Regarding calculations 2 and 3, to produce smooth σ profiles, ORCA discards the surface segments with an area < 0.01 Å^2. For these two calculations, the radii of several elements used to construct the cavity of solute and solvent are different than the defaults employed within non-COSMO-RS calculations (using C-PCM). This is due to the fact that the provided openCOSMO-RS binaries involve a special parametrization of COSMO-RS for ORCA 6.0, and this does not only affect the COSMO-RS parameters, but also the radii of several elements used in the C-PCM part.

After point 4 is done, the free energy of solvation (ΔG_S) of the solute in the solvent is printed in the ORCA output file. In ORCA 6.0 the use of openCOSMO-RS is restricted to the calculation of ΔG_S.

How to Run a ORCA/COSMO-RS Calculation

ORCA/COSMO-RS calculations are controlled through the `%cosmors` block. These type of calculations require two input structures, one for the solute and one for the solvent. The solute coordinates are provided in the input file as done for any other type of calculation. However, regarding the structure for the solvent, there are two options:

1. Retrieve the structure of the solvent from a database
2. Provide the structure in a separate file

To use strategy 1, we need to request the solvent via:

```
COSMORS(Solvent)
```

in the simple input or using the `%cosmors` block:

```
%cosmors
  solvent "Solvent"
end
```

The list of internal solvents available in ORCA are shown in Table 2.56. For instance, for a water molecule solvated by acetonitrile, the input file looks like:

```
!COSMORS(Acetonitrile)
* xyz 0 1
O    0.00000006589375       0.00157184228646       0.00000000004493
H    0.77316868532439      -0.58666889665624      -0.00000000000005
H   -0.77316876182122      -0.58666895650640      -0.00000000000005
*
```

If the user wants to provide a structure for the solvent (strategy 2), then a separate file with extension **.cosmorsxyz** should be available. The name of this file (without extension) is controlled by the tag `solventfilename` in the `%cosmors` block. For instance, if we want to calculate the free energy of solvation of acetone in water, the ORCA input file would look like this:

```
%cosmors
  solventfilename "water"
end

* xyz 0 1
H    1.99757808828569      0.25022586507917      0.72957579847856
C    1.44666788654273      0.06088176074125     -0.18975506731892
H    1.67115809398389      0.82690156694531     -0.93346420198265
H    1.76410010378270     -0.89580894800398     -0.61702931492419
C   -0.03218812888253     -0.00668250402989      0.08117960952503
O   -0.47126229461002     -0.06383203315654      1.21590571064657
C   -0.93671505604705     -0.00759565576779     -1.12174123739234
H   -0.88761294036774      0.97611724740670     -1.59852429171257
H   -0.58940208311204     -0.73299176093053     -1.85998395179737
H   -1.96332366957567     -0.22151553828368     -0.83026305352213
*
```

The structure of the solvent (water) is the one in the `water.cosmorsxyz` file:

```
3
0 1
O    0.00000006589375       0.00157184228646       0.00000000004493
H    0.77316868532439      -0.58666889665624      -0.00000000000005
H   -0.77316876182122      -0.58666895650640      -0.00000000000005
```

where the first line corresponds to the number of atoms, and in the second line the charge and multiplicity are provided.

The output for COSMO-RS is printed in the ORCA output file after the line that reads `OPENCOSMO-RS CALCULATION`. First of all, the information regarding the level of theory, the solute and the solvent is printed. For the example above (acetone in water), it reads as,

```
------------------------------------------------------------------------------
                          OPENCOSMO-RS CALCULATION
------------------------------------------------------------------------------


----------------------
GENERAL INFORMATION
----------------------
Calculation method                            ... DFT
Functional                                    ... BP86
Basis set                                     ... DEF2-TZVPD

----------------------
SOLUTE INFORMATION
----------------------
Number of atoms                               ...     10
Total charge                                  ...      0
Multiplicity                                  ...      1

CARTESIAN COORDINATES (ANGSTROEM)
  H      1.997578    0.250226    0.729576
  C      1.446668    0.060882   -0.189755
  H      1.671158    0.826902   -0.933464
  H      1.764100   -0.895809   -0.617029
  C     -0.032188   -0.006683    0.081180
  O     -0.471262   -0.063832    1.215906
  C     -0.936715   -0.007596   -1.121741
  H     -0.887613    0.976117   -1.598524
  H     -0.589402   -0.732992   -1.859984
  H     -1.963324   -0.221516   -0.830263

----------------------
SOLVENT INFORMATION
----------------------
Solvent name                                  ... water
Number of atoms                               ...      3
Total charge                                  ...      0
Multiplicity                                  ...      1

CARTESIAN COORDINATES (ANGSTROEM)
  O      0.000000    0.001572    0.000000
  H      0.773169   -0.586669   -0.000000
  H     -0.773169   -0.586669   -0.000000
```

After these lines, ORCA prints the final single point energy for each of the QM calculations, together with the output file to which the ORCA output is redirected:

```
----------------------------------------------
Single Point Calculation (solute / gas-phase)
----------------------------------------------
Output single point calculation redirected to >test.solute_vac.lastout
```

(continues on next page)

(continued from previous page)

```

FINAL SINGLE POINT ENERGY (Solute-gas-phase)     -193.233975714789

----------------------------------------------
Single Point Calculation (solute / CPCM)
----------------------------------------------
Output single point calculation redirected to >test.solute_cpcm.lastout

FINAL SINGLE POINT ENERGY (Solute-CPCM)     -193.243987123912

----------------------------------------------
Single Point Calculation (solvent / CPCM)
----------------------------------------------
Output single point calculation redirected to >test.solvent_cpcm.lastout

FINAL SINGLE POINT ENERGY (Solvent-CPCM)      -76.479155022866
```

Once this information is printed, ORCA calls the openCOSMO-RS executable and ΔG_S is printed in the following block:

```
----------------------
SOLVATION DATA
----------------------
Reference temperature              :                  298.15 K
Free energy of solvation (dGsolv)  :       -0.006626234385 Eh             -4.
↪158026 kcal/mol

Note: In order to calculate the free energy of the solvated solute (Gsolv), one␣
↪should add
      the computed dGsolv to the "Final Gibbs free energy" (Gvac = H-T*S) of the␣
↪solute in gas-phase.
      That is: Gsolv = Gvac + dGsolv. Here, the Gvac has been calculated␣
↪previously after a frequency
      calculation of the solute at a certain level of theory and printed in the
↪"THERMOCHEMISTRY" block.
```

As pointed out in the last paragraph of the COSMO-RS output, to calculate the Gibbs free energy of the solute solvated by the given solvent, one should add the calculated ΔG_S to the `Final Gibbs free energy` of the solute in the gas-phase.

Note: The workflow explained above for ORCA/openCOSMO-RS calculations involves the same structure for the solute in the gas-phase and in solution. However, these structures may differ substantially depending on the type of solute and solvent. In ORCA, it is possible to optimize the structures for each of the three calculations needed in the ORCA/openCOSMO-RS workflow. That is, (1) the solute in gas-phase, (2) the solute in a conductor, and (3) the solvent in a conductor. To do that, one needs to add the level of optimization via the `dftfunc` tag, which is exclusive for the DFT functional, but as an exception, can be extended with the optimization tag:

```
%cosmors
  dftfunc  "BP86 tightopt"
end
```

OpenCOSMO-RS Keywords

The parameters that can be defined in the `%cosmors` block in the ORCA input file are the following:

```
%cosmors
  aeff                     5.92500 # Effective contact area between surface segments␣
↪(Å^2)
  lnalpha                  0.20200 # Logarithm of the misfit prefactor
  lnchb                    0.16600 # Hydrogen bond (HB) strength parameter
  chbt                     1.50    # Parameter for the temperature dependence of the HB
  sigmahb                  9.61e-3 # HB threshold parameter (e/Å^2)
  rav                      0.50    # Radius to average ideal screening charges in Å
  fcorr                    2.40    # Parameter adjusted from dielectric screening␣
↪energies
  ravcorr                  1.00    # Additional radius to calculate the misfit energy␣
↪in Å
  astd                     41.6240 # Standard surface area (normalization factor) in Å^
↪2
  zcoord                   10.0    # Coordination number
  dgsolv_eta              -4.44800 # Offset for the solv. energy calculation
  dgsolv_omegaring         0.26300 # Correction for solv. energy of molecules with␣
↪rings
  temp                     298.15  # Reference temperature in Kelvin
  dftfunc                  "BP86"  # String for the DFT functional
  dftbas                   "def2-TZVPD"  # String for the basis set
  solvent                  "THF"   # Solvent from the internal database
  solventfilename          "water" # Name of the .cosmorsxyz solvent file
  orbs_vac                 false   # Reuse the gas-phase orbitals for the calculation
                                   # involving the solute in a conductor
end
```

It is not recommended to change the defaults of the COSMO-RS parameters.

2.13.7 Implicit Solvation in Coupled-Cluster Methods

The coupled-cluster Lagrangian, $\mathcal{L}$, for a system implicitly solvated reads as follows,[166, 167, 168]

$$\mathcal{L}(\Lambda, T) = \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}H_0\mathrm{e}^{T}|\psi_0\rangle + \frac{1}{2}\tilde{\mathbf{Q}}(\Lambda, T)\cdot\tilde{\mathbf{V}}(\Lambda, T) \tag{2.64}$$

where ψ_0 is the reference wave function, and H_0 is the Hamiltonian for the isolated molecule. The operator T for CCSD is defined in terms of single and double excitations ($T = T_1 + T_2$), and Λ is the de-excitation operator, defined in terms of the Lagrange multipliers:

$$T = T_1 + T_2 = \sum_{ia} t_a^i a_a^+ a_i + \sum_{ijab} t_{ab}^{ij} a_a^+ a_b^+ a_j a_i \tag{2.65}$$

$$\Lambda = \sum_{ia} \lambda_i^a a_i^+ a_a + \frac{1}{2}\sum_{ijab} \lambda_{ij}^{ab} a_i^+ a_a a_j^+ a_b \tag{2.66}$$

Here, t_a^i and t_{ab}^{ij} are the singles and doubles wave function amplitudes and a_i and a_a^+ are standard fermion annihilation and creation operators, respectively. Canonical occupied orbitals are denoted by the symbols $i, j, k, \ldots$, virtual orbitals by the symbols $a, b, c, \ldots$, and we use the symbols $p, q, r \ldots$ for general orbital indices.

The quantities $\tilde{\mathbf{Q}}$ and $\tilde{\mathbf{V}}$ are the CC expectation values of the C-PCM operators $\mathbf{Q}$ and $\mathbf{V}$, which are the solvation charges vector and solute potential vector defined at the position the charges, respectively.

$$\tilde{\mathbf{Q}}(\Lambda, T) = \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}\mathbf{Q}\mathrm{e}^{T}|\psi_0\rangle \tag{2.67}$$

$$\tilde{\mathbf{V}}(\Lambda, T) = \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}\mathbf{V}\mathrm{e}^{T}|\psi_0\rangle \tag{2.68}$$

Equation (2.64) can be rewritten by introducing the normal product form of an operator:

$$X_N = X - \langle\psi_0|X|\psi_0\rangle = X - X_0 \tag{2.69}$$

If one uses this result in eq (2.64), together with the fact that **Q** and **V** are related through eq (2.54), then eq (2.64) reads as,

$$\begin{aligned}\mathcal{L}(\Lambda,T) =& \langle\psi_0|H_0|\psi_0\rangle + \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}H_{0N}\mathrm{e}^{T}|\psi_0\rangle + \frac{1}{2}\mathbf{Q}_0\cdot\mathbf{V}_0 + \mathbf{Q}_0\cdot\tilde{\mathbf{V}}_N(\Lambda,T) + \frac{1}{2}\tilde{\mathbf{Q}}_N(\Lambda,T)\cdot\tilde{\mathbf{V}}_N(\Lambda,T) = \\ =& E_0 + \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}H_{0N}\mathrm{e}^{T}|\psi_0\rangle + \mathbf{Q}_0\cdot\tilde{\mathbf{V}}_N(\Lambda,T) + \frac{1}{2}\tilde{\mathbf{Q}}_N(\Lambda,T)\cdot\tilde{\mathbf{V}}_N(\Lambda,T)\end{aligned} \tag{2.70}$$

Here, $\mathbf{Q}_0$ and $\mathbf{V}_0$ are the **Q** and **V** vectors calculated with the ψ_0 wave function, and E_0 is the reference energy ($E_0 = \langle\psi_0|H_0|\psi_0\rangle + \frac{1}{2}\mathbf{Q}_0\cdot\mathbf{V}_0$). Different approximations can be adopted in eq (2.70) depending on how one calculates its last term $\frac{1}{2}\tilde{\mathbf{Q}}_N(\Lambda,T)\cdot\tilde{\mathbf{V}}_N(\Lambda,T)$.

In ORCA there are three different CCSD/CPCM approaches: (i) the PTE scheme, (ii) the PTE(S) scheme, and the (iii) the PTES scheme, being the last one the default. Here, the acronym PTE stands for "perturbation theory and energy" and "S" for singles. The choice of any of these approaches is controlled via the tag `CPCMccm`. Information about which CCSD/C-PCM is used by ORCA in a calculation is printed in the `ORCA-MATRIX DRIVEN CI` block in the output file in the line starting by `CPCM scheme`:

```
--------------------------------------------------------------------------------
                             ORCA-MATRIX DRIVEN CI
--------------------------------------------------------------------------------

--------------------------------
AUTOMATIC CHOICE OF INCORE LEVEL
--------------------------------

Memory available                           ...   2000.00 MB
Memory needed for S+T                      ...      9.26 MB
Memory needed for J+K                      ...     18.57 MB
Memory needed for DIIS                     ...    129.61 MB
Memory needed for 3-ext                    ...     72.69 MB
Memory needed for 4-ext                    ...    486.14 MB
Memory needed for triples                  ...      0.00 MB
 -> Final InCoreLevel    ... 5

Wavefunction type
-----------------
Correlation treatment                      ...      CCSD
Single excitations                         ... ON
Orbital optimization                       ... OFF
Calculation of Lambda equations            ... ON
Calculation of Brueckner orbitals          ... OFF
Perturbative triple excitations            ... OFF
CPCM scheme                                ... PTE(S)
Calculation of F12 correction              ... OFF
```

In the following subsections, we describe the different CCSD/C-PCM approaches available in ORCA and how to use them.

PTE scheme

In the "perturbation theory energy" (PTE) scheme, the last term in eq (2.70) is equal to zero (this term does not depend on Λ and T),

$$\mathcal{L}(\Lambda, T) = E_0 + \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}H_{0N}\mathrm{e}^{T}|\psi_0\rangle + \mathbf{Q}_0 \cdot \tilde{\mathbf{V}}_N(\Lambda, T) \tag{2.71}$$

The potential $\tilde{\mathbf{V}}_N$ can be written as follows:

$$\tilde{\mathbf{V}}_N(\Lambda, T) = \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}\sum_{pq}\mathbf{v}_{pq}\{p^+q\}\mathrm{e}^{T}|\psi_0\rangle = \sum_{pq}\mathbf{v}_{pq}\langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}\{p^+q\}\mathrm{e}^{T}|\psi_0\rangle = \sum_{pq}\mathbf{v}_{pq}\Gamma_{pq} \tag{2.72}$$

where we have used that $\mathbf{V}_N = \sum_{pq}\mathbf{v}_{pq}\{p^+q\}$ (second-quantized form of a normal ordered operator), with $\mathbf{v}_{pq}$ the components of the solute potential in the MO basis. The matrix Γ is the CCSD relaxed one-electron density matrix. Then, the contribution to the equations for the T amplitudes comes from the derivative of $\tilde{\mathbf{V}}_N(\Lambda, T)$ with respect to the Λ amplitudes ($\mathbf{Q}_0$ does not depend on the Lagrange multipliers). In this context, the Hamiltonian H_{0N} contains a term that depends on the elements of the Fock matrix ($\sum_{pq} f_{pq}\{p^+q\}$) and that has the same functional form as $\mathbf{V}_N$. As the Fock matrix is updated in the reference calculation with a C-PCM term that reads as (in AO basis) $F_{\mu\nu}^{\mathrm{CPCM}} = \mathbf{Q}_0 \cdot \mathbf{V}_{\mu\nu}$, then the term $\mathbf{Q}_0 \cdot \tilde{\mathbf{V}}_N(\Lambda, T)$ is added implicitly to eq (2.71).

Once the T amplitudes are obtained, the total energy, E, is calculated as

$$E = E_0 + \langle\psi_0|\mathrm{e}^{-T}(H_{0N} + \mathbf{Q}_0 \cdot \mathbf{V}_N)\mathrm{e}^{T}|\psi_0\rangle = E_0 + \sum_{ia} F_{ia}t_a^i + \frac{1}{4}\sum_{ijab}\langle ij||ab\rangle\left(t_{ab}^{ij} + 2t_a^i t_b^j\right) \tag{2.73}$$

Then, the C-PCM contribution to the CC energy within the PTE scheme occurs through the term $\frac{1}{2}\mathbf{Q}_0 \cdot \mathbf{V}_0$ in E_0 and implicitly through the Fock matrix elements F_{ia} ($F_{ia} = F_{ia}^0 + \mathbf{Q}_0 \cdot \mathbf{v}_{ia}$).

The PTE scheme corresponds to `CPCMccm 0`, and is implemented for canonical-CCSD (RHF and UHF) and DLPNO-CCSD (RHF and UHF). For instance, for a DLPNO-CCSD calculation (closed-shell reference) of a system solvated by water using the PTE scheme, the input file looks like:

```
! DLPNO-CCSD cc-pVTZ cc-PVTZ/C TightSCF CPCM(Water)

%cpcm
  CPCMccm 0
end

* xyzfile 0 1 water.xyz
```

PTE(S) scheme

In this scheme (where the "S" stands for singles), the last term in eq (2.70) depends on the T amplitudes, but not on the Λ amplitudes,

$$\mathcal{L}(\Lambda, T) = E_0 + \langle\psi_0|(1+\Lambda)\mathrm{e}^{-T}H_{0N}\mathrm{e}^{T}|\psi_0\rangle + \mathbf{Q}_0 \cdot \tilde{\mathbf{V}}_N(\Lambda, T) + \frac{1}{2}\tilde{\mathbf{Q}}_N(T) \cdot \tilde{\mathbf{V}}_N(T) \tag{2.74}$$

Again, in the same way as for the PTE scheme, the C-PCM contribution to the equations for the T amplitudes comes from the term $\mathbf{Q}_0 \cdot \tilde{\mathbf{V}}_N(\Lambda, T)$ in eq (2.74), which is implictly added to the Fock matrix elements in the MO basis. The last term in eq (2.74) does not depend on the Λ amplitudes and then does not contribute to the equations for the T amplitudes. However, this term depends on the T amplitudes through the elements γ_{ai} of the density matrix Γ,

$$\gamma_{ai} = t_a^i + \sum_{me}\left(t_{ae}^{im} - t_e^i t_a^m\right)\lambda_m^e - \frac{1}{2}\sum_{mnef}\lambda_{mn}^{ef}\left(t_{ef}^{in}t_a^m + t_{af}^{mn}t_e^i\right) \tag{2.75}$$

and then it contributes to the final energy E in the following way:

$$\frac{1}{2}\tilde{\mathbf{Q}}_N(T) \cdot \tilde{\mathbf{V}}_N(T) = \frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right) \cdot \left(\sum_{ai} t_a^i \mathbf{v}_{ai}\right) \tag{2.76}$$

That gives the final equation for the total energy of our system,

$$\begin{aligned} E &= E_0 + \langle\psi_0|\mathrm{e}^{-T}(H_{0N} + \mathbf{Q}_0 \cdot \mathbf{V}_N)\mathrm{e}^{T}|\psi_0\rangle + \frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right)\cdot\left(\sum_{ai} t_a^i \mathbf{v}_{ai}\right) = \\ &= E_0 + \sum_{ia} F_{ia}t_a^i + \frac{1}{4}\sum_{ijab}\langle ij||ab\rangle\left(t_{ab}^{ij} + 2t_a^i t_b^j\right) + \frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right)\cdot\left(\sum_{ai} t_a^i \mathbf{v}_{ai}\right) \end{aligned} \tag{2.77}$$

Therefore, the CC energy for a solvated system within the PTE(S) scheme involves three C-PCM contributions: (1) the term $\frac{1}{2}\mathbf{Q}_0 \cdot \mathbf{V}_0$ included in E_0, (2) the term $\sum_{ia} \mathbf{Q}_0 \cdot \mathbf{v}_{ia} t_a^i$ that occurs implicitly through $\sum_{ia} F_{ia} t_a^i$ and (3) the term $\frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right) \cdot \left(\sum_{ai} t_a^i \mathbf{v}_{ai}\right)$.

This scheme is available in ORCA for canonical-CCSD (RHF and UHF) and DLPNO-CCSD (RHF and UHF). In order to use it, the user needs to add the tag `CPCMccm 1` in the `%cpcm` block.

```
! DLPNO-CCSD cc-pVTZ cc-PVTZ/C TightSCF CPCM(Water)

%cpcm
  CPCMccm 1
end

* xyzfile 0 1 water.xyz
```

PTES scheme

In this scheme, both $\tilde{\mathbf{Q}}_N$ and $\tilde{\mathbf{V}}_N$ in the last term in eq (2.70) depend on the T amplitudes but just one of them depends on the Λ amplitudes,

$$\mathcal{L}(\Lambda, T) = E_0 + \langle\psi_0|(1+\Lambda)\mathrm{e}^{T} H_{0N}\mathrm{e}^{T}|\psi_0\rangle + \mathbf{Q}_0 \cdot \tilde{\mathbf{V}}_N(\Lambda, T) + \frac{1}{2}\tilde{\mathbf{Q}}_N(T)\cdot\tilde{\mathbf{V}}_N(\Lambda, T) \tag{2.78}$$

The C-PCM terms enter these equations on the one hand, implicitly, through the elements of the Fock matrix (like for the PTE and PTE(S) schemes), and on the other hand, explicitly through the derivatives of $\frac{1}{2}\tilde{\mathbf{Q}}_N(T)\cdot\tilde{\mathbf{V}}_N(\Lambda, T)$ with respect to Λ. If we call $\mathcal{L}_{\text{CPCM}}$ the last C-PCM term in eq (2.78), then the contribution from this term to the T amplitudes equations read as:

$$\frac{\partial\mathcal{L}_{\text{CPCM}}}{\partial\lambda_i^a} = \frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right)\cdot\left[-\sum_j t_a^j \mathbf{v}_{ji} + \sum_b t_b^i \mathbf{v}_{ab} + \mathbf{v}_{ia} + \sum_{bj}\left(t_{ba}^{ji} - t_a^j t_b^i\right)\mathbf{v}_{bj}\right] \tag{2.79}$$

$$\frac{\partial\mathcal{L}_{\text{CPCM}}}{\partial\lambda_{ij}^{ab}} = \frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right)\cdot\left[-\frac{1}{2}\sum_k t_{ab}^{kj}\mathbf{v}_{ki} + \frac{1}{2}\sum_c t_{ac}^{ij}\mathbf{v}_{bc} - \frac{1}{2}\sum_{ck}\left(t_{ab}^{kj}t_c^i + t_{cb}^{ij}t_a^k\right)\mathbf{v}_{ck}\right] \tag{2.80}$$

The contribution to the energy is the same as that for the PTE(S) scheme, but with different values for the T amplitudes (as the equations to calculate them differ slightly from those for the PTE(S) scheme).

$$\begin{aligned} E &= E_0 + \langle\psi_0|\mathrm{e}^{-T}(H_{0N} + \mathbf{Q}_0 \cdot \mathbf{V}_N)\mathrm{e}^{T}|\psi_0\rangle + \frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right)\cdot\left(\sum_{ai} t_a^i \mathbf{v}_{ai}\right) = \\ &= E_0 + \sum_{ia} F_{ia}t_a^i + \frac{1}{4}\sum_{ijab}\langle ij||ab\rangle\left(t_{ab}^{ij} + 2t_a^i t_b^j\right) + \frac{1}{2}\left(\sum_{ai} t_a^i \mathbf{q}_{ai}\right)\cdot\left(\sum_{ai} t_a^i \mathbf{v}_{ai}\right) \end{aligned} \tag{2.81}$$

This scheme is the default CCSD/C-PCM approach in ORCA and is available in ORCA for canonical-CCSD (RHF and UHF) and DLPNO-CCSD (RHF and UHF). In this case, the tag `CPCMccm` in the `%cpcm` block is equal to 2. However, as the PTES scheme is the default in ORCA, the user just needs to add the information about the solvent in the input file, in order to use this approach.

```
! DLPNO-CCSD cc-pVTZ cc-PVTZ/C TightSCF CPCM(Water)

* xyz 0 1
O  -0.00000018976103    0.00606010894837    0.00000000004527
H   0.76098169249695   -0.58891312953082   -0.00000000000022
H  -0.76098151333900   -0.58891299029372   -0.00000000000022
*
```

Notes regarding the use of the CCSD/C-PCM schemes:

- For calculations within the PTE(S) and the PTES schemes, the explicit C-PCM contribution to the total energy (see eqs. (2.77) and (2.81)) is printed in the `COUPLED CLUSTER ENERGY` block after the equations for the "T" amplitudes converge. In this case, this energy is labelled as `C-PCM corr. term` and is already included in the correlation energy, `E(CORR)`. For the input example from above, this information looks like:

  ```
  ----------------------
  COUPLED CLUSTER ENERGY
  ----------------------

  E(0)                                      ...    -76.066903687
  E(CORR)(strong-pairs)                     ...     -0.267203798
  E(CORR)(weak-pairs)                       ...     -0.000106106
  E(CORR)(corrected)                        ...     -0.267309904
  C-PCM corr. term (included in E(CORR))    ...     -0.000003158
  E(TOT)                                    ...    -76.334213591
  Singles Norm <S|S>**1/2                   ...      0.017784967
  T1 diagnostic                             ...      0.006287935
  ```

  This contribution does not represent the whole C-PCM contribution to the correlation energy, as this one also occurs, implicitly, through the "T" amplitudes.

- The C-PCM contribution to the Λ equations is implemented in ORCA for the PTE(S) and PTES schemes. Then, the user can request unrelaxed densities.

2.13.8 Complete Keyword List for the `%cpcm` Block

The available parameters/options that the user can request via the `%cpcm` block are:

```
%cpcm
  epsilon             80.0    # Dielectric constant
  refrac               1.0    # Refractive index
  rsolv                1.3    # Solvent probe radius in Angstroem (for SES-type␣
↪cavities)
  rmin                 0.5    # Minimal GEPOL sphere radius in Angstroem
  pmin                 0.1    # Minimal distance between two surface points in␣
↪Angstroem
  fepstype            cpcm    # Epsilon function type: cpcm, cosmo
  xfeps                0.0    # X parameter for the feps scaling function
  surfacetype vdw_gaussian    # Cavity surface: gepol_ses, gepol_sas
                                                vdw_gaussian, gepol_ses_gaussian
  ndiv                   5    # Maximum depth for recursive sphere generation
  num_leb              302    # Lebedev points for the Gaussian charge scheme
  radius[N]            1.3    # Atomic radius for atomic number N in Angstroem
  AtomRadii(N,1.4)            # Atomic radius for the Nth atom in Angstroem
  scale_gauss          1.2    # Scaling factor for the atomic radii in the
                                Gaussian charge scheme
  cut_area             0.0    # Cutoff for the area of a surface segment in a.u.
                                Only valid for the Gaussian charge scheme
  cut_swf             1e-7    # Cutoff for the switching function
                                Only valid for the Gaussian charge scheme
  thresh_h             5.0    # Threshold for the charge density on a hydrogen
```

(continues on next page)

(continued from previous page)

```
                                 atom in charges/Å^2 (isodensity scheme)
  thresh_noth           5.0    # Threshold for the charge density on non-hydrogen
                                 atoms in charges/Å^2 (isodensity scheme)
  solvent           "water"    # Solvent name (for C-PCM)
  smd                  true    # Turn on SMD
  smdsolvent      "ethanol"    # Solvent name for SMD
  soln               1.3611    # Index of refraction at optical frequencies at 293 K
  soln25             1.3593    # Index of refraction at optical frequencies at 298 K
  sola                 0.37    # Abraham's hydrogen bond acidity
  solb                 0.48    # Abraham's hydrogen bond basicity
  solg                31.62    # Relative macroscopic surface tension
  solc                 0.00    # Aromaticity, fraction of non-hydrogenic solvent␣
↪atoms
                               # that are aromatic carbon atoms
  solh                 0.00    # Electronegative halogenicity, fraction of non-
↪hydrogenic
                               # solvent atoms that are F, Cl, or Br
  SMD18               false    # Turn on SMD18
  CPCMccm                 0    # Coupled-cluster/C-PCM scheme (default = 2)
  cds_cpcm                0    # Use of the GVDW_nel or GSES_nel scheme (default =␣
↪0)
  draco                true    # Turn on DRACO (default = false)
  dracoisodens        false    # Use isodensity scheme for DRACO (default = false)
  draco_charges         ceh    # Charges used within DRACO (default = eeq)
  vopt           v_analytic    # Method to calculate the ESP at the position of the␣
↪charges:
                               #    v_analytic (default): via the analytic ESP␣
↪integrals
                               #    v_numeric: via numerical integration
                               #    v_ri: via the RI-density
                               #    v_multipole: via BUPO
 fopt            f_analytic    # Method to calculate the C-PCM contribution to the␣
↪Fock
                               # or Kohn-Sham matrix:
                               #    f_analytic (default): via the ESP integrals
                               #    f_numeric: via numerical integration
                               #    f_ri: via the RI-density
                               #    f_multipole: via BUPO
end
```

2.14 Integral Handling

As the number of nonzero integrals grows very rapidly and reaches easily hundreds of millions even with medium sized basis sets in medium sized molecules, storage of all integrals is not generally feasible. This desperate situation prevented SCF calculations on larger molecules for quite some time, so that Almlöf [169, 170, 171] made the insightful suggestion to repeat the integral calculation, which was already the dominant step, in *every SCF cycle* to solve the storage problem. Naively, one would think that this raises the effort for the calculation to $n_{\text{iter}}t_{\text{integrals}}$ (where n_{iter} is the number of iterations and $t_{\text{integrals}}$ is the time needed to generate the nonzero integrals). However, this is not the case because only the change in the Fock matrix is required from one iteration to the next, but not the Fock matrix itself. As the calculations starts to converge, more and more integrals can be skipped. The integral calculation time will still dominate the calculation quite strongly, so that ways to reduce this burden are clearly called for. As integrals are calculated in *batches*[1] the cost of evaluating the given batch of shells p, q, r, s may be estimated as:

$$\text{cost} \approx n_p n_q n_r n_s \left(2l_p + 1\right)\left(2l_q + 1\right)\left(2l_r + 1\right)\left(2l_s + 1\right) \tag{2.82}$$

[1] A *batch* is a set of integrals that arises from all components of the shells involved in the integral. For example a $\langle pp|pp\rangle$ batch gives rise to $3 \times 3 \times 3 \times 3 = 81$ integrals due to all possible combinations of p_x, p_y and p_z functions in the four shells. Computations based on batches lead to great computational advantages because the 81 integrals involved in the $\langle pp|pp\rangle$ batch share many common intermediate quantities.

Here, n_p is the number of primitives involved in shell p, and l_p is the angular momentum for this shell. *Large* integrals are also good candidates for storage, because small changes in the density that multiply large integrals are likely to give a nonzero contribution to the changes in the Fock matrix.

ORCA thus features two possibilities for integral handling, which are controlled by the variable `SCFMode`. In the mode `Conventional`, all integrals above a given threshold are stored on disk (in a compressed format that saves much disk space). In the mode `Direct`, all two-electron integrals are recomputed in each iteration.

Two further variables are of importance: In the `Conventional` mode the program may write enormous amounts of data to disk. To ensure this stays within bounds, the program first performs a so-called "statistics run" that gives a pessimistic estimate of how large the integral files will be. Oftentimes, the program will overestimate the amount of disk space required by a factor of two or more. The maximum amount of disk space that is allowed for the integral files is given by `MaxDisk` (in Megabytes).

On the other hand, if the integral files in `Conventional` run are small enough to fit into the central memory, it is faster to do this since it avoids I/O bottlenecks. The maximum amount of memory allocated for integrals in this way is specified by `MaxIntMem` (in Megabytes). If the integral files are larger than `MaxIntMem`, no integrals will be read into memory.

```
%scf
  MaxIter   100    # Max. no. of SCF iterations
  SCFMode Direct   # default, other choice: Conventional
  Thresh  1e-8     # Threshold for neglecting integrals / Fock matrix contributions
                   # Depends on the chosen convergence tolerance (in Eh).
  TCut    1e-10    # Threshold for neglecting primitive batches. If the prefactor
                   # in the integral is smaller than TCut, the contribution of the
                   # primitive batch to the total batch is neglected.
  DirectResetFreq 20   # default: 15
  MaxDisk  2500        # Max. amount of disk for 2 el. ints. (MB)
  MaxIntMem 400        # Max. amount of RAM for 2 el. ints. (MB)
end
```

The value of `DirectResetFreq` sets the number of incremental Fock matrix builds after which the program should perform a full Fock matrix build in a `Direct` SCF calculation. To prevent numerical instabilities that arise from accumulated errors in the recursively build Fock matrix, the value should not be too large, since this will adversely affect the SCF convergence. If the value is too small, the program will update more frequently, but the calculation will take considerably longer, since a full Fock matrix build is more expensive than a recursive one.

The thresholds `TCut` and `Thresh` also deserve a closer explanation. `Thresh` is a threshold that determines when to neglect two-electron integrals. If a given integral is smaller than `Thresh` Eh, it will not be stored or used in Fock matrix construction. Additionally, contributions to the Fock matrix that are smaller than `Thresh` Eh will not be calculated in a `Direct` SCF.

Clearly, it would be wasteful to calculate an integral, then find out it is good for nothing and thus discard it. A useful feature would be an efficient way to estimate the size of the integral *before it is even calculated*, or even have an estimate that is a *rigorous upper bound* on the value of the integral. Häser and Ahlrichs [172] were the first to recognize that such an upper bound is actually rather easy to calculate. They showed that:

$$\left|\left\langle ij\left|kl\right.\right\rangle\right| \leqslant \sqrt{\left\langle ij\left|ij\right.\right\rangle}\sqrt{\left\langle kl\left|kl\right.\right\rangle} \tag{2.83}$$

where:

$$\left\langle ij\left|kl\right.\right\rangle = \int\int \phi_i\left(\vec{r}_1\right)\phi_j\left(\vec{r}_1\right)r_{12}^{-1}\phi_k\left(\vec{r}_2\right)\phi_l\left(\vec{r}_2\right)d\vec{r}_1 d\vec{r}_2 \tag{2.84}$$

Thus, in order to compute an upper bound for the integral only the right hand side of this equation must be known. This involves only two index quantities, namely the matrix of two center exchange integrals $\left\langle ij\left|ij\right.\right\rangle$. These integrals are easy and quick to calculate and they are all $\geqslant$0 so that there is no trouble with the square root. Thus, one has a powerful device to avoid computation of small integrals. In an actual calculation, the Schwartz prescreening is not used on the level of individual basis functions but on the level of shell batches because integrals are always calculated in batches. To realize this, the largest exchange integral of a given exchange integral block is looked for and its square root is stored in the so called *pre-screening* matrix **K** (that is stored on disk in ORCA). In a `Direct` SCF this matrix is not recalculated in every cycle, but simply read from disk whenever it is needed. The matrix of exchange integrals on the level of individual basis function is used in `Conventional` calculations to estimate the disk requirements (the "statistics" run).

Once it has been determined that a given integral batch survives it may be calculated as:

$$\langle ij\,|kl\,\rangle = \sum_p d_{pi} \sum_q d_{qj} \sum_r d_{kr} \sum_s d_{sl} \,\langle i_p j_q\,|k_r l_s\,\rangle \tag{2.85}$$

where the sums p, q, r, s run over the primitive Gaussians in each basis function i, j, k, l and the d's are the contraction coefficients. There are more powerful algorithms than this one and they are also used in ORCA. However, if many terms in the sum can be skipped and the total angular momentum is low, it is still worthwhile to compute contracted integrals in this straightforward way. In equation (2.85), each primitive integral batch $\langle i_p j_q\,|k_r l_s\,\rangle$ contains a prefactor $I_{Gaussians}$ that depends on the position of the four Gaussians and their orbital exponents. Since a contracted Gaussian usually has orbital exponents over a rather wide range, it is clear that many of these primitive integral batches will contribute negligibly to the final integral values. In order to reduce the overhead, the parameter `TCut` is introduced. If the common prefactor I_{pqrs} is smaller than `TCut`, the primitive integral batch is skipped. However, I_{pqrs} is *not* a rigorous upper bound to the true value of the primitive integral. Thus, one has to be more conservative with `TCut` than with `Thresh`. In practice it appears that choosing `TCut=0.01*Thresh` provides sufficient accuracy, but the user is encouraged to determine the influence of `TCut` if it is suspected that the accuracy reached in the integrals is not sufficient.

Hint

- If the direct SCF calculation is close to convergence but fails to finally converge, this maybe related to a numerical problem with the Fock matrix update procedure – the accumulated numerical noise from the update procedure prevents sharp convergence. In this case, set `Thresh` and `TCut` lower and/or let the calculation more frequently reset the Fock matrix (`DirectResetFreq`).

Note

- For a `Direct` calculation, there is no way to have `Thresh` larger than `TolE`. If the errors in the Fock matrix are larger than the requested convergence of the energy, the change in energy can never reach `TolE`. The program checks for that.
- The actual disk space used for *all* temporary files may easily be larger than `MaxDisk`. `MaxDisk` only pertains to the two-electron integral files. Other disk requirements are not currently checked by the program and appear to be uncritical.

2.14.1 Compression and Storage

The data compression and storage options deserve some comment: in a number of modules including RI-MP2, MDCI, CIS, (D) correction to CIS, etc. the program uses so called "Matrix Containers". This means that the data to be processed is stored in terms of matrices in files and is accessed by a double label. A typical example is the exchange operator $\mathbf{K^{ij}}$ with matrix elements $K^{ij}(a,b) = (ia|jb)$. Here the indices i and j refer to occupied orbitals of the reference state and a and b are empty orbitals of the reference state. Data of this kind may become quite large (formally N^4 scaling). To store the numbers in single precision cuts down the memory requirements by a factor of two with (usually very) slight loss in precision. For larger systems one may also gain advantages by also compressing the data (e.g. use a "packed" storage format on disk). This option leads to additional packing/unpacking work and adds some overhead. The simple keywords to control this behavior are given in Table 2.58. For small molecules `UCDOUBLE` is probably the best option, while for larger molecules `UCFLOAT` or particularly `CFLOAT` may be the best choice. Compression does not necessarily slow the calculation down for larger systems since the total I/O load may be substantially reduced and thus (since CPU is much faster than disk) the work of packing and unpacking takes less time than to read much larger files (the packing may reduce disk requirements for larger systems by approximately a factor of 4 but it has not been extensively tested so far). There are many factors contributing to the overall wall clock time in such cases including the total system load. It may thus require some experimentation to find out with which set of options the program runs fastest with.

Caution

- It is possible that `FLOAT` may lead to unacceptable errors. Thus it is not the recommended option when MP2 or RI-MP2 gradients or relaxed densities are computed. For this reason the default is DOUBLE.
- If you have convinced yourself that `FLOAT` is OK, it may save you a factor of two in both storage and CPU.

2.14.2 Distance Dependent Pre-Screening

The "Direct" SCF procedure consists of re-calculating the electron-electron repulsion integrals in every SCF iteration. Avoiding to store these integrals is what made SCF calculations on large molecules feasible and therefore is a cornerstone of modern quantum chemistry. The recalculation is only feasible in reasonable turnaround times of the calculation of negligibly small integrals is avoided before calculation. Thus, there needs to be a reasonably cheap to compute estimate for these integrals that programs rely on for avoiding the actual integral calculation. This is the important subject of pre-screening. Pre-screening reduces the number of integrals to be calculated from O(N^4) (exact integrals) or O(N^3) (RI integrals) to O(N^2) because in a large molecule, there are only O(N) significant charge distributions. Of course, it is beneficial if the integral estimate provides a rigorous upper bound to the actual integral because in this case it is guaranteed that no contribution to the Fock or Kohn-Sham matrix that exceeds a pre-defined accuracy threshold is being missed.

The by far most popular integral estimate is the Schwartz estimator introduced by Häser and Ahlrichs:

$$|(\mu\nu|\kappa\tau)|_{Schwartz} \leq \sqrt{(\mu\nu|\mu\nu)}\sqrt{(\kappa\tau|\kappa\tau)}$$

Thus, all that is required in order to compute this estimate is the pre-screening matrix

$$Q_{\mu\nu} = \sqrt{(\mu\nu|\mu\nu)}$$

Which is cheap and not memory consuming. We note in passing that, since integrals are calculated in batches involving four shells, the pre-screening matrix is also shell contracted, meaning the maximum element over the members of both shells is taken as actual pre-screening element. The extension to RI integrals is trivial and will therefore not be written our here.

The Schwartz estimate is used in practically every quantum chemistry program. Despite it's success, it also has shortcomings. These shortcomings are best understood by considering that a given charge distribution $\mu\nu$ has a multipole structure. At sufficient distance, the interaction of the distributions $\mu\nu$ and $\kappa\tau$ therefore have a distance dependence that depends on the multipolar structure of the two distributions. For example, the monopole-monopole interaction goes as R-1 (R being the distance between the centers of the two distributions), dipole-dipole interaction fall off as R^{-3} etc. Evidently, there is no distance dependence in the Schwartz estimate and consequently, it will have a tendency to strongly overestimate integrals that involve non-negligible but distant charge distributions.

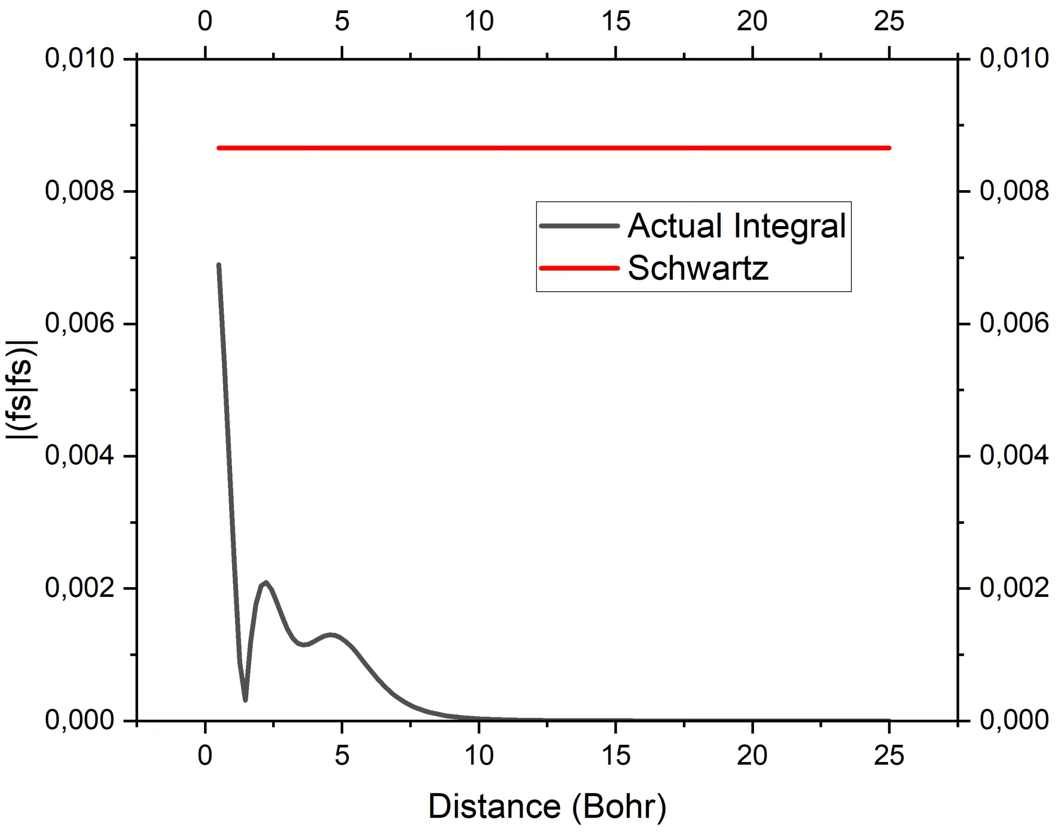


Fig. 2.2: Demonstration how the Schwartz-estimate can drastically overestimate integrals. The red trace Is the Schwartz-estimate while the black trace is the actual interaction of two one-center fs charge distributions as a function of distance.

Historically, a different, related estimator has been popular in the early days of direct SCF. This is the overlap estimator (OVLR-estimate). Consider the two-electron integral:

$$(\mu\nu|\kappa\tau) = \int\int \frac{\mu(\mathbf{r}_1)\nu(\mathbf{r}_1)\kappa(\mathbf{r}_2)\tau(\mathbf{r}_2)}{|\mathbf{r}_1 - \mathbf{r}_2|} d\mathbf{r}_1 d\mathbf{r}_2$$

And pretend that we can replace the inverse electronic distance by an effective distance between the two charge distributions:

$$|\mathbf{r}_1 - \mathbf{r}_2| \rightarrow R_{eff}$$

In this case, the integral simply becomes:

$$(\mu\nu|\kappa\tau) = \frac{S_{\mu\nu}S_{\kappa\tau}}{R_{eff}}$$

With S being the overlap integral. The estimator is then obtained by taking the maximum absolute element of the overlap matrix elements over the members of the involved shells.

$$|(\mu\nu|\kappa\tau)|_{OVLR} = \frac{\widetilde{S}_{\mu\nu}\widetilde{S}_{\kappa\tau}}{R_{eff}}$$

The effective distance is:

$$R_{eff} = R_{\mu\nu,\kappa\tau} - ext_{\mu\nu} - ext_{\mu\nu,\kappa\tau}$$

The estimator is applied if

$$R_{eff} > 1$$

And defaults back to the Schwartz estimate otherwise. In this equation $R_{\mu\nu,\kappa\tau} = |R_{\mu\nu} - R_{\kappa\tau}|$ is the distance between the centers of the two charge distributions ($R_{\mu\nu} = \langle\mu \vee r \vee \nu\rangle$) and $ext_{\mu\nu}$ is the extent of a charge distribution which is discussed in detail in the section on multipole approximations. In a nutshell $ext_{\mu\nu}$ defines a radius outside of which the product $\mu(r)\nu(r)$ is zero. Thus, the condition $R_{eff} > 1$ is the multipole permissible criterion. If it is met, the charge distributions are non-overlapping and the multipole estimate is convergent.

Based on this realization, Ochsenfeld and co-workers have developed a series of integral estimators that are designed to take this distance dependence into account. In their initial work, Lambrecht and Ochsenfeld provided a concise analysis of the multipolar structure of the charge distribution. After some experimentation, they settled on proposing **the QQR-estimator** which is a good compromise in terms of computational cost and efficiency in eliminating small contributions:

$$|(\mu\nu|\kappa\tau)|_{QQR} = \frac{Q_{\mu\nu}Q_{\kappa\tau}}{R_{eff}}$$

Which is applied if $R_{eff} > 1$, as above. This estimate is not completely rigorous, but still very safe because it assumes monopole moments on the two charge distributions. The Schwartz-integral instead of the overlap integral is assumed to mimic the multipole structure of the charge distributions.

Since experience indicates that the QQR estimator is not eliminating significantly more integrals than the Schwartz-estimate, Thomson and Ochenseld proposed a refined estimator that they refer to as "CSAM". It is given by:

$$|(\mu\nu|\kappa\tau)|_{CSAM} = Q_{\mu\nu}Q_{\kappa\tau}\,\text{max}\left(T_{\mu\kappa}T_{\nu\tau}, T_{\mu\tau}T_{\nu\kappa}\right)$$

Where the T-matrix elements are shell maxima of the two-center Coulomb pre-screening integrals:

$$T_{\mu\kappa} = \frac{\sqrt{(\mu\mu|\kappa\kappa)}}{\sqrt{Q_{\mu\mu}Q_{\kappa\kappa}}}$$

The T-factors bring in the distance dependence of the estimated integral.

Finally, there is a multipole estimator that takes the lowest multipole moment of each charge distribution into account.

$$|(\mu\nu|\kappa\tau)|_{multipole} = max\left(\frac{M^{\mu\nu}_{Lmin,m}M^{\kappa\tau}_{Lmin',m'}}{R^{Lmin+Lmin'+1}}\right)$$

Some idea about the tightness of the estimators can be seen from the following plot:

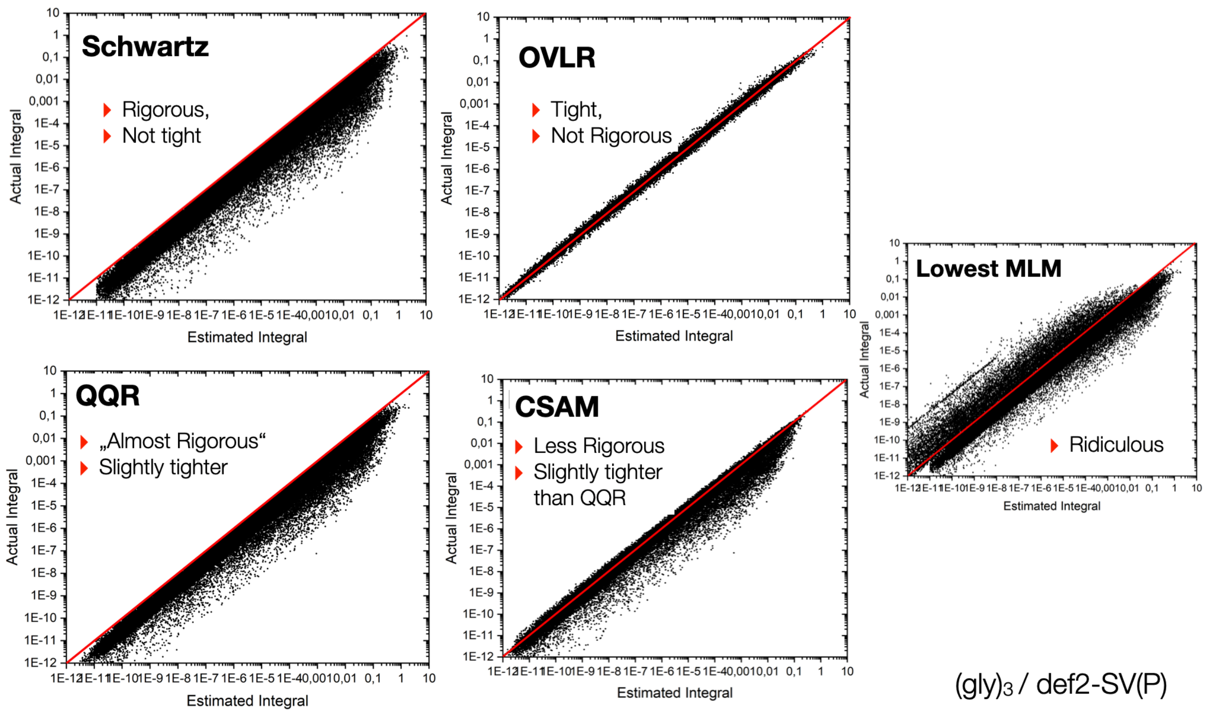
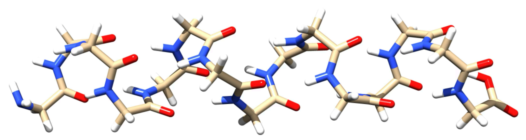
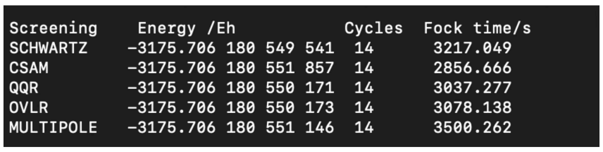


Fig. 2.3: Tightness of various integral estimators for the model system (glycine)$_3$ with the def2-SVP basis set.

One can see that the Schwartz-estimator is rigorous and the QQR estimator is nearly rigorous but not very much tighter. The OVLR estimator appears to be the by far tightest estimate but is scattering evenly above and below the correct value. CSAM leads to some underestimation of the integral values and appears to be slightly tighter than QQR. The lowest multipole estimate is ridiculous and of unusable quality.

Whether the revised estimators leads to actual computational savings can be seen in a model calculation on (glycine)$_{15}$/def2-SVP :





```
Screening    Energy /Eh            Cycles  Fock time/s
SCHWARTZ    -3175.706 180 549 541  14       3217.049
CSAM        -3175.706 180 551 857  14       2856.666
QQR         -3175.706 180 550 171  14       3037.277
OVLR        -3175.706 180 550 173  14       3078.138
MULTIPOLE   -3175.706 180 551 146  14       3500.262
```

Fig. 2.4: actual computation times and total Hartree-Fock energies for (glycine)$_{15}$ under the influence of different integral estimators.

The results show that only CSAM leads to savings on the order of 10%, although at the loss of some accuracy which in this example amounts to about 0.01 micro-Eh. The origin of the limited savings are certainly related to the fact that the lowest multipole of a two-center charge distribution is equal to the overlap integral between the two distributions. Thus, as soon as two basis functions have an overlap, they will have a monopole moment and the decay of the interaction between two such charge distributions is only R^-1^ which falls off so slowly that it is not of practical relevance in the elimination of small integrals.

The following input triggers the different estimates:

```
%shark Prescreening Schwartz
 OVLR
 QQR
 CSAM
 Multipole
End
```

Note

- In practice OVLR and Multipole are highly unstable and are not recommended
- The integral estimates are NOT yet available for RI integrals.

Relevant Papers:

1. Häser, Marco; Ahlrichs, Reinhart. Improvements on the direct SCF method. *Journal of Computational Chemistry*, **1989**, 10 (1), 104–111. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.540100111, DOI: 10.1002/jcc.540100111.
2. Lambrecht, Daniel S.; Ochsenfeld, Christian. Multipole-based integral estimates for the rigorous description of distance dependence in two-electron integrals. *The Journal of Chemical Physics*, **2005**, 123 (18), 184101. arXiv:https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/1.2079967/15373755/184101_1_online.pdf, DOI: 10.1063/1.2079967.
3. Thompson, Travis H.; Ochsenfeld, Christian. Distance-including rigorous upper bounds and tight estimates for two-electron integrals over long- and short-range operators. *The Journal of Chemical Physics*, **2017**, 147 (14), 144101. arXiv:https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/1.4994190/14837196/144101_1_online.pdf, DOI: 10.1063/1.4994190.

2.14.3 The "BubblePole" Approximation

The algorithms for Fock matrix construction in ORCA have been optimized to the point where the COSX approximation scales linearly with system size and so does the XC integration. Thus, asymptotically the quadratic scaling of the Coulomb term is dominating the Fock matrix construction time in very large systems. It is well-known that the introduction of hierarchical multipole approximations to the integrals can bring down the scaling of the Coulomb term to linear or near linear. This is the subject of the fast multipole method (FMM) approximation. The FMM approximation has been introduced in ORCA for the calculation of the point charge contribution to embedding calculations and significantly brings down the execution time.

While it may seem tempting to also introduce the FMM approximation in the quantum-quantum charge interactions, the ORCA developers have taken a different route. In FMM algorithms, real space is divided into hierarchical boxes followed by the calculation of the multipole interaction between boxes provided that a multipole allowedness criterion is met.

While it has been proven to work by several authors, it appeared to us that the boxing algorithm is not very natural to chemistry and leads to a number of problems that we intended to avoid in our alternative development.

The Bubblepole (BUPO) approximation is based on a different partitioning that is based on spheres ("Bubbles"). These bubbles fully enclose collections of "quantum objects". These quantum objects may be shell-pairs, auxiliary basis function shells or also point charges. The criterion for grouping a number of such objects together is spatial proximity. Since shell pairs and auxiliary shells are charge distributions, they have a spatial extent which must be taken into account in the group assembly algorithm. Taking the example of shell pairs, each surviving shell is assigned a shell pair center $R_{\mu\nu}$ and an extent $ext_{\mu\nu}$. The precise definition of these quantities is unique to ORCA and are fully discussed in the literature.

The BUPO algorithm then uses a variant of the Kmeans algorithm to group a predefined number of objects (e.g. 150) into each "bottom level" bubble. The bubble center is the arithmetic mean of all enclosed objects and the bubble radius is adjusted such that all objects are fully enclosed together with their extents. This ensures that there is no "leakage" of probability density outside of each bubble.

After setting up the bottom level bubbles, a bubble hierarchy is created in which multipoles are translated from the one bubble layer to the next. Super-bubbles contain lower-level bubbles (typically around three), until at the top layer, there only is a single bubble that encloses the entire molecule. This only happens one initially during system setup).

In the present ORCA implementation, the BUPO algorithm has been combined with ORCA's most efficient Coulomb-construction algorithm – the Split-RI-J algorithm. In the resulting RI-BUPO-J algorithm, the near-field is treated with the Split-RI-J method and the far-field by the hierarchical construction. The RI-J method generally contains three significant steps

"Projection" of the density on the auxiliary basis set:

$$g_K = \sum_{\mu\nu} P_{\mu\nu} \left(\mu\nu|K\right)$$

Solution of linear equations to get the aux-basis density:

$$P_K = \sum_{L} \left(V^{-1}\right)_{KL} g_L$$

Assembly of the Coulomb matrix:

$$J_{\mu\nu} = \sum_{K} P_K \left(\mu\nu|K\right)$$

The second step is computationally negligible and proceeds very efficienctly via the Cholesky decomposition. It is not linear scaling but about three to four orders of magnitude cheaper than the other steps such that for all treatable system sizes, it is insignificant. In the first step, the multipole approximation is applied to the density in the orbital basis and therefore the bubbles contain shell-pairs from the basis sets and multipoles derived from the molecular density expanded in the basis set. In the third step, the bubbles contain auxiliary basis shell pairs and multipoles derived from the aux-basis density. The subtleties that derive from this construction are discussed in detail in the original publication.

The numerical results indicate that RI-BUPO-J is asymptotically linear scaling with system size. On linear chains, the crossover with the Split-RI-J algorithm occurs at around 30-40 glycine units. In three dimensional systems, it will be significantly later. Hence, a real advantage of the RI-BUPO/J algorithm over Split-RI-J will only occur for very large systems. This is not due to shortcomings of the BUPO construction but is testimony of the exceptional efficiency of Split-RI-J.

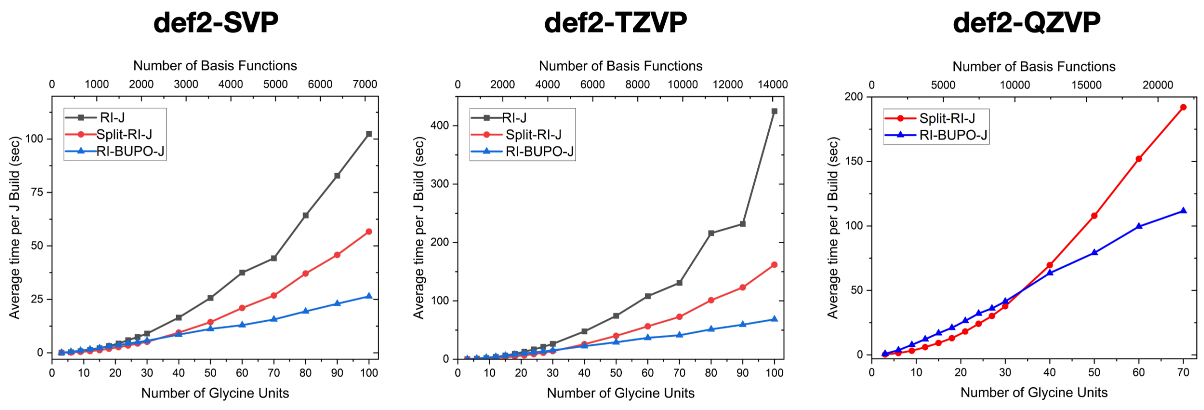


Fig. 2.5: Numerical results for the regular RI-J algorithm, Split-RI-J and RI-BUPO-J on linear glycine chains with basis sets of increasing quality.

There is a significant number of options for BUPO. However, we do NOT recommend to change the defaults. This is expert territory.

```
Simple Keyword line:

! RI-BUPO/J

Detailed parameter definitions:

%shark ExtentOpt Ext_SemiNumeric. # default: how to calculate the extents
Ext_analytic
Ext_Numeric
TSphere 1e-15      # shell pair extent cut-off
```

(continues on next page)

(continued from previous page)

```
MP_LAbsMax_BAS 32 # Max L allowed in BAS multipoles
MP_LAbsMin_BAS 0  # Min L allowed in BAS multipoles
MP_Lmax_BAS 10    # Bottom level expansion length
MP_Lincr_BAS 4    # Expansion length increase for
                  # subsequent bubble levels
MP_Rallow_BAS 1.0 # Far field met if effective bubble
                  # distance larger than this (Bohrs)
MP_TScreen_BAS 1e-10  # Multipole elimination threshold
MP_ClusterDim_BAS 150 # Number of objects bottom level bubble
MP_ClusterDim2_BAS 3  # Number of bubbles in super-bubble
                      # in higher level of the hierarchy
MP_NLevels_BAS 15     # Max number of bubble levels in the
                      # hierarchy
# these parameters apply to the first step of RI-BUPO-J
# the same variables exist w.r.t. the auxbasis using the
# postfix \_AUX and apply to the second step of RI-BUPO-J
End
```

Relevant Papers:

1. Colinet, Pauline; Neese, Frank; Helmich-Paris, Benjamin. Improving the Efficiency of Electrostatic Embedding Using the Fast Multipole Method. *J. Comput. Chem.*, **2025**, 46 (1), e27532. DOI: 10.1002/jcc.27532.
2. Neese, Frank; Colinet, Pauline; DeSouza, Bernardo; Helmich-Paris, Benjamin; Wennmohs, Frank; Becker, Ute. The “Bubblepole” (BUPO) Method for Linear-Scaling Coulomb Matrix Construction with or without Density Fitting. *J. Phys. Chem. A*, **2025**, 129 (10), 2618–2637. DOI: 10.1021/acs.jpca.4c07415.
3. White, Christopher A.; Johnson, Benny G.; Gill, Peter M.W.; Head-Gordon, Martin. The continuous fast multipole method. *Chemical Physics Letters*, **1994**, 230 (1), 8–16. DOI: 10.1016/0009-2614(94)01128-1.

2.14.4 Angular Momentum Limits

Energies and gradients in ORCA can now (starting from ORCA 6.1) be performed up to L=10 in the orbital and aux basis sets. Note that the nomenclature changed to the accepted spectroscopic notation:

L=0	1	2	3	4	5	6	7	8	9	10
S	P	D	F	G	H	I	K	L	M	N

Note

- no j-functions! The program will reject it.
- L is reserved for combined s,p shells for historical reasons. If you want to input L=8 functions, simply use the symbol “8” instead of “L”

Here is an example of how to use the angular momentum definition in an example of a user-specified basis set:

```
! RHF cc-pV6Z VeryTightSCF PrintBasis PModel
%basis

# Basis set for element : Ne
NewGTO Ne
S 11
  1 902400.0000000000 0.0000064569
  2 135100.0000000000 0.0000501790
  3 30750.0000000000 0.0002638325
  4 8710.0000000000 0.0011134543
```

(continues on next page)

(continued from previous page)

```
  5 2842.0000000000 0.0040396234
  6 1026.0000000000 0.0130374676
  7 400.1000000000 0.0377405416
  8 165.9000000000 0.0967943419
  9 72.2100000000 0.2108241400
 10 32.6600000000 0.3586501519
 11 15.2200000000 0.3985795167
S 11
  1 902400.0000000000 -0.0000045125
  2 135100.0000000000 -0.0000351554
  3 30750.0000000000 -0.0001851517
  4 8710.0000000000 -0.0007804845
  5 2842.0000000000 -0.0028452422
  6 1026.0000000000 -0.0092079116
  7 400.1000000000 -0.0271417038
  8 165.9000000000 -0.0715447701
  9 72.2100000000 -0.1678190251
 10 32.6600000000 -0.3267206864
 11 15.2200000000 -0.4994236511
S 1
  1 7.1490000000 1.0000000000
S 1
  1 2.9570000000 1.0000000000
S 1
  1 1.3350000000 1.0000000000
S 1
  1 0.5816000000 1.0000000000
S 1
  1 0.2463000000 1.0000000000
P 5
  1 815.6000000000 0.0014608751
  2 193.3000000000 0.0126013201
  3 62.6000000000 0.0668956161
  4 23.6100000000 0.2559896696
  5 9.7620000000 0.7470043852
P 1
  1 4.2810000000 1.0000000000
P 1
  1 1.9150000000 1.0000000000
P 1
  1 0.8476000000 1.0000000000
P 1
  1 0.3660000000 1.0000000000
P 1
  1 0.1510000000 1.0000000000
D 1
  1 13.3170000000 1.0000000000
D 1
  1 5.8030000000 1.0000000000
D 1
  1 2.5290000000 1.0000000000
D 1
  1 1.1020000000 1.0000000000
D 1
  1 0.4800000000 1.0000000000
F 1
  1 10.3560000000 1.0000000000
F 1
  1 4.5380000000 1.0000000000
F 1
  1 1.9890000000 1.0000000000
```

(continues on next page)

(continued from previous page)

```
F 1
  1 0.8710000000 1.0000000000
G 1
  1 8.3450000000 1.0000000000
G 1
  1 3.4170000000 1.0000000000
G 1
  1 1.3990000000 1.0000000000
H 1
  1 6.5190000000 1.0000000000
H 1
  1 2.4470000000 1.0000000000
I 1
  1 4.4890000000 1.0000000000
K 1
  1 3.0000000000 1.0000000000
# This would be L, but we cannot use the letter L
# because this is used for combined S-P shells
  8 1
  1 3.5000000000 1.0000000000
  M 1
1 2.0000000000 1.0000000000
  N 1
1 1.0000000000 1.0000000000
 end;
end

%shark FockFlag Force_SHARK
       end

* xyz 0 1
Ne 0 0 0
*
```

2.14.5 Keywords

Table 2.58: Simple input keywords related to integral handling

Keyword	Description
`Direct`	Selects an integral direct calculation
`Conv`	Selects an integral conventional calculation
`CheapInts`	Use the cheap integral feature in direct SCF calculations
`NoCheapInts`	Turn that feature off
`KeepInts`	Do not delete the integrals from disk after a calculation in conventional mode
`ReadInts`	Read the existing integrals from a previous calculation in conventional mode
`Float`	Set storage format for numbers to single precision (SCF, RI-MP2, CIS, CIS(D), MDCI)
`Double`	Set storage format for numbers to double precision (default)
`UCFloat`	Use float storage in the matrix containers without data compression
`CFloat`	Use float storage in the matrix containers with data compression
`UCDouble`	Use double storage in the matrix containers without data compression
`CDouble`	Use double storage in the matrix containers with data compression