(Please see section *Hints on the Use of Parallel ORCA* for what else has to be taken care of for a successfull parallel run of ORCA.)

2.5.1 Parallel and Multi-Process Modules

The following modules and utility programs are presently parallelized - or usable in Multi-Process Mode:

List of Parallelized Modules

AUTOCI - all methods
CASSCF / NEVPT2 / CASPT2 / CASSCFRESP
CIPSI
CIS/TDDFT / CISRESP
EDA
GRAD
GUESS
LEANSCF
MAGRELAX
MCRPA
MDCI (Canonical- and DLPNO-Methods)
MM
MP2 and RI-MP2 (including Gradients)
MRCI
PC
PLOT
PNMR
POP
PROP
PROPINT
REL
ROCIS
SCFGRAD
SCFRESP (with SCFHessian)
STARTUP
VPOT

(For a complete list of all modules and the description of their functionality, please refer to *Program Components*)

The efficiency of the parallel modules is such that for RI-DFT perhaps up to 16 processors are a good idea while for hybrid DFT and Hartree-Fock a few more processors are appropriate. Above this, the overhead becomes significant and the parallelization loses efficiency. Coupled-cluster calculations usually scale well up to at least 8 processors but probably it is also worthwhile to try 16.

List of Multi-Process Modules

Numerical Gradients, Frequencies, Overtones-and-Combination-Bands
VPT2
NEB (Nudged Elastic Band)
GOAT (Global Optimizer Algorithm)

For Numerical Frequencies or Gradient runs it makes sense to choose nprocs = 4 or 8 times 6*Number of Atoms. For VPT2 on larger systems you may well even try 16 times 6*Number of Atoms - if you use multiple processes per

displacement. (Please check out the section *Hints on the Use of Parallel ORCA* what you have to take care of for such kind of calculations.)

Note

Parallelization is a difficult undertaking and there are many different protocols that work differently for different machines. Please understand that we can not provide support for each and every platform. We are trying our best to make the parallelization transparent and provide executables for various platforms but we can not guarantee that they always work on every system. Please see the download information for details of the version.

2.5.2 Hints on the Use of Parallel ORCA

Many questions that are asked in the discussion forum deal with the parallel version of ORCA. Please understand that we cannot possibly provide one-on-one support for every parallel computer in the world. So, please make every effort to solve the technical problems locally together with your system administrator. Here are some explanations about what is special to the parallel version, which problems might arise from this and how to deal with them:

Single Node Runs

1. Parallel ORCA can be used with OpenMPI (on Linux and MAC) or MS-MPI (on windows) only. Please see the download information for details of the relevant OpenMPI-version for your platform.

   The OpenMPI version is configurable in a large variety of ways, which cannot be covered here. For a more detailed explanation of all available options, cf. http://www.open-mpi.org

2. Please note that the OpenMPI version is dynamically linked, that is, it needs at runtime the OpenMPI libraries (and several other standard libraries)! If you compile MPI on your own computer, you also need to have a fortran compiler, as `mpirun` will contain fortran bindings.

   (Remember to set `PATH` and `LD_LIBRARY_PATH` to mpirun and the mpi libraries)

3. Many problems arise, because ORCA does not find its parallel executables. To avoid this, it is crucial to call ORCA with its complete pathname. The easiest and safest way to do so is to include the directory with the ORCA-executables in your `$PATH`. Then start the calculation:

   ```
   - interactively:
     start orca with full path: "/mypath_orca_executables/orca MyMol.inp"
   - batch :
     set your path: `export PATH=/mypath_orca_executables:$PATH` (for bash) then
     start orca with full path: "$PATH/orca $jobname.inp"
   ```

   This seems redundant, but it really is important if you want to run a parallel calculation to call ORCA with the full path! Otherwise it will not be able to find the parallel executables.

4. It is recommended to run orca in local (not nfs-mounted) scratch-directories, (`/tmp1` or `/usr/local` e.g.) and to renew these directories for each run to avoid confusion with left-overs of a previous run.

5. It has proven convenient to use “wrapper” scripts. These scripts should

   ```
   - set the path
   - create local scratch directories
   - copy input files to the scratch directory
   - start orca
   - save your results
   - remove the scratch directory
   ```

   A basic example of such a submission script for the parallel ORCA version is shown later (13.) (this is for the Torque/PBS queuing system, running on Apple Mac OS X):

6. Parallel ORCA distinguishes the following cases of disk availability:

1. each process works on its own (private) scratch directory (the data on this directory cannot be seen by any other process). This is flagged by **"working on local scratch directories"**
2. all processes work in a common scratch directory (all processes can see all file-data) ORCA will distinguish two situations:
   - all processes are on the same node - flagged by **"working on a common directory"**
   - the processes are distributed over multiple nodes but accessing a shared filesysten - flagged by **"working on a shared directory"**
3. there are at least 2 groups of processes on different scratch directories, one of the groups consisting of more than 1 process - flagged by **"working on distributed directories"**

Parallel ORCA will find out, which of these cases exists and will handle the I/O respectively. If ORCA states disk availability differently from what you would expect, check the number of available nodes and/or the distribution pattern (fill_up/round_robin)

7. It is possible to pass additional MPI-parameters to mpirun by adding these arguments to the ORCA call - all arguments enclosed in a single pair of quotes:

```
/mypath_orca_executables/orca MyMol.inp "--bind-to core"
```

– or – for multiple arguments

```
/mypath_orca_executables/orca MyMol.inp "--bind-to core --verbose"
```

Multi-Node Runs - Remote Execution

1. If Parallel ORCA finds a file named "`MyMol.nodes`" in the directory where it's running, it will use the nodes listed in this file to start the processes on, provided your input file was "`MyMol.inp`". You can use this file as your machinefile specifying your nodes, using the usual OpenMPI machinefile notation.

```
node1 cpu=2
node2 cpu=2
```

or

```
node1
node1
node2
node2
```

If you run the Parallel ORCA version on only one computer, you do not need to provide a nodefile, and neither have to enable an rsh/ssh access, as in this case the processes will simply be forked! If you start ORCA within a queueing system, you also don't need to provide a nodefile. The queueing system will care for it.

2. If the ORCA-environment variables are not equally defined on all participating compute nodes it might be advisable to export these variables. This can be achieved by passing the following additional parameters to mpirun via the ORCA call:

```
/mypath_orca_executables/orca MyMol.inp "-x LD_LIBRARY_PATH -x PATH"
```

3. OpenMPI requires that the PATH environment variable be set to find executables on remote nodes. As it is not always possible to change the startup scripts accordingly, OpenMPI provides the additional option: "`--prefix`" Calling ORCA and requesting multiple processes for the parallelized modules should then look like

```
/mypath_orca_executables/orca MyMol.inp "--prefix /my-openmpi-folder"
/mypath_orca_executables/orca MyMol.inp "--prefix /my-openmpi-folder --
↪machinefile MyMol.nodes" (if not started via queueing system)
```

As ORCA is dynamically linked it also needs to know the location of the ORCA libraries This can be communicated via

```
/mypath_orca_executables/orca MyMol.inp "--prefix /my-openmpi-folder --
↪machinefile MyMol.nodes -x LD_LIBRARY_PATH"
```

Multi-Process Calculations

1. An additional remark on multi-process numerical calculations (Frequencies, Gradient, (Hybrid) Hessian), VPT2, NEB, GOAT:
   The processes that execute these calculations do not work in parallel, but independently, often in a totally asynchronous manner. The numerical calculations will start as many processes, as you dedicated for the parallel parts before and they will run on the same nodes. If your calculation runs on multiple nodes, you have to set the environment variable `RSH_COMMAND` to either "`rsh`" or "`ssh`". If `RSH_COMMAND` is not defined, ORCA will abort. This prevents that all processes of a multi-node run are started on the 'master'-node.

2. On multiple user request the 'parallelization' of NumCalc has been made more flexible. If before ORCA would start nprocs displacements with a single process each, the user can now decide on how many processes should work together on a single displacement.

   For this the nprocs keyword got a sibling:

   ```
   %pal nprocs       32 # or nprocs_world - total number of parallel processes
        nprocs_group  4 #                 - number of parallel processes per sub-
   ↪task
        end
   ```

   This setting will ORCA make use 32 processes, with 4 processes working on the same displacement, thus running 8 displacements simultaneously. The methods that can profit from this new feature are

   - all NumCalc-methods: as NumGrad, NumFreq, VPT2, Overtones, NEB, and GOAT.
   - the analytical (parallel) Hessian, leading to a nice increase of parallel performance for really large calculations.

   It is highly recommended to choose nprocs_group to be an integer divisor of nprocs_world!

   For convenient use a couple of standard 'groupings' are made available via simple input keyword:

   ```
   !PAL4(2x2)   # 2 groups a 2 workers
   !PAL8(4x2)   # 4 groups a 2 workers
   !PAL8(2x4)   # ...
   !PAL16(4x4)
   !PAL32(8x4)
   !PAL32(4x8)
   !PAL64(8x8)
   ```

   Note

   If your system-administration does not allow to connect via rsh/ssh to other compute nodes, you unfortunately cannot make use of parallel sub-calculations within NumCalc runs. This affects NEB as well as GOAT, VPT2, Overtone-and-Combination-Bands, as well as Numerical Frequencies and Gradients.

3. Wrapper script to start ORCA

   ```
   #!/bin/zsh

   setopt EXTENDED_GLOB
   setopt NULL_GLOB
   #export MKL_NUM_THREADS=1
   ```

(continues on next page)

(continued from previous page)

```
b=${1:r}

#get number of procs.... close your eyes... (it really works!)
if [[ ${$(grep -e '^!' $1):u} == !*(#b)PAL(<0-9>##)* ]]; then
  nprocs=$match
let "nodes=nprocs"
elif [[ ${(j: :) $(grep -v '^#' $1):u} == *%(#b)PAL*NPROCS' '#(<0-9>##)* ]];␣
↪then
  nprocs=$match
  let "nodes=nprocs"
fi

cat > ${b}.job <<EOF
#!/bin/zsh
#PBS -l nodes=1:ppn=${nodes:=1}
#PBS -S /bin/zsh
#PBS -l walltime=8760:00:00

setopt EXTENDED_GLOB
setopt NULL_GLOB
export PATH=$PBS_O_PATH

logfile=$PBS_O_WORKDIR/${b}.log
tdir=$(mktemp -d /Volumes/scratch/$USER/${b}__XXXXXX)

trap '
echo "Job terminated from outer space!" >> $logfile
rm -rf $tdir
exit
' TERM

cp $PBS_O_WORKDIR/$1 $tdir
foreach f ($PBS_O_WORKDIR/*.gbw $PBS_O_WORKDIR/*.pot) { cp $f $tdir }
cd $tdir

echo "Job started from ${PBS_O_HOST}, running on $(hostname) in $tdir using
$(which orca)" > $log
file
=orca $1 1>>$logfile 2>&1

cp ^(*.(inp|tmp*))  $PBS_O_WORKDIR/
rm -rf $tdir

EOF

qsub -j oe -o ${b}.job.out ${b}.job
```

2.6 Self-Consistent-Field (SCF)

SCF convergence is a pressing problem in any electronic structure package because the total execution times increases linearly with the number of iterations. Thus, it remains true that the best way to enhance the performance of an SCF program is to make it converge better. In some cases, especially for open-shell transition metal complexes, convergence may be very difficult. ORCA makes a dedicated effort to achieve reasonable SCF convergence for these cases without compromising efficiency.

Another issue is whether the solution found by ORCA is stable, i.e. a minimum on the surface of orbital rotations. Especially for open-shell singlets it can be hard to achieve a broken-symmetry solution. The SCF stability analysis (section *SCF Stability Analysis*) may be able to help in such situations. Please also note that if `! TRAH` is used the

solution must be a true local minimum though not necessarily a global.

Tip

The expectation value $\langle S^2 \rangle$ is an estimation of the spin contamination in the system. It is highly recommended for open-shell systems, especially with transition metal complexes, to check the UCO (unrestricted corresponding orbitals, see *UNO, UCO and QROs input*) overlaps and visualise the corresponding orbitals. Additionally, spin-population on atoms that contribute to the singly occupied orbitals is also an identifier of the electronic structure.

2.6.1 Convergence Tolerances

Before discussing how to converge a SCF calculation it should be defined what is meant by "converged". ORCA has a variety of options to control the target precision of the energy and the wavefunction that can be selected in the `% scf` block, or with a simple input line keyword that merges the criterion label with "SCF", e.g. `! StrongSCF` or `! VeryTightSCF`:

```
%scf
  Convergence             # The default convergence is between medium and strong
               Sloppy     # very weak convergence
               Loose      # still weak convergence
               Medium     # intermediate accuracy
               Strong     # stronger
               Tight      # still stronger
               VeryTight  # even stronger
               Extreme    # close to numerical zero of the computer
                          # in double precision arithmetic
 end
```

Like other keys, `Convergence` is a compound key that assigns default values to a variety of other variables given in the box below. In Table 2.9 we present the chosen values for each compound key. If the corresponding simple inputs are given (`!StrongSCF`, `!VeryTightSCF`, …etc), then in addition the values in Table 2.10 are also set, which also influence the behavior of post-SCF modules. The default convergence criteria are reasonable and should be sufficient for most purposes. For a cursory look at populations weaker convergence may be sufficient, whereas other cases may require stronger than default convergence. Note that `Convergence` does not only affect the target convergence tolerances but also the integral accuracy as discussed in the section about direct SCF and alike. **This is very important: if the error in the integrals is larger than the convergence criterion, a direct SCF calculation cannot possibly converge.**

The convergence criteria are always printed in the output. Given below is a list of the convergence criteria for `! TightSCF`, which is often used for calculations on transition metal complexes.

```
%scf
  TolE        1e-8  # energy change between two cycles
  TolRMSP     5e-9  # RMS density change
  TolMaxP     1e-7  # maximum density change
  TolErr      5e-7  # DIIS error convergence
  TolG        1e-5  # orbital gradient convergence
  TolX        1e-5  # orbital rotation angle convergence
  Thresh    2.5e-11 # integral prescreening threshold
  TCut      2.5e-12 # primitive integral prescreening cutoff
  ConvCheckMode  2  # = 0: check all convergence criteria
                    # = 1: stop if one of criterion is met, this is sloppy!
                    # = 2: check change in total energy and in one-electron energy
                    #      Converged if delta(Etot)<TolE and delta(E1)<1e3*TolE
  ConvForced        # = 0: convergence not mandatory for next calculation step
                    # = 1: break, if you did not meet the convergence criteria
end
%method
```

(continues on next page)

(continued from previous page)

```
  Z_Tol       1e-4  # CP-SCF solver tolerance
  BFCut       1e-11 # basis function cutoff for numerical integration
end

# Additional tolerances modified by the simple input
%casscf
  GTol       2.5e-4
  ETol       2.5e-8
end
%mdci
  STol         1e-5
end
%autoci
  STol         1e-5
end
%mrci
  ETol       2.5e-7
  RTol       2.5e-7
end
%cis
  ETol       2.5e-7
  RTol       2.5e-7
end
```

Table 2.9: Threshold choices for convergence keywords.

Variable	Value
Sloppy (`!SloppySCF/Convergence Sloppy`)	
TolE	3e-5
TolMAXP	1e-4
TolRMSP	1e-5
TolErr	1e-4
Thresh	1e-9
TCut	1e-10
BFCut	1e-10
TolG	3e-4
TolX	3e-4
Z_Tol	5e-3
Loose (`!LooseSCF/Convergence Loose`)	
TolE	1e-5
TolMAXP	1e-3
TolRMSP	1e-4
TolErr	5e-4
Thresh	1e-9
TCut	1e-10
BFCut	1e-10
TolG	1e-4
TolX	1e-4
Z_Tol	3e-3
Medium (`!MediumSCF/Convergence Medium`)	
ConvCheckMode	2
TolE	1e-6
TolMAXP	1e-5
TolRMSP	1e-6
TolErr	1e-5
Thresh	1e-10
TCut	1e-11

continues on next page

Table 2.9 – continued from previous page

Variable	Value
BFCut	1e-10
TolG	5e-5
TolX	5e-5
Z_Tol	1e-3
Strong (!StrongSCF/Convergence Strong)	
ConvCheckMode	2
TolE	3e-7
TolMAXP	3e-6
TolRMSP	1e-7
TolErr	3e-6
Thresh	1e-10
TCut	3e-11
BFCut	3e-11
TolG	2e-5
TolX	2e-5
Z_Tol	7e-4
Tight (!TightSCF/Convergence Tight)	
ConvCheckMode	2
TolE	1e-8
TolMAXP	1e-7
TolRMSP	5e-9
TolErr	5e-7
Thresh	2.5e-11
TCut	2.5e-12
BFCut	1e-11
TolG	1e-5
TolX	1e-5
Z_Tol	1e-4
VeryTight (!VeryTightSCF/Convergence VeryTight)	
ConvCheckMode	2
TolE	1e-9
TolMAXP	1e-8
TolRMSP	1e-9
TolErr	1e-8
Thresh	1e-12
TCut	1e-14
BFCut	1e-12
TolG	2e-6
TolX	2e-6
Z_Tol	3e-5
Extreme (!ExtremeSCF/Convergence Extreme)	
ConvCheckMode	0
TolE	1e-14
TolMAXP	1e-14
TolRMSP	1e-14
TolErr	1e-14
Thresh	3e-16
TCut	3e-16
TolG	1e-09
TolX	1e-09
Z_Tol	3e-06
BFCut	3e-16

Table 2.10: Additional threshold choices set by the simple input keys (`!StrongSCF`, …etc.)

Block	Variable	Value
Sloppy (`!SloppySCF`)		
`casscf`	`GTol`	5.0e-3
	`ETol`	1.0e-6
`mdci`	`STol`	1.0e-4
`mrci`	`ETol`	1.0e-5
	`RTol`	1.0e-5
`cis`	`ETol`	1.0e-5
	`RTol`	1.0e-5
Loose (`!LooseSCF`)		
`casscf`	`GTol`	5.0e-3
	`ETol`	1.0e-6
`mdci`	`STol`	1.0e-4
`autoci`	`STol`	1.0e-4
`mrci`	`ETol`	1.0e-5
	`RTol`	1.0e-5
`cis`	`ETol`	1.0e-5
	`RTol`	1.0e-5
Normal (`!NormalSCF`)		
`casscf`	`GTol`	1.0e-3
	`ETol`	1.0e-7
`mdci`	`STol`	2.5e-5
`autoci`	`STol`	2.5e-5
`mrci`	`ETol`	1.0e-6
	`RTol`	1.0e-6
`cis`	`ETol`	1.0e-6
	`RTol`	1.0e-6
Strong (`!StrongSCF`)		
`casscf`	`GTol`	5.00e-4
	`ETol`	6.66e-8
`mdci`	`STol`	7.50e-6
`autoci`	`STol`	7.50e-6
`mrci`	`ETol`	6.66e-7
	`RTol`	6.66e-7
`cis`	`ETol`	6.66e-7
	`RTol`	6.66e-7
Tight (`!TightSCF`)		
`casscf`	`GTol`	2.5e-4
	`ETol`	2.5e-8
`mdci`	`STol`	1.0e-5
`autoci`	`STol`	1.0e-5
`mrci`	`ETol`	2.5e-7
	`RTol`	2.5e-7
`cis`	`ETol`	2.5e-7
	`RTol`	2.5e-7
VeryTight (`!VeryTightSCF`)		
`casscf`	`GTol`	1.0e-5
	`ETol`	1.0e-8
`mdci`	`STol`	1.0e-6
`autoci`	`STol`	1.0e-6
	`D3Thresh`	1.0e-14
	`D4Thresh`	1.0e-14
	`D5Thresh`	1.0e-14
`mrci`	`ETol`	1.0e-7
	`RTol`	1.0e-7

continues on next page

Table 2.10 – continued from previous page

Block	Variable	Value
cis	ETol	1.0e-7
	RTol	1.0e-7
Extreme (!ExtremeSCF)		
casscf	GTol	1.0e-9
	ETol	1.0e-12
mdci	STol	1.0e-9
	TCutInt	0.0
autoci	STol	1.0e-9
	D3Thresh	1.0e-14
	D4Thresh	1.0e-14
	D5Thresh	1.0e-14
mrci	ETol	1.0e-12
	RTol	1.0e-12
cis	ETol	1.0e-12
	RTol	1.0e-12

There is an additional set of simple keywords, !SCFCONV<n> with n = 6–12 (e.g. !SCFCONV8), which sets TolE = 10^{-n} and *most* of the other tolerances listed above to appropriate values. However, some, like Z_Tol, TolRMSP, and TolMAXP are not adjusted! Hence, it is recommended to use the keywords given above instead.

If ConvCheckMode=0, *all* convergence criteria have to be satisfied for the program to accept the calculation as converged, which is a quite rigorous criterion. In this mode, the program also has mechanisms to decide that a calculation is converged even if one convergence criterion is not fulfilled but the others are overachieved. ConvCheckMode=1 means that one criterion is enough. This is quite dangerous, so ensure that none of the criteria are too weak, otherwise the result will be unreliable. The default ConvCheckMode=2 is a check of medium rigor — the program checks for the change in total energy and for the change in the one-electron energy. If the ratio of total energy and one-electron energy is constant, the self-consistent field does not fluctuate anymore and the calculation can be considered converged. If you have small eigenvalues of the overlap matrix, the density may not be converged to the number of significant figures requested by TolMaxP and TolRMSP.

ConvForced is a flag to prevent time consuming calculations on non-converged wave functions. It will default to ConvForced=1 for Post-HF methods, Excited States runs and Broken Symmetry calculations. You can overwrite this default behavior by setting ConvForced=0.

Irrespective of the ConvForced value that has been chosen, properties or numerical calculations (NumGrad, NumFreq) will not be performed on non-converged wavefunctions!

2.6.2 Dynamic and Static Damping

Damping is the oldest and simplest convergence aid. It was already invented by Douglas Hartree when he did his famous atomic calculations. Damping consists of mixing the old density with the new density as:

$$P_{\text{new, damped}} = (1 - \alpha)\, P_{\text{new}} + \alpha P_{\text{old}} \tag{2.1}$$

where α is the damping factor, which must have a value of less than 1. Thus the permissible range (not checked by the program) is 0 …0.999999. For α values larger than 1, the calculation cannot proceed since no new density is admixed. Damping is important in the early stages of a calculation where P_{old} and P_{new} are very different from each other and the energy is strongly fluctuating. Many schemes have been suggested that vary the damping factor dynamically to give strong damping in the beginning and no damping in the end of an SCF. The scheme implemented in ORCA is that by Hehenberger and Zerner [1] and is invoked with CNVZerner=true. Static damping is invoked with CNVDamp=true. These convergers are mutually exclusive. They can be used in the beginning of a calculation when it is not within the convergence radius of DIIS or SOSCF. Damping works reasonably well, but most other convergers in ORCA are more powerful.

If damping used in conjunction with DIIS or SOSCF, the value of DampErr is important: once the DIIS error falls below DampErr, the damping is turned off. In case SOSCF is used, DampErr refers to the orbital gradient value

at which the damping is turned off. The default value is 0.1 Eh. In difficult cases, however, it is a good idea to choose `DampErr` much smaller, e.g. 0.001. This is — to some extent — chosen automatically together with the keyword `! SlowConv`.

```
%scf
  # control of the Damping procedure
  CNVDamp   true    # default: true
  CNVZerner false   # default: false
  DampFac   0.98    # default: 0.7
  DampErr   0.05    # default: 0.1
  DampMin   0.1     # default: 0.0
  DampMax   0.99    # default: 0.98
  # more convenient:
  Damp fac 0.98 ErrOff 0.05 Min 0.1 Max 0.99 end
end
```

2.6.3 Level Shifting

Level shifting is a frequently used technique. The basic idea is to shift the energies of the virtual orbitals such that after diagonalization the occupied and virtual orbitals mix less strongly and the calculation converges more smoothly towards the desired state. Also, level shifting should prevent flipping of electronic states in near-degenerate cases. In a special context it has been shown by Saunders and Hillier [2, 3] to be equivalent to damping.

Similar to `DampErr` described in the previous section, `ShiftErr` refers to the DIIS error at which the level shifting is turned off.

```
%scf
  # control of the level shift procedure
  CNVShift   true  # default: true
  LShift     0.1   # default: 0.25, energy unit is Eh.
  ShiftErr   0.1   # default: 0.0
  # more convenient:
  Shift Shift 0.1 ErrOff 0.1 end
end
```

2.6.4 Direct Inversion in Iterative Subspace (DIIS)

The direct inversion in iterative subspace (DIIS) is a technique that was invented by Pulay [4, 5]. It has become the *de facto* standard in most modern electronic structure programs, because DIIS is robust, efficient and easy to implement. Basically DIIS uses a criterion to judge how far a given trial density is from self-consistency. The commutator of the Fock and density matrices [**F**,**P**] is a convenient measure for this error. With this information, an extrapolated Fock matrix from the present and previous Fock matrices is constructed, which should be much closer to self-consistency. In practice this is usually true, and better than linear convergence has been observed with DIIS. In some rare (open-shell) cases however, DIIS convergence is slow or absent after some initial progress. As self-consistency is approached, the set of linear equations to be solved for DIIS approaches linear dependency and it is useful to bias DIIS in favor of the SCF cycle that had the lowest energy using the factor `DIISBfac`. This is achieved by multiplying all diagonal elements of the DIIS matrix with this factor unless it is the Fock matrix/density which leads to the lowest energy. The default value for `DIISBfac` is 1.05.

The value of `DIISMaxEq` is the maximum number of old Fock matrices to remember. Values of 5-7 have been recommended, while other users store 10-15 Fock matrices. Should the standard DIIS not achieve convergence, some experimentation with this parameter can be worthwhile. In cases where DIIS causes problems in the beginning of the SCF, it may have to be invoked at a later stage. The start of the DIIS procedure is controlled by `DIISStart`. It has a default value of 0.2 Eh, which usually starts DIIS after 0-3 cycles. A different way of controlling the DIIS start is adjusting the value `DIISMaxIt`, which sets the maximum number of cycles after which DIIS will be started irrespective of the error value.

```
%scf
  # control of the DIIS procedure
  CNVDIIS     true  # default: true
  DIISStart   0.1   # default: 0.2
  DIISMaxIt   5     # default: 12
  DIISMaxEq   7     # default: 5
  DIISBFac    1.2   # default: 1.05
  DIISMaxC    15.0  # default: 10.0
  # more convenient:
  DIIS Start 0.1  MaxIt 5  MaxEq 7  BFac 1.2  MaxC 15.0  end
end
```

Note that for troublesome or lacking SCF convergence the TRAH algorithm should be used (see Sec. *Trust-Region Augmented Hessian (TRAH) SCF*). If not turned off explicitly, TRAH is switched on automatically whenever convergence problems are present by means of the AutoTRAH feature (see Sec. *Trust-Region Augmented Hessian (TRAH) SCF*).

2.6.5 Kolmar's DIIS (KDIIS)

An alternative algorithm that makes use of the DIIS concept is called KDIIS (Kolmar's DIIS[6, 7]) in ORCA. The KDIIS algorithm is designed to bring the orbital gradient of any energy expression to zero using a combination of DIIS extrapolation and first order perturbation theory. Thus, the method is diagonalization-free. In our hands it is superior to the standard DIIS algorithm in many cases, but not always. The algorithm is invoked with the keyword ! KDIIS and is available for RHF, UHF and CASSCF.

2.6.6 Approximate Second Order SCF (SOSCF)

SOSCF is an approximately quadratically convergent variant of the SCF procedure [8, 9]. The theory is relatively involved and will not be described here. In short – SOSCF computes an initial guess to the inverse orbital Hessian and then uses the BFGS formula in a recursive way to update orbital rotation angles. As information from a few iterations accumulates, the guess to the inverse orbital Hessian becomes better and better and the calculation reaches a regime where it converges superlinearly. As implemented, the procedure converges as well or slightly better than DIIS and takes a somewhat less time. However, it is also *a lot* less robust, so that DIIS is the method of choice for many problems (see also the description of the full second-order trust-region augmented Hessian (TRAH) procedure in the next section). On the other hand, SOSCF is useful when DIIS gets stuck at some error around ~ 0.001 or 0.0001. Such cases were the primary motive for the implementation of SOSCF into ORCA.

The drawback of SOSCF is the following: in the beginning of the SCF, the orbital gradient (the derivative of the total energy with respect to rotations that describe the mixing of occupied and virtual MOs) is large, so that one is far from the quadratic regime. In such cases, the procedure is not successful and may even wildly diverge. Therefore it is recommended to only invoke the SOSCF procedure in the very end of the SCF where DIIS may lead to "trailing" convergence. SOSCF is controlled by the variables SOSCFStart and SOSCFMaxIt. SOSCFStart is a threshold for the orbital gradient. When the orbital gradient, or equivalently the DIIS Error, fall below SOSCFStart, the SOSCF procedure is initiated. SOSCFMaxIt is the latest iteration to start the SOSCF even if the orbital gradient is still above SOSCFStart.

```
%scf
  # control of the SOSCF procedure
  CNVSOSCF    true  # default: false
  SOSCFStart  0.1   # default: 0.01
  SOSCFMaxIt  5     # default: 1000
  # more convenient:
  SOSCF Start 0.1  MaxIt 5  end
end
```

For many calculations on transition metal complexes, it is a good idea to be conservative in the startup criterion for SOSCF, it may diverge otherwise. A choice of 0.01 or lower is recommended.

2.6.7 Trust-Region Augmented Hessian (TRAH) SCF

The **Trust-Region Augmented Hessian (TRAH)** method[10, 11, 12, 13] is a robust SCF convergence algorithm that ensures reliable convergence to the electronic ground state by using information from the electronic Hessian. It is especially useful for difficult SCF cases where standard DIIS/SOSCF approaches fail, such as open-shell systems or molecules with complex electronic structures.

When to Use TRAH

The TRAH-SCF procedure can be:

- **Activated manually** using the `! TRAH` keyword.
- **Triggered automatically** (`AutoTRAH`) when standard SCF convergence is too slow or diverges.

```
%scf
  AutoTRAH true
end
```

AutoTRAH

The automatic mode (`AutoTRAH`) monitors the convergence behavior by performing a log-scale linear interpolation of the electronic gradient norm over several SCF iterations. After a minimum number of SCF iterations (`AutoTRAHIter`, default: 20), the algorithm analyzes the last `AutoTRAHNInter` (default: 10) values of the gradient norm. A linear fit is performed in $\log_{10}(||\mathbf{g}||)$ space, and the slope s of this interpolation is used to assess convergence. If the slope indicates a convergence rate slower than 10^{-s}, with $s <$ `AutoTRAHTol` (default: 1.125), standard SCF is stopped and TRAH-SCF is initiated using the current set of orbitals.

This mechanism allows ORCA to switch to the more robust TRAH-SCF method only when necessary, improving efficiency for difficult SCF cases while avoiding unnecessary overhead for easy ones.

To disable automatic activation:

```
! NOTRAH
```

or

```
%scf
  AutoTRAH false
end
```

Theoretical Basis

TRAH builds a quadratic model of the SCF energy as a function of orbital rotations $\mathbf{x}$:

$$E(\mathbf{x}) = E_0 + \mathbf{g}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x}, \quad ||\mathbf{x}|| \leq h$$

Minimizing $E(\mathbf{x})$ under the trust region constraint leads to level-shifted Newton equations:

$$(\mathbf{H} - \mu\mathbf{I})\mathbf{x} = -\mathbf{g}$$

Instead of solving these directly, the algorithm solves the eigenvalue problem of the scaled augmented Hessian matrix:

$$\begin{pmatrix} 0 & \alpha\mathbf{g}^T \\ \alpha\mathbf{g} & \mathbf{H} \end{pmatrix} \begin{pmatrix} 1 \\ \tilde{\mathbf{x}} \end{pmatrix} = \mu \begin{pmatrix} 1 \\ \tilde{\mathbf{x}} \end{pmatrix}$$

The Davidson algorithm is used for iterative diagonalization until the residual norm of $\tilde{\mathbf{x}}$ is below `TolFacMicro` $\times$ `||g||`. The scaling factor α is adjusted via bisection in the interval $[\alpha_0, \alpha_1]$ to ensure the trust region constraint is satisfied.

Performance and Implementation Details

- **Parallelization**: TRAH is MPI-parallel and supports large molecules using AO-based Fock matrices.
- **Acceleration**: TRAH supports `RIJ`, `RIJONX`, `RIJK`, and `RIJCOSX`, and is compatible with solvation models like `C-PCM` and `SMD`.
- **Available Methods**: Implemented for `RHF`, `RKS`, `UHF`, and `UKS`. `ROHF` and `ROKS` are not yet supported.
- **Efficient Hessian Operations**: Similar to CP-SCF and TD-DFT, gradient and Hessian operations use AO-basis sigma vectors for efficiency.

Grid parameters to speed up these steps:

```
%method
  Z_GridXC   1      // Lebedev Grid
  Z_IntAccXC 3.467  // eps parameter of radial grid
  Z_GridX    1      // Lebedev Grid
  Z_IntAccX  3.067  // eps parameter of radial grid
end
```

Input Parameters

```
%scf

 # AutoTRAH parameter
 AutoTRAH       true      # enables by default
 AutoTRAHTol    1.125     # slop of gradient convergence
 AutoTRAHIter   20        # first iteration for which TRAH could be called
 AutoTRAHNInter 10        # number of iteration for monitoring convergence progress

 # TRAH parameter
 trah
   MaxRed          24     # maximum number of Davidson micro iterations
   NStart          2      # number of start vectors for Davidson (at least 2)
   TolFacMicro     0.1    # Scaling factor for Davidson convergence
                          #  threshold =  TolFacMicro * || G ||
   MinTolMicro     1.e-2  # minimum accuracy of micro iterations
   QuadRegionStart 1.e-4  # start Newton-Raphson if || G || < QuadRegionStart
   tradius         0.4    # initial trust radius
   AlphaMin        0.1    # lower bound of gradient scaling parameter
   AlphaMax        1000.  # upper bound of gradient scaling parameter
   Randomize       true   # add white noise to Davidson start vectors
   PseudoRand      false  # use pseudo random numbers for comparibility
   MaxNoise        1.e-2  # maximum random number magnitude
   OrbUpdate       Taylor # orbital update algorithm (TAYLOR or CAYLEY)
   InactiveMOs     Canonical # CANONICAL or NOTSET
   Precond         Diag   # DIAG, FULL, or NONE
   PreconMaxRed    250    # maximum dimension of FULL Hessian for preconditioning
 end

 TolG 1.e-6  # SCF convergence threshold for gradient norm

end
```

- `Precond = FULL` uses the exact Hessian for a subset of important MO pairs, improving convergence for small RHF/UHF systems but not necessarily for large molecules or DFT.
- For `FULL`, ensure RI and auxiliary basis (`/C`) are provided.

Practical Notes

- **Quadratic Convergence**: Near the solution, TRAH achieves quadratic convergence.
- **Guaranteed Convergence**: In exact arithmetic and absent numerical noise, TRAH will always converge. If not, increase `MAXITER` or tighten numerical thresholds (e.g., grid accuracy).
- **LibXC Compatibility**: Some functionals lack ORCA-native kernels (e.g., `PWPB95`). Use `LIBXC(...)` for TRAH support.

Example Case

In a high-spin Rh_{12}^{+} cluster, DIIS fails to converge while TRAH steadily reduces the gradient norm below 10^{-6}:

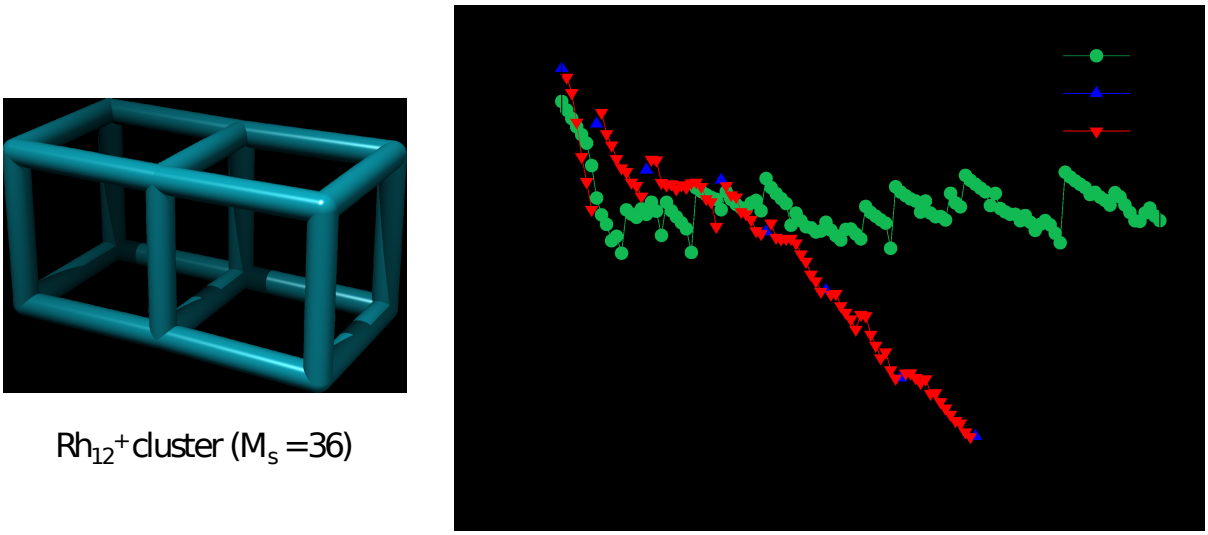

Fig. 2.1: TRAH-SCF gradient norm of a PBE/def2-TZVP calculation for a Rh_{12}^{+} cluster in high-spin configuration ($M_s = 36$). Structure from Ref. [14].

2.6.8 Finite Temperature SCF

A finite temperature can be used to apply a Fermi-like occupation number smearing over the orbitals of the system, which may sometimes help to get convergence of the SCF equations in near hopeless cases. Through the smearing, the electrons are distributed according to Fermi statistics among the available orbitals. The "chemical potential" is found through the condition that the total number of electrons remains correct. Gradients can be computed in the presence of occupation number smearing.

```
%scf SmearTemp 5000  # ``temperature'' in Kelvin
 end
```

Note

- Finite temperature SCF (fractional occupation numbers or FOD analysis, see sections *Fractional Occupation Numbers* and *Fractional Occupation Number Weighted Density (FOD)*, respectively) *cannot* be used together with `SOSCF` methods.

Fractional Occupation Numbers

Only a very basic implementation of fractional occupation numbers is presently provided. It is meant to deal with orbitally degenerate states in the UHF/UKS method. Mainly it was implemented to avoid symmetry breaking in DFT calculations on orbitally degenerate molecules and atoms. The program checks the orbital energies of the initial guess orbitals, finds degenerate sets and averages the occupation numbers among them. Currently the criterion for degenerate orbitals is 10^{-3} Eh. The fractional occupation number option is invoked by:

```
%scf
  FracOcc true
end
```

Clearly, the power of fractional occupation numbers goes far beyond what is presently implemented in the program and future releases will likely make more use of them. The program prints a warning whenever it uses fractional occupation numbers. The fractionally occupied orbitals should be checked to ensure they are actually the intended ones.

Note

- Using `GuessMode = CMatrix` will cause problems because there are no orbital energies for the initial guess orbitals. The program will then average over *all* orbitals — which makes no sense at all.

2.6.9 Tips and Tricks: Converging SCF Calculations

Despite all efforts you may still find molecules where SCF convergence is poor. These are almost invariably related to open-shell situations and the answer is almost always to provide "better" starting orbitals. Here is my standard strategy to deal with this (assuming a DFT calculation):

- Perform a small basis set (`SV`) calculation in using the `LSD` or `BP` functional and RI approximation with a cheap auxiliary basis set. Set `Convergence=Loose` and `MaxIter=200` or so. The key point is to use a large damping factor and damp until the DIIS comes into a domain of convergence. This is accomplished by `SlowConv` or even `VerySlowConv`. If you have an even more pathological case you may need to set `DampFac` larger and `DampErr` smaller than chosen by these defaults. This calculation is quite crude and may take many cycles to converge. It will however be rather quick in terms of wall clock time. If the DIIS gets stuck at some error 0.001 or so the `SOSCF` (or even better `TRAH`) could be put in operation from this point on.
- Use the orbitals of this calculation and `GuessMode=CMatrix` to start a calculation with the target basis set. In DFT we normally use a pure GGA functional (e.g. BP86). This calculation normally converges relatively smoothly.
- Use the target functional, grid etc. to get the final calculation converged. In many cases this should converge fairly well now.

Here are a few other things that can be tried:

- Try to start from the orbitals of a related closed-shell species. In general closed-shell MO calculations tend to converge better. You then hope to reach the convergence radius of another converger for the open-shell case.
- Try to start from the orbitals of a more positive cation. Cation calculations tend to converge better.
- Try to start from a calculation with a smaller basis set. Smaller basis sets converge better. Then you have the choice of `GuessMode=CMatrix` or `GuessMode=FMatrix` which will affect the convergence behavior.
- Use large level shifts. This increases the number of iterations but stabilizes the converger. (`shift shift 0.5 erroff 0 end`)
- If you are doing DFT calculations try to start from a Hartree-Fock solution for your molecule. HF calculations tend to converge somewhat better because they have a larger HOMO-LUMO gap (there are of course exceptions).

- Carefully look at the starting orbitals (`Print[P_GuessOrb]=1`) and see if they make sense for your molecule. Perhaps you have to reorder them (using `Rotate`) to obtain smooth convergence.
- Most of the time the convergence problems come from "unreasonable" structures. Did you make sure that your coordinates are in the correct units (Angström or Bohrs?) and have been correctly recognized as such by the program?
- If you have trouble with UHF calculations try ROHF (especially SAHF or CAHF) first and then go to the UHF calculation.
- Fool around with `Guess=Hueckel`, `PAtom` or even `HCore`.
- It may sometimes be better to converge to an undesired state and then take the orbitals of this state, reorder them (using `Rotate`) and try to converge to the desired state.
- Similarly, bad orbitals may be manipulated using the SCF stability analysis (section *SCF Stability Analysis*) to provide a new guess.
- Try to start the calculation with a large damping factor (`DampFac=0.90`; or even larger) and specify a relatively small DIIS error to turn damping off (say `DampErr=0.02`;). This will increase the number of cycles but may guide you into a regime were the calculation actually converges.
- The advices above mostly apply to Hartree-Fock and DFT. For CASSCF, the available options and how they can aid to overcome convergence problems are described in the CASSCF manual section. In many cases modifying the initial guess or adding a level shift will help. Do not hesitate to use large level-shifts (e.g 2.0 or even 3.0). The manual is accompanied by CASSCF tutorial that goes through many details of the process including practical advices on convergence. The choice of initial guess is crucial. Some guesses work better for organic molecules while others excel for transition-metal complexes. The tutorial therefore discusses various initial guess options available in ORCA.
- If you managed to converge your system in another quantum chemistry program, and do not want to waste time re-converging it in ORCA, then you may convert the converged orbitals in the other program to the ORCA format using third-party tools like MOKIT (section *orca_2mkl: Old Molekel as well as Molden inputs*). However, keep in mind that converging a wavefunction in program A and reconverging it in program B comes with an inevitable overhead: program B usually has to perform a few extra SCF iterations even if the wavefunction is fully converged in program A, because (among many other reasons) the two programs in general use different integration grids. Therefore, use this method only when you expect a significant time saving.
- If nothing else helps, stop, grab a friend and go to the next pub (you can also send me an unfriendly e-mail but this will likely not make your calculation converge any quicker; ◡̈).

2.6.10 Local-SCF Method

The Local SCF (LSCF) method developed by X. Assfeld and J.-L. Rivail ([15]) allows optimising single determinant wave functions within the constraint of keeping some selected orbitals frozen (i.e. unchanged), whilst all orbitals fulfill orthogonality requirements. The LSCF method may be applied to unrestricted Hartree-Fock or DFT calculations.

The use of the LSCF method consists of selecting the set of orbitals to freeze during the SCF process according to the guess orbitals of the calculation. The selection is done by employing the `LSCFalpha` and/or `LSCFbeta` keywords in the SCF bloc for alpha and beta orbitals, respectively, followed by the orbital selection. **Intervals** are used to define the selection and one may go up to five intervals (5×2 numbers separate with commas). In the example below, `LSCFalpha 0,1,4,4,10,12` corresponds to freezing the alpha orbitals from 0 to 1 (**0,1**,4,4,10,12), orbital 4 (0,1,**4,4**,10,12), and from 10 to 12 (0,1,4,4,**10,12**). Accordingly, `LSCFbeta 0,3,5,5` corresponds to freezing beta orbitals 0 to 3 and orbital 5.

```
%scf
LSCFalpha 0,1,4,4,10,12
LSCFbeta  0,3,5,5
end
```

The energy of frozen orbitals is meaningless since they are not eigenfunctions of the Fockian. During the LSCF process, the energy of the occupied frozen orbitals is artificially defined as the lowest occupied orbital energy of the guess orbitals minus 1000 Hartree, while for the virtual frozen orbitals, the energy is defined as the highest virtual

orbital energy plus 1000 Hartree. Hence, all occupied frozen orbitals are sorted at the beginning of the orbital distribution and the virtual frozen orbitals at the end.

2.6.11 Keywords

Simple input keywords related to the SCF procedure are summarized in Section 2.6.11.

Table 2.11: Simple input keywords for the SCF procedure

Keyword	Description
Convergence acceleration	
`DIIS`	Enable *DIIS*
`NoDIIS`	Disable *DIIS*
`KDIIS`	Enable *Kollmar's DIIS*
`TRAH`	Enable *TRAH*
`NoTRAH`	Disable *TRAH*
`SOSCF`	Enable *SOSCF*
`NoSOSCF`	Disable *SOSCF*
`Damp`	Enable *damping*
`NoDamp`	Disable *damping*
`LShift`	Enable *level shifting*
`NoLShift`	Disable *level shifting*
Solver parameter selection (see Section 2.6.2)	
`EasyConv`	Assumes no convergence problems.
`NormalConv`	Default solver parameters
`SlowConv`	Selects appropriate SCF solver parameters for difficult cases. Most transition metal complexes fall into this category.
`VerySlowConv`	Selects appropriate SCF solver parameters for very difficult cases.
`SCFCONV<n>`	Adjusts `TolE` to 10e-n and most of the other tolerances according to Table 2.9 and Table 2.10 (not recommended!)
Fractional occupation numbers	
`FracOcc`	Enable fractional occupations (also compute *FOD*)
`Smear`	Enable occupation number smearing with `SmearTemp=5000` (also compute *FOD*)
`NoSmear`	Disable occupation number smearing
`FOD`	Perform *FOD analysis* with default settings (TPSS/def2-TZVP, TightSCF, SmearTemp = 5000 K)

2.7 Basis Sets

ORCA provides a large number of natively implemented *orbital* and *auxiliary basis sets* alongside various *effective core potentials (ECPs)* that can be combined with them. For use with *scalar-relativistic methods like ZORA, DKH, or X2C*, specialized *relativistic basis sets* are available as well. Furthermore, other basis sets can be *read from external files*. Most built-in basis sets and ECPs were obtained from the Basis Set Exchange[16] or its predecessor, the EMSL library, and the input format in ORCA is closely related to the "GAMESS-US" format.

2.7.1 Basic Usage

The easiest way to use orbital and auxiliary basis sets in ORCA is via the simple input keywords. All available orbital basis set keywords can be found in Section 2.7.2 and all auxiliary basis set options in Section 2.7.4. For example, the Karlsruhe def2-TZVP basis set can be invoked via the `def2-TZVP` keyword.

```
! def2-TZVP
```

Note

Some basis sets like the *Karlsruhe def2* employ *ECPs* by default for heavy elements, in this case the `def2-ECP`. The explicit control of the ECPs is described in Section 2.7.5.

Auxiliary basis sets needed for *resolution-of-the-identity (RI)* can be defined in the same way. In this example, Weigend's universal `def2/J` auxiliary basis is used.

```
! def2-TZVP def2/J
```

Note

Note that for many calculations, RI is activated by default (e.g. *RIJCOSX* for *hybrid DFT*). Accordingly, the `def2/J` auxiliary basis is automatically invoked by default as well if not otherwise specified. In scalar relativistic calculations, the default is `SARC/J` instead.

Note that there are three separate slots for auxiliary basis sets for RI:

- `AuxJ` is the Coulomb-fitting basis for the *RI-J*, *RIJDX/RIJONX*, and *RIJCOSX* approximations.
- `AuxJK` is the Coulomb- and exchange-fitting basis used for *RI-JK*.
- `AuxC` is used for RI-based integral generation steps in post-SCF dynamical electron correlation methods, such as *RI-MP2*, *DLPNO-MP2*, and *DLPNO-CC*.

Finally, *F12 methods* require a complementary auxiliary basis set (`CABS`), in addition to the *specialized orbital basis* (and possibly `AuxC`), for example:

```
! F12-RI-MP2 cc-pVDZ-F12 cc-pVDZ-F12-CABS cc-pVTZ-F12-MP2Fit
```

Specifying an auxiliary basis with the simple input keyword, assigns it to the corresponding slot. However, each basis slot, as well as the ECP, can be assigned explicitly in the `%basis` block. For example, a "/JK" basis may be assigned to `AuxJ` in this way.

```
%basis
  Basis "def2-TZVP"          # The orbital expansion basis set
  ECP   "def2-ECP"           # Effective core potential
  AuxJ  "def2/J"             # RI-J auxiliary basis set
  AuxJK "def2/JK"            # RI-JK auxiliary basis set
  AuxC  "def2-TZVP/C"        # Auxiliary basis set for correlated
                             # calculations, e.g. RI-MP2
  CABS  "cc-pVDZ-F12-OptRI" # complementary auxiliary basis set
                             # for F12 calculations
end
```

If required, all basis sets can be decontracted via simple input (e.g. `! DECONTRACT`) or the `%basis` block with the respective keywords below. Note that if your basis set arises from general contraction, it will contain duplicate primitives in several contractions. These will be removed such that only unique primitives remain and there is no problem with redundancy.

```
%basis
  DecontractBas   false  # if chosen "true" the program will
                         # decontract the orbital basis set
  DecontractAuxJ  false  # if "true" - decontract the AuxJ basis set
  DecontractAuxJK false  # if "true" - decontract the AuxJK basis set
  DecontractAuxC  false  # if "true" - decontract the AuxC basis set
  DecontractCABS  true   # if "false" - do not decontract the CABS
  Decontract      false  # if "true" - decontract all basis sets
end
```

Tip

- Generally, basis sets can be *assigned to specific elements*, to individual *atoms*, or even to *structural fragments*.
- Any built-in basis can be exported via the *`orca_exportbasis` utility*.
- References for the built-in basis sets are usually printed at the start of the ORCA output, as well as by `orca_exportbasis`.

Warning

ORCA uses pure d and f functions (5D and 7F instead of Cartesian 6D and 10F) for all basis sets. This needs to be taken into account when results are compared with other programs, especially for Pople-style basis sets that were optimized with Cartesian (6D) functions.

2.7.2 Orbital Basis Sets

In the following, we will give an outline of natively implemented basis sets from various families, like the *Karlsruhe def2* and *correlation consistent* basis sets, and list which elements are covered by each. If an *ECP* is necessary for heavy elements, it is documented in the respective table entries and invoked by default if the basis is selected in the simple input or via the `basis` keyword in the `%basis` block.

Pople Basis Sets

Various basis sets of the Pople basis set family are available in ORCA. A list of all available Pople-style basis sets is given in Table 2.12.

Naming Convention of Pople Basis Sets

- * or (d) adds one set of first polarization functions on all atoms except H
- ** or (d,p) adds one set of first polarization functions on all atoms
- Further combinations: (2d), (2df), (2d,p), (2d,2p), (2df,2p), (2df,2pd)
- + before "G" includes diffuse functions on all atoms except H (e.g. 6-31+G)
- ++ before "G" includes diffuse functions on all atoms. Works only when H polarization is already included, e.g. 6-31++G(d,p)

Table 2.12: Available Pople-style basis sets.

Basis Set	Elem.	ECP	Comment
`STO-3G`	H–I	–	Minimal basis set
`3-21G`	H–Cs	–	
`3-21GSP`	H–Ar	–	
`4-22GSP`	H–Ar	–	
`6-31G`	H–Zn	–	
`6-31G*`	H–Kr	–	
`m6-31G`	Sc–Cu	–	Modified 6-31G for 3d transition metals (Sc–Cu)
`m6-31G*`	Sc–Cu	–	
`6-31G**`	H–Zn	–	
`6-31G(d)`	H–Zn	–	
`6-31G(d,p)`	H–Zn	–	
`6-31G(2d)`	H–Zn	–	
`6-31G(2d,p)`	H–Zn	–	
`6-31G(2d,2p)`	H–Zn	–	
`6-31G(2df)`	H–Zn	–	
`6-31G(2df,2p)`	H–Zn	–	
`6-31G(2df,2pd)`	H–Zn	–	
`6-31+G*`	H–Kr	–	
`6-31+G**`	H–Zn	–	
`6-31+G(d)`	H–Zn	–	
`6-31+G(d,p)`	H–Zn	–	
`6-31+G(2d)`	H–Zn	–	
`6-31+G(2d,p)`	H–Zn	–	
`6-31+G(2d,2p)`	H–Zn	–	
`6-31+G(2df)`	H–Zn	–	
`6-31+G(2df,2p)`	H–Zn	–	
`6-31+G(2df,2pd)`	H–Zn	–	

continues on next page