Table 2.38 – continued from previous page

ECP keyword	Core size[1]	Elements	Valence basis sets
	46	In–Lu	
	60	Hf–Hg	
	78	Tl–Rn	
CRENBL-ECP	2	Li–Mg	CRENBL
	10	Al–Zn	
	28	Rb–Cd	
	36	In–Xe	
	46	Cs–La	
	54	Ce–Lu	
	60	Hf-Hg	
	68	Tl-Rn	
	78	Fr-Ts	
	92	Og	
Legacy definitions			
def2-SD	28,MWB	Rb–Cd	
	28,MDF[3]	In–Xe	
	46,MWB	Cs–La	
	60,MWB	Hf–Pt	
	60,MDF[4]	Au–Rn	
def-SD	28,MWB	Rb–Cd	
	46,MWB	In–La	
	28,MWB	Ce–Lu	
	60,MWB	Hf–Pt	
	60,MDF[4]	Au, Hg, Rn	
	78,MWB	Tl–At	
	78,MDF	Fr, Ra	
	60,MWB	Ac–Lr	
SDD	2,SDF	Li, Be	
	2,MWB	B–Ne	
	10,SDF	Na, Mg	
	10,MWB	Al–Ca	
	10,MDF	Sc–Ni	
	10,MWB	Cu-Zn	
	28,MWB	Ga–Sr	
	28,MHF	Y–Cd	
	28,MDF	Ge–Br, Rb–Xe	
	46,MWB	In–Ba	
	28,MWB	La–Lu	
	60,MWB	Hf–Hg	
	78,MWB	Tl–Rn	
	60,MWB	Ac–Lr	
LANL1	10	Na–Ar	
	18	K–Zn	
	28	Ga–Kr	
	36	Rb–Cd	
	46	In–Xe	
	54	Cs–La	
	68	Hf–Tl	
	78	Pb, Bi	
LANL2	10	K–Cu	
	28	Rb–Ag	
	46	Cs–La	
	60	Hf–Au	

Note

Requesting some basis sets automatically assigns the matching ECP (except when using the NewGTO keyword): for example, “def2” basis sets use the def2-ECP. For others, see the respective *basis set table entries*.

The simplest way to assign ECPs is by using the ECP keyword within the simple input line. The ECP keyword itself

[1] Where applicable, reference method and data are given (S: single-valence-electron ion; M: neutral atom; HF: Hartree–Fock; WB: quasi-relativistic; DF: relativistic).

[2] Corresponds to LANL2 and to LANL1 where LANL2 is unavailable.

[3] I: OLD-SD(28,MDF) for compatibility with TURBOMOLE.

[4] Au, Hg: OLD-SD(60,MDF) for compatibility with TURBOMOLE.

assigns *only* the effective core potential, not a valence basis set! As an example for an explicitly named ECP you could use

```
! def2-TZVP def2-SD
```

This would assign the def2-SD ECP according to the definition given in the table above. Without the def2-SD keyword ORCA would default to def2-ECP.

Assignment of ECPs can be done within the `%basis` block using the `ECP` and `NewECP` keywords, as in the following example:

```
%basis
  ECP       "def2-ECP"     # All elements (for which the ECP is defined)
  NewECP Pt "def2-SD" end  # Different ECP for Pt
end
```

A variant of the `NewECP` keyword can be used for individual atoms inside the geometry definition:

```
* xyz ...
  ...
  S   0.0  0.0  0.0  NewECP "SDD" end
  ...
*
```

Note that these keywords also only affect the ECP and not the valence basis set!

In case the basis set for an element/atom has been changed using the `NewGTO` keyword (see sections *Assigning or Adding Basis Functions to an Element* and *Assigning or Adding Basis Functions to Individual Atoms* above) it may be necessary to remove the ECP from that element/atom. This can be done with the `DelECP` keyword in the `%basis` block or coordinates input, respectively:

```
! LANL2DZ                   # Uses HayWadt ECPs by default, starting from Na
%basis
  NewGTO S "def2-TZVP" end  # All-electron up to Kr
  DelECP S                  # Remove HayWadt ECP
end
* xyz ...
...
Cu   0.0  0.0  0.0
  DelECP                    # Remove HayWadt ECP
  NewGTO "def2-QZVPP" end   # All-electron up to Kr
...
*
```

To remove all ECPs loaded by default (e.g. in case no global basis set is chosen) you can use the `!NoECP` simple keyword.

Manual Input of ECP Parameters

To manually specify ECP parameters, the `NewECP` keyword is followed by the element for which an ECP is to be entered, the number of core electrons to be replaced (N_core) and the maximum angular momentum (lmax). The ECP specification is finished by giving the definitions of the individual shells that constitute the angular dependent potentials U_l.

```
%basis
    NewECP <element>
      N_core <number of core electrons>
      lmax <max. angular momentum>
      [shells]
    end
end
```

For each ECP shell, first the angular momentum l has to be given, followed by the number of primitives. The primitives themselves are then specified by giving a running index and the respective tuple of exponent a_{kl}, expansion coefficient d_{kl} and radial power n_{kl}.

```
# ECP shell
      l <number of primitives>
      1    a1l    d1l    n1l
      2    a2l    d2l    n3l
      ...
```

As an example, consider the SD(10,MDF) for Vanadium. The name indicates a Stuttgart–Dresden type ECP that replaces 10 core electrons and is derived from a relativistic calculation for the neutral atom. It consists of 4 shells with angular momentum s, p, d, and f. Note that the f shell has an expansion coefficient of 0.0 and thus will not contribute at all to this effective core potential. This is typical for all SD potentials (but may be different for program packages like TURBOMOLE that do not support arbitrary angular momentum with respect to the ECP and therefore use recontractions of the original parameter sets).

```
%basis
  # ECP SD(10,MDF) for V
  # M. Dolg, U. Wedig, H. Stoll, H. Preuss,
  # J. Chem. Phys. 86, 866 (1987).
  NewECP V
    N_core 10
    lmax f
    s 2
      1      14.4900000000    178.4479710000  2
      2       6.5240000000     19.8313750000  2
    p 2
      1      14.3000000000    109.5297630000  2
      2       6.0210000000     12.5703100000  2
    d 2
      1      17.4800000000    -19.2196570000  2
      2       5.7090000000     -0.6427750000  2
    f 1
      1       1.0000000000      0.0000000000  2
  end
end
```

ECPs and Ghost Atoms

When ghost atoms are defined in the input (see section *Special definitions*), ECPs are **not** added to these atoms by default. If that is somehow needed, please add `GhostECP true` under the `%basis` block.

```
%basis
 GhostECP true
 AllowGhostECP true # synonym
end
```

ECP Embedding

Computations on cluster models sometimes require the presence of embedding potentials in order to account for otherwise neglected repulsive terms at the border [107]. In order to simplify these kind of calculations with ORCA the ECP embedding can be accomplished quite easily:

```
*xyz ...
# atom>  charge x    y    z    optional ECP declaration
Zr>      4.0    0.0  0.0  0.0  NewECP "SDD" end
...
*
```

The declaration of such a coreless ECP center takes place in the coordinates section by appending a bracket ">" to the element symbol. Note that embedding ECPs are treated as point charges in ORCA, so the charge has to be given next. The coordinates of the coreless ECP center have to be specified as usual and may be followed by an optional ECP assignment. In general, calculations that employ an ECP embedding procedure should be single point calculations. However if the need arises to perform a geometry optimization, make sure to set up explicit Cartesian constraints for the coreless ECP centers.

2.7.6 Assigning or Adding Basis Functions to an Element

In order to assign a new basis set to a given element, use:

```
%basis
  NewGTO 8     # New basis for oxygen.
# NewGTO O     # This works as well.
   S   3       # s-shell
    1  910.10034975          0.03280967
    2  137.19711335          0.23422391
    3   30.85279077          0.81490980
   0   2       # also an s-shell
    1     1.72885887          0.27389659
    2     0.39954770          0.79112437
   P   1
    1     8.35065975          1.00000000
  end
end
```

For simplicity and consistency the input format is the same as that used in the *basis set files*. In this format, the first line carries the angular momentum of the shell to be added – either as an integer, or as a label (s, p, d, f, g, h, i, k) – and the number of primitives. Then for each primitive one line follows which has (a) the index of the primitive (1, 2, 3, …) (b) the exponent of the primitive and (c) the contraction coefficient (unnormalized). There also is the possibility to include a `SCALE X` statement after the number of primitives in the first line to indicate that the basis function exponents should be scaled.

Warning

- ORCA always uses spherical harmonic Gaussian functions.
- Angular momentum 7 is labeled as "k" – there are no j-shells in accepted spectroscopic conventions.
- Combined s- and p-shells are sometimes labelled as "L-shells" in other programs. This is not supported in the `NewGTO` format and to avoid confusion, shells with angular momentum 8 can only be specified with a number, and not with the label "l".

In order to add basis functions to the basis of a given element (for example because you do not like the standard polarization functions) use `AddGTO` instead of `NewGTO`. In `NewGTO` or `AddGTO` you can also use the nicknames of internally stored basis sets. An example is:

```
%basis
  NewGTO 8     # new basis for oxygen
   "6-31G"
    D  1
    1  0.4  1.0
  end
end
```

In this example the 6-31G basis is assigned to oxygen and in addition a polarization function with exponent 0.4 is added to the oxygen basis.

Note that the `NewGTO` keyword does not change the ECP for the given element – you must use `NewECP` or `DelECP` (see section *Effective Core Potentials*).

A similar mechanism was established for the auxiliary basis sets in RI calculations:

```
%basis
  NewAuxJGTO 8    # new auxiliary basis for oxygen
    s  1
    1  350  1.0
    ... etc
  end
  AddAuxJGTO 8   # add a shell to the auxiliary basis for
                 # oxygen
    D  1
    1  0.8  1.0
  end
end
```

New basis functions can be specifically assigned to any auxiliary basis sets. The keywords `NewAuxCGTO`, `AddAuxCGTO`, `NewAuxJKGTO`, `AddAuxJKGTO`, `NewCABSGTO`, `AddCABSGTO` are used in the same way. The keywords `NewAuxGTO` and `AddAuxGTO` are the same as `NewAuxJGTO` and `AddAuxJGTO`, that is, they only influence the Coulomb auxiliary basis (`AuxJ`)!

2.7.7 Assigning or Adding Basis Functions to Individual Atoms

Sometimes you may want to not treat all atoms of the same element with the same basis set but to assign a specific basis set to a specific atom in the molecules. This is also possible in ORCA and takes place in the coordinate section (`%coords`, `*xyz`, etc.). The format is the same as described above. An example may help to make things clear:

```
*int 0 1
  C  0 0 0 0.00   0.0   0.00
   AddGTO
     D  1
      1  1.0  1.0
   end
  O  1 0 0 1.13   0.0   0.00
   NewGTO
     "6-311G"
     D  1
      1  1.2  1.0
   end
*
```

In this example an extra d-shell with exponent 1.0 is added to the first carbon atom and the basis for the oxygen atom is changed to `6-311G` with an extra d-function of exponent 1.2 added.

Analogously, `AUX` basis functions can be assigned or added to individual atoms using the keywords `NewAuxJGTO`, `AddAuxJGTO`, `NewAuxCGTO`, `AddAuxCGTO`, `NewAuxJKGTO`, `AddAuxJKGTO`, `NewCABSGTO`, `AddCABSGTO`.

A note on the use of `AutoAux`: if you change the basis set on a given atom and want to generate a fitting basis, you have to specify it again in the coordinates section, even if `AutoAux` is already present in the simple input line or in the `%basis` block. For example:

```
! def2-SVP def2/JK
%basis
  NewAuxJKGTO H
    "AutoAux"
  end
end
*xyz 0 1
  O  0.00 0.00 0.00
  H -0.25 0.93 0.00
  H  0.96 0.00 0.00
```

(continues on next page)

(continued from previous page)

```
   AddGTO
     P 1
      1 1.6 1.0
     D 1
      1 1.0 1.0
   end
   NewAuxJKGTO
     "AutoAux"
   end
*
```

Here the oxygen atom is assigned the def2-SVP basis and the def2/JK fitting basis, the first hydrogen atom is assigned the def2-SVP basis and an automatically generated fitting basis and the second hydrogen atom is assigned the def2-SVP basis with two additional polarization functions and a larger automatically generated fitting basis that accounts for these functions.

Tip

When assigning custom basis sets it is always a good idea to print the basis set information (`%output print[p_basis] 2 end` or simply `!PrintBasis`) and check that everything is correct.

2.7.8 Assigning Basis Sets and ECPs to Fragments

In multi-level or QM/QM calculations it may be convenient to assign different basis sets to different *fragments*. This can be done with the keywords `FragBasis`, `FragAuxJ`, `FragAuxJK`, `FragAuxC`, `FragCABS`, and `FragECP` in the `%basis` block, followed by the number of the fragment (numbering starts at 1!) and a standard basis set or ECP from the ORCA library (see Section 2.7.2 and Table 2.38). Note that unlike the `NewGTO` keyword, `FragBasis` also changes the ECP, if applicable. Fragment basis sets will overload the global or element-specific (Section 2.7.6) choice but can be overloaded for individual atoms (Section 2.7.7). If AutoAux is requested for a fragment, it will be generated for the actual orbital basis set chosen for each atom, even if it is changed in the coordinates section. However, if AutoAux was requested for an element or in the simple input, the auxiliary basis will be generated before the fragment basis is assigned (which is not desired), therefore AutoAux must be requested again for the fragment. An example is given below:

```
! PrintBasis BP86 NoIter
! def2-SVP def2/J
%basis
  FragBasis 1 "def2-TZVP"
  FragBasis 2 "cc-pVTZ-PP"
  FragAuxJ  2 "AutoAux"
  FragECP   3 "SK-MCDHF-RSC"
  FragAuxJ  3 "def2/JK"
end
*xyz 0 1
  H(1) 0 0 0
  I(1) 0 0 1.6
  H(2) 0 5 0   NewGTO "cc-pVTZ" end
  I(2) 0 5 1.6
  H(3) 5 0 0
  I(3) 5 0 1.6
*
# Final basis sets:
# Atom Basis      ECP          AuxJ
# 0H   def2-TZVP  def2-ECP     def2/J
# 1I   def2-TZVP  def2-ECP     def2/J
# 2H   cc-pVTZ    -            AutoAux(cc-pVTZ)
# 3I   cc-pVTZ-PP SK-MCDHF-RSC AutoAux(cc-pVTZ-PP)
```

(continues on next page)

(continued from previous page)

```
# 4H   def2-SVP   -            def2/JK
# 5I   def2-SVP   SK-MCDHF-RSC def2/JK
```

It is also possible to read fragment-specific basis sets from a file. The syntax is analogous, using the keywords `ReadFragBasis`, `ReadFragAuxJ`, `ReadFragAuxJK`, `ReadFragAuxC`, `ReadFragCABS`, and `ReadFragECP`. In this case, the input string is expected to be an existing basis set file in GAMESS-US format (see section *Reading Basis Sets from a File*). All other details above (e.g., regarding ECPs and AutoAux) also apply here.

Note

- Details regarding the assignment of fragments can be found in *Fragment Specification* section.

2.7.9 Reading Basis Sets from a File

By using the variables `GTOName`, `GTOAuxJName`, `GTOAuxJKName`, `GTOAuxCName`, and `GTOCABSName` (`GTOAuxName` is a synonym for `GTOAuxJName`) a basis set can be read from an ASCII file. In this way you can construct or modify your favorite standard basis set and load it easily into the program.

```
%basis
  GTOName      = "MyBasis.bas"      # read orbital basis
  GTOAuxJName  = "MyAuxJBasis.bas"  # read Coulomb-fitting basis
  GTOAuxJKName = "MyAuxJKBasis.bas" # read Coulomb- and exchange-fitting basis
  GTOAuxCName  = "MyAuxCBasis.bas"  # read correlation-fitting basis
  GTOCABSName  = "MyCABSBasis.bas"  # read complementary auxiliary basis set
end
```

A word of caution: under Windows, backslashes directory assignments must be given twice to be correctly understood! The format is that used for "GAMESS-US" in the EMSL library [108]. To give an example of what this format looks like here is a part of the `3-21GSP` basis of Buenker and coworkers [109, 110]:

```
!                          lines in the beginning with '!' or '#' are comments
! BASIS="3-21GSP"
!Elements                             References
!--------                             ----------
! H - Ne: A.V. Mitin, G. Hirsch, R. J. Buenker, Chem. Phys. Lett. 259, 151 (1996)
! Na - Ar: A.V. Mitin, G. Hirsch, R. J. Buenker, J. Comp. Chem. 18, 1200 (1997).
!
$DATA         ! Optional
HYDROGEN      ! (3s) -> [2s]      Element symbols are also recognized
 S   2
  1           4.50036231          0.15631167
  2           0.68128924          0.90466909
 S   1
  1           0.15137639          1.00000000
CARBON        ! (6s,3p) -> [3s,2p]
 S   3
  1         499.24042249          0.03330322
  2          75.25419194          0.23617745
  3          16.86538669          0.81336259
 L   2        ! L shells are a s and a p shell with identical exponents
  1           0.89739483          0.24008573          0.46214684
  2           0.21746772          0.81603757          0.66529098
 L   1
  1           4.52660451          1.00000000          1.00000000
$END          ! Optional
```

The file format for the auxiliary basis sets is exactly the same. Basis sets can be also exported in GAMESS-US

format by the `orca_exportbasis` utility (section *orca_exportbasis*). Note that in order to read basis sets printed by ORCA (using `!PrintBasis`), the `NewGTO` and `end` keywords must be removed.

Warning

- Angular momentum 7 is labeled as "k" – there are no j-shells in accepted spectroscopic conventions.
- To avoid confusion with combined s- and p-shells, shells with angular momentum 8 can only be specified with a number, and not with the label "l".

2.7.10 Linear Dependence

The previous sections describe the assessment of a desired molecular basis set from appropriately parametrized functions at various locations within the molecule (normally centered on atoms). The parametrization of these functions is such that the chance for redundancy is minimal. Since however, one is limited to work with finite numerical precision, and furthermore these parameters also depend on the molecular geometry, redundancies cannot be completely eliminated in advance. Redundancy means that the subspace spanned by the given basis functions at given values of parameters (including geometry), can be identically spanned by a smaller number of *linear independent* basis functions. Linear dependent (redundant) function sets however may cause numerical instabilities. Linear dependence is normally identified by searching for zero eigenvalues of the overlap matrix. Note that the inverse of the overlap (or related matrices) are used for orthogonalization purposes, and it follows that if near zero eigenvalues are not treated properly, the inverse becomes ill-defined, and the SCF procedure numerically unstable.

From the previous discussion, it is evident that the crucial parameter for curing linear dependence is the threshold below which an overlap eigenvalue is considered zero. This parameter may be changed using the following keyword

```
%scf
  sthresh 1e-6 # default 1e-7
end
```

Although there is no strict limit for the value of the above parameter, it should reasonably be somewhere between 1e-5 and 1e-8 (the default is 1e-7). One may get away with 1e-9 or perhaps even lower without convergence problem, but there is a risk that the result is contaminated with noise caused by the near zero vectors. In difficult cases, an 1e-6 threshold was often found to work smoothly, and above that one risks throwing away more and more functions, which also influence comparability of results with other calculations. To monitor the behavior of the small eigenvalues, one should look for the following block in the output

```
Diagonalization of the overlap matrix:
Smallest eigenvalue                        ... -1.340e-17
Time for diagonalization                   ...    0.313 sec
Threshold for overlap eigenvalues          ... 1.000e-07
Number of eigenvalues below threshold      ... 1
Smallest eigenvalue above threshold        ... 6.013e-07
Time for construction of square roots      ...    0.073 sec
Total time needed                          ...    0.387 sec
```

Here, the smallest eigenvalue is printed, along with the currently used overlap threshold, and the number of functions below this (which will be dropped). It is a recommended consistency check to look for an equal number of zero entries among orbital energies once the SCF procedure converged. Note that for functions belonging to zero eigenvalues *no level shifts* are applied!

In case that redundant vectors were removed from the basis, `! MORead NoIter` should only be used in conjunction with the same `SThresh` as in the original calculation, otherwise the results will be inconsistent. `! MORead` may still be used together with a change in `SThresh`, but a few SCF iterations will be required.

Automatic Adjustments for Near Linear-Dependent Cases

Starting from ORCA6, there is now a keyword called DiffSThresh, which controls an automatic tightening of the integral cutoff parameters Thresh and TCut in case small eigenvalues of the overlap matrix are found. We found this to be important in some calculations using diffuse basis, and these parameters are set to a minimum value of `Thresh=1e-12` and `TCut=1e-13` in case the "Smallest eigenvalue" shown above gets below that number. If the cutoffs are already tighter than that, for instance when using `!VeryTightSCF`, than nothing will happen.

We found empirically that these are safe numbers to mitigate noise and increase the robustness of the SCF procedure, thus they are enforced by default. The default is 1e-6 and this can be turned off by setting `%SCF DiffSThresh -1 END` on the input in case you don't want this automatic adjustment to happen.

Removal of Redundant Basis Functions

While the approach described above is usually successful in removing linear dependencies from the orbital basis set, the auxiliary basis used in RI is not orthogonalized the same way. Instead, the RI linear equation system is solved using a Cholesky decomposition (CD) of the auxiliary basis Coulomb metric. If the auxiliary basis is redundant, the CD fails and the program usually aborts. One simple solution implemented in ORCA is to perform a pivoted Cholesky decomposition (PCD) of the metric, terminating at a given threshold. Then, the shells contributing to the nullspace are removed from the basis at the beginning of the calculation. This can be requested for any of the basis sets using either the overlap or the Coulomb metric. It is most appropriate for the AuxJ/AuxJK/AuxC basis using the Coulomb metric. The truncated basis can be examined using the `!PrintBasis` keyword. Often, functions may be removed for some atoms of a given element, but kept for others. As long as the threshold is low enough, i.e. only truly redundant functions are removed, this should not affect the molecular symmetry of the results.

```
%basis
  PCDTrimBas   Overlap # Trim the orbital basis in the overlap metric
  PCDTrimAuxJ  Coulomb # Trim the AuxJ basis in the Coulomb metric
  PCDTrimAuxJK Coulomb # Trim the AuxJK basis in the Coulomb metric
  PCDTrimAuxC  Coulomb # Trim the AuxC basis in the Coulomb metric
  PCDThresh    -1      # Threshold for the PCD: chosen automatically if <0
end
```

2.7.11 Which Methods Need Which Basis Sets?

ORCA offers a variety of methods and a large choice of *orbital* and *auxiliary* basis sets to go with them. Pure (GGA or meta-GGA) *DFT* functionals only require the calculation of Coulomb integrals, while *hybrid DFT*, HF (and by extension, all post-HF electron correlation methods, such as *MP2* and *coupled cluster*), as well as *CASSCF* (and *NEVPT2*), require the calculation of Coulomb and exchange integrals.

- An orbital basis set (`<basis>`) is always needed for these methods.
- If RI is used for Coulomb integrals (*RI-J*, *RIJDX/RIJONX*, *RIJCOSX*), `AuxJ` is needed (usually `<basis>/J` or `def2/J`).
- If RI is also used for exchange integrals (*RI-JK*), `AuxJK` is needed instead (usually `<basis>/JK` or `def2/JK`).
- If RI is used for integral generation in post-SCF correlation methods, as in *RI-MP2* (including *double-hybrid DFT*), *DLPNO-MP2*, and *DLPNO-CC*, `AuxC` is also needed (usually `<basis/C>`).
- In *F12 methods*, a specialized orbital basis is used (`<basis>-F12`) and `CABS` is needed in addition (usually `<basis>-F12-CABS` or `<basis>-F12-OptRI`).

An overview of auxiliary basis requirements for an inexhaustive list of methods and approximations is given in Table 2.39.

Table 2.39: Simple input keywords for basis sets and ECPs.

Method	Approximation	Basis sets
HF	NoRI (default)	<basis>
HF	RIJONX or RIJCOSX	<basis> + <basis>/J
HF	RI-JK	<basis> + <basis>/JK
pure DFT	RI (default)	<basis> + <basis>/J
hybrid DFT	NoRI	<basis>
hybrid DFT	RIJCOSX (default)	<basis> + <basis>/J
hybrid DFT	RI-JK	<basis> + <basis>/JK
CASSCF/NEVPT2		<basis>
CASSCF/NEVPT2	RI-JK	<basis> + <basis>/JK
CASSCF/NEVPT2	RIJCOSX	<basis> + <basis>/J + <basis>/C
CASSCF/NEVPT2	TrafoStep RI	<basis> + <basis>/JK or <basis>/C
NEVPT2-F12	TrafoStep RI	<basis>-F12 + <basis>-F12-CABS + <basis>/JK or <basis>/C
TDDFT		<basis>
MP2		<basis>
RI-MP2		<basis> + <basis>/C
RI-MP2	RI-JK	<basis> + <basis>/C + <basis>/JK
F12-MP2		<basis>-F12 + <basis>-F12-CABS
F12-RI-MP2		<basis>-F12 + <basis>-F12-CABS + <basis>/C
DLPNO-MP2		<basis> + <basis>/C
DLPNO-MP2	RIJCOSX	<basis> + <basis>/C + <basis>/J
F12-DLPNO-MP2		<basis>-F12 + <basis>-F12-CABS + <basis>/C
CCSD		<basis>
RI-CCSD		<basis> + <basis>/C
DLPNO-CCSD		<basis> + <basis>/C
DLPNO-CCSD	RIJCOSX	<basis> + <basis>/C + <basis>/J
F12-CCSD		<basis>-F12 + <basis>-F12-CABS
F12-RI-CCSD		<basis>-F12 + <basis>-F12-CABS + <basis>/C
F12-RI-CCSD	RI-JK	<basis>-F12 + <basis>-F12-CABS + <basis>/C + <basis>/JK

2.7.12 Keywords

Table 2.40: Simple input keywords for basis sets and ECPs.

Keyword	Description
<BasisName>	Assign the respective *orbial basis set* to all elements
<AUXJName>	Assign the respective *AuxJ basis set* to all elements
<AUXJKName>	Assign the respective *AuxJK basis set* to all elements
<AUXCName>	Assign the respective *AuxJC basis set* to all elements
<CABSName>	Assign the respective *CABS* to all elements
AutoAux	Automatically generate AuxJ, AuxJK, and AuxC auxiliary basis sets (see Section 2.7.4)
<ECPName>	Assign the respective ECP to all elements for which it is defined
NoECP	Remove the default ECP
Decontract	Decontract all (orbital and auxiliary) basis sets
DecontractBas	Decontract the orbital basis sets
NoDecontractBas	Do not decontract the basis set
DecontractAuxJ	Decontract the AuxJ basis set
NoDecontractAuxJ	Do not decontract the AuxJ basis
DecontractAuxJK	Decontract the AuxJK basis set
NoDecontractAuxJK	Do not decontract the AuxJK basis
DecontractAuxC	Decontract the AuxC basis set
NoDecontractAuxC	Do not decontract the AuxC basis

Table 2.41: `%basis` block input keywords for basis sets and ECPs.

Keyword	Options	Description
`Basis`	`"<BasisName>"`	Define the *orbital basis set*
`AuxJ`	`"<AuxName>"`	Define the *J auxiliary basis set*
`AuxJK`	`"<AuxName>"`	Define the *JK auxiliary basis set*
`AuxC`	`"<AuxName>"`	Define the *correlation auxiliary basis set*
`CABS`	`"<CABSName>"`	Define the *complementary auxiliary basis set* for F12 calculations
`ECP`	`"<ECPName>"`	Assign the respective *ECP* to all elements for which it is available
`GhostECP`	`false`	Activate ECPs on ghost atoms
`AllowGhostECP`	`false`	Equivalent to `GhostECP`
Decontraction options		
`Decontract`	`false`	If `true`, decontract all basis sets
`DecontractBas`	`false`	If `true`, 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
Setting basis sets for elements (see Section 2.7.6)		
`NewGTO`	`<Element> "<BasisName>" <shells> End`	Define new `Basis` for element via built-in name and/or custom shells
`AddGTO`	`<Element> <shells> End`	Add GTO shells to basis for element
`NewAuxJGTO`	`<Element> "<AuxName>" <shells> End`	Define new `AuxJ` set for element via built-in name and/or custom shells
`AddAuxJGTO`	`<Element> <shells> End`	Add GTO shells to `AuxJ` for element
`NewAuxJKGTO`	`<Element> "<AuxName>" <shells> End`	Define new `AuxJK` set for element via built-in name and/or custom shells
`AddAuxJKGTO`	`<Element> <shells> End`	Add GTO shells to `AuxJK` for element
`NewAuxCGTO`	`<Element> "<AuxName>" <shells> End`	Define new `AuxC` set for element via built-in name and/or custom shells
`AddAuxCGTO`	`<Element> <shells> End`	Add GTO shells to `AuxC` for element
`NewCABSGTO`	`<Element> "<CABSName>" <shells> End`	Define new `CABS` set for element via built-in name and/or custom shells
`AddCABSGTO`	`<Element> <shells> End`	Add GTO shells to `CABS` for element
`NewECP`	`<Element> "<ECPName>" End`	Define new built-in ECP for element
	`<Element> <shells> End`	Manually define new ECP for element (see Section 2.7.5)
`DelECP`	`<Element>`	Remove the ECP for the element
Setting basis sets for fragments (see Section 2.7.8)		
`FragBasis`	`<FragID> "<BasisName>"`	Define `Basis` for fragment
`FragAuxJ`	`<FragID> "<AuxName>"`	Define `AuxJ` for fragment
`FragAuxJK`	`<FragID> "<AuxName>"`	Define `AuxJK` for fragment
`FragAuxC`	`<FragID> "<AuxName>"`	Define `AuxC` for fragment
`FragCABS`	`<FragID> "<CABSName>"`	Define `CABS` for fragment
`FragECP`	`<FragID> "<ECPName>"`	Define `ECP` for fragment
`ReadFragBasis`	`<FragID> "<filename.bas>"`	Read `Basis` for fragment from file
`ReadFragAuxJ`	`<FragID> "<filename.bas>"`	Read `AuxJ` for fragment from file
`ReadFragAuxJK`	`<FragID> "<filename.bas>"`	Read `AuxJK` for fragment from file
`ReadFragAuxC`	`<FragID> "<filename.bas>"`	Read `AuxC` for fragment from file
`ReadFragCABS`	`<FragID> "<filename.bas>"`	Read `CABS` for fragment from file
`ReadFragECP`	`<FragID> "<filename.bas>"`	Read `ECP` for fragment from file
Reading basis sets from a file (see Section 2.7.9)		
`GTOName`	`<filename.bas>`	Read orbital basis from file
`GTOAuxJName`	`<filename.bas>`	Read `AuxJ` from file
`GTOAuxName`	`<filename.bas>`	Equivalent to `GTOAuxJName`
`GTOAuxJKName`	`<filename.bas>`	Read `AuxJK` from file
`GTOAuxCName`	`<filename.bas>`	Read `AuxC` from file
`GTOCABSName`	`<filename.bas>`	Read `CABS` from file
Removal of linear dependence (see Section 2.7.10)		
`PCDTrimBas`	`Overlap`	Trim the orbital basis in the overlap metric
`PCDTrimAuxJ`	`Coulomb`	Trim the AuxJ basis in the Coulomb metric
`PCDTrimAuxJK`	`Coulomb`	Trim the AuxJK basis in the Coulomb metric
`PCDTrimAuxC`	`Coulomb`	Trim the AuxC basis in the Coulomb metric
`PCDThresh`	`-1`	Threshold for the PCD (1e-16 to 1e-10 makes sense): chosen automatically if `<0`
AutoAux-related keywords (see Section 2.7.4)		
`AutoAuxSize`	0	Use minimal effective rather than minimal primitive exponent (suitable for ANO basis sets)
	1	(default) Increases the maximal exponent for the shells with low angular momenta.
	2	Increases the maximal exponent for all shells
	3	Directly uses the primitives and produces the largest fitting basis

continues on next page

Table 2.41 – continued from previous page

Keyword	Options	Description
AutoAuxLmax	false	If true, increase the maximal angular momentum of the fitting basis set to the highest value permitted by ORCA and by the orbital basis set.
AutoAuxLLimit	-1	If >0, do not exceed the given angular momentum.
AutoAuxF[0]	20.0	The factor to increase the maximal s-exponent
AutoAuxF[1]	7.0	Same for the p-shell
AutoAuxF[2]	4.0	Same for the d-shell
AutoAuxF[3]	4.0	Same for the f-shell
AutoAuxF[4]	3.5	Same for the g-shell
AutoAuxF[5]	2.5	Same for the h-shell
AutoAuxF[6]	2.0	Same for the i-shell
AutoAuxF[7]	2.0	Same for the j-shell
AutoAuxB[0]	1.8	Even-tempered expansion factor for the s-shell
AutoAuxB[1]	2.0	Same for the p-shell
AutoAuxB[2]	2.2	Same for the d-shell
AutoAuxB[3]	2.2	Same for the f-shell
AutoAuxB[4]	2.2	Same for the g-shell
AutoAuxB[5]	2.3	Same for the h-shell
AutoAuxB[6]	3.0	Same for the i-shell
AutoAuxB[7]	3.0	Same for the j-shell
AutoAuxTightB	true	Only use AutoAuxB[l] for shells with high l and AutoAuxB[0] for the rest
OldAutoAux	false	If true, selects the ORCA 3.1 generation procedure (deprecated)

2.8 Resolution-of-the-Identity (RI)

A very efficient and well-established way to speed up DFT calculations are resolution-of-the-idendity (RI) techniques also know as density-fitting.[111, 112, 113, 114, 115, 116, 117] Various variants to address the Coulomb part and Hartree-Fock exchange parts (cf. *Hartree-Fock* and *Hybrid DFT*). Available options available in ORCA include *RI-J*, *Split-RI-J*, *RIJONX*, *RI-JK*, and *RIJCOSX*.

Some Notes on RI in ORCA

- Any RI approximation requires the choice of a sufficiently large *auxiliary basis set*.
- *Split-RI-J* using the def2/J auxiliary basis is the default for *non-hybrid DFT*.
- *RIJCOSX* using the def2/J auxiliary basis is the default for *hybrid DFT*.
- When *scalar relativistic Hamiltonians* are used with all-electron basis sets, then SARC/J can be chosen as ageneral-purpose *auxiliary basis set*. Other choices are documented in the *basis sets section*.
- The usage of RI is generally recommended as the introduced error is very small. The speedup further enables the usage of larger *basis sets* for large systems which will improve the results significantly.
- The default usage of RI approximations can be disabled by the !NORI keyword (not recommended!).
- If you do not want to depend on the RI approximation, a reasonable yet inconvenient approach is to converge a RI-J calculation and then take the resulting orbitals as initial guess for a calculation with exact Coulomb term. This should converge within a few cycles and the total execution time should still be lower than just converging the calculation directly with exact Coulomb treatment.

2.8.1 RI-J

> **Note**
>
> This is the default for non-hybrid DFT! Can be turned off by using `!NORI`.

A very useful approximation that greatly speeds up DFT calculations unless the molecule gets very large is the so called "RI-approximation"

RI stands for "Resolution of the identity". In short, charge distributions arising from *products of basis functions* are approximated by a linear combination of *auxiliary basis functions*.

$$\phi_i\left(\vec{r}\right)\phi_j\left(\vec{r}\right) \approx \sum_k c_k^{ij}\eta_k(\mathbf{r}) \tag{2.2}$$

There are a variety of different possibilities to determine the expansion coefficients c_k^{ij}. A while ago, Almlöf and coworkers [118] have shown that for the approximation of electron repulsion integrals, the best choice is to minimize the *residual repulsion*.

> **Note**
>
> The basic theory behind the RI method has been known for a long time and since at least the late sixties, methods similar to the RI approximation have been used — mainly in the context of "approximate ab initio methods" such as LEDO, PDDO, and MADO, but also in density functional theory in the mid and late seventies by Baerends, Dunlap, and others [111, 112, 113, 114]

Define:

$$R_{ij} \equiv \phi_i\left(\vec{r}\right)\phi_j\left(\vec{r}\right) - \sum_k c_k^{ij}\eta_k\left(\vec{r}\right) \tag{2.3}$$

and

$$T_{ij} = \int\int R_{ij}\left(\vec{r}\right)\frac{1}{|\vec{r}-\vec{r}'|}R_{ij}\left(\vec{r}\right)d^3r d^3r' \tag{2.4}$$

Determining the coefficients that minimize T_{ij} leads to

$$\mathbf{c}^{ij} = \mathbf{V}^{-1}\mathbf{t}^{ij} \tag{2.5}$$

where:

$$t_k^{ij} = \left\langle \phi_i\phi_j \left| r_{12}^{-1} \right| \eta_k \right\rangle \tag{2.6}$$

$$V_{ij} = \left\langle \eta_i \left| r_{12}^{-1} \right| \eta_j \right\rangle \tag{2.7}$$

Thus, an ordinary two-electron integral becomes

$$\begin{aligned}\left\langle \phi_i\phi_j \left| r_{12}^{-1} \right| \phi_k\phi_l \right\rangle &\approx \sum_{p,q} c_p^{ij}c_q^{kl}V_{pq} \\ &= \sum_{p,q} V_{pq}\sum_r \left(\mathbf{V}^{-1}\right)_{pr} t_r^{ij}\sum_s \left(\mathbf{V}^{-1}\right)_{qs} t_s^{kl} \\ &= \sum_{r,s}\left(\mathbf{V}^{-1}\right)_{rs} t_r^{ij}t_s^{kl}\end{aligned} \tag{2.8}$$

and the total Coulomb energy becomes

$$\begin{aligned} E_J &= \sum_{i,j}\sum_{k,l} P_{ij}P_{kl}\left\langle \phi_i\phi_j \left| r_{12}^{-1} \right| \phi_k\phi_l \right\rangle \\ &\approx \sum_{i,j}\sum_{k,l} P_{ij}P_{kl}\sum_{r,s}\left(\mathbf{V}^{-1}\right)_{rs} t_r^{ij}t_s^{kl} \\ &= \sum_{r,s}\left(\mathbf{V}^{-1}\right)_{rs} \underbrace{\sum_{i,j} P_{ij}t_r^{ij}}_{\mathbf{X}_r} \underbrace{\sum_{k,l} P_{kl}t_s^{kl}}_{\mathbf{X}_s} \end{aligned} \tag{2.9}$$

where **P** is the total density matrix.

In a similar way, the Coulomb contribution to the Kohn-Sham matrix is calculated. There are substantial advantages from this approximation: the quantities to be stored are the matrix $\mathbf{V}^{-1}$ — which depends only on two indices — and the three-index auxiliary integrals t_r^{ij}. This leads to a tremendous reduction of storage requirements compared to a four-index list of repulsion integrals. In addition, the two- and three-index electron repulsion integrals are easier to compute than the four-index integrals, leading to further reductions in processing time. Furthermore, the Coulomb energy and the Kohn-Sham matrix contributions can be quickly assembled by simple vector/matrix operations, leading to large time savings. This arises because each auxiliary basis function $\eta_k\left(\vec{r}\right)$ appears in the expansion of many charge distributions $\phi_i\left(\vec{r}\right)\phi_j\left(\vec{r}\right)$. Unfortunately, a similar strategy is less easily applied (or, at least, with less benefit) to the Hartree-Fock exchange term.

If the auxiliary basis set $\{\eta\}$ is large enough, the approximation is also highly accurate. Since any DFT procedure already has a certain, sometimes sizable, error from the noise in the numerical integration of the XC part, it might be argued that a similarly large error in the Coulomb part is perfectly acceptable without affecting the overall accuracy of the calculation much. Furthermore, the errors introduced by the RI method are usually much smaller than the errors in the calculation due to basis set incompleteness. It is therefore recommended to use the RI procedure for pure DFs. However, one should probably not directly mix absolute total energies obtained from RI and non-RI calculations as the error in the total energy accumulates and will rise with increased molecular size, while the errors in the relative energies will tend to cancel.

There are several choices for auxiliary basis sets described in the next section, which depend on the choice of the primary GTO basis set used to expand the molecular orbitals[1]. The RI procedure requires the inversion of the auxiliary basis metric **V**, which is often the most expensive matrix operation in the calculation, due to the O(N^3) scaling and the large size of the auxiliary basis. However, in ORCA this is done via an efficient Cholesky decomposition, which is only performed once during the startup phase of a single point caclulation. Hence, this step is usually of no concern in practice.

In ORCA, the RI approximation is toggled by the input

```
%method
  RI  on   # do use the RI-J approximation
      off  # do not use the RI-J approximation
end
```

Note

If you use RI, you *must* specify an auxiliary basis set (in the `%basis` section or using the appropriate simple keyword) or use the `!AutoAux` simple keyword.

[1] It probably should be noted that a slightly awkward step in the procedure is the inversion of the auxiliary integral matrix V, which can easily become very large. Matrix inversion is an $O(N^3)$ process such that for large molecules, this step takes some real time. However, in ORCA, this is only done once during the calculation, whereas other programs that constrain the fit to also exactly reproduce the number of electrons perform a similar process each iteration. Starting from ORCA 2.2.09, the Cholesky decomposition is used in favor of matrix inversion, removing any bottleneck concerning the solution of the linear equation system.

2.8.2 Split-RI-J

There is an improved version of the RI-algorithm that has been implemented since ORCA 2.2.09. This Split-RI-J algorithm yields the same Coulomb energy as the standard RI-algorithm, but is significantly faster if the basis set contains many high angular momentum functions (d-, f-, g-functions). For small basis sets, there is virtually no difference between the two algorithms, except that Split-RI-J uses more memory than standard RI. However, calculations with ca. 2000 basis functions only need about an extra 13 MB for Split-RI-J, which is a trivial requirement on present-day hardware.

The Split-RI-J algorithm is invoked with the `!Split-RI-J` simple keyword. Split-RI-J is presently only available for SCF and gradient calculations.

Note

- The Split-RI-J algorithm is the default if RI is turned on via `!RI`. If you do not want to use Split-RI-J, please also use the keyword `!NoSplit-RI-J`

2.8.3 RIJONX

Alternatively, the RI method can be used for the Coulomb term and the standard treatment for the exchange term. This method is called RIJONX since the exchange term should tend towards linear scaling for large molecules. RIJONX can be invoked via the `!RIJONX` keyword for Hartree-Fock and hybrid DFT calculations.

```
! HF RIJONX
```

The requirements for the auxiliary basis are the same as for the normal RI-J method.

2.8.4 RIJCOSX

Frustrated by the large difference in execution times between pure and hybrid functionals, we have been motivated to study approximations to the Hartree-Fock exchange term. The method that we have finally come up with is called the “chain of spheres” COSX approximation and may be thought of as a variant of the pseudo-spectral philosophy. Essentially, in performing two electron integrals, the first integration is done numerically on a grid and the second (involving the Coulomb singularity) is done analytically. For algorithmic and theoretical details see Refs. [119] and [120]. Upon combining this treatment with the *Split-RI-J* method for the Coulomb term (thus, a Coulomb fitting basis is needed!), we have designed the RIJCOSX approximation that can be used to accelerate Hartree-Fock and hybrid DFT calculations. Note that this introduces another grid on top of the DFT integration grid which is usually significantly smaller.

Note

Since ORCA 5, RIJCOSX is the default option for hybrid DFT (can be turned off by using `!NOCOSX`). However, it is by default NOT turned on for HF.

In particular for large and accurate basis sets, the speedups obtained in this way are very large - we have observed up to a factor of sixty! The procedure is essentially linear scaling such that large and accurate calculations become possible with high efficiency. The RIJCOSX approximation is basically available throughout the program. The default errors are on the order of 0.05 ± 0.1 kcal mol^{-1} or less in the total energies as well as in energy differences and can be made smaller with larger than the default grids or by running the final SCF cycle without this approximation. The impact on bond distances is a fraction of a pm, angles are better than a few tenth of a degree and soft dihedral angles are good to about 1 degree. To the limited extent to which it has been tested, vibrational frequencies are roughly good to 0.1 wavenumbers with the default settings.

The aim of this approximation is to efficiently compute the elements of exchange-type matrices as described in refs. [119, 120]:

$$K_{\mu\nu} = \sum_{\kappa\tau} P_{\kappa\tau}(\mu\kappa|\nu\tau) \tag{2.10}$$

where **P** is some kind of density-type matrix (not necessarily symmetric) and the two-electron integrals are defined over the basis set $\{\varphi\}$ by:

$$(\mu\kappa|\nu\tau) = \int \mu(\mathbf{r}_1)\kappa(\mathbf{r}_1)\nu(\mathbf{r}_2)\tau(\mathbf{r}_2)r_{12}^{-1}d\mathbf{r}_1 d\mathbf{r}_2 \tag{2.11}$$

The approximation pursued here can be written as follows:

$$K_{\mu\nu} \approx \sum_g X_{\mu g} \sum_\tau A_{\upsilon\tau}(\mathbf{r}_g) \sum_\kappa P_{\kappa\tau} X_{\kappa g} \tag{2.12}$$

Here, the index g refers to grid points $\mathbf{r}_g$ and:

$$X_{\kappa g} = w_g^{1/2}\kappa(\mathbf{r}_g) \tag{2.13}$$

$$A_{\upsilon\tau}(\mathbf{r}_g) = \int \frac{\nu(\mathbf{r})\tau(\mathbf{r})}{|\mathbf{r}-\mathbf{r}_g|}d\mathbf{r} \tag{2.14}$$

where w_g denotes the grid weights. Thus, the first integration is carried out numerically and the second one analytically. Note that this destroys the Hermitian character of the two-electron integrals.

Equation (2.12) is perhaps best evaluated in three steps:

$$F_{\tau g} = (\mathbf{PX})_{\tau g} \tag{2.15}$$

$$G_{\nu g} = \sum_\tau A_{\nu\tau}(\mathbf{r}_g)F_{\tau g} \tag{2.16}$$

$$K_{\mu\nu} = (\mathbf{XG}^+)_{\mu\nu} \tag{2.17}$$

As such, the equations are very similar to the pseudo-spectral method extensively developed and discussed by Friesner and co-workers since the mid 1980s and commercially available in the Jaguar quantum chemistry package. The main difference at this point is that instead of $X_{\kappa g}$ there appears a least-square fitting operator $Q_{\kappa g}$ in Friesner's formulation. Note that an analogue of the fitting procedure has also been implemented in ORCA and — in contrast to Friesner's pseudo-spectral method — does not need specially optimized grids. The basic idea is to remove the grid errors within the basis set by "fitting" the numerical overlap to the analytical one. Due to its nature, overlap fitting is supposed to work better with larger basis sets.

Basic Usage

The RIJCOSX approximation can be invoked by the `!RIJCOSX` keyword:

```
! HF RIJCOSX
```

By default, RIJCOSX will make use of the `def2/J` *auxiliary basis set*. Note that as the requirements for the SCF and correlation fitting bases are quite different an *correlation auxiliary basis set* has to be defined additionally. This is e.g. the case for *RI-MP2* or *double-hybrid DFT*.

```
! RI-MP2 def2-TZVPP def2/J def2-TZVPP/C RIJCOSX
```

Note

This is the default for hybrid DFT! The COSX part can be turned off by using the `!NOCOSX` keyword.

RIJCOSX Gradient

Given the exchange matrix, the exchange energy is given by (a sum over spin cases is left out here for simplicity):

$$E_{\mathrm{X}} = \frac{1}{2}\sum_{\mu\nu} P_{\mu\nu} K_{\mu\nu}(\mathbf{P}) \tag{2.18}$$

Previous to ORCA6, the gradient of the COSX contribution to the energy was taken as an approximation:

$$\frac{\partial E_X}{\partial \lambda} \approx 2\sum_{g}\sum_{\mu\nu} \frac{\partial F_{\mu g}}{\partial \lambda} G_{\nu g} \tag{2.19}$$

with

$$\frac{\partial F_{\mu g}}{\partial \lambda} = w_g^{1/2} \sum_{\kappa} P_{\kappa\mu} \frac{\partial X_{\mu g}}{\partial \lambda} \tag{2.20}$$

as published in the original implementation paper [119].

Starting from ORCA6, this was updated to the full derivative of the energy component, including the derivative of all terms: grid weight derivatives, derivative of the SFitting matrices and all derivatives of F and G[2]. The gradient is thus now more accurate and less noisy. In case one wants to revert to the previous approximated version (not recommended), just set:

```
%method
  cosxgradtype grad_n
  useqgradfit false
end
```

COSX Numerical Grids

For expert users, the grid parameters for the exchange grids can be even more finely controlled:

```
%method
  IntAccX  Acc1, Acc2, Acc3
  GridX    Ang1, Ang2, Ang3
end
```

There are three grids involved: the smallest grid (Acc1, Ang1) that is used for the initial SCF iterations, the medium grid (Acc2, Ang2) that is used until the end of the SCF and the largest grid (Acc3, Ang3) that is used for the final energy and the gradient evaluations. `UseFinalGridX` can be used to turn this last grid on or off, though changing this is not generally recommended. More details about the grid constructions can be found in *Numerical Integration*.

SFitting Parameters

To modify the overlap fitting parameters, the following input can be specified:

```
%method
  UseSFitting false   # Same as NoSFitting in the simple input
                      # (Default is true)
  UseQGradFit false   # Uses the SCF fitting matrix for gradient calculations
                      # (Default is true)
end
```

Note that overlap fitting also works for HF and MP2 gradients without specifying any additional keyword. The `UseQGradFit` parameter uses the same fitting matrix for the gradients as for the energy calculation and is the default behavior since ORCA6.

[2] The theory is not yet published, but will be soon.

Partial Contraction Scheme

Since ORCA 5.0, generally-contracted basis sets can be handled efficiently by using an intermediate partially contracted (pc) atomic-orbital basis for the exchange-matrix computation without affecting the results [120]. Depending on the basis set and element type, computational speedups by many orders of magnitude are possible. For testing or benchmark purposes, the K matrix computation can be done in the original basis by using the flag

```
%method
  COSX_PartialContraction false   # No intermediate basis for generally contracted␣
↪shells
                                  # (Default is true)
end
```

Restoring Full Symmetry

The semi-numerical integration scheme in the default COSX algorithm breaks the permutational symmetry of the two-electron integrals. We have observed that this flaw is often the cause of convergence problems for iterative algorithms, in particular for multi-reference theories [121]. An input option is available since ORCA 6.0 to preserve the full eight-fold permutational symmetry of the two-electron integrals:

```
%method
  COSX_IntSymmetry Full      # Fully symmetrized integrals
                   Standard  # Original COSX algorithm
end
```

The full-symmetrization algorithm often improves the numerical stability and is enabled by default for TRAH-CASSCF and CASSCF linear-response calculations. However, the full-symmetrization algorithm may come with additional costs that depend on the number of **F** intermediates. The number of **F** intermediates depends on the symmetry of the density matrix (symmetric (S), anti-symmetric (A), and non-symmetric (N)) and whether overlap fitting is employed, as summarized in the table below.

Table 2.42: Number of COSX **F** intermediates per density matrix

S fitting	P symmetry (S, A, N)	Number of **F** intermediates
No	S / A	1
No	N	2
Yes	S / A	2
Yes	N	4

Note that for symmetric (S) and anti-symmetric (A) densities, we symmetrize and anti-symmetrize, respectively, exchange matrices at the end, which reduces the number of **F** intermediates by a factor of 2. The actual computational costs usually do not increase linearly with the number of **F** intermediates since we compute the costly analytic integrals ($A_{\nu\tau}(\mathbf{r}_g)$) only once and then contract them with the additional **F** intermediates. From our experience the overhead of the full-symmetrization algorithm is roughly between 1.2 and 1.5 times that of the original algorithm.

COSX Grid and Convergence Issues

Symptoms of convergence issues: Erratic convergence behavior, often starting from the first SCF step or possibly setting in at a later stage, which produce crazy energy values with "megahartree" jumps. If overlap fitting is on, the following error message may also be encountered: "Error in Cholesky inversion of numerical overlap".

Convergence issues may arise when the chosen grid has difficulties in representing the basis set. This is the "grid equivalent" of a linear dependence problem, as discussed in *Linear Dependence*. It should also be mentioned that the grid-related problem discussed here often goes hand in hand with basis set linear dependence, although not necessarily. The most straightforward way of dealing with this is to increase the size of the integration grid. This, however, is not always desirable or possible.

One way to avoid the Cholesky inversion issue is to turn overlap fitting off (`!NoSFitting`). However, this means that the numerical problems are still present, but are ignored. Due to the fact that overlap fitting operates with the numerical overlap and its inverse, it is more sensitive to linear dependence issues, so turning off the fitting procedure may lead to convergence. This may be a pragmatic — but by no means clean — solution, since it relies on the assumption that the numerical errors are small.

On the other hand, overlap fitting also gives a similar tool to deal with linear dependence issues as the one discussed for basis sets. The eigenvalues of the numerical overlap can be similarly inspected and small values screened out. There is unfortunately no universal way to determine this screening parameter, but see *Linear Dependence* for typical values.

The parameters influencing the method used for inversion and obtaining the fitting matrix are:

```
%method
  SFitInvertType Cholesky    # Cholesky inversion (default)
                 Cholesky_Q  # Cholesky + full Q matrix
                 Diag        # Inversion via diagonalization
                 Diag_Q      # Diag + full Q matrix
  SInvThresh     1e-8        # Inversion threshold for Diag and Diag_Q (default 1e-
↪8)
end
```

By default, the inversion procedure proceeds through Cholesky decomposition as it is the fastest option. Ideally, the overlap matrix is non-singular if the basis set is not linearly dependent. For singular matrices, the Cholesky procedure will fail. It should be noted at this point that the numerical overlap can become linearly dependent even if the overlap of basis functions is not, and so a separate parameter will be needed to take care of grid-related issues. To achieve this, a diagonalization procedure (`Diag`) can be used instead of Cholesky with the corresponding parameter to screen out eigenvectors belonging to eigenvalues below a threshold (`SInvThresh`). For both Cholesky and diagonalization procedures, a "full Q" approach is also available (`Cholesky_Q` and `Diag_Q`), which corresponds to the use of a more accurate (untruncated) fitting matrix.

2.8.5 RI-JK

An alternative algorithm for accelerating the HF exchange in hybrid DFT or HF calculations is RI-JK developed by Kendall and Früchtl[115]. It will use the RI approximation for both Coulomb and exchange and allows for a fair approximation to the Hartree-Fock exchange matrix. It can be difficult to make this approximation highly accurate. It is, however, usefully fast compared to direct SCF if the molecule is "dense" enough. There are special auxiliary basis sets for this purpose (see section *Basis Sets*). RI-JK is implemented in ORCA for SCF single point energies but not for gradients. The speedups for small molecules are better than for *RIJCOSX*, for medium sized molecules (e.g. $(\mathrm{gly})_4$) similar, and for larger molecules RI-JK is less efficient than RIJCOSX. A detailed comparison can be found in Section 2.8.6. The errors of RI-JK are usually below 1 mEh and the error is very smooth (smoother than for *RIJCOSX*). Hence, for small calculations with large basis sets, RI-JK is a good idea, for large calculations on large molecules *RIJCOSX* is better.

Note

- RI-JK requires a larger auxiliary basis set. For the *Karlsruhe basis set*, the universal `def2/JK` and `def2/JKsmall` basis sets are available. They are large and accurate.
- For UHF RI-JK is roughly twice as expensive as for RHF. This is not true for RIJCOSX.
- RI-JK is available for conventional and direct runs and also for ANO bases. There the conventional mode is recommended.

Basic Usage

The RI-JK approximation can be invoked by the `!RI-JK` keyword:

```
! RHF def2/JK RI-JK
```

2.8.6 Speed Comparison of RIJCOSX and RI-JK

A comparison of the *RIJCOSX* and *RI-JK* methods is shown in Table 2.43.

Table 2.43: Comparison of wall clock times in seconds for a full SCF for $(gly)_2$, $(gly)_4$, and $(gly)_8$. Data taken from Ref. [122]

		def2-SVP	def2-TZVP(-df)	def2-TZVPP	def2-QZVPP
$(gly)_2$	*Default*	105	319	2574	27856
	RI-JK	44	71	326	3072
	RIJCOSX	70	122	527	3659
$(gly)_4$	*Default*	609	1917	13965	161047
	RI-JK	333	678	2746	30398
	RIJCOSX	281	569	2414	15383
$(gly)_8$	*Default*	3317	12505	82774	
	RI-JK	3431	5452	16586	117795
	RIJCOSX	1156	2219	8558	56505

It is obvious from the data that for small molecules the *RI-JK* approximation is the most efficient choice. For $(gly)_4$ this is already no longer obvious. For up to the def2-TZVPP basis set, *RI-JK* and *RIJCOSX* are almost identical and for def2-QZVPP *RIJCOSX* is already a factor of two faster than *RI-JK*. For large molecules like $(gly)_8$ with small basis sets *RI-JK* is not a big improvement but for large basis set it still beats the normal 4-index calculation. *RIJCOSX* on the other hand is consistently faster. It leads to speedups of around 10 for def2-TZVPP and up to 50-60 for def2-QZVPP. Here it outperforms *RI-JK* by, again, a factor of two.

2.8.7 Keywords

Table 2.44: Simple input keywords for the RI approximations.

Keyword	Description
`RI`	Activates *RI-J* (uses *Split-RI-J* by default)
`NORI`	Deactivates RI treaments
`SplitRIJ`, `Split-RI-J`	Activates *Split-RI-J*
`NoSplitRIJK`, `NoSplit-RI-J`	Activates *Split-RI-J*
`RIJONX`, `RIJDX`	Activates *RIJONX*
`RIJCOSX`, `RIJDX`	Activates *RIJCOSX*
`NORIJCOSX`, `NOCOSX`	Deactivates RIJCOSX, will activate RIJONX instead
`RIJK`, `RI-JK`	Activates *RI-JK*
`NoSFitting`	Deactivates *SFitting*