that has perfect symmetry. Starting with ORCA 6, the optimizer will clean up the gradient at every step of the optimization if requested by setting option `CleanUpGradient true` in the `%Symmetry` input block. The gradient cleanup is done by projecting out all components that are not totally symmetric. This way the symmetry of the molecule cannot decrease during the optimization.

By default, the point group is determined from scratch again after the geometry has been updated at every step of the optimization. This behaviour may be switched off by setting option `SymRelaxOpt false` in the `%Symmetry` input block. In this case the point group of the molecule is actually frozen during the entire optimization.

The following table summarizes the behaviour of the optimizer depending on the options `SymRelaxOpt` and `CleanUpGradient`:

`SymRelaxOpt`	`CleanUpGradient`	Behaviour
`true`	`true`	Symmetry may increase but not decrease.
`true`	`false`	Symmetry may change freely.
`false`	`true`	Symmetry will be frozen.
`false`	`false`	Setting not recommended.

Setting both switches `false` would allow the point group to change during the optimization but at the same time, a change would be impossible to detect. Therefore this setting is strongly discouraged.

2.19.3 Default alignment of the symmetry elements with the coordinate system

If ORCA determines the point group of a molecule and the user has not selected any special options, the following principles apply to the manner in which the symmetry elements of the full point group are aligned with the coordinate system:

1. The center of mass of the molecule will be shifted into the origin by default.[1] If the point group leaves one *unique* vertex invariant to all symmetry operations, the center of mass agrees with this vertex. This is the case for all point groups except C_s, C_n $(n \geq 1)$, C_{nv} $(n \geq 2)$, and $C_{\infty v}$.
2. If the molecule exhibits a unique axis of symmetry with the highest number of positions, this axis will become the z axis. This applies to all point groups except C_1, C_i, C_s, D_2, D_{2h}, the cubic point groups, and K_h.
3. For point group C_s, the mirror plane will become the xy plane.
4. For point groups C_{nv} $(n \geq 2)$, one of the vertical mirror planes will become the xz plane.
5. For point groups D_n $(n \geq 3)$, D_{nh} $(n \geq 3)$, and D_{nd} $(n \geq 2)$, one of the two-fold rotation axes perpendicular to the axis with the highest number of positions will become the x axis.
6. For point groups D_2, D_{2h}, T, and T_h, the three mutually orthogonal C_2 axes will become the coordinate axes.
7. For point groups T_d, O, and O_h, the three mutually orthogonal four-fold rotation or rotation-reflection axes will become the coordinate axes.
8. Finally, for point groups I and I_h, one of the five sets of three mutually orthogonal C_2 axes will become the coordinate axes. The pair of C_5 or S_{10} axes closest to the z axis will be located in the yz plane.
9. In general the orientation of the molecule will be changed as little as possible to meet the criteria above. If the input geometry meets these criteria already, the molecule will not be moved or rotated at all.

If the point group of the system is D_{nd} with $n \geq 2$ or T_d and the user has selected subgroup C_{2v} using option `PreferC2v`, the following rules apply instead:

- For point group D_{nd} with $n \geq 2$, one of the diagonal mirror planes will become the xz plane.
- For point group T_d, one of the diagonal mirror planes containing the z axis will become the xz plane, i. e. the molecule will be rotated by 45 degrees about the z axis compared to the default orientation.

Table 2.63 gives an overview over all point groups and the way in which the symmetry elements of the reduced point group (the largest common subgroup of D_{2h}) are aligned with the coordinate system.

Table 2.63: Point groups and corresponding subgroups suitable for electronic-structure calculations.

Full point group	Index n	Unique center[Page 195, 2]	Consistent with planar molecule[Page 195, 3]	Chosen subgroup	Alignment of the subgroup[Page 195, 4]
C_1		no	no	C_1	
C_i		i	no	C_i	
C_s		no	yes	C_s	
C_n	odd	no	no	C_1	
	even	no	no	C_2	z axis
C_{nv}	odd	no	no	C_s	xz plane
	even	no	for $n = 2$	C_{2v}	z, xz, yz
C_{nh}	odd	yes	yes	C_s	xy plane
	even	i	yes	C_{2h}	z, xy
D_n	odd	yes	no	C_2	x axis
	even	yes	no	D_2	
D_{nh}	odd	yes	yes	C_{2v}	x, xy, xz
	even	i	yes	D_{2h}	
D_{nd}	odd	i	no	C_{2h}	x, yz
	even	yes	no	D_2	

continues on next page

[1] In the very special case that the Z matrix contains no atoms with mass, the geometric center will be used instead.

Table 2.63 – continued from previous page

Full point group	Index n	Unique center[2]	Consistent with planar molecule[3]	Chosen subgroup	Alignment of the subgroup[4]
				C_{2v}	z, xz, yz
S_{2n}	odd	i	no	C_i	
	even	yes	no	C_2	z axis
T		yes	no	D_2	
T_h		i	no	D_{2h}	
T_d		yes	no	D_2	
				C_{2v}	z, xz, yz
O		yes	no	D_2	
O_h		i	no	D_{2h}	
I		yes	no	D_2	
I_h		i	no	D_{2h}	
$C_{\infty v}$		no	no	C_{2v}	z, xz, yz
$D_{\infty h}$		i	no	D_{2h}	
K_h		i	no	D_{2h}	

2.19.4 Irreducible representations of D_{2h} and subgroups

Table 2.64, Table 2.65, and Table 2.66 contain lists of the irreducible representations (also called species) and the corresponding characters of the point groups supported for electronic structure calculations in ORCA, and the product tables of these irreducible representations. Where the data depends on the alignment of the symmetry elements with the coordinate system, Mulliken's recommendations [176] are followed. This approach is in line with the recommendations by the IUPAC [177].

Table 2.64: Species and species product table of point group C_{2v}. The species table for C_{2v} corresponds to Table III in [176]. The directions of the two-fold axis and the mirror planes in each column are related to each other by cyclic permutations.

C_{2v}	E	$C_2(z)$ $C_2(x)$ $C_2(y)$	$\sigma_v(xz)$ $\sigma_v(xy)$ $\sigma_v(yz)$	$\sigma_v(yz)$ $\sigma_v(xz)$ $\sigma_v(xy)$
A_1	$+1$	$+1$	$+1$	$+1$
A_2	$+1$	$+1$	-1	-1
B_1	$+1$	-1	$+1$	-1
B_2	$+1$	-1	-1	$+1$

$\times$	A_1	A_2	B_1	B_2
A_1	A_1	A_2	B_1	B_2
A_2	A_2	A_1	B_2	B_1
B_1	B_1	B_2	A_1	A_2
B_2	B_2	B_1	A_2	A_1

Table 2.65: Species and species product table of point group D_2. The species table for D_2 has been obtained by dropping the center of inversion and the mirror planes from the species table for D_{2h} (see Table 2.66).

D_2	E	$C_2(z)$	$C_2(y)$	$C_2(x)$
A	$+1$	$+1$	$+1$	$+1$
B_1	$+1$	$+1$	-1	-1
B_2	$+1$	-1	$+1$	-1
B_3	$+1$	-1	-1	$+1$

$\times$	A	B_1	B_2	B_3
A	A	B_1	B_2	B_3
B_1	B_1	A	B_3	B_2
B_2	B_2	B_3	A	B_1
B_3	B_3	B_2	B_1	A

[2] A center of inversion is denoted i. "yes" indicates the existence of a *unique* vertex that remains invariant to all symmetry operations of the point group.

[3] This column indicates whether the given point group may be the *full* point group of a planar molecule.

[4] This column contains the elements (axes or planes) of the coordinate system that coincide with the symmetry elements of the reduced point group (the largest common subgroup of the full point group and D_{2h}) by *default*. If the full point group contains a unique principle axis of symmetry (with the highest number of positions), this axis is presumed to coincide with the z axis.

Table 2.66: Species and species product table of point group D_{2h}. The species table for D_{2h} corresponds to Table IV in [176].

D_{2h}	E	$C_2(z)$	$C_2(y)$	$C_2(x)$	i	$\sigma(xy)$	$\sigma(xz)$	$\sigma(yz)$
A_g	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$	$+1$
B_{1g}	$+1$	$+1$	-1	-1	$+1$	$+1$	-1	-1
B_{2g}	$+1$	-1	$+1$	-1	$+1$	-1	$+1$	-1
B_{3g}	$+1$	-1	-1	$+1$	$+1$	-1	-1	$+1$
A_u	$+1$	$+1$	$+1$	$+1$	-1	-1	-1	-1
B_{1u}	$+1$	$+1$	-1	-1	-1	-1	$+1$	$+1$
B_{2u}	$+1$	-1	$+1$	-1	-1	$+1$	-1	$+1$
B_{3u}	$+1$	-1	-1	$+1$	-1	$+1$	$+1$	-1

$\times$	A_g	B_{1g}	B_{2g}	B_{3g}	A_u	B_{1u}	B_{2u}	B_{3u}
A_g	A_g	B_{1g}	B_{2g}	B_{3g}	A_u	B_{1u}	B_{2u}	B_{3u}
B_{1g}	B_{1g}	A_g	B_{3g}	B_{2g}	B_{1u}	A_u	B_{3u}	B_{2u}
B_{2g}	B_{2g}	B_{3g}	A_g	B_{1g}	B_{2u}	B_{3u}	A_u	B_{1u}
B_{3g}	B_{3g}	B_{2g}	B_{1g}	A_g	B_{3u}	B_{2u}	B_{1u}	A_u
A_u	A_u	B_{1u}	B_{2u}	B_{3u}	A_g	B_{1g}	B_{2g}	B_{3g}
B_{1u}	B_{1u}	A_u	B_{3u}	B_{2u}	B_{1g}	A_g	B_{3g}	B_{2u}
B_{2u}	B_{2u}	B_{3u}	A_u	B_{1u}	B_{2g}	B_{3g}	A_g	B_{1u}
B_{3u}	B_{3u}	B_{2u}	B_{1u}	A_u	B_{3g}	B_{2g}	B_{1g}	A_g

2.19.5 Options available in the `%Symmetry` input block

Table 2.67 contains a list of the options available in the `%Symmetry` (or `%Sym`) input block. Options `SymThresh` and `SymRelax` (same as `SymRelaxSCF` below) can also be accessed in the `%Method` input block for backward compatibility. This use is deprecated and not recommended in new input files, however.

Table 2.67: List of options in the `%Symmetry` (`%Sym`) input block

Option	Type	Default	Description
`UseSymmetry`	Boolean	`True`	By setting this option to `False`, symmetry may be switched off even though the `%Symmetry` block is present in the input file.
`UseSym`	Boolean	`True`	Same as `UseSymmetry`.
`SymThresh`	Real	10^{-4}	Two vertices with a distance shorter than this threshold (in atomic units) are considered identical during point group recognition.
`PreferC2v`	Boolean	`False`	Indicates whether to prefer subgroup C_{2v} over D_2 for electronic-structure calculations where both choices are appropriate (point groups D_{nd} with odd n and T_d).
`PointGroup`	String	Empty string	If the user specifies a point group using this option, point group recognition will be skipped and the user must make sure that the molecule is oriented in the coordinate system in agreement with the conventions in *Default alignment of the symmetry elements with the coordinate system*. Note that the point group label must be enclosed in double quotes. Otherwise ORCA will complain about an invalid assignment.
`SymRelaxSCF`	Boolean	`False`	Indicates whether orbital occupation numbers of each irreducible representation are allowed to change during SCF.
`SymRelaxOpt`	Boolean	`True`	Indicates whether the point group will be determined from scratch in every step of a geometry optimization. A value of `True` will allow the point group to change in an arbitrary manner. Otherwise the initial point group will be imposed at every step no matter how far the distances between the current and the ideal structure exceed `SymThresh`.

continues on next page

Table 2.67 – continued from previous page

Option	Type	Default	Description
CleanUpCoords	Boolean	True	Determines whether the molecular geometry will be cleaned up using the automatically detected or user-specified point group. Even if CleanUpCoords is False, symmetrized coordinates will still be computed temporarily and a warning will be printed if the largest deviation from the original geometry exceeds SymThresh.
CleanUpGeom	Boolean	True	Same as CleanUpCoords.
CleanUpGrad	Boolean	True	Indicates whether the full point group of the molecule shall be used to remove all non-totally symmetric components from the gradient. This ensures that the point group will not decrease throughout the optimization.
CleanUpGradien	Boolean	True	Same as CleanUpGrad.
Print	Integer	1	Determines the output size for symmetry handling in general and point group detection in particular; 0 – No output during point group detection; 1 – Normal output; 2 – Detailed information; 3 – Debug print.
PrtSALC	Integer	0	Specifies the output size for the construction of symmetry-adapted linear combinations (SALCs) of atomic orbitals; 0 – No output for symmetry-adapted orbitals; 1 – Normal output; 2 – Detailed information (e. g. the SALCs themselves); 3 – Debug print.

2.20 Choice of Initial Guess and Restart of SCF Calculations

The initial guess is an important issue in each SCF calculation. If this guess is reasonable, the convergence of the procedure will be much better. ORCA makes some effort to provide a good initial guess and gives the user enough flexibility to tailor the initial guess to his or her needs.

The initial guess is also controlled via the %scf block and the variables Guess, MOInp and GuessMode.

```
%scf
  Guess     HCore       # One electron matrix
            Hueckel     # Extended H\"uckel guess
            PAtom       # Polarized atomic densities
            PModel      # Model potential
            MORead      # Restart from an earlier calc.
  MOInp     "Name.gbw"  # orbitals used for MORead
  GuessMode FMatrix     # FMatrix projection
            CMatrix     # Corresponding orbital projection
  AutoStart true        # try to use the orbitals from the existing
                        # GBW file of the same name (if possible)
                        # (default)
            false       # don't use orbitals from existing GBW file
end
```

2.20.1 One Electron Matrix Guess

The simplest guess is to diagonalize the one electron matrix to obtain starting orbitals. This guess is very simple but usually also a disaster because it produces orbitals that are far too compact.

2.20.2 Basis Set Projection

The remaining guesses (may) need the projection of initial guess orbitals onto the actual basis set. In ORCA there are two ways this can be done. `GuessMode FMatrix` and `GuessMode CMatrix`. The results from the two methods are usually rather similar. In certain cases `GuessMode CMatrix` may be preferable. `GuessMode FMatrix` is simpler and faster. In short the `FMatrix` projection defines an effective one electron operator:

$$\hat{f} = \sum_p \varepsilon_p a_p^\dagger a_p \tag{2.86}$$

where the sum is over all orbitals of the initial guess orbital set, $a_p^\dagger$ is the creation operator for an electron in guess MO p, a_p is the corresponding annihilation operator and ε_i is the orbital energy. This effective one electron operator is diagonalized in the actual basis and the eigenvectors are the initial guess orbitals in the target basis. For most wavefunctions this produces a fairly reasonable guess.

`CMatrix` is more involved. It uses the theory of corresponding orbitals to fit each MO subspace (occupied, partially occupied or spin-up and spin-down occupied) separately [178, 179]. After fitting the occupied orbitals, the virtual starting orbitals are chosen in the orthogonal complement of the occupied orbitals. In some cases, especially when restarting ROHF calculations, this may be an advantage. Otherwise, it is not expected that `CMatrix` will be grossly superior to `FMatrix` for most cases.

2.20.3 PModel Guess

The PModel guess (chosen by `Guess PModel` in the `%scf` block or simply a keyword line with `!PModel`) is one that is usually considerably successful. It consists of building and diagonalizing a Kohn–Sham matrix with an electron density which consists of the superposition of spherical neutral atoms densities which are predetermined for both relativistic and nonrelativistic methods. This guess is valid for both Hartree–Fock and DFT methods, but not for semiempirical models. However, due to the complexity of the guess it will also take a little computer time (usually less than one SCF iteration). The model densities are available for most atoms of the periodic table and consequently the PModel guess is usually the method of choice (particularly for molecules containing heavy elements) unless you have more accurate starting orbitals available.

2.20.4 Hückel and PAtom Guesses

The extended Hückel guess proceeds by performing a minimal basis extended Hückel calculation and projecting the MOs from this calculation onto the actual basis set using one of the two methods described above. The minimal basis is the STO-3G basis set. The Hückel guess may not be very good because the STO-3G basis set is so poor. There is also accumulating evidence that the superposition of atomic densities produces a fairly good initial guess. The critique of the atomic density method is that the actual shape of the molecule is not taken into account and it is more difficult to reliably define singly occupied orbitals for ROHF calculations or a reasonable spin density for UHF calculations. Therefore ORCA chooses a different way in the `PAtom` guess (which is the default guess): the Hückel calculation is simply carried out *for all electrons* in a minimal basis of atomic SCF orbitals. These were determined once and for all and are stored inside the program. This means that the densities around the atoms are very close to the atomic ones, all orbitals on one center are exactly orthogonal, the initial electron distribution already reflects the molecular shape and there are well defined singly occupied orbitals for ROHF calculations.

2.20.5 Restarting SCF Calculations

To restart SCF calculations, it can be very helpful and time-saving to read in the orbital information of a previous calculation. To do this, specify:

```
! moread
%moinp "name.gbw"
```

This is done by default for single-point calculations if the `.gbw` file of the same name exists – see *AutoStart feature*.

The program stores the current orbitals in every SCF cycle. Should a job crash, it can be restarted from the orbitals that were present at this time by just re-running the calculation to use the present `.gbw` file. In addition, an effort has been made to make `.gbw` files from different releases compatible with each other. If your input `.gbw` file is from an older release, use `! rescue moread noiter` with `% moinp "name.gbw"` to produce an up-to-date `.gbw`. When the `rescue` keyword is invoked, only the orbital coefficients are read from the `.gbw` file, and everything else from the input file. Thus, make sure that the geometry and the basis set of the old `.gbw` file and the new input match.

Within the same ORCA version, neither the geometry nor the basis set stored in `name.gbw` need to match the present geometry or basis set. The program merely checks if the molecules found in the current calculation and `name.gbw` are consistent with each other and then performs one of the possible orbital projections. If the two basis sets are identical the program by default only reorthogonalizes and renormalizes the input orbitals. However, this can be overruled by explicitly specifying `GuessMode` in the `% scf` block as `CMatrix` or `FMatrix`.

If redundant components were removed from the basis (see *Linear Dependence*), then `! moread noiter` must not be used to read SCF orbitals from a previous calculation, as it is going to lead to wrong results. In that case, `! moread` may be used (without `noiter`) if doing the entire calculation in one go is not possible.

If `!moread` and `%moinp` are used, the input `.gbw` file from the earlier calculation must have a different name than the new calculation, because in the very beginning of a calculation, a new `.gbw` file is written. If the names are the same, the `.gbw` file from the earlier calculation is overwritten and all information is lost. Therefore, if you want to restart a calculation with an input file of the same name as the previous calculation, you have to rename the `.gbw` file first. This is good practice anyway to avoid surprises, particularly for expensive calculations. Alternatively, you can use the *AutoStart feature* instead of explicitly specifying `!moread` and `%moinp`.

There is an additional aspect of restarting SCF calculations — if you have chosen `SCFMode = Conventional` the program stores a large number of integrals that might have been time consuming to calculate on disk. Normally the program deletes these integrals at the end of the calculation. However, if you want to do a closely related calculation that requires the same integrals (i.e. the *geometry*, the *basis set* and the *threshold* `Thresh` are the same) it is faster to use the integrals generated previously. This is done by using `KeepInts = true` in the `% scf` block of the first calculation and then use `ReadInts = true` in the `% scf` block of the second calculation. If the second calculation has a different name than the first calculation you have to use `IntName = "FirstName"` to tell the program the name of the integral files. Note that the file containing the integrals does not have an extension — it is simply the name of the previous input file with `.inp` stripped off.

```
%scf
  KeepInts true          # Keep integrals on disk
  ReadInts true          # Read integrals from disk
  IntName  "MyInts"      # Name of the integral files without extension
end
```

Note that, in general, restarting calculations with old integral files requires the awareness and responsibility of the user. If properly used, this feature can save considerable amounts of time.

2.20.6 AutoStart feature

Older versions of ORCA always created a new GBW file at the beginning of the run no matter whether a file of the same name existed or perhaps contained orbitals. Now, in the case of single-point calculations the program automatically checks if a `.gbw` file of the same name exists. If yes, the program checks if it contains orbitals and all other necessary information for a restart. If yes, the variable `Guess` is set to `MORead`. The existing `.gbw` file is renamed to `BaseName.ges` and `MOInp` is set to this filename. If the `AutoStart` feature is not desired, set `AutoStart false` in the `%scf` block or give the keyword `!NoAutoStart` in the simple input line. Note that `AutoStart` is ignored for geometry optimizations: in this case, using previously converged orbitals contained in a `.gbw` file (of a different name) can be achieved via `MORead` and `MOInp`.

2.20.7 Changing the Order of Initial Guess MOs and Breaking the Initial Guess Symmetry

Occasionally you will want to change the order of initial guess MOs — be it because the initial guess yielded an erroneous occupation pattern or because you want to converge to a different electronic state using the orbitals of a previous calculation. Reordering of MOs and other tasks (like breaking the symmetry of the electronic wavefunction) are conveniently handled with the `Rotate` feature in ORCA. `Rotate` is a subblock of the SCF block that allows you to linearly transform pairs of MOs.

```
%scf
  Rotate
   { MO1, MO2, Angle }
   { MO1, MO2, Angle, Operator1, Operator2 }
   { MO1, MO2} # Shortcut to swap MO1 and MO2. Angle=90 degrees.
  end
end
```

Here, `MO1` and `MO2` are the indices of the two MOs of interest. Recall that ORCA starts counting MOs with index 0, i.e. the MO with index 1 is the *second* MO. `Angle` is the rotation angle in degrees. A rotation angle of 90° corresponds to flipping two MOs, an angle of 45° leads to a 50:50 mixture of two MOs, and a 180° rotation leads to a change of phase. `Operator1` and `Operator2` are the orbitals sets for the rotation. For UHF calculations spin-up orbitals belong to operator 0 and spin-down orbitals to operator 1. RHF and ROHF calculations only have a single orbital set.

Among others, the `Rotate` feature can be used to produce broken-symmetry solutions, for example in transition metal dimers. In order to do that, first perform a high-spin calculation, then find the pairs of MOs that are symmetric and antisymmetric combinations of each other. Take these MOs as the initial guess and use rotations of 45° for each pair to localize the starting MOs. If you are lucky and the broken symmetry solution exists, you have a good chance of finding it this way. See section *Broken-Symmetry Wavefunctions and Exchange Couplings* for more details on the broken-symmetry approach.

2.20.8 Automatically Breaking of the Initial Guess Symmetry

Another simple way to break the initial guess symmetry for more trivial cases, is to simply use the keyword `! GUESSMIX`. This will automatically mix 50% of the alpha LUMO into the alpha HOMO. That is equivalent to a 45 degree rotation as done above and only for these orbitals. It might be useful when one wants an open-shell singlet and needs the alpha and beta orbitals to start differently.

The specific angle of rotation can be controlled with:

```
%scf
  guessmix 75 # angle in degrees, default is 45
end
```

2.20.9 Calculating only the energy of an input density

In case you want to give the result of a previous SCF and recalculate the energy, or maybe some other property (like the MP2 energy) using that density without changing the orbitals, you can use the flags `!CALCGUESSENERGY NOITER`.

The SCF program will read the orbitals, compute the density and one Fock matrix necessary to get the energy and move on with no orbital updates. This can be used to combine DFT orbitals with DLPNO-CCSD(T) for example. Be careful with the results you get from this because these orbitals are not variational anymore!

2.20.10 Keywords

Table 2.68: Simple input keywords related to the initial guess.

Keyword	Description
`PAtom`	Selects the polarized atoms guess
`PModel`	Selects the model potential guess
`Hueckel`	Selects the extended Hückel guess
`HCore`	Selects the one-electron matrix guess
`MORead`	Read MOs from a previous calculation (use together with `%moinp "myorbitals.gbw"`)
`AutoStart`	Try to start from an existing GBW file of the same name as the present one (default; only for single-point calculations)
`NoAutoStart`	Don't try to do that
`NoIter`	Sets the number of SCF iterations to 0. This works together with MOREAD and means that the program will work with the provided starting orbitals.
`KeepInts`	Do not delete the integrals from disk after a calculation in conventional mode
`ReadInts`	Read the existing integrals from a previous calculation in conventional mode (use together with `%scf IntName "PreviousBaseName" end`)
`CalcGuessEner`	Calculate the energy of the initial guess orbitals (use together with `NoIter`)

2.21 Frozen Core Options

The frozen core (FC) approximation is usually applied in correlated calculation and consists in neglecting correlation effects for electrons in the low-lying core orbitals. The FC approximation and the number of core electrons per element can be adjusted in the %method block. The default number of core electrons per element is listed in Table 2.69.

Table 2.69: Default values for number of frozen core electrons.

H																He
0																0
Li	Be											B	C	N	O	F	Ne
0	0											2	2	2	2	2	2
Na	Mg											Al	Si	P	S	Cl	Ar
2	2											10	10	10	10	10	10
K	Ca	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Cu	Zn	Ga	Ge	As	Se	Br	Kr
10	10	10	10	10	10	10	10	10	10	10	10	18	18	18	18	18	18
Rb	Sr	Y	Zr	Nb	Mo	Tc	Ru	Rh	Pd	Ag	Cd	In	Sn	Sb	Te	I	Xe
18	18	28	28	28	28	28	28	28	28	28	28	36	36	36	36	36	36
Cs	Ba	Lu	Hf	Ta	W	Re	Os	Ir	Pt	Au	Hg	Tl	Pb	Bi	Po	At	Rn
36	36	46	46	46	46	46	46	46	46	46	46	68	68	68	68	68	68
Fr	Ra	Lr	Rf	Db	Sg	Bh	Hs	Mt	Ds	Rg	Cn						
68	68	68	100	100	100	100	100	100	100	100	100						
Lan-thanide			La	Ce	Pr	Nd	Pm	Sm	Eu	Gd	Tb	Dy	Ho	Er	Tm	Yb	
			36	36	36	36	36	36	36	36	36	36	36	36	36	36	
Ac-tinides			Ac	Th	Pa	U	Np	Pu	Am	Cm	Bk	Cf	Es	Fm	Md	No	
			68	68	68	68	68	68	68	68	68	68	68	68	68	68	

For systems containing heavy elements, core electrons might have higher orbital energies compared to the orbital energies of valence MOs of some lighter elements. In that case, core electrons might be included in the correlation calculation, which ultimately leads to large errors in correlation energy. In order to prevent this, the MO ordering is checked: Do all lower energy MOs in the core region have core electron character, i.e. are they strongly localized on the individual elements? For post-(CAS)SCF calulations, this check is always performed both after the SCF calculation, and after the initial guess (because the SCF may be skipped with `!NoIter`). For other calculations, the check is off by default but may be switched on with the `CheckFrozenCore` keyword in the `%method` block. If core orbitals are found in the valence region, while more delocalized orbitals are found in the core region, the corresponding MO pairs are swapped. This behavior can be disabled using the `CorrectFrozenCore` keyword.

```
%method
  FrozenCore FC_ELECTRONS #Freeze all core electrons
             FC_EWIN      #Freeze selected core electrons via an energy window
                          #e.g. for MP2: %mp2 EWin EMin,EMax
             FC_NONE      #No frozencore approximation
             -n           #Freeze a total of n electrons

  NewNCore Bi 68 end      #Set the number of core electrons for Bi to 68
  CheckFrozenCore   true  #Check whether frozen core orbitals are ordered correctly
                          #Default: true only for post-(CAS)SCF calculations
  CorrectFrozenCore true  #Whether to rotate valence orbitals out of the core␣
↪region
end
```