# SAPT2020: An *Ab Initio* Program for Symmetry-Adapted Perturbation Theory Calculations of Intermolecular Interaction Energies. Sequential and parallel versions.

## User's Guide

Revision SAPT2020.1

**Robert Bukowski**, **Wojciech Cencek**, **Javier Garcia**, **Piotr Jankowski**, **Małgorzata Jeziorska**, **Bogumił Jeziorski**, **Stanisław A. Kucharski**, **Victor F. Lotrich**, **Michael P. Metz**, **Alston J. Misquitta**, **Robert Moszyński**, **Konrad Patkowski**, **Rafał Podeszwa**, **Fazle Rob**, **Stanisław Rybak**, **Krzysztof Szalewicz**, **Hayes L. Williams**, **Richard J. Wheatley**, **Paul E.S. Wormer**, **and Piotr S. Żuchowski**

Department of Physics and Astronomy,
University of Delaware, Newark, Delaware 19716

Department of Chemistry, University of Warsaw,
ul. Pasteura 1, 02-093 Warsaw, Poland

February 7, 2020

# Contents

## 1 Introduction

SAPT2020 is a computer code implementing Symmetry-Adapted Perturbation Theory (SAPT). SAPT is designed to calculate the interaction energy of a dimer, i.e., a system consisting of two arbitrary closed-shell or high-spin open-shell monomers. Calculations can also be performed for trimers. Each monomer can be an atom or a molecule. In SAPT, the interaction energy is expressed as a sum of perturbative corrections in the intermolecular interaction operator $V$, each correction resulting from a different physical effect. This decomposition of the interaction energy into distinct physical components is a unique feature of SAPT which distinguishes this method from the popular supermolecular approach. The SAPT methodology and its applications are discussed in several review papers [1–7] where complete references to the original developments can be found. Most of the formulas programmed in SAPT2020 are given in the paper published in the book accompanying the METECC collection of computer codes [8]. The METECC paper is available on the SAPT web page `http://www.physics.udel.edu/∼szalewic/SAPT/SAPT.html` and can be also found in the SAPT2020 distribution (`SAPT2020/doc/METECC.ps`). Note that the formulas for the SAPT corrections in the METECC paper contain several misprints—for an errata, see Ref. 9.

The METECC project [8] was the first distribution of the SAPT codes. The next version of SAPT, called SAPT96[1], was available since 1996. Compared to SAPT96, the next version, SAPT2002, was about a factor of two faster in medium size (about 200 functions) bases. It also allowed calculations with up to 1023 basis functions (SAPT96 was restricted to 255), was interfaced with a larger number of front-end SCF packages, and ran on a larger number of platforms. A parallel version of the SAPT suite [9], referred to as pSAPT2K2, became available in SAPT2002. This version runs on SGI Origin, IBM SP, and on Linux clusters and scales well up to about 32 processors. See Sec. 14 for a detailed description of this version. The SAPT2006 edition added a new, powerful version of SAPT based on the density-functional description of monomers and called SAPT(DFT) [10–14]. The density fitting implementation of SAPT(DFT) [15, 16], allowing calculations for dimers with nearly 100 atoms, was the main addition appearing in SAPT2008. The current version of the programs became available in 2016 and is denoted as SAPT2020. The list of changes relative to SAPT2002 is given in Sec. 2.

A version of SAPT has also been developed [17–19] which allows calculations of the nonadditive portion of the interaction energy for an arbitrary trimer consisting of closed-shell monomers. Thus, SAPT can now be used to calculate the two leading terms in the many-body expansion of the interaction energy of a cluster. The package SAPT3B containing the three-body SAPT is distributed

[1]During work on SAPT2002 a small error was detected in the `ccsdt` module of SAPT96 which slightly affected the correction $\epsilon^{(1)}$(CCSD), especially in calculations for small systems in large basis sets. In all tested cases the magnitude of this error was small enough not to change the published results.

optionally with SAPT2020. The package also includes a set of codes for three-body SAPT based on the Kohn-Sham description of the monomers [SAPT(DFT)] with or without density-fitting [20]. Unfortunately, no further description of the three-body codes is available, but users should be able to use these programs following the examples provided. For details of the methods, we refer to the original papers [17, 19, 20].

The two-body SAPT(DFT) can tackle high-spin open-shell systems [21]. This option is available since the SAPT2008 edition. However, density fitting has not been implemented in open-shell SAPT(DFT). No manual is available for open-shell SAPT(DFT), but it is quite similar from a user point of view to closed-shell SAPT(DFT) and therefore, with the examples provided in the distribution package, the use of these codes should not be problematic.

The calculation of the interaction energy of a dimer using SAPT2020 involves four steps. In the first step, one and two-electron integrals are computed in a chosen orbital basis set and then SCF calculations are performed on both monomers. The SCF calculation for the dimer can also be performed at this stage. Several integral/SCF packages are interfaced to SAPT2020 and can be used, including free packages such as GAMESS [22] (see `http://www.msg.ameslab.gov/GAMESS/GAMESS.html`) and ATMOL [23] (a modified version of the latter package, ATMOL1024, is included in the SAPT2020 distribution and can be downloaded from the SAPT web page). After the SCF calculations are completed, the “atomic” integrals (i.e., integrals between the functions of the basis set used) are transformed into molecular integrals using the 4-index transformation program `tran`. In the third step, the Coupled-Cluster (CC) program `ccsdt` is invoked to calculate the many-body perturbation theory (MBPT, known also as the MP method since it is based on the Møller-Plesset partition of the Hamiltonian) and/or CC amplitudes for monomers A and B. Finally, the `sapt.x` program is run to compute the interaction energy components. Several short interface programs are also invoked between the calls to `tran` and `ccsdt`. An alternative (to that implemented in the `sapt.x` module) way of calculating the electrostatic component of the interaction energy computes the Coulomb interaction of the monomer charge densities obtained at the relaxed CCSD level. If such a calculation is desired, additional two codes are run after `ccsdt`. These are `ccsdm`, producing the relaxed CCSD densities, and `e1dcbs`, performing the integration of these densities.

SAPT approach described above is a double perturbation expansion in powers of the operator $V$ and in powers of the intramonomer correlation operator $W = W_A + W_B$, the sum of the Møller-Plesset fluctuation operators for monomers A and B. The expansion in $V$ is truncated at the third-order terms. The expansion in $W$ is truncated at different orders depending on the particular component and the most important components (the electrostatic, first-order exchange, and dispersion energies) are selectively summed to infinity in $W$. Thus, one may call this approach SAPT(MP/CC). A complete SAPT(CC) approach at the CCSD (CC with single and double excitations) level was developed by Korona *et al.* [24–31] and is available in SAPT2020.

The computational cost of the SAPT corrections scales as a product of some powers of the number of occupied orbitals and of the virtual orbitals. Therefore, with a given basis set size, calculations for larger systems will take longer. At the present time (2012), the largest runs performed at the full theory level included about 500 (300) virtual orbitals for systems with monomers containing about 10 (20) occupied orbitals, whereas routine runs typically use basis sets of about 250 functions. Larger systems can be tackled by using pSAPT2K2 and a few dozen of processors or by neglecting some effects of intramonomer correlation, i.e., using the SAPT2 or SAPT0 levels defined later on. However, the recommended option is to use SAPT(DFT) [14] (see Sec. 16), which gives interaction energies similarly accurate as those given by the full SAPT at the costs close to those of SAPT0. Complete SAPT(DFT) calculations have been performed for dimers with nearly 100 atoms and the dispersion energies can be computed for dimers with about 300 atoms [32].

In the asymptotic region, i.e., for large intermonomer separations, SAPT calculations may be significantly simplified by means of the multipole expansion which allows to express the interaction energy as a series of inverse powers of the intermonomer separation. Coefficients of this series, the so-called van der Waals constants, depend only on monomer multipole moments and polarizabilities (static and dynamic) and can be computed using the POLCOR suite of codes by Wormer and Hettema [33, 34]. The POLCOR suite, as well as a fitting program developed in our group and utilizing the *ab initio* asymptotic information, comprise the independent package ASYMP_SAPT, distributed optionally with SAPT2020.

A version of the asymptotic codes based on DFT calculations is included with SAPT2020 and is described in Sec. 18

This document is intended to provide a basic introduction to the SAPT method and the instructions on how to download, compile, and run the SAPT2020 and pSAPT2K2 codes. Also included are some details on the types of computers, compilers, and integral plus Hartree-Fock self-consistent field (SCF) packages that SAPT has been tested with. Whereas extensive tests of SAPT have been performed, there may appear unforeseen difficulties with the installation and running of the codes. As SAPT2020 and pSAPT2K2 are products of a research project, no resources are available to provide support for users. The authors of the code will *try* to provide limited help within the restrictions of their schedules.

## 2 What's new since SAPT2002

### 2.1 New in version SAPT2020.1

- A new FORTRAN program, `SAPT-fastdf`, has been developed that serves as a platform for SAPT(DFT) calculations. It is interfaced with ORCA, and includes algorithms aimed at computing interaction energies of large systems. SAPT(DFT) with non-hybrid functionals is now possible for dimers containing hundreds of atoms. Users interested in these new algorithms should refer to sec. 17 for installation and usage instructions. Some of the features included in the `fastdf` program are:
  - SAPT(DFT) with $O(N^2)$ memory requirements and $O(N^3)$ disk space requirement when using non-hybrid functionals (such as PBE). This includes the computation of $E^{(2)}_{\text{exch-disp}}$(CKS) with no approximations other than the $S^2$ one.
  - Re-adapted the algorithms from SAPT2016.2 for SAPT(DFT) with hybrid functionals. These have $O(N^3)$ and $O(N^4)$ requirements for memory and disk space, respectively. Some efficiency has been gained by rewriting the 4 index transformation part and using the fully density-fitted algorithm for $E^{(2)}_{\text{exch-disp}}$(CKS).
  - Ability to compute $\delta E^{\text{HF}}_{\text{int,resp}}$ with the same requirements. This includes a new density-fitted algorithm for the computation of $E^{(20)}_{\text{ind,resp}}$ from the coupled Hartree-Fock equations.
  - Uses `libint` for Coulomb integrals. This combined with the modifications to the `kernel-gaussian` program allows for calculations with basis sets up to $5\zeta$.
- The `kernel-gaussian` program has been modified to compute kernel integrals involving gaussian orbitals with angular momentum up to 6 (from the 4 available in SAPT2016.2).
- Performed a small modification to keep compatibility with `gfortran v9.2.0`.

### 2.2 New in revision SAPT2016.2

- This is a bugfix release. No main features were added.
- Restored compatibility with the `pgf90` compiler for the parts of SAPT that were compatible.
- Slightly improved warnings and outputs for the ORCA interface.
- Fixed minor bugs in `SAPT`, `SAPTdf` and `SAPTdf.orca` scripts.

## 2.3 New in revision SAPT2016.1

- Compatibility with autoPES package (see Ref. [35])
- Compatibility with distributed multipole asymptotic package
- Inclusion of DFT-based center of mass multipole expansion codes for arbitrary molecules
- Added option to input memory limits in units other than single words and removed 32-bit limitation
- Added option to automatically allocate memory quantity
- Added OpenMP parallelization option of $E_{\mathrm{exch-ind}}$ and $E_{\mathrm{exch-disp}}$ corrections (experimental)
- Fixed issue with ORCA input method and modified input file format
- Fixed bug in compilation using QUADINV option
- Numerous other bug fixes

## 2.4 New in revision SAPT2012.2

- This is a bugfix release. No new features were added.
- Fixed `gfortran` compilation problem caused by the use of `entry` statements in integer∗8 functions.
- Fixed several `gfortran` runtime errors.
- Restored compatibility with Fortran 77 compilers (such as g77).

## 2.5 New in revision SAPT2012.1

- SAPT(CC): interaction energy contributions from relaxed monomer CCSD density matrices and their cumulants [24–31]. This functionality requires MOLPRO and the patch supplied here.
- Relaxed third-order induction correction $E^{(30)}_{\mathrm{ind,resp}}$ [36].
- Removed limitation of 1024 orbitals (current limitation is 65535). The option has to be compiled in (not enabled by default).

- Added code for calculating CKS exchange-dispersion (both non-DF and DF versions) with amplitudes obtained from CKS propagators in addition to version scaled from uncoupled versions. This correction, together with analogous exchange-induction corrections, is enabled by default when SAPT(DFT) calculations are requested. Old versions are retained for compatibility.
- ORCA interface for DF-SAPT(DFT).
- Gaussian09 interface.
- Gradient-regulated asymptotic correction (GRAC) [37] for the DALTON program.
- Optional support for `dimer.cnf` in Angstroms.
- Lowered memory requirements in DF-SAPT(DFT) transformation. Added automatic automatic calculation of required memory in DF-tran.
- Added trailing zeroes to the name of output of `Runlot`. Helps with sorting.
- Fixed timing routines with Intel Fortran.
- Renamed `sapt` program to `sapt.x` enabling correct use of SAPT2012 on filesystems that are not case-sensitive (e.g., Mac OS X).
- Fix for DF-SAPT(DFT) going into infinite loop in some cases when compiled with `gfortran`.
- Fixed DF-SAPT(DFT) code in some calculations with large intermonomer distances.
- Fixed SAPT(DFT) geometry code with some versions of `awk`.
- Fixed integer overflows for large-scale SAPT(DFT) calculations.

## 2.6 New in revision SAPT2008.2

- This is a bugfix release. No new features were added.
- Fixed crashing of very large (over 1000 shells in auxiliary basis sets) SAPT(DFT) jobs.
- Fixed crashing of DF-SAPT(DFT) for calculations with large intermonomer distances.
- Removed incorrect “integral unsupported for DF-TRAN” in density-fitting calculations for third-order induction calculations that prevented such calculations with DF-SAPT(DFT).
- Fixed a bug in the MOLPRO interface to prevent a crash when MOLPRO is compiled with `ifort`.

- Fixed a compilation failure for `TARGET=gfortran` and `GAMSI8='YES'`.
- Fixed a bug in the `e1dcbs` program that resulted in a crash of `ccsddSAPT` when `GAMSI8='YES'`.
- Fixed `SAPT` script to correctly handle energies printed by MOLPRO.
- Small fixes in the examples and scripts for running the examples.
- Attached new DALTON patch with small compilation fixes and a fix to correctly handle ECP DALTON runs.

## 2.7 New in revision SAPT2008.1

- Density-fitting SAPT(DFT) [16], optionally with quadruple precision electrostatic component.
- Three-body SAPT(DFT) with or without density fitting [20].
- SAPT(DFT) for open-shell high-spin complexes [21].
- Faster transformation algorithm for non-four-virtual integrals, reusing partially transformed integrals whenever possible. Optimal performance is achieved when twice as much memory as for the regular in-core transformation is available. When additional memory is not available, the new implementation is still faster than the one from SAPT2006.1, especially in dimer-centered basis sets.
- The four-virtual diagram in cc can now be evaluated in atomic orbitals, eliminating the need for a four-virtual transformation. The AO algorithm currently works only with ATMOL1024 and MOLPRO interfaces.
- Faster ($\sim o^2v^3$, not $\sim o^3v^3$) implementation of the ${}^tE^{(22)}_{\rm ind}$ correction, backported from parallel SAPT.
- Support for monomer-centered basis sets in the MOLPRO interface.
- The `C6H6_H2O_DCBS` example replaced by a similar, `C6H6_H2O_ADZM` one. In the old example, the basis set was nearly linearly dependent which caused problems with the coupled-cluster convergence.
- A new example showing how to run SAPT with effective core potentials (ECPs) using the MOLPRO interface.
- A new example showing how to calculate a relativistic SAPT interaction energy utilizing the second-order Douglas-Kroll-Hess Hamiltonian, with the relevant integrals generated by MOLPRO.

- CKS program can now allocate more than 2 Gwords of memory.
- Fixed a bug resulting in a crash of the `sapt.x` program in an (extremely unlikely) situation when no transformed integrals of a given type were larger than the threshold so that no integrals of this type were written to disk.
- Fixed a bug causing the `ccsdm` program to allocate too little memory in some cases.
- Fixed a bug in `misc/atmolstuff.F` resulting in a compile error under HPUX.
- Fixed a bug in `cc/tpdrvn.F` resulting in a compile error when setting `EXTRADEFS='-DTWOGIGAMAX'`.
- Fixed a bug in the MOLPRO interface which caused SAPT to produce nonsensical results when used with MOLPRO2006.1. Updated example input files for use with MOLPRO2006.1 and MOLPRO2008.1.
- Updated the GAMESS interface for use with the 2008 release of GAMESS.
- Corrected a POSIX-noncompliant use of the `tail` command in the `Compall` script that resulted in compile errors on some of the platforms.
- Replaced an advanced regular expression in the SAPT script by a simpler one which is compatible with `mawk`. The previous version was problematic for systems for which `mawk` was the default `awk` program.
- Added a preliminary version of the library with common utilities (`saptlib/`).

## 2.8 New in SAPT2006

- SAPT(DFT) [14] calculations available, with DFT quantities extracted from DALTON 2.0 [38].
- Third-order SAPT corrections: $E^{(30)}_{\rm ind}$, $E^{(30)}_{\rm exch-ind}$, $E^{(30)}_{\rm ind-disp}$, $E^{(30)}_{\rm exch-ind-disp}$, $E^{(30)}_{\rm disp}$, and $E^{(30)}_{\rm exch-disp}$ [39].
- Optional computation of $E^{(1)}_{\rm elst}$ from relaxed-monomer CCSD densities. If a pure monomer basis set is used, the densities may be precomputed and reused, after translation and rotation, to compute the whole surface of $E^{(1)}_{\rm elst}$(CCSD) using direct integrals.
- Optional calculation of the second-order dispersion energy using converged CCD amplitudes [40].
- The $E^{(10)}_{\rm exch}(S^2)$ correction reimplemented using a formula valid in both dimer-centered basis set (DCBS) and monomer-centered basis set (MCBS). The implementation of this correction in SAPT2002 was valid in DCBS only, and for MCBS a wrong result was printed. This correction does not enter the final SAPT interaction energy, as the formula of infinite order in $S$ is used.

- New, much faster algorithm for the calculation of $E^{(20)}_{\rm exch-disp}$—the most demanding correction at the SAPT0 or SAPT(DFT) level. An out-of-core algorithm utilizing much less memory is also available for this correction.

- New, faster and more memory-efficient CHF routines.

- Frozen core implemented for all standard SAPT corrections [41].

- New interfaces: DALTON 2.0, MOLPRO [42], and GAUSSIAN03 [43].

- New architectures: AMD64 (Opteron, Athlon64) and Intel EMT64 with g77 (32 and 64 bit) and pgf compilers (32 and 64 bit), as well as HPUX (Itanium).

- DIIS [44] in coupled-cluster calculations. Turned on by default if compiled in.

- More efficient algorithms for the four-virtual diagram in CC calculations.

- Both energy-based and amplitude-based convergence criteria for CC available.

- Smarter scripts for running SAPT.

- Fixed a memory allocation bug on x86 Linux machines with kernel 2.6.x using g77.

- Several small bug fixes and enhancements.

Note that turning DIIS on/off, as well as changing the CC convergence criterion from energy-based to energy- and amplitude-based, affects the converged CC amplitudes as well as the $E^{(1)}_{\rm exch}$(CCSD) SAPT correction if CC convergence thresholds are not very tight (as it is in the default case). This may lead to slight differences in the value of $E^{(1)}_{\rm exch}$(CCSD) between SAPT2006 and older versions of SAPT.

## 3 Short overview of theory

In SAPT, the total Hamiltonian for the dimer is partitioned as $H = F + V + W$, where $F = F_A + F_B$ is the sum of the Fock operators for monomers $A$ and $B$, $V$ is the intermolecular interaction operator, and $W = W_A + W_B$ is the sum of the Møller-Plesset fluctuation operators. The latter operators are defined as $W_X = H_X - F_X$, where $H_X$ is the total Hamiltonian of monomer $X$. The interaction energy, $E_{\rm int}$, is expanded as a perturbative series

$$E_{\rm int} = \sum_{n=1}^{\infty}\sum_{j=0}^{\infty}(E^{(nj)}_{\rm RS} + E^{(nj)}_{\rm exch}) \qquad (1)$$

with the indices $n$ and $j$ denoting the orders in the operators $V$ and $W$, respectively. The energies $E^{(nj)}_{\rm RS}$ are the corrections defined by the regular Rayleigh-Schrödinger perturbation theory. These terms were named "polarization" energies by Hirschfelder [45] and this terminology was used in earlier editions of SAPT, but was dropped later due to the confusion with the induction interactions often called polarization interactions. The exchange corrections, $E^{(nj)}_{\rm exch}$, arise from the use of a global antisymmetrizer to force the correct permutational symmetry of the dimer wave function in each order, hence the name "symmetry adaptation". Whereas the double perturbation theory expansion of Eq. (1) is very convenient for analyzing the results, SAPT is actually a triple perturbation theory as the $W_A$ and $W_B$ operators appear individually in SAPT expressions. The resulting triple index corrections, $E^{(nij)}$, with the consecutive indices referring to $V$, $W_A$, and $W_B$, respectively, will occasionally appear later on in this manual.

The RS corrections of the first order in $V$, $E^{(1j)}_{\rm RS}$, describe the classical electrostatic interaction and are denoted by $E^{(1j)}_{\rm elst}$. An alternative to expanding the electrostatic energy in powers of the intramonomer correlation operator is to calculate monomer electron charge densities $\rho_A$ and $\rho_B$ at a certain level of correlation and then use these densities in the formula

$$E^{(1)}_{\rm elst} = \int \rho_A(\mathbf{r}_1)\frac{1}{r_{12}}\rho_B(\mathbf{r}_2)d\mathbf{r}_1 d\mathbf{r}_2 + \int \rho_A(\mathbf{r})V_B(\mathbf{r})d\mathbf{r} + \int \rho_B(\mathbf{r})V_A(\mathbf{r})d\mathbf{r} + V_0, \tag{2}$$

where $V_A$ and $V_B$ denote the electrostatic potential of the nuclei of monomer $A$ and $B$, respectively, and $V_0$ is the nuclear repulsion term. In SAPT2020, the densities in Eq. (2) can be computed at the relaxed CCSD level. The quantity $E^{(1)}_{\rm elst,resp}$(CCSD) obtained in this way contains all the second- and third-order intramonomer correlation corrections as well as some other classes of diagrams (diagrams resulting from single and double coupled-cluster excitations) summed up to infinite order [46]. Similarly, the $E^{(1)}_{\rm exch}$(CCSD) correction sums up the respective exchange contributions [47].

The second-order corrections can be decomposed into the induction and dispersion parts:

$$E^{(2j)}_{\rm RS} = E^{(2j)}_{\rm ind} + E^{(2j)}_{\rm disp} \quad \text{and} \quad E^{(2j)}_{\rm exch} = E^{(2j)}_{\rm exch-ind} + E^{(2j)}_{\rm exch-disp}. \tag{3}$$

The induction component is the energy of interaction of the permanent multipole moments of one monomer and the induced multipole moments on the other, whereas the dispersion part comes from the correlation of electron motions on one monomer with those on the other monomer. Similarly, the third-order polarization corrections are decomposed as

$$E^{(3j)}_{\rm RS} = E^{(3j)}_{\rm ind} + E^{(3j)}_{\rm ind-disp} + E^{(3j)}_{\rm disp} \tag{4}$$

and the same holds for the corresponding exchange corrections. A detailed discussion of the physical interpretation of various parts of the third-order polarization energy can be found in Refs. 1 and 39.

The SAPT interaction energy can be computed at different levels of intramonomer correlation and an approximate correspondence can be made between these levels and the correlation levels of the supermolecular methods. It can be shown [48], for example, that an appropriate sum of the polarization and exchange corrections of the zeroth order in $W$ provides a good approximation to the supermolecular Hartree-Fock interaction energy, $E_{\rm int}^{\rm HF}$:

$$E_{\rm int}^{\rm HF} = E_{\rm elst}^{(10)} + E_{\rm exch}^{(10)} + E_{\rm ind,resp}^{(20)} + E_{\rm exch-ind,resp}^{(20)} + \delta E_{\rm int,resp}^{\rm HF}, \tag{5}$$

where $\delta E_{\rm int,resp}^{\rm HF}$, defined by the equation above, collects all third- and higher-order induction and exchange-induction terms. The subscript "resp" means that the coupled Hartree-Fock-type response of a perturbed system is incorporated in the calculation of this correction. Including the intramonomer correlation up to a level roughly equivalent to the supermolecular second-order MBPT calculation, we obtain the interaction energy referred to as SAPT2:

$$E_{\rm int}^{\rm SAPT2} = E_{\rm int}^{\rm HF} + E_{\rm elst,resp}^{(12)} + \epsilon_{\rm exch}^{(1)}(2) + {}^{t}E_{\rm ind}^{(22)} + {}^{t}E_{\rm exch-ind}^{(22)} + E_{\rm disp}^{(20)} + E_{\rm exch-disp}^{(20)}, \tag{6}$$

where the notation $\epsilon^{(n)}(k) = \sum_{j=1}^{k} E^{(nj)}$ has been used, ${}^{t}E_{\rm ind}^{(22)}$ is the part of $E_{\rm ind}^{(22)}$ not included in $E_{\rm ind,resp}^{(20)}$, and ${}^{t}E_{\rm exch-ind}^{(22)}$ is the estimated exchange counterpart of ${}^{t}E_{\rm ind}^{(22)}$:

$${}^{t}E_{\rm exch-ind}^{(22)} \approx E_{\rm exch-ind,resp}^{(20)} \frac{{}^{t}E_{\rm ind}^{(22)}}{E_{\rm ind,resp}^{(20)}}. \tag{7}$$

The highest routinely used level of SAPT, approximately equivalent to the supermolecular MBPT theory through fourth order, is defined by:

$$E_{\rm int}^{\rm SAPT} = E_{\rm int}^{\rm SAPT2} + E_{\rm elst,resp}^{(13)} + [\epsilon_{\rm exch}^{(1)}({\rm CCSD}) - \epsilon_{\rm exch}^{(1)}(2)] + \epsilon_{\rm disp}^{(2)}(2), \tag{8}$$

where $\epsilon_{\rm exch}^{(1)}({\rm CCSD}) = E_{\rm exch}^{(1)}({\rm CCSD}) - E_{\rm exch}^{(10)}$ is the part of $\epsilon_{\rm exch}^{(1)}(\infty)$ with intramonomer excitations at the CCSD level only.

The SAPT2 level of theory takes much less time than the full SAPT calculation and therefore it is recommended for large systems. If still faster calculations are required, the correction ${}^{t}E_{\rm ind}^{(22)}$ can be omitted, as it is usually fairly small.

The corrections $E_{\rm ind,resp}^{(20)}$, $E_{\rm exch-ind,resp}^{(20)}$, $E_{\rm elst,resp}^{(12)}$, and $E_{\rm elst,resp}^{(13)}$ can also be computed in non-response versions, but these forms are not recommended and are not calculated unless explicitly requested.

In case the sum $E_{\rm disp}^{(20)} + \epsilon_{\rm disp}^{(2)}(2)$ is not converged well enough, the CCD+ST(CCD) approach developed in Ref. 40 is also available in SAPT2020. In this method, first, the dispersion energy is approximated at a level corresponding to the dimer CCD calculation. The energy $E_{\rm disp}^{(2)}({\rm CCD})$ obtained in this way can be shown [40] to contain the full corrections $E_{\rm disp}^{(20)}$ and $E_{\rm disp}^{(21)}$ and the so-called "DQ" part of $E_{\rm disp}^{(22)}$. Next, to take into account the remaining, "S" and "T" contributions to $E_{\rm disp}^{(22)}$, the expressions for these contributions (Eqs. (91) and (98), respectively, of Ref. 49) are evaluated with the converged CCD dispersion amplitudes replacing the first-order ones [hence the name CCD+ST(CCD)].

The corrections listed above constitute the set typically used in SAPT calculations. Recently, it has become possible to calculate also the corrections of the third order in $V$ and zeroth order in $W$ [39],

$$E^{(30)}_{\rm SAPT} = E^{(30)}_{\rm ind} + E^{(30)}_{\rm ind-disp} + E^{(30)}_{\rm disp} + E^{(30)}_{\rm exch-ind} + E^{(30)}_{\rm exch-ind-disp} + E^{(30)}_{\rm exch-disp}. \quad (9)$$

The first and fourth of these corrections constitute a part of the $\delta E^{\rm HF}_{\rm int,resp}$ quantity, however, for some systems it is advantageous to replace $\delta E^{\rm HF}_{\rm int,resp}$ by the sum $E^{(30)}_{\rm ind} + E^{(30)}_{\rm exch-ind}$ [39] or by their response versions [36]. Note that the third-order polarization and exchange corrections tend to cancel each other to a large extent, and one should not include any part of $E^{(30)}_{\rm RS}$ without including the corresponding exchange correction.

A few other corrections have been developed by the authors of SAPT but these are either not working in the current version of the program or for some other reasons are not recommended to be computed. These corrections include in particular various parts of $E^{(14)}_{\rm elst,resp}$ [50].

The theory presented above was restricted to SAPT(MP/CC) for dimers. The SAPT(DFT) approach is actually simpler since the intramonomer correlation effects are accounted for by DFT and the only operator is $V$. The SAPT approach is quite similar for trimers, except that there is a total of six perturbation operators: $V_{AB}$, $V_{AC}$, $V_{BC}$, $W_A$, $W_B$, $W_C$ in SAPT(MP/CC) and the three former operators in SAPT(DFT).

At intermonomer separations $R$ large enough for the exchange effects to be negligible, the SAPT results become identical to those of the regular Rayleigh-Schrödinger perturbation theory. The calculation of the interaction energies in this region can be substantially simplified by neglecting the overlap effects and expanding $V$ in the multipole series. The long-range part of the interaction energy becomes then expressed as a power series in $R^{-1}$, with coefficients that can be obtained using only monomer properties (viz. multipole moments and polarizabilities). These monomer properties can be calculated *ab initio* at the correlation level consistent with finite-$R$ SAPT calculations [51, 52] using the monomer parts of the basis set and the POLCOR suite of codes developed by Wormer and Hettema [33, 34] and distributed as a part of the package ASYMP_SAPT.

## 4 Downloading SAPT2020

The SAPT2020 distribution, the parallel version pSAPT2K2, the ASYMP_SAPT asymptotic package, and the ATMOL1024 integral and SCF code can be obtained from the web page

`http://www.physics.udel.edu/~szalewic/SAPT/SAPT.html`.

All these codes are distributed free of charge but we require users to sign a license agreement which can be downloaded from this web site and mailed or faxed to us as described there. We will then email to the interested party the password needed to complete the download. The users who download the ASYMP_SAPT and ATMOL1024 modules will be also asked to notify the authors of the POLCOR and ATMOL suites of the intended use of their codes.

## 5 Packages included in the distribution

There are several options for downloading SAPT2020 and the accompanying programs, so that users can download only the parts of interest to them:

1. SAPT2020. This file contains only the SAPT2020 codes including the new SAPT(DFT) codes. Users will have to obtain some integral/SCF package (like GAMESS, MOLPRO, GAUSSIAN, etc.) or download the ATMOL1024 code (see below) before running SAPT2020. On decompression, this file expands into `./SAPT2020/`.

2. ATMOL1024. This file contains the ATMOL1024 package. This package is a subset of the ATMOL codes [23] modified by us to handle basis sets of up to 1023 orbitals. On decompression, this file expands into `./SAPT2020/atmol1024`, so users should decompress it in the root directory that SAPT2020 is in.

3. AUTOPES. The SAPT automatic PES generation package. See Ref. [35] for details. If autoPES is required, it must be installed in addition to SAPT2020 using the included documentation.

4. ASYMP_SAPT. Contains the POLCOR suite [33, 34] and the accompanying programs necessary for computation of asymptotic coefficients. Also included is the potential energy fitting program `genfit_v1`, developed in our group. On decompression this file expands into `./asymp_SAPT/`. Documentation for this package is located in `./asymp_SAPT/doc`. This version of the asymptotic codes is included for legacy purposes. The latest version of DFT-based asymptotic codes are included in the SAPT2020 package.

5. **Complete set of sequential two-body codes**
   This file contains all the above modules. On decompression it expands into `./SAPT2020/` and `./asymp_SAPT/`.

6. SAPT3B. Contains the three-body SAPT [17] and three-body SAPT(DFT) [20]. Requires SAPT2020. On decompression this file expands into `./sapt3b/`.

7. pSAPT2K2
   Contains the parallel version of the SAPT codes, pSAPT2K2. To make the most of this version, you will have to obtain and install parallel GAMESS(US) as the integral/SCF package. See Sec. 14 and Ref. 9 for a detailed description of pSAPT2K2.

8. SAPT.OS

   Contains SAPT(DFT) code for open-shell high-spin complexes [21].

9. SAPT(CC):

   SAPT(CC) codes [24–31]. This is actually not a separate package, but a part of SAPT2020, point 1 above. The codes are provided as a patch to MOLPRO in the directory `misc/patch.ccsapt` of the SAPT2020 package, see Sec. 15.

The instructions on unpacking these files can be found on the SAPT web page in the Download Area.

## 6 Structure of `./SAPT2020` directory

After unpacking, the `./SAPT2020` main directory will contain the following files and subdirectories:

- `Cleandirs`: use this script to clean the entire SAPT2020 directory tree before recompiling from scratch.
- `Compall`: script used to build the package (see Sec. 8).
- `Makefile`: a generic makefile used by `Compall`.
- `UPDATES`: log of the history of changes and updates.
- `atmol1024/`: present if ATMOL1024 has been downloaded, this directory contains the sources of the ATMOL1024 integral and SCF code.
- `tran/`: program performing the one- and two-electron integral transformation.
- `cc/`: program performing the coupled cluster singles and doubles calculations for the monomers. The first few iterations are performed perturbatively, in this way producing MBPT order-by-order amplitudes needed in SAPT. Both CCSD and MBPT amplitudes are later used by the `sapt.x` module to compute intramonomer correlation contributions to various interaction energy components.
- `sapt.x/`: program computing the SAPT corrections.
- `e2d/`: program computing the dispersion energy with the intramonomer correlation effects included at the CCD level [40].
- `ccsdd/`: programs computing the relaxed CCSD densities of the monomers and the integrals of Eq. (2) in DCBS bases.

- `elsden/`: programs computing the electrostatic energy from monomer charge densities precomputed in MCBS bases, translated and rotated to monomers' positions in the dimer.
- `cks/`: program computing the coupled Kohn-Sham (CKS) dispersion and induction energies used in SAPT(DFT).
- `df/`: program computing the density version of the CKS dispersion and induction energies used in SAPT(DFT).
- `asymp_com`: program and scripts computing asymptotic COM-COM multipole expansion with monomer calculations based on DFT
- `misc/`: contains various interface and utility programs. Most integral/SCF packages need an interface program to extract one-electron integrals and SCF orbital energies and coefficients from files created by these packages and transform them into a standard form readable by the transformation code (two-electron integrals are read by the transformation program directly, without such preprocessing). Other programs present in `misc/` include `int` and `sort`, interfacing the transformation to the coupled cluster code, the memory estimator `memcalc`, a set of geometry converters which can be used with the `bin/Runlot*` scripts for automatic generation of potential energy surfaces (see Sec. 10 for details), and a few scripts helpful in creating DALTON input files.
- `fastdf/`: This folder contains the files necessary to build the new `fastdf` program.
- `bin/`: utility scripts for running SAPT. After compilation, this directory will also contain the executables used in a SAPT run.
- `doc/`: documentation for SAPT2020; contains this document and the METECC paper [8] in the postscript form. Documentation for ASYMP_SAPT can be found in the directory `asymp_SAPT/doc/`.
- `examples/`: input and output files for a set of systems and a variety of integral/SCF packages. This is a good source of templates for users runs.

## 7 SAPT installations at a glance

Table 1 presents a summary of hardware configurations, compilers, and integral/SCF packages with which SAPT has been tested. This list is meant to be used as a guide only. In case one needs to port SAPT to a new architecture, some important hints are provided in Appendix A.

Table 1: Grid of operating systems (OS) and front-end programs with which SAPT2020 has been tested. Symbols used: '+' – tested and working, '±' – tested, some problems, '−' – not working, '0' – not tested. This table is only for SAPT(MP/CC). SAPT(DFT) is interfaced only with DALTON 2.0 and ORCA.

| OS/processor/compiler[a] | ATMOL1024 | GAMESS(US) | GAUSSIAN03/09 | DALTON 2.0 | MOLPRO |
|---|---|---|---|---|---|
| **Linux/amd64 gfortran** | + | 0 | 0 | + | 0 |
| **Linux/i386 g77**[b] | + | + | 0[c] | + | −[d] |
| **Linux/amd64 g77**[e] | + | ±[f] | 0[e] | + | −[d] |
| **Linux/i386 pgf77**[g] | + | −[h] | + | 0 | −[d] |
| **Linux/amd64 pgf90**[i] | + | ±[f] | + | + | + |
| **Linux/ifort**[j] | ±[k] | 0 | − | +/0[l] | +[m] |
| **SUN OS** | + | + | +[n] | 0 | 0 |
| **IBM AIX** | + | + | 0 | + | 0 |
| **HPUX/Itanium** | + | 0 | +[n] | 0 | 0 |

[a] The systems presented in the table were tested with SAPT2020. Some obsolete systems such as **SGI OnK**, **Alpha**, and **IBM RS/6000** were tested at the time of the SAPT2002 version and may still work with SAPT2020.

[b] Tested on various i386 Linux systems with `g77` versions 3.3.x and 3.4.x. `gfortran` 4.x should work with SAPT2008.2 and above.

[c] GAUSSIAN does not support compilation with `g77`.

[d] MOLPRO requires a Fortran 90 compiler.

[e] AMD Opteron, AMD Athlon64, and Intel EM64T. Both 32-bit and 64-bit mode available. Compilation requires use of proper target flags: `g77_32` or `g77_64`.

[f] The SCF package does not work in 64-bit mode. Integral files calculated with 32-bit g77 GAMESS can be processed with 32-bit `g77` or 32 and 64-bit `pgf90` SAPT.

[g] Portland Fortran Compiler `pgf77` on i386 Linux systems.

[h] GAMESS does not work.

[i] Portland Fortran Compiler `pgf90` on AMD64 Linux systems.

[j] Intel Fortran Compiler `ifort` version 8.1 and above on i386 and AMD64 Linux systems.

[k] ATMOL does not work in 64-bit mode on AMD64.

[l] 64-bit mode not tested.

[m] Tested on AMD64 only.

[n] Tested with GAUSSIAN98 but should work with newer versions just as well.

# 8 Installing SAPT2020

Installation of SAPT2020 is controlled by a universal script `Compall`, a small portion of which has to be customized by the user. The `Compall` script sets the compilation options appropriate for a given hardware platform and for the integral/SCF interfaces chosen. It then compiles and links all the pieces of the code, which sometimes requires access to the I/O libraries of the integral/SCF packages. If ATMOL1024 has been downloaded and the `./SAPT2020/atmol1024` directory is present, this package is also built. Finally, `Compall` updates the scripts used to run SAPT2020 (`SAPT, Runlot*`) by inserting updated paths to executables. (The ASYMP_SAPT package is installed by a separate script).

## 8.1 `Compall` installation script

The following adjustments must be made by the user to the `Compall` script:

1. **Execution shell**: Change the first line of the script to one of the following:
   - `#!/bin/bash` : The Bash shell on a Linux box, or
   - `#!/bin/ksh` : The Korn shell on all other platforms.
2. **Integral/SCF program**: Declare the SCF packages you wish SAPT2020 to be interfaced with. An integral/SCF package is activated by simply setting the corresponding variable in the script to the complete path to the directory where the libraries (or executables, depending on the SCF package) of a given package are located (for example, `GAMESS=/home/local/gamess`). Otherwise set the variable to 'NO' (note the capitals). In addition, for GAMESS, the so-called "version number" environmental variable, `VERNO=XX`, needs to be specified (SAPT2020 assumes that the name of the GAMESS executable is `gamess.$VERNO.x`). The list of SCF codes in `Compall` does not include ATMOL1024. This latter option is activated automatically if ATMOL1024 has been downloaded and is present in the `./SAPT2020/atmol1024` directory. Thus, you may set all the integral/SCF variables to 'NO', provided that ATMOL1024 has been downloaded. Otherwise at least one integral/SCF variable must be set by specifying the path. If ATMOL1024 is present and in addition one or more integral/SCF packages are selected, both ATMOL1024 and the explicitly selected interfaces should work with the created SAPT2020 executables. In most cases, the SAPT2020 codes use I/O libraries of the integral/SCF packages in order to read the data created by these packages. Therefore, a given SCF package has to be installed *before* SAPT2020 and its libraries must be accessible. Four exceptions are CADPAC, GAMESS, DALTON, and ORCA, which generate data read by SAPT2020 using its own subroutines. The SCF programs that can be used to generate atomic integrals and SCF vectors for SAPT2020 are

- ATMOL1024: This is the current and maintained by us version of ATMOL [23], extended to handle up to 1023 basis functions. ATMOL1024 is the default integral/SCF package. The `Compall` script checks if the ATMOL1024 source is present in `./SAPT2020/atmol1024` subdirectory and compiles it automatically. (If for some reasons ATMOL1024 has to be kept in another location, and compiled separately, it can still be used by SAPT2020 if the line `ATMOL1024=NO` in `Compall` is changed to reflect the current location of the ATMOL1024 main directory.) ATMOL1024 code has been tested with SAPT2020 more extensively than any other package and is the recommended choice.

- GAUSSIAN: GAUSSIAN94 (`G94`) and GAUSSIAN98, GAUSSIAN03, or GAUSSIAN09 [43] (`G03`).

  **A full GAUSSIAN installation including the source code must be present at the location specified by the `G94`/`G03` variable. A binary-only GAUSSIAN installation will not suffice.** Set at least *one* of the variables `G94`, `G03` to `NO` as these packages are mutually exclusive. The path specified here should contain the GAUSSIAN's library `util.a` (usually it is the main directory of the GAUSSIAN distribution). Additionally, the variable `GAUEXE` holds the name of the GAUSSIAN executable (no path is necessary) so that the SAPT scripts know which executable to call. The default value `GAUEXE=g09` must be changed if a different version is used. Finally, the `Compall` script declares a variable `EXTRADEFS` which is an empty string by default. If GAUSSIAN was compiled with the option `-DI64` (64-bit integers, default on several architectures) and/or `-DPACK64` (64-bit packing of integral indices, also default on a few platforms), the same set of options (`-DI64` and/or `-DPACK64`) must be given to the variable `EXTRADEFS` so that the GAUSSIAN interface and the transformation program are compiled to work with this particular configuration of GAUSSIAN. These options do not affect running SAPT with any other integral and SCF front-ends.

  One should note that, as the SAPT codes are linked to the GAUSSIAN's `util.a` library, the user must make sure that this library (and the whole GAUSSIAN code) has been compiled for exactly the same architecture as SAPT. For example, the Sun SPARC architecture (`TARGET=sunf90`) supports generation of codes for several different flavors of SPARC via the compilation option `-xarch=PLATFORM`. If GAUSSIAN was compiled for an architecture other than the default `-xarch=generic`, the user needs to edit all occurrences of `-xarch=generic` in both `Compall` and `atmol1024/Makefile.sunf90`, replacing `generic` by the architecture for which GAUSSIAN was compiled. It is not a problem if Fortran 77 (rather than Fortran 90 as in the case of SAPT) was used to compile GAUSSIAN for `TARGET=sunf90`, as the compatibility library `-lf77compat` is automatically linked in where necessary.

SAPT does work with the latest releases of GAUSSIAN03 that support the AMD64 architecture. Note, however, that the `TARGET=pgf90` architecture has to be chosen in this case despite the fact that for this platform, GAUSSIAN itself is compiled with `pgf77`. **The `TARGET=pgf77` setting is restricted to the 32-bit i386 architecture and will not produce a valid code on an AMD64 machine.**

- GAMESS [22]: SAPT2008.1 works with the most recent release of GAMESS (dated 11 Apr 2008). If you want to use an older version of GAMESS, you may need to edit the file `misc/gamint/gamsintf.F`, subroutine `OPENDA`, changing the value of the variable `IRECLN` from 4090 to whatever is returned by the GAMESS function `NRASIZ(10)`; see GAMESS source code, file `iolib.src`.

  If the GAMESS interface is used, set the variable `GAMESS` to the path where the GAMESS executable is located. Also set the variable `VERNO` which is the middle part of the name of this executable (e.g., for `gamess.01.x`, `VERNO=01`). The sole purpose of the variable `GAMESS` in `Compall` is to pass the path and name of the GAMESS executable to the `SAPT` execution script. No access is needed to any GAMESS files at the compilation time. Thus, compilation will proceed even if GAMESS is not installed beforehand.

  **64-bit** GAMESS: if you intend to use GAMESS compiled with a 64-bit integer (this happens by default on the ALPHA platform, but may also happen on SGI when "sgi64" is specified during compilation of GAMESS), replace the line `GAMSI8=NO` with `GAMSI8=YES`.

- CADPAC [53]: This package does not need any interface programs since the files created by it can be read directly by SAPT2020 codes. Also, SAPT2020 does not need access to any CADPAC libraries, so that the compilation will proceed even if CADPAC is not installed on your system.

- DALTON [38]: SAPT2020.1 is compatible with Dalton version 2.0. This older version of Dalton can be obtained by a special request to the Dalton authors (visit [38] for more details). If used, set the path to wherever the DALTON executable script is located. The sole purpose of the variable `DALTON` in `Compall` is to pass the path and name of DALTON executable to the `SAPT` execution script. No access is needed to any DALTON files at the compilation time. Thus, compilation will proceed even if DALTON is not installed on your system beforehand.

- ORCA [54]: If used, set the path with the ORCA executable files. Similarly as for DALTON, the only use of the variable `ORCA` during the compilation is to pass the path to ORCA executable files to the `SAPT` execution script.

- MOLPRO [42]: If the MOLPRO interface is used, set the variable `MOLPRO` to the full name of the MOLPRO executable (including the path). All versions of MOLPRO starting from MOLPRO2002.1 up to and including MOLPRO2010.1 are supported. During the compilation of SAPT2020, no access to any MOLPRO libraries is needed so the compilation will proceed even if MOLPRO is not installed on your system beforehand. However, the MOLPRO program itself must be modified and recompiled before using it with SAPT2020 as the SAPT interface to this front-end has the form of a MOLPRO patch. This interface is located in the `misc/patch.molpro2sapt` subdirectory and consists of four files: `src/common/sapt1`, `src/common/sapt2`, `src/util/user.f`, and `src/util/user_sapt.f`. The first two of these files must be placed in the `src/common` subdirectory, and the last two—in the `src/util` subdirectory of your parent MOLPRO directory before recompiling MOLPRO. Note that the standard distributions of MOLPRO already contain a dummy version of the file `src/util/user.f`. It may safely be replaced by the file from SAPT2020 if, as it is the case for a default MOLPRO installation, no other user-supplied subroutines have been placed there. As a Fortran 90 compiler is required to build MOLPRO, this interface has been tested only for such compilers (Portland pgf90 and Intel ifort on a 64-bit AMD architecture running Linux) so far.

- ACES: This interface has not been tested recently but might work.

It is also possible to build the SAPT2020 package for use with the older version of ATMOL (such a version, limited to 255 basis functions, is included in the ASYMP_SAPT distribution). This can be accomplished by setting the variable `ATMOL` in `Compall` equal to the path of the main ATMOL directory (change the line `ATMOL=NO`) and removing or renaming the directory `./SAPT2020/atmol1024` (if present). The ATMOL package has to be compiled prior to the compilation of SAPT2020. SAPT2016 cannot be interfaced with ATMOL1024 and ATMOL simultaneously.

3. **PLATFORM**: Two variables need to be set:

   - `TARGET` is the system on which SAPT2020 will be compiled. This can be one of (all lowercase): `sgi, ibm32, ibm64, alpha, g77, pgf77, pgf90, ifort, g77_32, g77_64, sunf90, hpux` or `gfortran`. `sgi` stands for the Silicon Graphics ORIGIN or POWER CHALLENGE series computers with the MIPSPRO 7.3 or higher compiler, `ibm32` – for the IBM RS6000 or SP machines (on 64-bit IBM platforms, `ibm64` is also possible), `alpha` – for the (formerly DEC) ALPHA platform. `g77` and `pgf77` are used for a 32-bit LINUX box with the compiler `g77` or `pgf77` (Portland Group Fortran), respectively. If one needs to run SAPT on an old Linux machine which does not support files larger than 2 GB, splitting of all potentially large temporary files into 2 GB chunks can be turned on at compile time by adding the `-DTWOGIGAMAX` declaration to the variable `EXTRADEFS` at the beginning of the `Compall` file. On a 64-bit AMD platform running Linux, one may compile SAPT using the Portland Group Fortran 90 (Portland Group Fortran 77 will not suffice in this case) or the Intel Fortran Compiler (versions 8.1 and above) by setting `TARGET=pgf90` and `TARGET=ifort`, respectively. For the compilation on 64-bit AMD using `g77`, the user needs to specify the `g77_32` or `g77_64` target flag for a 32-bit or a 64-bit binary, respectively. `TARGET=pgf90` and `TARGET=ifort` work in the 64-bit mode by default; note, however, that ATMOL1024 does not work with `TARGET=ifort` in 64-bit mode. GNU Fortran95, `gfortran` is supported starting with SAPT2008.2, except for ATMOL, where some combinations of compiler versions/systems do not work. Since `gfortran` changed the file binary format between versions 4.1 and 4.2, it is *strongly recommended* that all programs (including SCF/DFT interface programs) are compiled with the same version of the compiler. `TARGET=sunf90` is for the Sun SPARC machines equipped with the FORTRAN90 compiler. Finally, `TARGET=hpux` corresponds to the 64-bit Itanium architecture running HPUX and equipped with the HP Fortran90 compiler. Note that for this platform several necessary system calls (used, e.g., in the timing routines) are contained in the `libU77.a` library whose location may vary on different machines (`/lib/hpux64` or other). To compile SAPT for `TARGET=hpux`, the user must make sure that the correct, 64-bit version of the `libU77.a` library is present in one of the directories in which the linker searches for libraries.

- `BLAS` points to the Basic Linear Algebra Subprograms (BLAS) and the Linear Algebra PACKage (LAPACK) static libraries. **The efficiency of the BLAS library is crucial for the performance of SAPT and users are encouraged to use the fastest BLAS library on their machines.** On SGIset `BLAS=' -llapack -lblas '`. On an ALPHA, set `BLAS=' -ldxml '`, on IBM AIX (RS6000 or SP4): `BLAS=' -lessl '`, and on SPARC: `BLAS=' -xlic_lib=sunperf '`. On Linux machines, we recommend one of the following three libraries: GotoBLAS (or its successor, OpenBLAS), AMD's Core Math Library (ACML), or Intel's Math Kernel Library (MKL). On machines equipped with Intel processors, we've found the MKL library to be the most efficient; if not available, GotoBLAS tends to be adequate enough, but in any case we recommend the user to try linking SAPT against different BLAS libraries and evaluating which one works better. Please consult the manual for the specific version of the chosen library for the proper linking options. The simplest setting, `BLAS=' -llapack -lblas '`, (or just `BLAS=' -lblas '` if LAPACK is not available) should work on most Linux distributions but will link the generic BLAS and (usually) significantly hurt the performance of SAPT. The performance of LAPACK, however, is of minor importance and if optimized LAPACK is unavailable, one can set `BUILDLAPACK=YES` in the `Compall` script and the subset of LAPACK required for SAPT will be compiled. Please note, however, that both ACML and Intel's MKL provide subroutines that replace LAPACK's own, and `BUILDLAPACK` should be set to `NO` if either of these is used.

All three listed libraries exist also in multithreaded versions that can automatically (*i.e.*, without any changes in the user program calling the BLAS routines) make use of multi-core architecture of modern processors (or any other type of shared-memory architecture) and we recommend the use of these versions. With the exception of the fastdf program, the number of cores actually used by the program is controllable by a shell variable so that a number smaller than physically available may be requested while starting the SAPT calculation (see Sec. 10.8).

4. Several additional preprocessor definitions may be specified during the compilation by listing them in the variable `EXTRADEFS` in the `Compall` script. This variable, equal to an empty string by default, may contain the options `-DI64` and/or `-DPACK64` defining the format of files imported from GAUSSIAN, the option `-DTWOGIGAMAX` which requests splitting of all large temporary files into chunks smaller than 2 GB, and/or, the definition `-DTPDRVN` used to switch on the old algorithm of calculating the four-virtual contribution to the monomer CC amplitudes using unsorted integrals. The default algorithm uses sorted integrals and is faster, especially when a well-optimized BLAS library is available, but it requires more disk space during the CC stage of the calculation.

Once all the variables mentioned above are set, simply type:

- **C-Shell users:** `./Compall >& compall.log &`
- **K-Shell or Bash users:** `./Compall > compall.log 2>&1 &`

and the compilation should begin. Check `compall.log` to see if all is well. The `Compall` script will create a file '`./SAPT2020/stamp.intf`' containing a summary of the settings you have used in the compilation. A subsequent invocation of `Compall` will detect any changes made to these settings since `stamp.intf` was last created and only those parts of the code which were affected by these changes will be rebuilt. Running the script `./SAPT2020/Cleandirs` will restore the `./SAPT2020` directory to its "distribution" state, i.e., all object files and executables (except shell scripts) will be deleted and a subsequent invocation of `Compall` will start the compilation from scratch.

One more customization step may be required before SAPT2020 is run with GAMESS as the SCF front-end: the `./SAPT2020/bin/runGAMESS` script must be modified by setting the `TARGET` variable, which depends on the platform and on the way GAMESS has been compiled. In most cases, the default `TARGET=sockets` will be appropriate, although on SGI machines `TARGET=sgi-mpi` is also popular. Consult your local GAMESS installation. Further customization of `runGAMESS` will be needed for other targets (if you need to do such a customization, see examples given in both `runGAMESS` and the standard `rungms` script for the GAMESS distribution).

### 8.2 `Compall_asymp` installation script

If you have downloaded the ASYMP_SAPT package, it will expand into `./asymp_SAPT` directory. Change to this directory and use `Compall_asymp` script to build all programs in this package. The subdirectory `doc` contains the user's manual for these programs. The asymptotics calculations are presently limited to 255 orbitals per monmer and the relevant codes are interfaced only with the older version of ATMOL (included in the ASYMP_SAPT package). Furthermore, the POLCOR program computing the dynamic polarizabilities works only under SGI's IRIX and IBM AIX operating systems.

### 8.3 Testing SAPT2020 installation

Once the compilation has been completed successfully (if unsure, just `grep` the `compall.log` file for the word *error*), we **strongly** recommend that you perform as many tests as possible before starting to use SAPT2020 for production runs. A suite of test jobs of varying size and for various SCF front-ends has been provided for this purpose in the subdirectory `examples/`. Sample outputs from different platforms can also be found there. For more information on running the test jobs, see Sec. 13.

## 9 Using SAPT2020 with different front-end packages

### 9.1 ATMOL1024

A good place to look for the description of input to ATMOL1024 is Paul Wormer's web page, `http://www.theochem.ru.nl/~pwormer` (strictly speaking, this is a description of the older version of ATMOL, but the input options are the same as for ATMOL1024). Below we discuss several options relevant for a SAPT2020 run.

It is convenient to reduce printouts from the integral (`integw`) module of ATMOL1024 by adding the directive `NOPRINT GROU GTOS` to the "intinp" files. This will suppress printing of the groups or ordered Gaussian Type Orbitals (GTOs). Adding the word `BASI` to the list will also suppress the basis set printout. To avoid computation of the multipole integrals, not needed in a SAPT calculation, include the directive `BYPASS PROPERTY` (this may be important especially on SGI machines, where the absence of such directive may cause a run-time error). Inclusion of a line `FPRINT NVCT NEIG NFTE NITE NPOP` in the `*.scfinp` input files will ask the `scf` program to skip the printout of the vectors, eigenvalues, Fock trial matrix, iterations, and populations for that run.

When the supermolecular SCF interaction energy is calculated during a SAPT run, i.e., when the `scfcp` option is in effect (most runs are like this), the files `nameA.intinp` and `nameB.intinp` should contain the directive `BYPASS TWO` to avoid recalculation of two-electron integrals. This directive should *not* be present in the file `name.intinp` or, in the case of a MC$^+$BS calculation – in `nameMA.intinp` and `nameMB.intinp` (`name` denotes here the name of the job). If the supermolecular SCF calculations are *not* performed, i.e., if the `scfcp` option is *not* in effect, then the directive `BYPASS TWO` must be also removed from the file `nameA.intinp`. See Sec. 10 for further explanation.

On old Linux platforms with the `g77` compiler, there is a 2 GB restriction on the size of any file. In such a case, if the two-electron integral file generated by ATMOL1024 is likely to exceed this size, it has to be split into several pieces each 2 GB or less. For example, if the integrals are anticipated to take 5 GB of disk space, one should have them distributed over at least three files. This can be done by specifying

```
MAINFILE MT3 MT4 MT5
MAXBLOCK 499999
```

in the `*.intinp` files, somewhere below the basis set specification but before the `ENTER` directive. The first of the lines above means that the files called `MT3, MT4, MT5` will be used to store the integrals, and that the size of each will not exceed `MAXBLOCK*511*8` bytes, which in this case amounts to somewhat less than 2 GB (the number 511 comes from some internal data structures of ATMOL1024). Now we have to tell the transformation module of SAPT2020 how many files ATMOL1024 produced by including `NMFILES=3` in the `TRN` namelist in the `*P.data` input file (see Secs. 10 and 10.2 for description of this file). Of course, the number "3" would have to be changed if some other number of `MT*` files were used. ATMOL1024 and SAPT2020 support up to 18 such files, `MT3` through `MT20`. This option is left for compatibility but is not necessary on modern Linux distributions.

## 9.2 CADPAC

The CADPAC interface to SAPT2020 is not maintained anymore, however, it is likely to work. Notice that CADPAC can use only Cartesian basis functions, and only angular functions up to $f$ are allowed. The input file used for CADPAC runs must include the `SAPT` keyword. This keyword makes CADPAC write out to a file the SCF vectors and other information in the format required by the SAPT2020 transformation code `tran`. Thus, there is no interface program required for CADPAC.

CADPAC was the first front-end for SAPT(DFT) and the use of this package may be still be of some interest despite the slowness of this interface due to the large number of DFT eXchange Correlation (XC) functionals implemented in CADPAC. Notice, however, that the complete SAPT(DFT) [14], with coupled Kohn-Sham dispersion and induction energies, does not work anymore with CADPAC (it works with DALTON 2.0 and ORCA only).

To run SAPT2020 with CADPAC as the front-end, a special script `/SAPT2020/bin/SAPT_CADPAC` should be used instead of `SAPT` used for all other SCF programs.

## 9.3 GAUSSIAN

When using GAUSSIAN with SAPT2020, a symbolic link in the compilation script `Compall` is made to point to the `util.a` file in the GAUSSIAN directory structure. The transformation module `tran` of SAPT2020 links to this library to be able to read the `rwf` and two-electron integral files. In the input to the post-Hartree-Fock stage of the calculation (the file `nameP.data`), in the `TRN` namelist, the user must specify either `ISITG94=T` for GAUSSIAN94 or `ISITG03=T` for GAUSSIAN98, GAUSSIAN03, or GAUSSIAN09. In the latter case, the variable `GAUEXE`, set up during the compilation of SAPT, is passed to the script so that it knows whether to execute `g98`, `g03`, or `g09`.

In GAUSSIAN94 and in newer versions of this code, the default method of SCF calculations is direct (two-electron integrals calculated in-core and not written to disk). Since SAPT2020 always needs the two-electron integrals, the command `SCF=(CONV)`, which stands for "do conventional SCF", must be used to force the two-electron integrals to be written to disk. Note that in the conventional (non-direct) mode GAUSSIAN is limited to $s, p, d, f$ basis functions and one must use a different integral and SCF front-end when $g$ (or higher) functions are present in the basis set. We recommend the `SCF=(TIGHT,CONV)` keyword to be used to tighten the SCF iterations convergence criteria. Moreover, the keywords `SYMM=NOINT` and `NORAFF` must be present in all inputs to GAUSSIAN since SAPT does not use the point group symmetry of integrals and accepts integrals in the regular (non-Raffenetti) format only. For unknown reasons, these settings do not always work in GAUSSIAN03 if the default method of selecting the initial guess for the SCF iterations is used. In case of such problems, the user should specify `GUESS=INDO` so that the initial guess is obtained using the method which was the default in GAUSSIAN98 and previous releases (note that the `GUESS=INDO` keyword is not understood by GAUSSIAN98 and older versions). Finally, we remind the user that the `MASSAGE` keyword should be used in GAUSSIAN when setting the charges to zero (e.g., to perform an SCF run for a monomer in a dimer-centered basis set).

## 9.4 GAMESS

### 9.4.1 Optional modification of GAMESS source

In order to ensure that the SCF energies from GAMESS are printed with a sufficient number of decimals, we recommend one `FORMAT` change in the `rhfuhf.src` module, prior to compilation of GAMESS. In the subroutine `RHFCL`, the statement

```
 8000 FORMAT('--- CLOSED SHELL ORBITALS --- GENERATED AT ',3A8/10A8/
     *       'E(',A,')=',F20.10,', E(NUC)=',F16.10,',',I5,' ITERS')
```

should be replaced by:

```
 8000 FORMAT('--- CLOSED SHELL ORBITALS --- GENERATED AT ',3A8/10A8/
     *       'E(',A,')=',F24.16,', E(NUC)=',F16.10,',',I5,' ITERS')
```

### 9.4.2 Required and recommended input options

Please note that standard (non-direct) SCF calculations must be performed and some options in the GAMESS input must have specific values:

- In the `$CONTRL` input group, `NOSYM=1` option *must* be present (i.e., *no symmetry* must be requested).
- In the `$INTGRL` input group, `NOPK=1` (integrals *must not* be in supermatrix form) and `NINTMX=2048` options must be present. Maximum number of integrals in a record block `NINTMX` must be the same as `linrec` variable in the SAPT2020 module `trans.f` which is set in the `IF(isitgams) THEN` block. The `linrec` is currently set to 2048 (but can be changed to a different value).
- We recommend that the following thresholds for the two-electron integrals, linear dependence, and SCF convergence are set instead of the GAMESS default values:
  - in the `$CONTRL` input group, `ICUT=24, ITOL=26`
  - in the `$CONTRL` input group, `QMTTOL=1.0E-30`; this will prevent GAMESS from unexpectedly removing quasi-linearly dependent combinations of basis functions from the variational space (SAPT2020 does not work correctly if such a removal occurs)
  - in the `$SCF` input group, `NCONV=9`.
- We recommend using the option `ISPHER=1` in the `$CONTRL` input group, which forces GAMESS to do SCF calculations in the basis set of *spherical* Gaussian orbitals instead of the default *Cartesian* ones and thus reduces the risk of linear dependencies. Unfortunately, although using this option reduces the dimension of the variational space available to the system, the atomic integral file produced by GAMESS remains "Cartesian" and thus its size is not reduced.
- In the MC$^+$BS-type run (see Sec. 10 for explanations), the variable `SPHG` in the namelist `TRN` in the `*P.data` file (see Secs. 10 and 10.2 for a detailed description of this file) *must* be set to `F` or `.FALSE.`, even if GAMESS was run with `ISPHER=1`. This is because of the "Cartesian" character of the integrals file, as mentioned above. The variable `SPHG`, which defaults to `.TRUE.` (i.e., spherical basis is assumed), is only important for MC$^+$BS and MCBS runs and has no effect in the DCBS case.

### 9.4.3 `runGAMESS` script

Whereas most other integral/SCF packages are invoked in the `SAPT` script by just executing a given program, GAMESS needs its own script, `./SAPT2020/bin/runGAMESS`, called from the `SAPT` script. `runGAMESS` is a slightly modified version of the standard script `rungms` distributed with GAMESS. As already pointed out in Sec. 8, the user must edit this script and supply the appropriate value of the `TARGET` variable. The targets `sockets` and `sgi-mpi` have been extensively tested, while other targets may require some additional customization to make GAMESS run. These additional changes are independent of the SAPT2020-related portions of `runGAMESS` and, if needed, can be introduced with the help of the standard GAMESS documentation.

If you rather prefer your own GAMESS script instead of `runGAMESS`, you can easily adapt it for use with SAPT2020 by following these steps:

- Copy your script into the `./SAPT2020/bin` directory. In the `SAPT` script, change the name of `runGAMESS` to that of your own script.
- The `SAPT` script communicates with `runGAMESS` (or your custom-made script) through the syntax of the type

  ```
  runGAMESS $JOB $VERNO $NNODES $SCR $PUNCHDIR $GMSPATH
  ```

  where

  - `JOB` is the name of the input file, like `xxx.inp`, give only the `xxx` part
  - `VERNO` is the "version number" of the executable (usually the name of the executable is something like `gamess.$VERNO.x`)
  - `NNODES` is the number of compute processes to be run (only 1 can be used with sequential SAPT2020 and that's what the `SAPT` script is requesting)
  - `SCR` is the scratch directory for GAMESS
  - `PUNCHDIR` is the punch directory for GAMESS
  - `GMSPATH` is the path to the GAMESS executable

  All the parameters in the invocation of `runGAMESS` are set automatically in the `SAPT` script upon compilation of SAPT2020 or at run time. Your custom-made replacement of `runGAMESS` should recognize these command-line parameters instead of having them hard-coded, as it is usually the case with the standard `rungms`.

- In your script, after the SCF calculation is done but before file cleanup, some GAMESS files should be saved with appropriate names:

– `$SCR/$JOB.F05` as `$SCR/$JOB.INP`,

– `$SCR/$JOB.F08` as `$SCR/inttw.data`, and

– `$SCR/$JOB.F10` as `$SCR/$JOB.DAF`,

where `SCR` denotes GAMESS scratch directory and `JOB` is a script variable which will be set by the `SAPT` script on each call to GAMESS.

### 9.4.4 Interface

The sources of GAMESS—SAPT2020 interface are located in the `./SAPT2020/misc/gamint` subdirectory. The interface consists of the Fortran program `gamsintf.f` which extracts one-electron integrals and SCF vectors from the "dictionary" file of GAMESS, and two simple `awk` scripts, `gms_awk1` and `gms_awk2`, which scan the GAMESS standard output for the number of basis functions, occupied orbitals, and system geometry.

## 9.5 DALTON

The DALTON 2.0 interface was tested on machines with 32-bit integers only (but most of the '64-bit' architectures, including AMD64 and IBM64, have 32-bit integers). DALTON must be compiled without `-DVAR_SPLITFILES` option and the operating system must support files larger than 2 GB. The input file `.dal` must include `.INTERFACE` directive in `**WAVE FUNCTIONS` and `.NOSUPSYM` in `*ORBITAL`. If the job has more than 255 basis functions, `.NOSUPMAT` in `*AUXILLIARY INPUT` is required, see `examples/DALTON/He2`. The symmetry must be switched off. For larger basis sets, prone to linear dependencies, `.AO DELETE` and `.CMOMAX` in `*ORBITAL INPUT` should be set to a small and a large value (e.g., `1.D-8` and `1.D+5`), respectively, to suppress removing of quasi-linear dependencies from the basis set combinations since SAPT2020 would not work if such a removal is performed. It is recommended to compile DALTON with the same compiler as SAPT.

## 9.6 MOLPRO

When using MOLPRO as an integral and SCF front end, only one MOLPRO input file that takes care of the dimer and both monomers is required irregardless of whether the DCBS/DC$^+$BS (Sec. 10.1.1) or MCBS/MC$^+$BS (Sec. 10.1.2) approach is used. The SAPT script assumes that this file is named `name.molpro` where `name` is the name of the job. Example `name.molpro` files can be found in the `examples/MOLPRO` subdirectory of SAPT2020. Several important things should be kept in mind when writing the `name.molpro` file:

- The integral symmetry must be switched off using the `SYMMETRY,NOSYM` keyword.

- After the SCF calculation for each monomer, the records containing eigenvectors and other important quantities have to be saved so that these records are accessible to the interface routine. Furthermore, the number of electrons, the spin multiplicity, the nuclear repulsion energy, and the total SCF energy must be saved to appropriate variables. This is accomplished (in case of the DCBS/DC$^+$BS approach) by a series of keywords in the MOLPRO input file

  ```
  orbital,2110.3
  NelecA=nelec
  NspinA=spin
  nucmonoA=enuc
  emonoA=energy
  data,copy,1200.1,1202.1
  data,copy,1410.1,1412.1
  data,copy,700.1,702.1
  ```

  for monomer A and

  ```
  orbital,2111.3
  NelecB=nelec
  NspinB=spin
  nucmonoB=enuc
  emonoB=energy
  data,copy,1200.1,1201.1
  data,copy,1410.1,1411.1
  data,copy,700.1,701.1
  ```

  for monomer B. Note that the order of some of these keywords is important as MOLPRO2006 appears to forget the SCF energy after the `data,copy` command. The interface also needs the variable `nucrep` containing the nuclear repulsion term between monomers:

  ```
  nucrep=nucdimer-nucmonoA-nucmonoB
  ```

- If the CP-corrected supermolecular SCF interaction energy is requested apart from the SAPT corrections, the SCF energy for the dimer must be calculated and saved:

  ```
  hf
  edim=energy
  ```

Later in the script, all three SCF energies should be listed with sufficient precision

```
show[1,d25.15],edim
show[1,d25.15],emonoA
show[1,d25.15],emonoB
```

as the SAPT programs extract these values from the output of the MOLPRO part of calculation. If the supermolecular SCF interaction energy is not requested, the dimer SCF energy `edim` is not needed, however, the nuclear repulsion energy for the dimer `nucdimer` is still required. The simplest way to get `nucdimer` is to request a zero-iteration SCF calculation for the dimer in some minimal basis set like STO-3G:

```
basis=sto-3g
hf;maxit=0
nucdimer=enuc
```

- At the very end of the MOLPRO input, the interface routine has to be called using the keyword

  ```
  user
  ```

Starting from the SAPT2008.1 release, the MOLPRO interface supports the use of a monomer-centered basis set as well as of a dimer-centered one. An example of an MC$^+$BS SAPT2020 run employing the MOLPRO interface can be found in the directory `./SAPT2020/examples/MOLPRO/CO2D_MCBS`. Several changes in the MOLPRO input file `name.molpro` are needed to perform an MC$^+$BS run:

- Before (not after) the regular set of dimer and monomer integral/SCF calculations in the full DC$^+$BS basis is done, separate monomer SCF calculations have to be performed in restricted, MC$^+$BS parts of the full dimer basis. The only data that is needed from these calculations are the SCF orbitals and orbital energies, and these quantities must be saved using the keywords `orbital,2110.3` for monomer A and `orbital,2111.3` for monomer B.
- The same `orbital` keywords must be removed for the subsequent monomer SCF calculations employing full DC$^+$BS set to avoid overwriting the MC$^+$BS orbitals.
- One should note that a `name.molpro` input file for an MC$^+$BS run contains three basis specifications (the MC$^+$BS set for monomer A, the MC$^+$BS set for monomer B, and the full DC$^+$BS set). The user must make sure that these three bases have exactly the same ordering of functions apart from some DC$^+$BS functions being omitted in the MC$^+$BS sets. Special care should be taken when specifying different basis sets for different atoms of the same type, as under some circumstances MOLPRO changes the ordering of atoms compared

to the ordering given in the geometry specification. The user is strongly advised to check in the MOLPRO output if the ordering of atoms is consistent with the basis set specification.

The post-SCF input file `nameP.data` must in this case contain the `BLKMB=F` directive so that the assignment of the basis functions to monomer A/monomer B/dimer sets is specified using the 'tags' mechanism, cf. Sec. 10.1.2.

One should note that the geometry section of the final SAPT summary table looks a little different than usual when MOLPRO is used as the integral and SCF front-end. The charges on the nuclei are not given, and all atoms (those of monomer A, those of monomer B, and dummy atoms) are listed together. Nevertheless, all energies calculated by SAPT should be correct.

# 10 How to run SAPT2020

To perform a SAPT2020 calculation for one dimer geometry, one has to run a dozen or so programs: integral/SCF calculations for the monomers and possibly for the dimer, interface programs (in most cases) rewriting integral/SCF files into different forms, one- and two-electron integral transformations, MBPT/CCSD calculations for monomers, and finally, the "proper" SAPT calculations. All of this is performed automatically using the script `SAPT` from `./SAPT2020/bin` directory (note, however, that a special script `doSAPT_CADPAC` has to be used if CADPAC is the front-end SCF program). This script calls other executables and scripts which can be found in the same place. Calculation of the interaction potential energy surface of a dimer involves multiple invocations of the `SAPT` script for different dimer geometries. This process can be simplified and automated with the help of the `Runlot` utility scripts, described in Sec. 10.6.

One of the scripts called by `SAPT` is the script `bin/Clean` that cleans up unnecessary files after a SAPT run (it will erase the SAPT-related files from the directory in which it is run). The files are not automatically erased at the end of of each run in order to enable restarts (which has to be done on a case-by-case basis by modifying the `SAPT` script except when starting from the transformation step). Therefore, `bin/Clean` is called at the beginning of the `SAPT` script, so that a consecutive calculation can be performed in the same directory (do not forget to change the name of the output file). This is necessary since several temporary files are named with no reference to the job name. Thus, two simultaneous calculations cannot be done in the same directory (but can be run, of course, in separate directories). It is also prudent to run `bin/Clean` itself (just by executing `./SAPT2020/bin/Clean`) to release unnecessary disk space after finishing calculations in a given working directory.

The actual running of the program when using the `SAPT` script is very simple. On most installations runs are performed in a "working" or "scratch" directory designed to hold large temporary files. We find it simplest to make a subdirectory there, copy the input files (created by the user or taken from the `./SAPT2020/examples`) to this subdirectory, and either execute the `SAPT` script in this directory using the full path (e.g., `/home/local/SAPT2020/bin/SAPT ...`) or simply copy the `SAPT` script to the working directory (several other possibilities exist, for example users can add the `./SAPT2020/bin` directory to their `PATH` environment variable). However, the SAPT script requires the `vars.cfg` file present in the same directory so this file must be copied accordingly if the script is not run from the `SAPT2020/bin` directory.

During the compilation, the script `Compall` updates `vars.cfg` file with proper paths to executables and the scripts should run properly on any installation without changes if `vars.cfg` is present in the same directory. If the paths need to be changed for some reasons, the file `vars.cfg` should be edited and the variable `MAIN_SAPT_DIR` and the `SCF_HOME_DIRECTORIES` changed to reflect user's directory structure (if GAMESS is used, it concerns also additional directories relevant for this program).

Also note that the large core memory requested typically by the SAPT2020 programs requires on some systems the use of the `ulimit` command to change the default user resources. This command, built into the `SAPT` script, is currently commented out but it may be reactivated and adjusted as needed.

The `SAPT` script is written in `ksh` although on Linux platforms, some of which are not equipped in `ksh`, it is actually executed under `bash`. A SAPT2020 calculation is launched by typing `SAPT` with appropriate options. Typing `SAPT` without any options will produce a brief description of the necessary input. A typical run statement might be something like (in Unix `ksh`):

```
SAPT jobname [opt1] [opt2] >output.file 2>&1 &
```

The keyword `jobname` has to be the same as the beginning of the name of the input files for the system considered (we use a naming scheme to reference the needed input files). For example, let the keyword be `name`. Then, as the script is running, it will look for the `nameP.data` file, which contains the input data to the programs `tran`, `ccsdt`, and `sapt.x`. Similarly, the `SAPT` script will look for input files to the integral/SCF parts of a calculation starting with `name`. The number and full names of such files depend on the type of basis set used in calculations and on the choice of the integral/SCF code.

For all SCF front-ends except GAMESS, the keyword `opt2` can be skipped and `opt1` is optional. Using the string `scfcp` for `opt1` will request, in addition to the standard SAPT calculation, also the CP-corrected supermolecular SCF interaction energy in the dimer-centered basis set. If such a calculation is not required, leave `opt1` blank or, better yet, use `noscfcp` instead. Using the keyword `gototran` as `opt1` will result in restarting a SAPT run from the transformation step, i.e., from the program `tran`. Both parameters, `opt1` and `opt2`, are required when GAMESS is used as the SCF program. `opt1` can assume one of the values listed above (but `noscfcp` *must* now be used instead of a blank), while `opt2` must be the full path of the scratch directory in which the whole calculation is taking place. `opt2` has the same meaning when DALTON is used. However, for DALTON `opt2` is optional and the place where `SAPT` is launched will be assumed as the scratch directory if this parameter is omitted.

The output from all programs executed by `SAPT` is written to a file `output.file` (this name can be set by the user to an arbitrary string, usually connected with the system being computed and its geometry). The last two elements in the command line given above are Unix `ksh` constructs which indicate that standard error messages will be written to the same file as the standard output and that the process will be run in the background.

Most modern computer clusters do not allow command line submission of jobs. Instead, one has to use a queuing system (batch processor). In this case, the sequential SAPT should be submitted in a way analogous to that described in Sec. 14 on pSAPT2K2, of course, requesting just one core.

## 10.1 Calculations of integrals and SCF energies

A SAPT2020 run starts with calculations of "atomic" integrals, i.e., integrals involving the basis functions, and proceeds to calculations of the SCF orbitals and orbital energies for monomers. Optionally, if the `opt1` keyword is set to `scfcp`, a counterpoise (CP) corrected supermolecular calculation of the HF-SCF interaction energy is also performed. The number and type of SCF calculations depend on the chosen approach to the basis set construction and on the selection of the supermolecular calculation. For a description of possible choices of basis sets, see the following subsections and Ref. 55.

For information on the input to the integral/SCF programs please refer to the original documentation for those codes. Some remarks on this subject can also be found in Sec. 9. For an ATMOL manual see

`http://www.theochem.ru.nl/∼pwormer` .

Please note that symmetry considerations are not built into the SAPT2020 codes, so the SCF packages should be asked to output the integrals with no symmetry assumptions.

The SAPT2020 codes make an implicit assumption that for each monomer the number of occupied orbitals is smaller or equal to the number of virtual orbitals and a crash will occur if this condition is not met. It is thus **not recommended** to run SAPT2020 calculations in **minimal basis sets**.

Regardless of the interface program, SAPT2020 is limited to 65535 molecular orbital indices (or 1024, if the setting `BEYOND1024=YES` was not used in the `Compall script`).

### 10.1.1 DCBS and DC$^+$BS approaches

If a dimer-centered basis set (DCBS), possibly including "bond" functions (denoted then by DC$^+$BS), is used and the supermolecular SCF interaction energy is not needed, then only two integral/SCF calculations, for monomer A and for monomer B, will be performed. The DCBS/DC$^+$BS approach means that the calculations for monomer X are performed using the orbital basis set consisting of *all* functions, those "belonging" to monomer X (i.e., centered on the nuclei of monomer X), those of the interacting partner (i.e., centered at the positions where the partner's nuclei are in the dimer), and the bond functions (in DC$^+$BS case). In the inputs, the bond functions and the functions of the interacting partner are typically connected with zero-charge centers and are sometimes called "ghost" orbitals. The script will then look for files `nameA.*` and `nameB.*` for the respective monomers. The number and full names of the files depend on the integral/SCF code used. Here is a list of files needed for some of the front-end codes:

ATMOL1024: `nameA.intinp`, `nameA.scfinp`, `nameB.intinp`, and `nameB.scfinp`

GAMESS: `nameA.inp` and `nameB.inp`

GAUSSIAN: `nameA.data` and `nameB.data`

DALTON: `nameA.dal`, `nameA.mol`, `nameB.dal`, and `nameB.mol`

MOLPRO: `name.molpro`

In each case, the file `nameP.data` will also be needed containing input for the post-SCF part of the calculation (`tran`, `cc`, and `sapt.x` stages). In a DC$^+$BS run, the variable `DIMER` in the namelist `TRN` in this file must be set to `T` (or `.TRUE.`).

If DCBS or DC$^+$BS is used and a supermolecular SCF interaction energy calculation *is* requested (the `scfcp` option is set), the script will additionally look for a dimer input file (or files) `name.*` (notice the convention: `nameA`, `nameB`, and `name` for the monomers A, B, and for the dimer, respectively). Thus, for example for ATMOL1024, the complete set of input files necessary for this type of run are: `name.intinp`, `name.scfinp`, `nameA.intinp`, `nameA.scfinp`, `nameB.intinp`, `nameB.scfinp`, and `nameP.data`.

In the DCBS or DC$^+$BS approach, the basis set specifications for all integral/SCF calculations are identical except for different charges set to zero in different files and for different numbers of electrons in each run. If the `scfcp` option is used, the file of two-electron integrals can be computed only once, during the dimer run, and the subsequent SCF calculations for monomers A and B can then use this file (most integral/SCF programs are flexible enough to allow such a route). Note, however, that the *one-electron* integrals must be computed separately for monomers A and B. If the `scfcp` option is not used, only the monomer A calculation needs to compute the two-electron integrals.

### 10.1.2 MCBS and MC$^+$BS approaches

An alternative, and strongly recommended, way of performing SAPT calculations is to use the so-called monomer-centered 'plus' basis set (MC$^+$BS) [55]. To understand this approach, first notice that the conceptually simplest (and most natural) method is to use in SAPT the monomer-centered basis set (MCBS), i.e., a basis that includes only functions that one would have used if the calculations involved only the energies and properties of a given monomer (calculations for monomer X involve only basis orbitals centered on this monomer). In the limit of infinite orbital basis set, the MCBS approach converges to the exact values for each SAPT correction. However, for several reasons this convergence is slow for some corrections [55]. The simplest cure for this problem is to use the DCBS or, even better, the DC$^+$BS approach. The advantage of such a method is a close relation to the supermolecular approach with the CP correction. The disadvantage is that the basis set is increased by a factor of two (between MCBS and DCBS methods for identical monomers). Reference [55] has shown that many of the functions used in DCBS/DC$^+$BS are not needed for the SAPT convergence. If a DCBS/DC$^+$BS is reduced to some intermediate size, it is called an MC$^+$BS. It has been recommended in Ref. 55 that MC$^+$BS's are constructed from an MCBS by adding all the midbond functions and only a part of the basis set on the interacting monomer (so-called 'farbond' functions). The simplest choice for the farbond part is the "isotropic" part of the basis set, i.e., orbitals with symmetries appearing in the occupied orbitals of constituent atoms (i.e., *s* part of the basis for H and He and *sp* part of the basis centered on the first and second row atoms). In practice, MC$^+$ basis sets match the accuracies of DC$^+$ bases and at the same time reduce several times the costs of SAPT calculations.

An MC$^+$BS approach requires a little more work with setting up the basis sets than a DC$^+$BS calculation. The reason is that in the former approach the basis functions appear in three different roles: a given basis function can belong only to monomer A, only to monomer B, or to both monomers. A calculation of the integrals in a set with repeated functions would be unnecessarily time and disk space consuming, but a method has been developed to avoid this problem. Below, the individual files needed for an MC$^+$BS calculation are listed first and then the dependence between the structure of these files and the control parameters in the file `nameP.data` is described.

If MC$^+$BS calculations are performed and the supermolecular SCF interaction energy is *not* requested (`scfcp` keyword not present), the `SAPT` script will look for input files `nameMA.*`, `nameMB.*` to perform integral/SCF calculations for monomers A and B, respectively. The input files for these runs should contain the basis of a given monomer plus the midbond and farbond functions on the "ghost" centers. Since the basis sets are different for A and B, the two-electron integrals from A cannot be reused for B. The eigenvalues and eigenvectors from these two runs are saved, whereas both one- and two-electron integrals are discarded. The script will next look for an input file `nameA.*` to calculate one- and two-electron integrals for monomer A in the DC$^+$BS equivalent of the MC$^+$BS used (this basis is identical to the one described in the previous subsection, but in some cases the functions have to be ordered in a special way or the proper "tag" parameters have to be specified in the file `nameP.data`, as discussed below). No SCF calculations are needed (if performed, the results will be discarded). This step is needed to produce the integrals for the SAPT calculations. Next the script will look for a file `nameB.*` to calculate one-electron integrals only for the monomer B in the DC$^+$ basis set (the two-electron integrals need not be calculated since these are identical as for monomer A). Again, no SCF step is needed. Thus, if for example ATMOL1024 is used as the integral/SCF program, the needed input files are: `nameMA.intinp`, `nameMA.scfinp`, `nameMB.intinp`, `nameMB.scfinp`, `nameA.intinp`, `nameB.intinp`, and `nameP.data`. In the last of these files, the variable `DIMER` in namelist `TRN` should be set to `F` (or `.FALSE.`).

If MC$^+$BS calculations are performed and the supermolecular HF-SCF interaction energy *is* requested (the `scfcp` option is used), the script will, as in the previous case, first look for input files `nameMA.*` and `nameMB.*` to perform integral/SCF runs for monomers A and B in an appropriate MC$^+$BS. Next, the script will look for the file(s) `name.*` to perform integral/SCF calculations for the dimer in the DC$^+$BS equivalent of the MC$^+$BS used. In the next two steps the files `nameA.*` and `nameB.*` containing the same DC$^+$ basis set will be used, as in the calculations described in the previous paragraph. However, now neither the monomer A nor B calculation need to compute the two-electron integrals since these are already available from the dimer calculation. Also, in contrast to the case of the previous paragraph, the SCF calculations are performed for each monomer since the total SCF energies are needed to compute the supermolecular HF-SCF interaction energy. The SCF orbital energies and coefficients from these runs are discarded. Thus, if for example ATMOL1024 is used as the integral/SCF program, the following input files are needed: `nameMA.intinp`, `nameMA.scfinp`, `nameMB.intinp`, `nameMB.scfinp`, `name.intinp`, `name.scfinp`, `nameA.intinp`, `nameA.scfinp`, `nameB.intinp`, `nameB.scfinp`, and `nameP.data`. If GAUSSIAN is used as the integral/SCF program, the input files needed are: `nameMA.data`, `nameMB.data`, `name.data`, `nameA.data`, `nameB.data`, and `nameP.data`. If DALTON is used, the following input files are needed: `nameMA.dal`, `nameMA.mol`, `nameMB.dal`, `nameMB.mol`, `nameA.dal`, `nameA.mol`, `nameB.dal`, `nameB.mol`, `nameP.data`, `name.dal`, and `name.mol`.

The possibility of skipping parts of integral/SCF calculations varies between programs. In the case of ATMOL1024, the calculations of two-electron integrals are omitted by simply including the `BYPASS TWO` statement in the proper `*.intinp` file and an SCF calculation is omitted by not executing the corresponding binary (`scf`). The two-electron files from a previous ATMOL1024 run will be recognized by a subsequent SCF run due to the naming convention. For DALTON, the appropriate keyword is `.NOTWO` and it can be used in the file `nameB.dal` in MC$^+$BS calculations. It appears that some integral/SCF programs, e.g., GAMESS, do not have an input option to skip the calculation of two-electron integrals.

There are two ways of arranging basis functions in an MC$^+$BS (or MCBS) run. The first way is known to work with ATMOL1024 and to fail with GAUSSIAN, DALTON, and MOLPRO. It may work with other packages, but it has not been tested. This method is chosen by specifying the keyword `BLKMB=T` in the `TRN` namelist read from the file `nameP.data` (this is also the default and therefore this keyword can be omitted). If this path is used, the basis functions in the DC$^+$BS-type input files specified above (`name.*, nameA.*`, and `nameB.*`) have to be ordered as follows:

$$pol_A \;\; iso_A \;\; mid \;\; iso_B \;\; pol_B$$

where $iso_X$ is the isotropic part of the basis set of monomer X (as defined above), $pol_X$ are the polarization functions on monomer X (orbitals with symmetries $p$ and higher for H and He and $d$ and higher for the first and second-row atoms), and $mid$ are midbond functions (optional). In the calculations producing the orbital energies and coefficients for monomers A and B (files `nameMA.*` and `nameMB.*`), the basis set should be ordered as:

$$pol_A \;\; iso_A \;\; mid \;\; iso_B$$

and

$$iso_A \;\; mid \;\; iso_B \;\; pol_B,$$

respectively. The method is in fact more general than the names $iso_X$ and $pol_X$ indicate. As it should be clear from the above, the basis set for each monomer can be divided into two arbitrary subsets. Examples of MC$^+$BS input files set up using this strategy can be found in the directories `./SAPT2020/examples/ATMOL1024/HF2_MCBS`, `./SAPT2020/examples/ATMOL1024/CO2D_MCBS`, and also `./SAPT2020/examples/ATMOL1024/ArH2O_MCBS`.

A more general method that works for all integral/SCF front-ends is the "tags" method. In this method, chosen by `BLKMB=F`, the ordering of functions is arbitrary, except that it has to be the same in all input files. Of course, the "MCBS files", `nameMA.*` and `nameMB.*` will contain only subsets of the whole basis. It is, however, required that the basis specifications in these files differ from the specification in the files `name.*`, `nameA.*`, and `nameB.*` by *only* the deletion of functions which belong exclusively to the other monomer, i.e., the sequence of functions in the subset remaining after such deletion is not altered with respect to the whole DC$^+$BS set. The role of a given function is specified by "tagging" it in the `TRN` namelist read from the file `nameP.data`. This is achieved by first setting the variable `BASIS` to a string containing letters `spdf...`, one letter for each shell in the DC$^+$BS-type basis (thus, for a part of the basis with $3s2p1d$, the string should be `sssppd`). Since the program expands each symbol into a number of orbitals, one has to provide information on the type of basis set, spherical or Cartesian, which is done by setting the variable `SPHG` to `.TRUE.` or `.FALSE.`, respectively. Note that when running SAPT2020 with GAMESS as a front-end, `SPHG` *must* always be set to `.FALSE.`, even if the `ISPHER=1` option was applied during the SCF calculations to enforce removal of spurious spherical components from the Cartesian basis used by GAMESS. The other string variable to be specified is `TAGS`. It has to contain exactly the same number of characters as the `BASIS` variable. The allowed characters are `a`, `b`, and `m`, denoting basis functions appearing only in the MC$^+$BS of monomer A (which can be the same as in $pol_A$ part of the basis discussed above), only in the MC$^+$BS of B (like $pol_B$), and in the MC$^+$BS sets of both monomers, respectively. Each of the variables `BASIS` and `TAGS` should end with a `+` sign denoting the end of a string.

For an example of using tags with GAMESS, see the files in the directory `./SAPT2020/examples/GAMESS/HF_NH3_MCBS`. This example does not use the *iso*/*pol* splitting, to show the more general character of the tags method. Here, in the files `HF_NH3.inp, HF_NH3A.inp`, and `HF_NH3B.inp`, the DC$^+$BS basis functions are entered in the order F ($5s3p2d$), H of HF ($3s2p$), midbond ($2s1p$), N ($5s3p2d$), H1 of $NH_3$ ($3s2p$), H2 of $NH_3$ ($3s2p$), and H3 of $NH_3$ ($3s2p$). Now, in this example, the MC$^+$BS for monomer A (HF) is constructed by deleting the last two $s$, the last $p$, and the last $d$ functions of N, as well as the last $s$ and $p$ functions on each H of $NH_3$ (see the file `HF_NH3MA.inp`). The functions deleted here happen to be the most diffuse ones for each symmetry, although in principle other choices could have been made as well. Similarly, the MC$^+$BS for monomer B ($NH_3$) is constructed from the whole DC$^+$BS set by deleting the last two $s$, the last $p$, the last $d$ functions of F, and the last $s$ and $p$ functions of HF's hydrogen (see the file `HF_NH3MB.inp`). Note that the sequence of basis functions in the input files is determined by the sequence of atomic centers and the sequence of functions within each center. Thus, simply deleting a contraction does not change the relative sequence in the remaining set, as required. The part of the `TRN` namelist corresponding to this setup is

```
BLKMB=F, SPHG=F,
BASIS='ssssspppddsssppsspssssspppddsssppsssppssspp+',
 TAGS='mmmaammamammamammmmmmbbmmbmbmmbmbmmbmbmmbmb+'
```

Note that the variable `SPHG` has been set to `.FALSE.` since GAMESS always generates integrals in the Cartesian basis (even if the SCF calculations are performed in variational space spanned by only pure spherical components, as it is done with the option `ISPHER=1` in GAMESS input files `*.inp`). If the "tags" method is used with some other SCF program, make sure that the value of `SPHG` matches the actual type of basis set for which the atomic integrals are generated.

Other examples of MC$^+$BS input files using the `BASIS` and `TAGS` options with GAMESS can be found in directories `./SAPT2020/examples/GAMESS/*_MCBS`. For ATMOL1024, GAUSSIAN, and MOLPRO, the examples are in `./SAPT2020/examples/ATMOL1024/ArHF_MCBS`, `./SAPT2020/examples/GAUSSIAN/CO2D_MCBS`, and `./SAPT2020/examples/MOLPRO/CO2D_MCBS`, respectively.

For large basis sets (several hundreds or more), the length of the strings `BASIS` and `TAGS` can cause problems with certain compilers (for instance, some versions of Portland Fortran have a limit of 299 characters for namelist strings). In such cases, alternative 250-character-long variables named `BASIS1`, `BASIS251`, `BASIS501`, etc. (and similarly for `TAGS`), can be used, so that, for instance, `BASIS251` contains elements 251–500 of the complete `BASIS` string. The current limit of `BASIS` and `TAGS` strings (defined either in one piece or using the 250-character segments) is 16384.

## 10.2 Input for post-Hartree-Fock part

The input for the transformation `tran`, the MBPT/CC code `cc`, and the proper SAPT program `sapt.x` is supplied in the file `nameP.data`. This input consists of a title line and three namelist sets. The first line is reserved for the title of the run (up to 80 characters). The three namelists that follow are `TRN`, which is for the input to the transformation program, `CCINP`, which passes information to the MBPT/CC program, and `INPUTCOR`, which informs the perturbation program which corrections are to be computed.

It should be noted that the exact syntax of a namelist statement depends on the platform. For example, on SGI and Linux platforms, each namelist should start with the name preceded by `&` (e.g., `&TRN`) and end with `&END`. On the other hand, IBM compilers require the forward slash (`/`) as the namelist end marker. Throughout this manual, the former convention will be followed.

### 10.2.1 Namelist TRN

The major function of this namelist is to tell the `tran` program which integral/SCF package is in use. Currently supported codes are listed in Table 1. In addition, a number of legacy codes should still work, although several of them have not been used for years.

Out of the 10 possible integral/SCF selections the user should select only one by setting the corresponding variable equal to `.TRUE.` and all the other variables equal to `.FALSE.` The selection variables are given by:

- `ISITANEW` selects ATMOL1024
- `ISITGAMS` selects GAMESS
- `ISITG03` selects GAUSSIAN98, GAUSSIAN03, or GAUSSIAN09
- `ISITDALT` selects DALTON 2.0
- `ISITMOLP` selects MOLPRO

- ISITG94 selects GAUSSIAN94

- ISITCADP selects CADPAC.

- ISITACES selects ACES

- ISITATM selects the older version of ATMOL

- ISITORCA selects ORCA [only works in SAPT(DFT)].

Of course, selecting a given SCF package in the TRN namelist will work only if SAPT2020 has been *compiled* for use with this package (see Sec. 8).

The variable DIMER is used to determine whether the transformation type is dimer (default) or monomer (DIMER=.FALSE.). The dimer-type transformation is chosen for DCBS/DC$^+$BS calculations and the monomer-type one for MC$^+$BS calculations.

If monomer type of transformation is set, several other options become available. These are BLKMB, BASIS, TAGS, and SPHG variables connected with the tags system. The use of these variables was described in Sec. 10.1.2. Also, a description is provided in comments of the subroutine mkoffset of the module trans.F (in subdirectory ./SAPT2020/tran).

Setting the OUT variable to .TRUE. will give more extensive printing from tran, showing mainly memory partitioning and the input vectors.

The variable TOLER sets the threshold for writing the transformed integrals to disk. The input is an integer $n$ which is then used to discard all integrals which are less than $10^{-n}$ in absolute value. An input of –1 gives a threshold of identically 0. We recommend values slightly tighter than normally used for similar purposes in isolated molecules calculations, of about $10^{-12}$ around the van der Waals minimum and progressively smaller as one proceeds farther away from the minimum. Note that the usual default thresholds in most integral/SCF programs will be larger than this value and must therefore be lowered so that the atomic integral files which are the input to the transformation contain sufficiently small integrals. It is also advisable to tighten the threshold for the convergence on the density matrices in order to increase the accuracy of the SCF eigenvectors.

If the SCF program used is ATMOL1024 and if the two-electron integrals produced by this program are stored in more than one file, the number of such files has to be given in TRN as the variable NMFILES. For example, NMFILES=3 will tell the transformation to look for ATMOL1024 integrals in the files MT3, MT4, and MT5. The most likely use of this option is on old Linux platforms which do not support files larger than 2 Gbytes so that the integrals have to distributed over several smaller files.

Finally, the MEMTRAN variable can be used to dynamically allocate memory for the transformation section (tran) of SAPT2020. As a default, the memory for the tran program is set automatically depending on the basis size. If possible, the memory should be set large enough so that the transformation program uses the faster "in core" path. When `DIMER=.TRUE.`, an even faster algorithm is switched on if twice as much memory as for the ordinary "in core" algorithm is available. The amount of memory needed for the available transformation paths can be calculated beforehand using the program `memcalc` in `./SAPT2020/bin` (see Sec. 10.7). Declaring more than the amount needed for the "fast in core" algorithm when `DIMER=.TRUE.`, or more than needed for the standard "in core" approach for `DIMER=.FALSE.`, makes no difference for the performance of transformation (and may increase the probability of crashing due to requesting more memory than available on a system at a given time). The transformation will work in smaller memory, except that the slower "out of core" pathway will be chosen (see the source module `memory.F` for more details). One can read from the transformation output of a test run whether the "out of core" or "in core" (faster) pathway has been chosen and what was the largest size of memory used. This method can also be used to adjust the `MEMTRAN` variable appropriately (look for lines containing phrases similar to:

```
...  Mem:  8182228 CPU: 463.7
```

or

```
AABBOVVV Integrals.  Out of core.  2 passes Mem:  29848693
```
).

The minimal information absolutely necessary for the namelist `TRN` is given by: `&TRN ISITx=T &END`, where x is one of the symbols denoting the integral/SCF program listed above.

### 10.2.2 Namelist `CCINP`

The purpose of this namelist is to pass information to the MBPT/CC program which generates the required cluster amplitudes for the intermolecular perturbation theory program `sapt.x`. The namelist variable `CCPRINT` tells the program whether or not to do extra printing. This printing involves information about the memory partitioning and the integral class which is currently being read in. The namelist variable `VCRIT` gives the tolerance for retaining cluster amplitudes. Here we recommend a tolerance of $1 \times 10^{-10}$.

The variables `TOLITER` and `TOLAMP` determine the convergence criteria of the CC iterations if the converged amplitudes are required, *i.e.*, if the variable `CONVAMP` (see namelist `INPUTCOR`) is set to `.TRUE.` In such a calculation, the CC program continues the iterations until one of the following two conditions is fulfilled:

1. The iteration number, $k$, exceeds 39 (no convergence).

2. All of the conditions listed below are met:

$$DE^{(k)} = \left| \frac{E_{\rm CC}^{(k)} - E_{\rm CC}^{(k-1)}}{E_{\rm CC}^{(k)} - E_{\rm SCF}} \right| \quad < \quad \texttt{TOLITER}$$

$$D(D)^{(k)} = \sqrt{\frac{\sum_{ijab}(t_{ij}^{ab\,(k)} - t_{ij}^{ab\,(k-1)})^2}{o^2v^2}} \quad < \quad \texttt{TOLAMP}$$

$$D(S)^{(k)} = \sqrt{\frac{\sum_{ia}(t_{i}^{a\,(k)} - t_{i}^{a\,(k-1)})^2}{ov}} \quad < \quad \texttt{TOLAMP}$$

where $t_{ij}^{ab}$ and $t_i^a$ are the double- and single-excitation amplitudes, respectively. Note that $DE^{(k)}$ is the *relative* correlation energy change. The default values are `TOLITER`=$10^{-5}$ and `TOLAMP`=$10^{-6}$.

DIIS convergence acceleration in the coupled cluster iterations can be turned off by the `DIIS=F` directive. DIIS is turned on by default (recommended). The variables `RPA`, `CCD`, `CCSD`, and several others can be set to `.TRUE.` or `.FALSE.` to choose one of the possible forms of CC theory. At this moment only `CCSD=T` and `CCD=T` are guaranteed to work and are needed for the SAPT corrections. Since `CCSD=T` and all other variables of this type are false by default, these keywords can be omitted from the namelist. The variables `SITEA` and `SITEB` allow to perform MBPT/CC calculations for one of the monomers only (for testing purposes). The default values are `.TRUE.` and therefore these variables can be omitted from the list. The variable `WRTEACH` allows writing of the cluster amplitudes from each iteration into separate disk files for future use. It is `.FALSE.` by default and is not needed for the ordinary SAPT run; however, it must be set to `.TRUE.` if one wants to calculate the dispersion energy at the CCD level - see Sec. 10.4. Finally, the logical variable `AOCC` switches on/off the atomic-orbital (AO) based algorithm for calculating the most time-consuming four-virtual contribution to the CCSD amplitudes. This slows down the CC calculations slightly, but the `tran` program requires much less time since no four-virtual transformation is needed. At present, `AOCC=.TRUE.` works only when ATMOL1024 or MOLPRO has been used as the integral/SCF interface, and when either the DCBS/DC$^+$BS approach is used, or, in case of an MC$^+$BS calculation, the basis functions are properly grouped (i.e., the variable `BLKMB` in the `TRN` namelist is equal to its default value of `.TRUE.`). Note that this implies that the `AOCC=.TRUE.` algorithm with the MOLPRO interface works only for the DCBS/DC$^+$BS approach. The default value of `AOCC` is `.TRUE.` if the above conditions are satisfied and `.FALSE.` otherwise, so an explicit specification of this variable is not needed unless one wants to turn off the AO-based pathway for ATMOL1024 or MOLPRO interfaces. The defaults should be sufficient for this namelist, so all that is absolutely necessary to specify would be a statement of the form: `&CCINP &END`, i.e., no input.

### 10.2.3 Namelist `INPUTCOR`

This namelist tells the SAPT program which of the perturbation theory corrections are to be computed. All the variables are by default set to `.FALSE.`, so only those corrections that one wants to be computed have to be mentioned in the namelist. The list of variables and of the associated currently available corrections is as follows:

1. `E1TOT` — $E^{(10)}_{\rm elst}$ and $E^{(10)}_{\rm exch}$. By default, the so-called $S^2$ approximation to $E^{(10)}_{\rm exch}$ is also computed. This computation can be turned off by setting `E1S2=.FALSE.` in the `INPUTCOR` namelist.
2. `E2IND` — $E^{(20)}_{\rm ind}$
3. `E2INDR` — $E^{(20)}_{\rm ind,resp}$
4. `EEX2I` — $E^{(20)}_{\rm exch-ind}$
5. `EEX2IR` — $E^{(20)}_{\rm exch-ind,resp}$
6. `EEX2D` — $E^{(20)}_{\rm exch-disp}$
7. `EEX2` — $E^{(20)}_{\rm exch-disp}$ and $E^{(20)}_{\rm exch-ind}$, i.e., `EEX2=.TRUE.` implies `EEX2I=.TRUE.` and `EEX2D=.TRUE.`
8. `EEX2R` — $E^{(20)}_{\rm exch-disp}$ and $E^{(20)}_{\rm exch-ind,resp}$, i.e., `EEX2R=.TRUE.` implies `EEX2IR=.TRUE.` and `EEX2D=.TRUE.`
9. `E12` — $E^{(12)}_{\rm elst}$
10. `E12R` — $E^{(12)}_{\rm elst,resp}$
11. `E13PL` — $E^{(13)}_{\rm elst}$
12. `E13PLR` — $E^{(13)}_{\rm elst,resp}$
13. `E11` — $E^{(11)}_{\rm exch}$
14. `E111` — $E^{(111)}_{\rm exch}$
15. `E12X` — $E^{(120)}_{\rm exch} + E^{(102)}_{\rm exch}$
16. `CONVAMP` — $E^{(1)}_{\rm exch}$(CCSD)
17. `E2DSP` — $E^{(20)}_{\rm disp}$
18. `E21D` — $E^{(21)}_{\rm disp}$
19. `E22D` — $E^{(22)}_{\rm disp}$
20. `EMP2` — $E^{(02)}$ = MBPT2 correction to monomer's correlation energy (for tests only)

21. `E22I` — ${}^{t}E^{(22)}_{\rm ind}$ (see Ref. 2)

22. `E300D` — $E^{(30)}_{\rm disp}$

23. `E3IND` — $E^{(30)}_{\rm ind}$

24. `DSPIND` — $E^{(30)}_{\rm ind-disp}$

25. `E30XD` — $E^{(30)}_{\rm exch-disp}$

26. `E30XDSDQ` — $E^{(30)}_{\rm exch-disp}$ without the most time-consuming 'triples' term

27. `E30XI` — $E^{(30)}_{\rm exch-ind}$

28. `DSPIX` — $E^{(30)}_{\rm exch-ind-disp}$

29. `E3INDR` — $E^{(30)}_{\rm ind,resp}$ and a scaled approximation to $E^{(30)}_{\rm exch-ind,resp}$

Notice that the electrostatic energies $E^{(1i)}_{\rm elst}$ are sometimes denoted as $E^{(1i)}_{\rm RS}$ as these terms result from the Rayleigh-Schrödinger perturbation theory that in this context is often called *polarization* theory. The acronym "pol" in several places in the program, as well as the letters "PL" in `E13PL`, are resulting from this terminology. If ${}^{t}E^{(22)}_{\rm ind}$ is requested, its exchange counterpart will be estimated from the formula of Eq. (7). The triple superscripts appearing for example in the correction $E^{(111)}_{\rm exch}$ denote the order with respect to the operators $V$, $W_A$, and $W_B$, respectively.

The programs calculating the following corrections either have not been sufficiently tested, or are known to contain errors. Therefore, calculations of the following corrections are *not* recommended:

1. `E122` — $E^{(122)}_{\rm elst}$

2. `E14PLR` — $E^{(140)}_{\rm elst,resp} + E^{(104)}_{\rm elst,resp}$ (without triples)

3. `TE14` — Triples for $E^{(14)}_{\rm elst,resp}$

4. `E1CC` — calculates the electrostatic energy from the formulas analogous to those appearing in $E^{(12)}_{\rm elst}$ using the converged CCSD amplitudes in place of the second-order ones. Instead, use $E^{(1)}_{\rm elst,resp}$(CCSD) described in Sec. 10.5.

Six variables are provided to select groups of corrections: `SAPT0`, `SAPT2`, `SAPTNOCC`, `DELTASCF`, `SAPTKS`, and `SAPT`, and only one of them should be set to `.TRUE.` The settings `SAPT2=T` and `SAPT=T` select the groups of corrections defined by Eqs. (6) and (8), respectively. The choice `SAPTNOCC=T` is equivalent to `SAPT=T` except that the intramonomer correlation contribution to $E^{(1)}_{\rm exch}$ is approximated as $E^{(11)}_{\rm exch} + E^{(12)}_{\rm exch}$ instead of $\epsilon^{(1)}_{\rm exch}$(CCSD). The choice `SAPT=T` is approximately equivalent in accuracy of predictions to using supermolecular MBPT through fourth order. It has been used for most published recent SAPT calculations for small and medium-size systems. The choice `SAPT2=T`, equivalent to MBPT2, requires significantly less computer time and can be recommended for larger dimers if time of calculations at the `SAPT=T` level becomes an issue. In most cases, we recommend that the $\delta E^{\rm HF}_{\rm int,resp}$ term defined by Eq. (5) is included as implied by Eq. (6) (this requires the `scfcp` keyword as a command-line parameter to the `SAPT` script). Only for nonpolar or nearly nonpolar monomers this term should be omitted, see Ref. 39 and the text below for a discussion of this issue.

The choice `SAPT0=T` selects all available SAPT corrections of the first and second order in $V$ and of zeroth order in $W$, i.e.,

$$E^{\rm SAPT0}_{\rm int} = E^{(10)}_{\rm elst} + E^{(10)}_{\rm exch} + E^{(20)}_{\rm ind,resp} + E^{(20)}_{\rm exch-ind,resp} + E^{(20)}_{\rm disp} + E^{(20)}_{\rm exch-disp}. \qquad (10)$$

This level is recommended for calculations for very large systems when a higher level of theory would be too time-consuming (however, note that SAPT(DFT) described in Sec. 16 would offer much better accuracy at similar computational costs). One can expect that this level of theory may introduce about 20-30% errors with respect to the exact interaction energies, but such an accuracy can be acceptable for large systems. Also the errors due to the basis set truncation will most likely be of the same order of magnitude for systems that large. For increased accuracy, we recommend the addition of the $\delta E^{\rm HF}_{\rm int,resp}$ term.

The choice `DELTASCF=T` selects all SAPT corrections necessary for computing the $\delta E^{\rm HF}_{\rm int,resp}$ term, i.e. $E^{(10)}_{\rm elst}$, $E^{(10)}_{\rm exch}$, $E^{(20)}_{\rm ind,resp}$, and $E^{(20)}_{\rm exch-ind,resp}$. This setting should be selected when only $\delta E^{\rm HF}_{\rm int,resp}$ is required since it skips calculations of the dispersion and exchange-dispersion corrections. Such separate calculations of $\delta E^{\rm HF}_{\rm int,resp}$ may be required for certain SAPT(DFT) jobs, see Sec. 16.3. Since $\delta E^{\rm HF}_{\rm int,resp}$ is calculated automatically also for `SAPT0`, `SAPT2`, `SAPTNOCC`, and `SAPT` (as long as the keyword `scfcp` has been specified upon the submission of the SAPT script), it is not necessary to select `DELTASCF=T` when the former options are used. For additional keywords required for SAPT(DFT), see Sec. 16.2.

The `SAPTx` variable takes precedence over the settings of individual corrections in the sense that a correction included in a given level will be computed even if it is explicitly set to `.FALSE`. However, a correction not included will be computed if it is explicitly set to `.TRUE.`

The variable `CONVAMP` selects the use of the converged CCSD amplitudes in the expressions analogous to those appearing in the corrections $E^{(111)}_{\rm exch}$, $E^{(120)}_{\rm exch}$, and $E^{(102)}_{\rm exch}$. Setting this variable to `.TRUE.` results in the calculation of the $\epsilon^{(1)}_{\rm exch}$(CCSD) correction regardless of whether $E^{(111)}_{\rm exch}$, $E^{(120)}_{\rm exch}$, and $E^{(102)}_{\rm exch}$ terms are calculated or not. The logical variable `DCONVAMP` selects the calculation of the dispersion energy with the inter- and intramonomer correlation treated at the CCD level. However, this energy must be computed in a separate run, see Sec. 10.4.

The six leading components of the third-order SAPT energy, Eq. (9), are calculated when the variables `E3IND`, `DSPIND`, `E300D`, `E30XI`, `DSPIX`, and `E30XD`, respectively, are set to `.TRUE.` It is also possible to calculate the relaxed third-order induction correction $E^{(30)}_{\rm ind,resp}$ [36] by setting `E3INDR=.TRUE.` In this case, the corresponding relaxed exchange-induction energy $E^{(30)}_{\rm exch-ind,resp}$ has to be estimated by a scaling of the nonrelaxed quantity,

$$E^{(30)}_{\rm exch-ind,resp} = E^{(30)}_{\rm exch-ind} \cdot \frac{E^{(30)}_{\rm ind,resp}}{E^{(30)}_{\rm ind}}. \tag{11}$$

Note that the polarization corrections, especially $E^{(30)}_{\rm ind}$ (or $E^{(30)}_{\rm ind,resp}$), tend to cancel to a large extent with the corresponding exchange corrections, so calculating also the latter is highly recommended. The effects described by $E^{(30)}_{\rm ind,resp}$ and $E^{(30)}_{\rm exch-ind,resp}$, together with the higher-order induction and exchange-induction contributions, are approximately included in the $\delta E^{\rm HF}_{\rm int,resp}$ term resulting from the supermolecular Hartree-Fock interaction energy. For the interactions of polar molecules, where the induction component of the interaction energy plays an important role, $\delta E^{\rm HF}_{\rm int,resp}$ tends to provide a more accurate description of the high-order induction interactions than the third-order approximation. For nonpolar and nearly nonpolar systems, the inclusion of $E^{(30)}_{\rm ind}$ and $E^{(30)}_{\rm exch-ind}$, or of $E^{(30)}_{\rm ind,resp}$ and $E^{(30)}_{\rm exch-ind,resp}$, usually gives more accurate results than the addition of $\delta E^{\rm HF}_{\rm int,resp}$. Whereas the corrections $E^{(30)}_{\rm ind}$ , $E^{(30)}_{\rm ind,resp}$, $E^{(30)}_{\rm ind-disp}$, $E^{(30)}_{\rm exch-ind}$, and $E^{(30)}_{\rm exch-ind-disp}$ are fairly inexpensive, the terms $E^{(30)}_{\rm disp}$ and $E^{(30)}_{\rm exch-disp}$ scale like $o^2v^4$ and $o^3v^4$, respectively, with the numbers $(o, v)$ of occupied and virtual orbitals for a given monomer. Moreover, these two corrections require the mixed-monomer four-virtual integrals over the molecular orbital basis that are expensive to produce. Because of this, SAPT has a built-in mechanism to calculate these terms in a semi-direct way, utilizing the integrals over AOs and removing the need for a mixed-monomer four-virtual transformation. Currently, this mechanism, just like the AO-based algorithm for the four-virtual diagram in the monomer CCSD calculations (controlled by the variable `AOCC` in the `CCINP` namelist), works only when ATMOL1024 or MOLPRO are employed as integral and SCF front-ends, and when either the DCBS/DC$^+$BS approach is used, or, in case of an MC$^+$BS calculation, the basis functions are properly grouped (i.e., the variable `BLKMB` in the `TRN` namelist is equal to its default value of `.TRUE.`). If this is the case, the semi-direct calculation is chosen by default; the user can change this behavior by setting `DIRECTE3=.FALSE.` in the `INPUTCOR` namelist. The computation of the most expensive $o^3v^4$ contribution to the $E^{(30)}_{\rm exch-disp}$ correction can be omitted by specifying `E30XD=.FALSE.` and `E30XDSDQ=.TRUE.` This contribution usually constitutes about 10% of $E^{(30)}_{\rm exch-disp}$ and converges fast with the basis set size [39]. The remaining part of $E^{(30)}_{\rm exch-disp}$ scales like $o^2v^4$ with the number of orbitals.

The variables `FROZEN`, `NFOA`, `NFOB`, `NFVA`, and `NFVB` control the range of orbitals used for electron excitations [41]. If `FROZEN` is set to `.FALSE.`, an all-electron calculation is performed and the values of `NFOA`, `NFOB`, `NFVA`, and `NFVB` are irrelevant; this is the default setting. If `FROZEN` is set to `.TRUE.`, `NFOA` lowest orbitals on A and `NFOB` lowest orbitals on B are treated as core and no excitations out of these orbitals are taken into account. Additionally, `NFVA` highest virtual orbitals on A and `NFVB` highest virtual orbitals on B are also excluded from the excitation space. One may set `NFVA=NFVB=0` to perform a conventional frozen-core calculation with excitations onto all virtual orbitals allowed. Setting `NFVA=NFOB` and `NFVB=NFOA` is another reasonable choice. Note that setting `FROZEN=.TRUE.` affects not only the SAPT program, but also the integral transformation and the calculation of monomer CC amplitudes. However, the parameters `FROZEN`, `NFOA`, `NFOB`, `NFVA`, and `NFVB` need, and can, be set only in the `INPUTCOR` namelist.

SAPT2020 can be used with effective core potentials (ECPs). So far, this option has only been tested with the MOLPRO front end, however, using other ECP-enabled integral front ends interfaced with SAPT2020 should be pretty straightforward. Contrary to a frozen-core SAPT run, the ECP effects are taken care of at the integral program level and no special input or options are needed for SAPT. In particular, the variable `FROZEN` in the `INPUTCOR` namelist should be set to its default value of `.FALSE.` unless freezing of a larger core on top of a small-core ECP is requested. The frozen-core and ECP SAPT approaches are described in Ref. 41. Note that when using SAPT with ECPs, the inclusion of the $\delta E^{\rm HF}_{\rm int,resp}$ term, Eq. (5), is strongly recommended [41] (this requires the `scfcp` keyword as a command-line parameter to the `SAPT` script).

The relativistic contributions to the SAPT interaction energy can be estimated by using relativistic ECPs. Alternatively, one can calculate a relativistic SAPT interaction energy by employing the second-order Douglas-Kroll-Hess relativistic one-electron Hamiltonian as implemented in MOLPRO. This requires a special syntax of the MOLPRO integral and SCF input: please see the example in the directory `MOLPRO/ArHF_AVDZ` for details.

The variable `PRINT` is used to print more information about intermediate results and memory partitioning. The variable `MEMSAPT` can be used to dynamically allocate memory for the perturbation theory stage (`sapt.x`) of SAPT2020. As a default, the memory for the `sapt.x` program is set automatically according to the basis size. The amount of memory required by the `sapt.x` program may be computed using the `memcalc` utility, as described in Sec. 10.7.

## 10.3 How to read the output

The values of all the calculated SAPT corrections are summarized at the end of the output file in the section entitled `Summary Table`. An example of the `Summary Table` can be found in Appendix D. The first part of the table lists the numbers of orbitals, the Cartesian geometry of the dimer, and the SCF energies of the monomers and the dimer (if computed) obtained in the full DC$^+$BS. What follows is a set of low-order SAPT corrections which, when summed up, approximate the supermolecular SCF interaction energy (also printed as `E^{HF}_{int}` if the keyword `scfcp` was used on job submission). The quantity `SAPT SCF_{resp}` is equal to the sum of `E^{(10)}_{elst}`, `E^{(10)}_{exch}`, `E^{(20)}_{ind,resp}`, and `E^{(20)}_{ex-ind,r}`, i.e., the first 4 terms on the rhs of Eq. (5). The quantity `\delta^{HF}_{int,r}` represents the last term in this equation. If the third-order induction and exchange-induction corrections have been calculated, the quantity `SAPT SCF(3)_{resp}`, equal to a sum of `SAPT SCF_{resp}`, `E^{(30)}_{ind}`, and `E^{(30)}_{exch-ind}`, is displayed, and the appropriate third-order effects are subtracted from `\delta^{HF}_{int,r}` to form the `\delta3^{HF}_{int,r}` quantity. If the non-response corrections `E^{(20)}_{ind}` and `E^{(20)}_{ex-ind}` have been computed, the corresponding non-response approximation to the SCF interaction energy, `SAPT SCF`, and `\delta^{HF}_{int}` are also given.

The `CORRELATION` part of the `Summary Table` contains all the computed SAPT corrections of order higher than zero in the intramonomer correlation operator, and the dispersion and induction-dispersion energies, as defined in Sec. 3. Two partial sums are also provided, `SAPT_{corr,resp}` and `SAPT_{corr}`. The first of these quantities is the sum of `\eps^{(1)}_{elst,r}(k)`, `\eps^{(1)}_{exch}(k)` or, if available, `\eps^{(1)}_{exch}(CCSD)`, `^tE^{(22)}_{ind}`, `^tE^{(22)}_{ex-ind}`, `E^{(2)}_{disp}(k)`, `E^{(20)}_{exch-disp}`, `E^{(30)}_{disp}`, `E^{(30)}_{ind-disp}`, `E^{(30)}_{exch-disp}`, and `E^{(30)}_{exch-ind-disp}`. The definition of the second partial sum is completely analogous, except that `\eps^{(1)}_{elst}(k)` is used. Note that if any of the corrections appearing in the definition of a partial sum is not computed (for example, because it has not been requested in the namelist `INPUTCOR`), this partial sum will not contain this correction. For example, if only `SAPT=T` is requested in the `INPUTCOR` namelist, the non-response electrostatic corrections, as well as the components of $E^{(30)}$, will not be computed and the quantity `SAPT_{corr}` will not contain any correlation contribution to electrostatics, and neither `SAPT_{corr}` nor `SAPT_{corr,resp}` will include any contribution of the third order in the intermolecular interaction operator.

If the $\tilde{E}^{(1)}_{\rm elst}(\rm CCSD)$ energy is requested by setting `E1CC=T` in the namelist `INPUTCOR`, the quantity $\epsilon^{(1)}_{\rm elst}(\rm CC) = \tilde{E}^{(1)}_{\rm elst}(\rm CCSD) - E^{(10)}_{\rm elst}$ will be reported in the correlation section of `Summary Table`, but it will *not* be counted as a part of `SAPT_{corr,resp}`. An example of using the `E1CC` option can be found in `examples/GAMESS/HF2_MCBS`.

Finally, the last set of entries in the `Summary Table` gives the hybrid interaction energies, i.e., the sums of the supermolecular SCF and the correlation parts, represented by `SAPT_{corr,resp}` or `SAPT_{corr}`. If all the relevant corrections have been computed (such as when one of the `SAPTx` variables is set to `.true.` in `INPUTCOR`), the quantity `SCF+SAPT_{corr,resp}` should be considered the recommended SAPT interaction energy. If the third-order induction and exchange-induction energies have been computed, the sums `SAPT SCF(3)+SAPT_{corr}` and `SAPT SCF(3)_{resp} +SAPT_{corr,resp}`, with the terms `\delta^{HF}_{int}` and `\delta^{HF}_{int,r}`, respectively, replaced by their third-order component `E^{(30)}_{ind}+E^{(30)}_{exch-ind}`, are also displayed.

## 10.4 Calculations of dispersion energy at CCD+ST(CCD) level

An algorithm for calculating the second-order dispersion energy with the inter- and intramonomer correlation effects included at the CCD level has been developed in Ref. 40. This energy cannot be calculated along with the other SAPT corrections: instead, a separate run of the `SAPT` script must be performed. For this run, the user needs to supply a set of integral and SCF inputs just like for the ordinary SAPT calculation, and the file `nameP.data` must contain, apart from the `TRN`, `CCINP`, and `INPUTCOR` namelists, a namelist `E2DINP` which contains the necessary input for the `e2disp` program. IMPORTANT! The `e2disp` program only works with the older transformation subroutines, so it can only be compiled with the `BEYOND1024` option set to `NO`. Please also note that the integral files produced by the transformation programs compiled with `BEYOND1024=YES` won't work with `e2disp`. Since the `BEYOND1024=NO` option would break some modern functionalities, we recommend making a separate installation of SAPT2020 exclusively for this application.

The logical variables `RPA`, `MPENER`, `CCD`, and `CCSD` in the `E2DINP` namelist control the level of theory employed in the `e2disp` program. Currently, only the `CCD=.TRUE.` version works, so this variable should be set to `.TRUE.` and all others to `.FALSE.` Note that the default values are `CCD=.TRUE.` and `RPA=MPENER=CCSD=.FALSE.` so that no explicit specification of these variables in the `E2DINP` namelist is necessary. The logical variable `PT` governs the perturbative/nonperturbative methodology of the iterations; currently, only the first one works so this variable should be always set to `.TRUE.` The logical variables `PMEM`, `PNRM`, `PKL`, `PSEP`, and `PCOMP` control the printing of various intermediate quantities. Finally, the variable `TOLITER` states the (relative) convergence threshold for the dispersion iterations; the recommended default is `TOLITER=1.0d-5`.

The CCD+ST(CCD) path of the calculation is turned on by specifying `DCONVAMP=.TRUE.` in the namelist `INPUTCOR`. Note that this turns off all SAPT corrections other than the second-order dispersion energy since the values of some of these corrections would be incorrect when CCD, rather than CCSD, amplitudes for monomers are calculated. Note also that the frozen core (`FROZEN=.TRUE.`) option has not been implemented for the CCD+ST(CCD) dispersion energy.

The namelist `CCINP` controlling the behavior of the CC program also needs to be adjusted when one performs the calculation of the dispersion energy at the CCD+ST(CCD) level. First of all, the single-excitation contribution needs to be omitted from the calculation, i.e. *both* `CCD=.TRUE.` and `CCSD=.FALSE.` need to be specified in the `CCINP` namelist. Furthermore, partial amplitudes from each coupled-cluster iteration need to be saved for the further use in the `e2disp` program. To save these amplitudes, one needs to set the option `WRTEACH` to `.TRUE.` Finally, a sufficiently tight convergence threshold for the CCD equations is required: we recommend setting `TOLITER=1.0d-9` in the `CCINP` namelist.

The computation of the CCD+ST(CCD) dispersion energy proceeds as follows. After the integral transformation, the CC program is invoked to produce CCD amplitudes for both monomers. Next, the `e2disp` program is run which perturbatively calculates the converged intermolecular dispersion amplitudes and the CCD part of the dispersion energy. Finally, the amplitudes are used by the SAPT program, with the option `DCONVAMP=.TRUE.` specified in the `INPUTCOR` namelist, to calculate the singles and triples contribution to the dispersion energy with converged doubles amplitudes, i.e., the ST(CCD) term. The final result for the CCD+ST(CCD) dispersion energy is then listed in the SAPT summary table.

## 10.5 Calculation of electrostatic energy from relaxed CCSD densities

In order to compute the energy $E^{(1)}_{\rm elst,resp}$(CCSD) from the relaxed CCSD monomer densities, it is necessary to use a special script `ccsddSAPT` from the `bin` subdirectory. At present, such a calculation can only be performed using GAMESS as the SCF program.

### 10.5.1 DCBS calculation

In the DCBS or DC$^+$BS approach, the user needs to supply a set of GAMESS SCF input files, just like for the ordinary SAPT calculation in DCBS or DC$^+$BS basis. The namelist `TRN` in the file `nameP.data` has to set `ISITGAMS=T` and `DIMER=T`. In the namelist `INPUTCOR`, the option `CONVAMP=.TRUE.` must be requested and the namelist `CCINP` has to be set up so that the CCSD calculations for the monomers are performed (i.e., the variables `CCSD` and `CCD` should be either omitted or set to `CCSD=.TRUE.` and `CCD=.FALSE.`). The computation of the $E^{(1)}_{\rm elst,resp}$(CCSD) electrostatic energy using the `ccsddSAPT` script proceeds as follows. After the integral transformation, the `ccsdt` program is invoked to produce the CCSD amplitudes for both monomers. Next, the `ccsdm` program is run twice (for each monomer) to calculate the relaxed CCSD densities. The latter are stored in the AO representation in the unformatted sequential files `ccsd_dena.out.A` and `ccsd_dena.out.B`. The obtained densities are then used by the program `e1dcbs`, together with the atomic one- and two-electron integrals, to compute $E^{(1)}_{\rm elst,resp}$(CCSD), reported in the output as `CCSD electrostatics:`. The correction $E^{(10)}_{\rm elst}$ is computed here as well and reported as `SCF electrostatics:`. The script `ccsddSAPT` then proceeds to invoke the `sapt.x` module, which calculates the other SAPT corrections, as specified by namelist `INPUTCOR`.

An example of a DCBS calculation of the electrostatic energy from the CCSD densities can be found in `examples/GAMESS/CO_E1DEN_DCBS`.

### 10.5.2 MCBS calculation

In the (pure) MCBS basis set (i.e., without midbond or farbond functions), it is possible to precompute the electron densities for both monomers and then reuse them, after suitable translations and rotations, for a whole set of dimer geometries. In this way, the expensive calculation of CCSD densities has to be done only once per monomer. A calculation of this type can be accomplished using the `ccsddSAPT` script described above and a separate program `elstdenrot`, which computes the electrostatic energies using the AO density files produced by `ccsdm` and basis set information extracted for this purpose from GAMESS output files. The script `ccsddSAPT` follows the appropriate path once the option `DIMER=F` is detected in the `TRN` namelist. The steps involved in the calculation are the following:

1. Assuming that the name of the job is `name`, prepare the files `nameA` and `nameB` containing the specifications of the geometries and basis sets of monomers A and B in GAMESS(US) format. Remember to put blank lines after the basis set of each atom, including the blank line at the end of the file. From these files, the `ccsddSAPT` script will construct the GAMESS input files needed to calculate the SCF vectors and generate integrals. A spherical basis set will be assumed (i.e., `ISPHER=1`) and the name of the job will be used as the comment line. Geometries of the monomers specified in `nameA` and `nameB` should correspond to whatever is considered to be the initial configuration of a given monomer, i.e., the COM (or other characteristic point around which rotations will be performed) should coincide with the origin of the coordinate system, and all Euler angles describing the orientation of the monomer will be assumed zero at this geometry.

2. Prepare the file `nameP.data`, as for the regular MCBS SAPT calculation (note that in the namelist `TRN`, the variables `basis` and `tags` have to be specified for such a calculation, as well as `DIMER=F`). In the namelist `INPUTCOR`, the option `CONVAMP=.TRUE.` must be requested and the namelist `CCINP` has to be set up so that the CCSD calculations for the monomers are performed (i.e., the variables `CCSD` and `CCD` should be either absent or set to `CCSD=.TRUE.`, `CCD=.FALSE.`). The corrections specified in `INPUTCOR` will be calculated and printed in the `Summary Table` for one dimer geometry, obtained by shifting the position of monomer B by 10 bohr along the positive $z$ axis with respect to the position specified in the `nameB` file. The $E^{(10)}_{\mathrm{elst}}$ result from the `Summary Table` may be later compared to what comes out from the density algorithm.

3. Prepare the file `input.edi`, specifying the dimer geometries for which the electrostatic energies are to be calculated by `elstdenrot` out of the precomputed monomer densities. The file `input.edi` consists of lines (one for each dimer geometry) containing, in this order, the

COM-COM separation (in Å), the $\beta$ and $\gamma$ Euler angles (in degrees) for monomer A ($\alpha$ is assumed as zero), followed by the $\alpha, \beta$, and $\gamma$ Euler angles of monomer B. It is assumed that the COM of monomer A coincides with the origin of the coordinate system and that monomer B is shifted in the positive direction of the $z$ axis. For example,

```
5.29177249 0.          0.            0.          0.           0.
2.38521611 86.80692336 90.00000000 180.00000000 93.37890335 360.00000000
2.63658901 87.13779711 90.00000000 180.00000000 81.94039222 360.00000000
2.89422970 87.42491802 90.00000000 180.00000000 75.80434134 359.99999915
```

would be a valid `input.edi` file containing 4 geometries. The first of these geometries is exactly the same as the one for which the interaction energies are computed by the `sapt.x` module during the `ccsddSAPT` run.

4. Run the `ccsddSAPT` job the same way a "regular" job would be run. For example, to run in /scratch/mydir from `ksh`, one would type

   ```
   ccsddSAPT name noscfcp /scratch/mydir > name.out 2>&1 &
   ```

   Note that the `noscfcp` keyword has been used since the supermolecular SCF interaction energy is of no interest here.

The result of running the `ccsddSAPT` script in MCBS will be the file `name.out` containing the standard output from the whole run. In this file, the SAPT corrections requested in the `nameP.data` file will be reported (in the `Summary Table` section) for one specific dimer configuration, described earlier. Furthermore, the calculated monomer densities will be packed for a further use into a file `name.den.tar.gz`, which, after running

```
gzip -d name.den.tar.gz
tar -xvf name.den.tar
```

will decompress info the following files:

1. unformatted sequential files `ccsd_dena.out.A` and `ccsd_dena.out.B`, containing the SCF and relaxed CCSD densities of monomer $A$ and $B$, respectively,

2. formatted files `infoa_m.data` and `infob_m.data` (in the format readable by the code `elstdenrot`, run subsequently) containing geometry and basis set info for the monomers.

After collecting the files listed above and `input.edi` in one scratch directory, one can run the `elstdenrot` program by issuing a command similar to

```
elstdenrot > dimers.out 2>&1
```

modifying, of course, the paths and the output file name appropriately. As usual, the executable `elstdenrot` can be found in `SAPT2020/bin` directory. The electrostatic energies at the $E^{(10)}_{\mathrm{elst}}$ and CCSD levels, calculated from the monomer densities translated and rotated into appropriate positions, will be reported in the standard output from `elstdenrot` (is this example—in `dimers.out`).

An example of MCBS CCSD densities calculation using the `ccsddSAPT` script, followed by a batch of electrostatic energy calculations with the `elstdenrot` program, can be found in `examples/GAMESS/CO_E1DEN_MCBS`.

## 10.6 Submitting a sequence of SAPT2020 jobs

The directory `./SAPT2020/bin` contains five utility scripts, `RunlotATMOL`, `RunlotCADPAC`, `RunlotDALTON`, `RunlotDALTONdf`, and `RunlotORCAdf`, which allow automatic calculations for multiple points on a potential energy surface in a single submission of a computer task. Some members of our group found it also convenient to utilize `RunlotATMOL2`, a more elaborate and flexible version of `RunlotATMOL`. Currently available only for ATMOL1024, CADPAC, and DALTON, these scripts can be fairly easily extended to other integral/SCF programs. An example of using `RunlotATMOL` can be found in the directory

`./SAPT2020/examples/ATMOL1024/ArH2O_MCBS`.

The idea behind the `Runlot*` scripts is to construct the input files (appropriate for a given SCF front-end) by combining universal, geometry-independent templates containing basis set and all the necessary integral/SCF options with Cartesian geometries calculated from a minimal set geometric data. Once all the input files are generated, the `SAPT` script is invoked to calculate the interaction energy for a given dimer geometry and the results are recorded in a file with a name unique for this geometry. Then, the input files are prepared and the calculation performed for the next geometry from the set, and the process continues until the set of geometries is exhausted.

Below, using the `ArH2O_MCBS` example, we describe steps needed to set up a `RunlotATMOL` job.

1. Prepare the files containing the basis sets in ATMOL1024 format:

    - `iso.d`: "isotropic" part of the whole (DCBS) basis, i.e., the functions which will be used in SCF calculations for both monomers; the sequence should be A, midbond (if present), and B;
    - `polA.d` and `polB.d`: polarization functions for A and B, respectively; functions for `polA.d` will not be used in expansion of molecular orbitals of B, and vice versa;
    - `head.d`: a simple file containing a header common to all `*.intinp` files;

- `end.d, endA.d, endB.d, endMA.d`, and `endMB.d`: end parts of `*.intinp` files, specific for a given system (e.g., `end.d` corresponds to the dimer, `endMA.d` – to monomer A in its MC$^+$BS, `endA.d` – to monomer A in DC$^+$BS, and so on; note that in this example the files `endA.d` and `endB.d` contain the ATMOL1024 directive `BYPASS TWO` which allows to avoid recalculation of two-electron integrals generated during the dimer SCF run);
- `*.scfinp` files needed for SCF calculations on all subsystems;

2. Prepare the file `dimer.cnf` which specifies the Cartesian geometry (in bohr) of monomers A and B in their "initial" configurations, i.e., for all Euler angles equal to zero. The charge, atomic mass, and an up to two-character symbol are also given for each atom. Each monomer is automatically shifted to its own center of mass, therefore only the orientation, and not the overall translation, of the monomers is important in the `dimer.cnf` file.

3. Prepare the file `geoparm.d` which contains dimer geometries for which the SAPT calculations are to be performed. Each such geometry is specified by a single line containing the separation (in Å) between the centers of mass (as defined by the masses and geometries supplied in `dimer.cnf`) of the monomers, the $z-y-z$ Euler angles [56] $\beta_A$ and $\gamma_A$ of monomer A, and the $z-y-z$ Euler angles $\alpha_B$, $\beta_B$, and $\gamma_B$ of monomer B. These angles (to be supplied in degrees) are measured with respect to the coordinate system in which the $z$ axis points from A to B. The Euler angle $\alpha_A$ is not needed, since the intermolecular potential depends only on the difference $\alpha_B - \alpha_A$, so that $\alpha_A$ may be assumed zero without loss of generality. At the end of each line of `geoparm.d` there is also a string `'DOIT'` (including the apostrophes). If for some reason you do not need the calculation for a given geometry but still want to keep it in the `geoparm.d` file, put `'DONE'` instead. The file `geoparm.d` may contain any number of lines.

Once all these files are prepared, copy them, along with the `RunlotATMOL` script itself, to your scratch directory where the whole calculation is to take place. Then start the job with the command

```
nohup time RunlotATMOL name >name.con 2>&1 &
```

in `ksh` or `bash`, or

```
time RunlotATMOL name >& name.con &
```

in `csh`, where, as usual, `name` is how the job is called and how the names of all input files start.

The calculation will then proceed as follows. First, the program `getgeoATMOL` is invoked by `RunlotATMOL` (this program is automatically compiled during the installation of SAPT2020 and the path to it is inserted into `RunlotATMOL`) which analyzes the `geoparm.d` file looking for the first line containing the `'DOIT'` directive. When such a line is found, `getgeoATMOL` reads the COM separation and Euler angles and uses them, along with the data of `dimer.cnf`, to produce a set of files `geo*.d` containing Cartesian coordinates of all atoms comprising the given dimer configuration with appropriately set charges. A ghost (i.e., zero-charge) site called `Mb` is also inserted halfway between the COMs of the monomers. A copy of `geoparm.d` is also produced (`fort.7`) in which the just processed input line is given the label `'DONE'`. The `RunlotATMOL` script then proceeds to overwrite the old `geoparm.d` with this modified copy and create all the needed `*.intinp` files out of the previously prepared pieces (`head.d, end*.d, iso.d, polA.d, polB.d`), and the Cartesian geometry files `geo*.d` generated by `getgeoATMOL`. Once all the `*.intinp` files are created, the `SAPT` script is called (the path pointing to this script is automatically set up during installation) to do the proper SAPT2020 calculation for this geometry. The output is written to the file `name1.out`. After the `SAPT` script completes its task, the whole cycle is repeated, i.e., `geoparm.d` is searched for the new geometry labeled `'DOIT'`, the input files for this geometry are generated and the `SAPT` script invoked. The process continues automatically until all the geometries in `geoparm.d` are taken care of. Consecutive output files are named `name1.out, name2.out, name3.out, ...`, in accordance with the position of geometries in `geoparm.d`. In this way, using the `Runlot` utility, the whole surface (or a significant portion of it) can be filled up with data points with a minimum interference on the user's part.

For larger systems, placing of the midbond 'ghost' atom halfway between monomer COMs can result in significant linear dependencies since the location chosen in this way can be close or overlapping with the monomers. In such cases the `'DOR6'` directive can be used instead of `'DOIT'`. With the `'DOR6'` directive, `getgeoATMOL` places the midbond according to the $1/r^6$ weights based on distances from all atoms. With this scheme, the midbond is placed in a more balanced way [57]. The rest of the atoms and the behavior of the `getgeoATMOL` program is identical to the `'DOIT'` directive. The `'DOR6'` scheme is recommended for all non-trivial monomers.

The philosophy behind the scripts `RunlotCADPAC` and `RunlotDALTON` (and the related `getgeoCADPAC` geometry conversion program) is very similar, consult the scripts themselves for details. The structure of both scripts is simple enough to allow relatively easy modifications.

## 10.7 Memory and disk requirements

When setting up a SAPT2020 computational project, it is imperative to know how much computational resources this project will consume and whether or not it will fit into the memory and disk of the machine at hand. Memoy allocations of various pograms may be assigned by setting the appropriate namelist entry in the `nameP.data` file. If a memory quantity is not specified manually (or if it is set to zero), the amount of memory will be automatically set. Memory values are by default given as a number of 8-byte words, but the suffixes 'K', 'M', 'G', 'KB', 'MB', and 'GB' may be used to indicate units of kilowords, megawords, gigawods, kilobytes, megabytes, or gigabytes, respectively. For example, 'MEMSAPT=1024' is equivalent to 'MEMSAPT=8KB'.

Memory requirements of SAPT2020 can be estimated based on the total number of basis functions and the numbers of occupied and virtual orbitals. The transformation code `tran` should work even with small memory, although it may then choose the "out-of-core" path which significantly increases the time of this step. To allow the transformation to work entirely in-core, the amount of available memory should be slightly more than $on^3/2$ 8-byte words, where $n$ is the dimension of the basis set and $o$ is the number of occupied orbitals of the larger monomer. For DCBS or DC$^+$BS calculations, it is advantageous to set memory (if possible) to about twice as much, i.e., slightly more than $on^3$ words, as a somewhat faster algorithm is chosen when additional memory is available. For the `cc` part, memory requirements can be roughly estimated as $2o^2v^2 + 2o^2n^2$ or $o^2v^2 + 2o^2n^2 + n^3$ words, whichever is greater ($v$ is the number of virtual orbitals for the larger monomer.) The `sapt.x` code will ask for about $5o^2v^2$ or $v^3 + 3o^2v^2$ words, whichever is greater, and $\max(2o^2v^2, v^3 + ov) + 6\,000\,000$ words will be needed to generate the relaxed CCSD densities using the `ccsdm` code.

More precise calculation of the memory required for an efficient run can be done using the program `memcalc` (built automatically during installation). In order to use this program, prepare the regular `nameP.data` file with the first (title) line containing five numbers, in the following order: the total number of DC$^+$BS basis functions, the total number of functions for monomer A (will be different than the previous number in a MC$^+$BS-type run), the number of occupied orbitals for monomer A, and then the same information for monomer B. Then run the program by typing

```
memcalc < nameP.data > memcalc.out
```

The file `memcalc.out` will contain detailed information about memory requirements of different parts of the code. The variables `MEMTRAN` from namelist `TRN` and `MEMSAPT` from namelist `INPUTCOR` should then be adjusted accordingly. The memory needed to run the `cc` program is calculated there explicitly so that no separate namelist variable is needed.

The memory requirements of the SCF programs are much smaller than those of SAPT2020. For specific information, consult the manuals distributed with these programs.

Disk space requirements of SAPT2020 are more difficult to estimate as these depend not only on the size of the monomers and basis set dimensions, but also on the assumed integral thresholds and the dimer configuration. For larger intermonomer separations, more two-electron integrals will be neglected, and the integral files will be smaller than for "close" configurations. While providing an accurate estimate of the required disk space is not possible, some guidance can be obtained by considering the upper limits on sizes of different scratch files involved and their scaling with the system and basis sizes.

With the dimension of the atomic basis (DC$^+$BS) equal to $n$, the size of the raw integral file produced by an SCF run scales as $n^4/8$. This should be multiplied by 12 to 16 bytes (each integral is represented by an 8-byte real number plus 4 to 8 bytes for index storage) to give an upper bound (in bytes) for the file size. In practice, with usual integral thresholds, this maximum size is often reduced by about 50%.

In the transformation step of the calculation, the atomic integral file has to coexist with files produced by the transformation (it is deleted after the `tran` program completes unless the AO-based calculation of $E^{(30)}_{\rm disp}$ or $E^{(30)}_{\rm exch-disp}$ has been requested). During the most demanding four-virtual transformations (performed in the beginning), the upper bound on the disk space becomes roughly $(3n^4/8 + v^2n^2/4 + v^4/8) \times 13$ bytes. Once the four-virtual transformations are completed, the files (including raw integrals) will take at most about $(n^4/8 + v^4/4) \times 13$ bytes, which will have to coexist with new "incoming" integrals being generated subsequently. At the end of a typical transformation for a dimer consisting of identical monomers, the disk space taken up by the raw and transformed integrals will scale approximately as $2.5o^4 + 8o^3v + 8o^2v^2 + 2.5ov^3 + 0.25v^4$, which should be multiplied by 13 to give the maximum number of bytes. Again, due to nonzero thresholds, this number has a good chance to be reduced by about 50%.

In the subsequent stages of the calculation, the `ccsdt` and `sapt.x` codes will generate additional scratch files of the order of $o^2v^2$, $ov^3$, and smaller, which will have to share disk space with the transformed integrals. Moreover, unless the non-default algorithm has been switched on at compile time by adding the `-DTPDRVN` definition to the `EXTRADEFS` variable, one more file of the order $v^4/4$ will be temporarily created by the `ccsdt` program. However, the original file of raw atomic integrals ($n^4/8$) will be removed beforehand unless the corrections $E^{(30)}_{\rm disp}$ and/or $E^{(30)}_{\rm exch-disp}$ (employing the AO integrals) are to be calculated.

## 10.8 Exploiting multithreading in SAPT

If SAPT was compiled with a multithreaded BLAS (see Sec. 8), you can use more than one processor core (if available) to reduce the wall time of the calculation. To turn on the BLAS multithreading, you have to set the shell variable `OMP_NUM_THREADS` by including the following line in the queuing system batch script (or executing the line from the shell prompt in the case of a manual SAPT submission) before invoking SAPT:

- **C-Shell users:** `setenv OMP_NUM_THREADS num`
- **K-Shell or Bash users:** `export OMP_NUM_THREADS=num`

(with no spaces around the '=' character), where `num` is the desired number of cores. The same number of cores must also be requested from the job scheduler (details depend on the type of the queuing system) and all the requested cores must belong to the same computing node, as the BLAS-level multithreading uses the shared-memory parallelism. Since the fraction of work performed by BLAS routines varies throughout a SAPT calculation, the relative time gain achieved by BLAS multithreading in a specific job is hard to predict, but should be generally higher for larger problems. However, for optimal use of the resources it is probably not advisable to use more than 8 cores. See Sec. 16.4.2 for an example of parallel scaling.

## 11 Description of some internal data sets

We present here a short schematic of the program run sequence with emphasis on the description of various files used. First, the integral/SCF program performs the appropriate calculations for the dimer, monomer A, and monomer B (depending on the type of basis set used and on the choice whether to calculate the supermolecular SCF interaction energy). The output of the integral/SCF calculations is included in the file `output.file` with the name selected on the command submitting the `SAPT` script. In the later stages, output from other programs as well as output of some system commands will be added to this file. Next, with the aid of an interface program written specifically for that integral/SCF package, several new files are created by extracting information from the data sets of the integral/SCF program just run. The `vecta.data` and `vectb.data` files contain the eigenvalues (orbital energies) and eigenvectors (molecular orbital coefficients) produced by the SCF step. The `onela.data` and `onelb.data` files contain the one-electron integrals of site A and B, respectively. The file `inta.data` contains the two-electron integrals (in some cases the original two-electron file from a given package is used). The `infoa.data` and `infob.data` files contain some general information about the system being investigated like the number of occupied orbitals, the geometry, and charges of each of the monomers.

These files (`vecta.data`, `vectb.data`, `onela.data`, `onelb.data`, `infoa.data`, `infob.data`, and `inta.data`) serve as input files to the four-index transformation program. This program transforms the one- and two-electron atomic integrals into molecular integrals and produces the files `f2e.000.xxx` (direct access, integrals other than four-virtual), `f72.000.xxx` (sequential, $vvvv$ integrals for monomer A), and `f73.000.xxx` ($vvvv$ for monomer B) as output. `xxx` is normally always `001`, but on legacy systems with file-size limitations, a series of files with consecutive labels is produced instead. The numbers of these chunks for different file types are stored in the file `nfiles`, which is used by the other programs.

Next, two interfacing programs, `int` and `sort`, are called to extract different types of integrals from `f2e.000.xxx` and to put them in `ccsorta.000.xxx`, `ccsortb.000.xxx` ($ov^3$ integrals, direct access files) and in a number of sequential files named `o????__a` and `o????__b`. Additionally, small formatted files `ccloca.000` and `cclocb.000` with indexing information are created. The infoa.data, infob.data, and nfiles files are required by both interfaces.

Then, the MBPT/CC program is run. It uses the files generated by int and sort, as well as f72.000.xxx, f73.000.xxx, infoa.data, infob.data, vecta.data, vectb.data, and nfiles. The generated cluster amplitudes are stored in sequential files locx_y_m, where x is s for singles and d for doubles, m is a or b depending on the monomer, and y can be 1, 2, 3, or i (amplitudes from the first 3 iterations or from the last one).

The last step of the process is the calculation of the SAPT corrections. The needed input files are vecta.data, vectb.data, infoa.data, infob.data, loc*, and f2e.000.xxx. These files contain the output from each of the previous steps. The perturbation program uses some temporary files and prints to output.file the values of the corrections as well as time needed to compute them. Near the end of the output file, there is a table collecting all the corrections expressed in various energy units.

## 12 Performance of SAPT2020

The computational cost of SAPT2012 calculations depends on the system size in the following way. The integral/SCF calculation time scales at most as $(o+v)^4$, with a somewhat lower power expected for large systems. When the AO-based algorithm for the four-virtual diagram in CC is used (which is the default in SAPT2012.2 when ATMOL1024 or MOLPRO are used as the integral/SCF program), the transformation step scales as $o(o+v)^4$ and the CCSD step as $o^2(o+v)^4$. Otherwise, the scalings of the transformation and CCSD steps are proportional to $v(o+v)^4$ and $o^2v^4$, respectively. Finally, the scaling of the perturbation theory code strongly depends on the required level of theory: $o^3v^2$ for SAPT(0), $o^2v^3$ for SAPT2, and $o^3v^4$ for the full SAPT. For a general orientation, Table 2 contains timings from an Opteron 252 Linux system, obtained with the SAPT2020 suite compiled in the 64-bit mode with Portland pgf90 compiler. The examples chosen are described in Sec. 13 under the labels ArHF_DCBS, CO2D_MCBS, and C6H6_H2O_DCBS. For the sake of completeness, the amounts of memory and disk space required in each calculation are also reported (see Sec. 10.7 for a general discussion of these two issues). The memory requirements of the integral/SCF calculations are omitted because they are negligible compared to the other steps. Table 2 is restricted to regular SAPT calculations on a single core. See Ref. 58 for a detailed analysis of the SAPT(DFT) performance and Ref. 9 for a discussion of the parallel SAPT scaling.

## 13 Tests and example input and output files

A set of test and example runs is provided in the distribution of SAPT2020. These include several very small cases which should finish in seconds or minutes and which can be used to check the

Table 2: Wall-clock execution times and memory/disk requirements of different parts of the SAPT2020 suite on the Opteron 252 Linux system. The integral/SCF package used is ATMOL1024.

| system | Ar-HF | $(CO_2)_2$ | $C_6H_6$-$H_2O$ |
|---|---|---|---|
| occupied orbitals (A/B) | 9/5 | 11/11 | 21/5 |
| DCBS size | 86 | 200 | 262 |
| $MC^+BS$ size | - | 149 | - |
| integral/SCF time | 21 s | 13 min | 53 min |
| transformation time | 15 s | 21 min | 53 min |
| CC time | 128 s | 32 min | 16 h 25 min |
| sapt.x time | 84 s | 41 min | 8 h 9 min |
| transformation memory[a] | 5.8 Mwords | 40 Mwords | 380 Mwords |
| CC memory | 2.3 Mwords | 11 Mwords | 112 Mwords |
| sapt.x memory | 3.0 Mwords | 12 Mwords | 128 Mwords |
| integral/SCF disk | 42 MB | 1.1 GB | 7.8 GB |
| transformation disk | 138 MB | 1.9 GB | 17 GB |
| CC disk | 300 MB | 4.8 GB | 38 GB |
| sapt.x disk | 268 MB | 2.0 GB | 23 GB |

[a]Memory amount required for the fastest 'in core' algorithm is given. `tran` can also run is significantly smaller memory at the expense of execution time.

correctness of the SAPT2020 installation. In each case, both the complete set of input files and the output files (at least from one platform) are provided. With the large number of front-end SCF programs interfaced with SAPT2020 and with the large number of hardware platforms on which it can be run, it is not practical to provide examples for all possible cases. The mix of combination of various factors chosen should be sufficient to establish the correctness of installation and help in preparation of user's production runs.

In some cases, the test files compute all currently available SAPT corrections. This is *not* needed for calculations of interaction potentials. In such calculations, use one of the `SAPTn` control variables to select the minimal needed set of corrections. This will reduce the time of calculations.

## 13.1 The `examples` directory

The directory `./SAPT2020/examples` is divided into subdirectories corresponding to different SCF front-end programs, e.g., GAMESS or ATMOL1024. Each of those subdirectories contains the following test job directories (named the same for all SCF programs):

- `BER`: a very small test calculation for the beryllium dimer in a DCBS consisting of a $2s1p$ set on each atom. The run will complete in a few seconds using 0.1 Mwords of memory and should be tried first. We note again the naming convention for clarity: for ATMOL1024 the dimer files are called `BER.intinp` and `BER.scfinp`, the monomer A files are called `BERA.intinp` and `BERA.scfinp`, the monomer B files are called `BERB.intinp` and `BERB.scfinp`, and the perturbation file input is given by `BERP.data`. The input files needed when using GAMESS are called `BER.inp`, `BERA.inp`, and `BERB.inp` (integral/SCF input files) and `BERP.data` (perturbation input file). Monomer A and monomer B input files are created by setting the charge to zero on the proper site. For example, in the case of monomer A the charges on site B should be set to zero. The perturbation program should produce the results given below in Appendix D. These results can be found in the `Summary Table` at the end of an output.

- `HF2_DCBS`: the HF basis set $[4s2p1d/2s1p]$ is taken from Ref. 49, and a $2s1p$ set of midbond functions is added. Needs about 1 Mword of memory and takes less than a minute to complete. In the GAMESS input files, the option `ISPHER=1` is used, so that the results of this test performed with GAMESS should agree with those from an ATMOL1024 run.

- `HF2_MCBS`: the MC$^+$BS version of the above: monomer basis set (files `HF2MA.*` and `HF2MB.*`) are obtained by removing the $d$ and $p$ orbitals of F and H, respectively, of the "ghost" molecule. In the GAMESS version of this test, the "tags" technique is used to enforce MC$^+$BS, whereas in the ATMOL1024 version, the basis functions are arranged in blocks *polA*, *isoA*, *mid*, *isoB*, *polB*, as described in Sec. 10.1.2. As in the DC$^+$BS variant, the space spanned by the basis functions is always restricted to spherical Gaussian functions, even in the GAMESS version.

- `ArHF_DCBS`: the basis set $[8s5p2d1f/6s3p2d1f/3s2p1d]$ has been taken from Ref. 59. No midbond functions are present here. This run will need about 6 Mwords of memory and about 4 minutes to complete (on an Opteron 252 machine).

- `ArHF_MCBS`: an MC$^+$BS version of the above. Monomer basis sets (in files `ArHFMA.*` and `ArHFMB.*`) are obtained by deleting the polarization functions ($d$ and $f$ on F and Ar and $p$ and $d$ on H) of the "ghost" molecule. The "tags" technique is used for MC$^+$BS for both the GAMESS and ATMOL1024 examples.

- `CO2D_MCBS`: an MC$^+$BS run for the $CO_2$ dimer in the basis of Ref. 52 (DCBS size: 200, MC$^+$BS size: 149). This example is larger than the previous ones, it requires about 40 Mwords of memory and takes slightly less than two hours to complete on an Opteron 252 machine. The GAUSSIAN and MOLPRO versions of this example are also provided (note that these two require the specification of monomer A/monomer B/dimer sets using the "tags" mechanism presented in Sec. 10.1.2).

- `C6H6_H2O_ADZM`: the benzene-water dimer in a 262-term basis. This is the largest example included and will take about 27 hours (on an Opteron 252) to complete.

Some additional examples are also included, namely

- `GAMESS/HF_NH3_MCBS`: an example illustrating the use of the MC$^+$BS technique in GAMESS with the "tags" and "basis" options, as described in Sec. 10.1.2. About 7 minutes on a 2.80 GHz processor.

- `ATMOL1024/ArH2O_MCBS`: an example of using the `RunlotATMOL` script, as described in Sec. 10.6. Consists of two jobs, each taking about 6 minutes on an Opteron 252.

- `ATMOL1024/Ar2_FROZENCORE_3ORDER`: an example of calculating the third-order SAPT corrections in Eq. (9), and of performing a frozen core calculation. The files `Ar2P.data.allelectron` and `Ar2P.data.frozencore` present in this example's subdirectory correspond to an ordinary all-electron calculation and a frozen core calculation, respectively. Rename either one to `Ar2P.data` and invoke the SAPT script. The basis set employed is aug-cc-pVQZ+$(3s3p2d2f1g)$ midbond, and the MC$^+$BS approach is used. The calculation requires about 1.5 hours (all-electron) or half an hour (frozen core) on an Opteron 252.

- `ATMOL1024/HE2_E2DISPCCD`: an example of the CCD+ST(CCD) dispersion energy calculation (Sec. 10.4) for the helium dimer in the d-aug-cc-pVQZ basis set. This calculation will take about 15 minutes on an Opteron 252. A GAUSSIAN version of this example is also provided.

- `MOLPRO/ArHF_AVDZ`: a simple example for a MOLPRO–SAPT2020 calculation. Note that for MOLPRO a single input file `ArHF.molpro` takes care of all integral and SCF calculations and only the usual `ArHFP.data` post-SCF input file is needed as well. This example takes less than two minutes on an Opteron 252. A relativistic version of this example, showing how to use SAPT with the MOLPRO front end and the second-order Douglas-Kroll-Hess relativistic Hamiltonian, can be found in the same directory (files `ArHFrel*`). An ATMOL1024 equivalent of this example (the nonrelativistic version) is also provided.

- `MOLPRO/KR2_ECP`: an example of using SAPT2020 with an effective core potential. An ECP basis set aug-cc-pVTZ-PP [60] supplemented by a $(3s3p2d2f1g)$ midbond set is employed for the van der Waals minimum of the krypton dimer. This example takes slightly more than two hours on an Opteron 250.

- `DALTON/He2`: helium dimer example with basis set larger than 255 basis functions. The DALTON input files have `.dal` and `.mol` extensions.

- `saptdft`: a few examples utilizing SAPT(DFT) with DALTON interface and DF-SAPT(DFT) with ORCA interface. `ArHF` is a quick example of basic SAPT(DFT) usage. Directories `Ne2-aTZ-GGAKER` and `Ne2-aTZ-LDAKER` contain neon dimer with GGA kernel or LDA kernel (default), respectively (see Sec. 16.2). `H2O2_MCBS` contains water dimer example with calculation of $\delta E^{\rm HF}_{\rm int,resp}$. Finally, `C6H6` contains a large calculation for benzene dimer. All examples except `ArHF` utilize `RunlotDALTON`, described in Sec. 10.6.
- `dfsaptdft`: examples utilizing DF-SAPT(DFT) with DALTON interface.
- `ORCA`: water dimer examples utilizing DF-SAPT(DFT) with ORCA interface, compared with analogous jobs with DALTON interface.

## 13.2 Running test jobs

The simplest way to run an example is to copy *all* files from the corresponding directory (e.g., `./SAPT2020/examples/ATMOL1024/BER`) to a scratch directory, then cd to this scratch directory and submit the job using the submit line described in Sec. 10, for example, in `ksh`

```
SAPT BER scfcp > BER.out 2>&1 &
```

Recall that the string `BER` is the same as the initial part of the name of all input files. If the `./SAPT2020/bin` directory is not in your `PATH`, you may need to supply the full path to the `SAPT` script.

To simplify the process of running multiple tests, two simple `ksh` scripts, `runtstGAMESS` and `runtstATMOL` are provided in the directories `./examples/GAMESS` and `./examples/ATMOL1024`, respectively. The scripts execute loops over the subdirectories selected in the `for` statement, running the `SAPT` script inside these directories, and cleaning up after each such run. To select the jobs to be executed, adjust the `for` statement in `runtstGAMESS` and `runtstATMOL`. The names of output files from the tests are given an ending (see the variable `MACHINE`) which can be used, e.g., to distinguish between the runs performed on different platforms.

Under Unix, the output file can be viewed while the program is run. This allows to check how far the program has advanced. Note, however, that Unix first writes the data to fairly large buffers, and only when the buffers are filled, to the output file. It is important to remember about it when debugging the program (some info may not appear in the output after a crash, although it may come from a successful part of the run). SAPT uses instructions flushing buffers in several places but more such statements may have to be added for debugging. Also, for larger runs, one should monitor the disk use (by just doing `ls -lt` or `du -k` in the working directory). The memory actually used by the program can be checked by using the `top` command (called `prstat` or `monitor` on some systems). When the program crashes, our practice shows that in most cases it is due to errors in the integral/SCF parts of the code. Thus, make first sure that these stages have finished successfully. When there is an error there, consecutive stages of the code do start (and immediately fail), creating an impression that the code crashed in a later stage than it was the case.

Once the test runs are completed, the results, especially the `Summary Table` at the end of each output file, should be compared to the reference ones, provided in each test directory for at least one platform. One can notice slight differences between the results obtained with different front-ends and/or on different platforms. In any case, reproducibility of at least 5 significant digits should be expected. The main differences come in the corrections using the converged CCSD amplitudes. The CC convergence threshold has sometimes to be adjusted to obtain several digit agreement. Notice also that in different versions of SAPT the threshold was changing between relative and absolute, which of course makes a significant difference.

## 14 Parallel SAPT: pSAPT2K2

The parallel implementation of SAPT has been described in detail in Ref. 9. The parallel pSAPT2K2 codes have essentially the same functionality as their sequential counterpart SAPT2002. Slight differences exist in user interface-related areas, such as compilation (more libraries are needed in pSAPT2K2), input options (SAPT2002 is more user-friendly when it comes to input defaults), or output presentation. Since pSAPT2K2 parallel runs are more prone to various types of problems, the output of pSAPT2K2 contains a lot of debugging and profiling information. SAPT2020 has been tested with a large number of (sequential) SCF front-end programs, while in the case of pSAPT2K2 the parallel GAMESS(US) is effectively the only SCF code which can be used. pSAPT2K2 should also work with the sequential SCF codes, like ATMOL1024, although quite often the time cost of such a calculation would be dominated by the SCF step. As stated above, pSAPT2K2 is essentially the parallel version of SAPT2002. Thus, the extensions introduced in later versions of SAPT codes are not available for parallel execution.

All modules of the pSAPT2K2 code have been parallelized using only the MPI library and thus the suite is very portable and should run on any parallel architecture. It is also recommended that the program is linked with the SCALAPACK/BLACS libraries, as these are needed for the CHF routine to run in parallel. In the case these libraries are not available, the CHF routines will run in sequential mode, which may have substantial impact on the timings, especially for larger numbers of occupied orbitals. The SCALAPACK/BLACS libraries are also necessary to build the code generating the static and dynamic CHF propagators, also referred to as susceptibility functions. This code will not be built if the libraries are missing.

To date, the program has been tested on the SGI Origin2000 and Origin3800 shared-memory machines, the IBM SP3 and SP4 platforms (shared/distributed memory), as well as on Beowulf clusters running Linux. From the standpoint of that latter machine, it is **important** that pSAPT2K2 is able to distribute its temporary files over file-systems local to the nodes, so that the scratch file-systems do not have to be NFS-mounted across the whole cluster.

Novel algorithms have also been developed for efficient calculation of the electrostatic, induction, and dispersion energies, see Sec. 14.7. The idea behind these algorithms is that all these interaction components are expressible through monomer charge densities and dynamic susceptibility functions. These monomer properties can be calculated just once for each monomer at a high level of correlation, then simplified, stored, and reused many times for different dimer geometries. Scalable *beta* versions of these new algorithms are also included in this distribution. As mentioned before, the induction/dispersion module requires the SCALAPACK/BLACS libraries to be installed.

## 14.1 Structure of `./psapt2K2` directory

After unpacking, the `./psapt2K2` main directory will contain the following files and subdirectories:

- `bin/`: utility scripts for running SAPT. After compilation, this directory will also contain the executables used in a SAPT run.
- `pcksdisp/`: contains sources and detailed documentation for the code generating the static and dynamic susceptibility functions (currently at the CHF level).
- `cleandirs`: use this script to clean the entire `psapt2k2` directory tree before recompiling from scratch.
- `compall`: script used to build the package (see Sec. 14.2).
- `doc/`: documentation for SAPT2002; contains this document and the METECC paper [8] in the postscript form.
- `edi_notran/`: contains sources of the code implementing the new transformationless algorithms for calculations of electrostatic, induction, and dispersion energies using the precomputed densities, susceptibility functions, and the Casimir-Polder formula.
- `examples/`: input and output files for a set of systems and platforms. This is a good source of templates for your runs.
- `misc/`: contains various interface and utility programs. Most integral/SCF packages need an interface program to extract one-electron integrals and SCF orbital energies and coefficients from files created by these packages and transform them into a standard form readable by the

transformation code (two-electron integrals are read by transformation directly without such preprocessing). Other programs present in `misc/` include `int`, `sort`, and `even`, interfacing the transformation to coupled cluster code, memory estimator `memcalc`, and a utility program `tmerge` which can be used to merge the transformed integral files if these are to be used further by sequential codes.

- `pcc/`: program performing the coupled cluster singles and doubles calculations for the monomers. The first few iterations are performed perturbatively, in this way producing MBPT order-by-order amplitudes needed in SAPT. Both CCSD and MBPT amplitudes are later used by the `psapt` module to compute intramonomer correlation contributions to various interaction energy components as well as electron densities at various MBPT levels.
- `psapt/`: program computing the SAPT corrections.
- `ptran/`: parallel program performing the one- and two-electron integral transformation.
- `updates.log`: log of the history of changes and updates.

## 14.2 Installing psapt2k2

Installation of pSAPT2K2 is controlled by the `ksh` script `compall`, a small portion of which has to be customized by the user. The `compall` script sets the compilation options appropriate for a given hardware platform. It is assumed that the integral/SCF code used is GAMESS(US). No other SCF programs are currently supported by the `compall` script. The use of other SCF packages is still possible although it would require some hand-tuning of the compilation process. After the compilation options have been set, the `compall` script compiles and links all the pieces of the code. Finally, the scripts used to run pSAPT2K2 are updated by inserting the current paths to the executables.

### 14.2.1 `compall` installation script

The following environment variables must be adjusted by the user in the top section of the `compall` script:

1. **Execution shell**: Change the first line of the script to one of the following:
   - `#!/bin/bash` : The Bash shell on a LINUX box, or
   - `#!/bin/ksh` : The Korn shell on all other platforms
2. `TARGET`: the machine the program is being compiled on. Choose one from `o2k` (will work for O3K also), `sp` (will work for SP3 and SP4), or `mpich` (on Beowulf clusters).

3. `BLAS`: if `TARGET=mpich`, specify how you want the BLAS library to be linked (usually this is just `-lblas`, but Beowulf configurations differ, so beware!). For other `TARGET`s – ignore this option.

4. `GAMESS`: path containing the GAMESS exec you will be using.

5. `VERNO`: GAMESS "version number" (`compall` assumes that the name of GAMESS executable is `gamess.$VERNO.x`).

6. `TRGT_GMS`: if `TARGET=o2k`, specify how GAMESS should be run (pick from `sockets` and `sgi-mpi`, consistently with your GAMESS installation). For other `TARGET`s this will be set automatically.

7. `POE_COM`: if `TARGET=sp`, specify which `poe` command (`poe`, `grd_poe`, or `sge_mpirun`) will be used to run parallel codes. Usually an MPI code on an SP machine is launched by the Load Leveler queuing system using the `poe` command. At ARL, a wrapper `sge_mpirun` is used instead, which works with the GRD queuing system (or grid engine). On IBM SP3 brainerd at ARL, the command `grd_poe` also works. For `TARGET`s other than SP – ignore this variable.

8. `SCLPCK`: specify how the SCALAPACK and BLACS libraries are to be linked. If this is set to 'NO', then the sequential CHF procedure will be used (will spoil scaling of `psapt` for large systems), and the `pcksdisp` code (for susceptibility functions) will not be built at all.

9. `LPCKLIB`: specify how the LAPACK library is to be linked. This library is needed to build the program `tdenfit`, fitting the densities and propagators in terms of auxiliary bases. If `LPCKLIB` is set to `'NO'`, `tdenfit` will not be built. On SP machines, LAPACK is a part of ESSL library and hence `LPCKLIB` does not have to be given at all (just set it to `YES`).

Once all the variable mentioned above are set, simply type:

- **C-Shell users:** `./compall >& compall.log &`
- **K-Shell users:** `./compall > compall.log 2>&1 &`

and the compilation should begin. Check `compall.log` to see if all is well. The `compall` script works by invoking the `make` command with platform-specific makefiles. Thus, any subsequent invocation of `compall` will detect changes made to the sources and only those parts of the code will be rebuilt which were affected by these changes. Running the script `./psapt2K2/cleandirs` will restore the `./psapt2K2` directory to its "distribution" state, i.e., all object files and executables (except shell scripts) will be deleted and a subsequent invocation of `compall` will start the compilation from the beginning.

### 14.2.2 Testing pSAPT2K2 installation

Once the compilation has been completed successfully (if unsure, just `grep` the `compall.log` file for the word *error*), we recommend that you perform as many tests as possible before starting to use pSAPT2K2 for production runs. A suite of test jobs of varying size and for different parallel platforms has been provided for this purpose in the subdirectory `examples/`. Sample outputs from different machines (`*out_ref.*`) and queuing system submission scripts can be also found there. Examples of using the new, monomer property-based algorithms for electrostatics, induction, and dispersion can be found in directories `./psapt2K2/examples/PLATFORM/EDI`, where `PLATFORM` is one of `O3K, SP3, SP4`, and `BEOWULF`.

## 14.3 Using pSAPT2K2 with GAMESS as a front-end

GAMESS(US) is currently the only *parallel* SCF package pSAPT2K2 is interfaced with. See Sec. 9 for information about setting up the GAMESS input files for pSAPT2K2 runs and about the structure of the GAMESS driver script `runGAMESS` and of the `GAMESS-ptran` interface program `gamsintf`.

There are a few minor differences between the `runGAMESS` script used with the sequential program SAPT2020 (described in Sec. 9) and the one used with pSAPT2K2. Although these differences are completely transparent to the user, we list them here for the sake of completeness:

- Unlike in the sequential version of the code, the variable `TARGET` in `runGAMESS` is set automatically by the `compall` script, so no additional action is needed here.

- A separate version of the `runGAMESS` script, called `runGMS.mpich`, has been created for `samson` – our Beowulf cluster here at UD. The GAMESS submission instruction present in this script were causing syntax errors on other platforms - hence the need to create a separate script for MPICH implementation of MPI. `runGMS.mpich` will be called automatically from within the `pSAPT.mpich` driver script, so the user does not have to worry about these details.

- A separate version of the `runGAMESS` script, called `runGMS.mpich.maui`, has been created for `huinalu` – a Beowulf cluster at Maui supercomputer center. `runGMS.mpich.maui` differs from `runGMS.mpich` slightly in details of how the `HOSTLIST` is constructed and by the usage of `ssh` instead of `rsh` for interhost communication. `runGMS.mpich.maui` will be called automatically from within the `pSAPT.mpich.maui` and `pSAPT.mpich.maui.sep` driver scripts, so the user does not have to worry about these details.

- Should you prefer to use your own script as a driver for GAMESS, adapt it for pSAPT2K2 following the instructions of Sec. 9 with one exception: the integral file generated by process "0", `$JOB.F08`, has to be renamed into `inta.0`, while the integral files from the other processes, `$JOB.F08.X` – into `inta.X`. This can be accomplished in C-shell using the following syntax:

```
mv $JOB.F08 inta.0
set m=1
foreach fle ( ‘ls -1 $JOB.F08.*‘ )
 mv $fle inta.$m
 @ m++
end
```

- On some `BEOWULF` clusters, such as `huinalu` at Maui center, the communication between the nodes is furnished in terms of `ssh` rather than `rsh`. In such cases, GAMESS's parallel launcher `ddikick.x` must be modified to use `ssh` instead of `rsh` and recompiled, as suggested in the source `ddikick.c`.

## 14.4 How to run pSAPT2K2

To perform a pSAPT2K2 calculation for one dimer geometry, one has to run a dozen or so programs: integral/SCF calculations for the monomers and possibly for the dimer, interface programs (in most cases) rewriting integral/SCF files into different forms, 1- and 2-electron integral transformations, MBPT/CCSD calculations for monomers, and finally the “proper” SAPT calculations. All of this is performed automatically using the script `pSAPT.X` from the `./psapt2K2/bin` directory, where `X` denoting a platform is one of `o2k`, `sp`, `mpich`, `mpich.maui`, and `mpich.maui.sep`. The last two cases pertain to the `huinalu` cluster at Maui center. While `pSAPT.mpich.maui` controls a calculation utilizing the common scratch file-system `/scratch/huinalu`, the other script, `pSAPT.mpich.maui.sep`, makes use of local scratch disk (`/scratch/local`) on every node. The `pSAPT.X` script is executed on one “master” processor which in turn calls other executables and scripts, also found in the `./psapt2K2/bin` directory. The MPI executables (such as `ptran, pcc, psapt`) are launched by the “master” using the `mpirun` command (on SP systems `poe`, `grd_poe`, or `sge_mpirun` is used for this purpose). The details of the calculation “flowchart”, input files, and the use of various types of basis sets are presented in Sec. 10. Here we only expose the features characteristic of the parallel version of the code.

The executables invoked by the `pSAPT.X` script communicate with one another through various (usually large) temporary files. Most of these files are written and read by only one process and do not need to be accessed directly by other processes. This means that such files can be stored locally in some scratch directory accessible only to one process. This is indeed the case on a Beowulf cluster, where scratch file-systems are usually local to the nodes. Another class of (smaller) temporary and input files have to be accessed from all processes. On a Beowulf cluster such files have to be replicated in each processes' local scratch area. This problem does not exist on most other multiprocessor platforms (like the SGIs and SPs in the DoD centers), where all temporary files are stored in a common large file-system accessible to all processes. All the details of how the files are handled are taken care of by the `pSAPT.X` scripts and the MPI executables themselves and are completely transparent to the user (with the exception of the input files - *vide infra*).

During the compilation, the script `compall` adapts the appropriate `pSAPT.X` script by inserting the proper global paths to executables and should run properly on any installation without changes. Currently, this feature is not functional in the case of scripts used on `huinalu`, namely `pSAPT.mpich.maui` and `pSAPT.mpich.maui.sep`. The paths in these scripts need to be changed manually so that the variables `MAIN_SAPT_DIR` and `SCF_HOME_DIRECTORIES` reflect user's directory structure (if GAMESS is used, it concerns also additional directories relevant for this program). These variables are set about 500 lines down in the `pSAPT.X` script. The `pSAPT.X` script is written in `ksh`, although on Linux platforms, some of which are not equipped with `ksh`, it is actually executed under `bash`.

### 14.4.1 Running pSAPT2K2 on SGI

We first consider a situation where there is no queuing system installed and a pSAPT2K2 job can be submitted interactively from the command line.

Before launching a pSAPT2K2 run, the scratch directory should be created and all the input files should be copied to this directory. The pSAPT2K2 calculation is then launched *in this scratch directory* by typing `pSAPT.o2k` with appropriate options. If `ksh` is used as the login shell and an SGI machine is the platform, one would type

```
pSAPT.o2k jobname opt1 nproc opt2 >output.file 2>&1 &
```

(if `psapt2K2/bin` is not in your `$PATH` then the full path to `pSAPT.o2k` will have to be given.) The keyword `jobname` has to be the same as the beginning of the name of the input files for the system considered (we use a naming scheme to reference the needed input files). For example, let the keyword be `name`. Then, as the script is running, it will look for the `nameP.data` file, which contains the input data to the programs `ptran`, `pcc`, and `psapt`. Similarly, the `pSAPT.o2k` script will look for GAMESS input files to the integral/SCF parts of a calculation starting with `name`. The number and full names of such files depend on the type of basis set used in calculations. See Sec. 10 for details on how to prepare GAMESS input files for different types of runs. Using the string `scfcp` for `opt1` will request, in addition to the standard SAPT calculation, also the CP-corrected supermolecular SCF interaction energy in the dimer-centered basis set. If such a calculation is not required, use `noscfcp` instead. Using the keyword `gototran` as `opt1` will result in restarting a `pSAPT.o2k` run from the transformation step, i.e., from the program `ptran`. The parameter `nproc` is the number of processors to launch the job on. The parameter `opt2` must be the full path of the scratch directory in which the whole calculation is taking place.

The output from all programs executed by `pSAPT.o2k` is written to a file `output.file` located in the scratch directory (this name can be set by the user to an arbitrary string, usually connected with the system being considered and its geometry). The last two elements in the command line given above are Unix `ksh` constructs which indicate that standard error messages will be written to the same file as standard output and that the processes will be run in the background.

A more common situation occurs when jobs have to be submitted through some sort of a queuing system, such as GRD (or, more recently, the *grid engine*). In this case, all the operations described above should be performed by a script submitted to the queue. A sample script, `ARL_script`, could look like

```
#!/bin/ksh
#$ -S /bin/ksh
#$ -N ArHF_Tc
#$ -l o3k
#$ -pe pe_4hr 8

###############################################
#        Set the these parameters:
###############################################

NODT=8                          # number of processors to run everything
CURDIR=/home/bukowski/ArHF_Tc   # Starting directory
WRKROOT=/usr/var/tmp/bukowski   # root of the scratch directory
SAPTSCRIPT=/home/bukowski/psapt2K2/bin/pSAPT.o2k # driver script
JOB=ArHF                        # Core of the job name

###################################################
# No need to modify anything below this line
###################################################

WRKDIR=$WRKROOT/$JOB   # scratch directory where
                       # the calculation will take place
cd $CURDIR

# Create the job-specific scratch directory

if [ ! -d $WRKDIR ]
  then
  mkdir $WRKDIR
fi

# Copy the input files from HOME to scratch

cp $JOB'.inp' $WRKDIR
cp $JOB'A.inp' $WRKDIR
cp $JOB'B.inp' $WRKDIR
cp $JOB'P.data' $WRKDIR

# Run the job in scratch
```

```
cd $WRKDIR
$SAPTSCRIPT $JOB scfcp $NODT $WRKDIR >>$JOB'.out_test.'$NODT 2>&1

# Copy output file back to HOME

cp $JOB'.out_test.'$NODT $CURDIR
```

The first few lines starting with `#$` are options passed on to the queuing system (in this case: GRD), specifying the name of the job, what machine it should be run on, in what queue, and on how many processors. The syntax of these options depends of the queuing system at hand and so these few lines must be adjusted accordingly. The script is submitted typically using the command

```
qsub ARL_script
```

After the script enters execution, it first creates a scratch directory named after the name of the job, then copies the input files from the home subdirectory (CURDIR) to the scratch directory, and runs the `pSAPT.o2k` script (SAPTSCRIPT). Upon completion of the pSAPT2K2 calculation, the output file is copied back to the home subdirectory CURDIR. Note that after the job is finished, the scratch directory will contain all kinds of temporary files which are kept there in case a restart is needed. These files should be manually cleaned up.

### 14.4.2 Running pSAPT2K2 on an SP3/SP4

The job submission technique here is very similar to the one used on an SGI platform, except that interactive submissions are generally not possible and everything must be handled by a queuing system. Consult the `ARL_script` and `NAVO_script` in the `psapt2K2/examples/SP?` directories for examples of the GRD and Load Leveler submission scripts.

### 14.4.3 Running pSAPT2K2 on a Beowulf cluster

Configurations of Linux-based Beowulf clusters may vary between installations. Presented below are techniques of running pSAPT2K2 on our local cluster `samson` at University of Delaware, and on the `huinalu` cluster at Maui supercomputer center.

#### `samson` cluster

The machine consists of 132 compute nodes equipped with 1 GHz AMD Athlon processors and connected by Ethernet. Each node is equipped with a local scratch file-system not accessible from other nodes. On each node, this file-system is mounted as `/tmp`. In addition, there is a login node which hosts home directories of the users and some shared resources, like the MPI implementation MPICH. The home and shared directories are NFS-mounted on all compute nodes. Commands and programs can be executed on compute nodes from the login node through the `rsh` utility. It is also possible to `rlogin` to the compute nodes, although it is not necessary for running pSAPT2K2. The consecutive modules of pSAPT2K2 are launched from within the script `pSAPT.mpich` using the `mpirun` command, while GAMESS is started using the `ddikick.x` utility of the DDI package.

It is a good rule on `samson` that jobs are to be run on compute nodes only and *not* on the login node. For any MPI application, this may be accomplished by executing the `mpirun` command on the login node with the option `-nolocal`, supported by MPICH. Unfortunately, this option (or a similar one) is not available in the DDI package which is the GAMESS parallelization tool. This creates a problem if an MPI application depends on files generated by GAMESS and both are to be run from the same driver script, such as `pSAPT.mpich`. This means that `pSAPT.mpich` has to be submitted directly on one of the compute nodes rather than on the login node, and that the MPI programs should be launched *without* the `-nonlocal` option.

The process of submitting a pSAPT2K2 run on `samson` consists of the following steps:

1. In your `$HOME`, create a subdirectory for the job and copy the input files `job*.inp` and `jobP.data` to this subdirectory.

2. Create file `calcnodes` listing the nodes about to be used in the calculation, for example,

   ```
   sam2
   sam4
   sam5
   sam6
   ```

   Make sure there are no blank lines in this file.

3. Make sure that the appropriate scratch directory (we will use `/tmp/bukowski` as an example) is present on each of the compute nodes listed in the file `calcnodes`.

4. Submit the job using a command `./submit JOB`, where `JOB` is the job name (the core of the input file name) and `submit` is a script similar to the following:

```
#!/bin/bash

########## Customize scratch directory on compute nodes
########## and the directory where the driver script resides.

SCRDIR=/tmp/bukowski      # scratch directory
SAPT_SCRIPT=/home/bukowski/psapt2K2/bin/pSAPT.mpich
                        # SAPT driver script

####### Paths to the SAPT executable directory

SAPTDIR=/home/bukowski/psapt2K2/bin

########## End of customized part #####################################
```

```
#########################################################################

#### Create all the needed PROC files based on calcnodes

PTRAN=$SAPTDIR/ptran
PCC=$SAPTDIR/pcc
PSAPT=$SAPTDIR/psapt
INT=$SAPTDIR/int
SORT=$SAPTDIR/sort
EVEN=$SAPTDIR/even

FRSTNODE=`head -1 calcnodes`

echo $FRSTNODE 0 > PROC
echo $FRSTNODE 0 > PROCc
echo $FRSTNODE 0 > PROCe
echo $FRSTNODE 0 > PROCi
echo $FRSTNODE 0 > PROCs
echo $FRSTNODE 0 > PROCso

for i in `tail +2 calcnodes`
do

echo $i 1 $PTRAN >> PROC
echo $i 1 $PCC   >> PROCc
echo $i 1 $EVEN  >> PROCe
echo $i 1 $INT   >> PROCi
echo $i 1 $PSAPT >> PROCs
echo $i 1 $SORT  >> PROCso

done

########### PROCs file ready - run the job now ###########################

JOB=$1                  # job name

# copy all SAPT input files and the PROC file onto the scratch dir
# on every node specified in PROC file

for i in `awk '{print $1}' PROC`
do
echo Copying input to $i
 rcp *inp $i:$SCRDIR
 rcp $JOB'P.data' $i:$SCRDIR
 rcp  PROC $i:$SCRDIR
 rcp  PROCs $i:$SCRDIR
 rcp  PROCi $i:$SCRDIR
 rcp  PROCc $i:$SCRDIR
 rcp  PROCso $i:$SCRDIR
 rcp  PROCe $i:$SCRDIR
done

# Set the number of nodes

NPROC=`wc PROC | awk '{print $1}'`

# Determine the master node (the first one in PROC)
```

```
MASTER=`head -1 PROC | awk '{print $1}'`
echo Submitting job on $MASTER

# Submit the job on master node. Make sure that master knows what
# LOGNAME is (needed for psapt).

rsh $MASTER "LOGNAME=bukowski; export LOGNAME; echo $LOGNAME; cd $SCRDIR;
          $SAPT_SCRIPT $JOB scfcp $NPROC $SCRDIR > OUT 2>&1 " &

exit
```

The variables `SCRDIR` (the scratch directory, its name assumed to be the same on each node), `SAPT_SCRIPT` (full path to the driver script), and `SAPTDIR` (location of pSAPT2K2 executables) have to be customized by the user. Using the node information from `calcnodes`, the script first creates the files `PROC, PROCc, PROCe, PROCi, PROCs`, and `PROCso` which will be used by MPICH to run the consecutive modules of pSAPT2K2. The input files and the `PROC*` files are then copied from the home subdirectory to the scratch directory on each node. Based on analyzing the `PROC` file, the script detects the number of processors and the "master" node on which `pSAPT.mpich` will be submitted (this is the first node listed in `calcnodes`). Finally, having set up some environment parameters, it launches `pSAPT.mpich` on the "master" node. The temporary files produced by GAMESS and pSAPT2K2 will be created and stored in (local) directories `/tmp/bukowski`. The output from the run will be written to the file `OUT` in the scratch directory (`/tmp/bukowski`) of the "master" node (in this case: `sam2`) only. It will have to be copied (by `rcp`, for example) back to the home subdirectory.

### `huinalu` cluster

The machine consists of 256 batch nodes and 4 interactive (or login) nodes, each equipped with two 933 MHz Pentium III processors and 1 GB of memory. The nodes are connected via a 200 MB/s Myrinet Switch and also via 100 Mbit/s Ethernet. Each node is equipped with a local scratch file-system not accessible from other nodes. On each node, this filesystem is mounted as `/scratch/local`. In addition to the local scratch, all nodes have access to a shared scratch area, mounted as `/scratch/huinalu`, with the capacity of 239 GB. All nodes have also access to users' home directories and other shared resources. The machine features the MPICH implementation of MPI and a variety of Fortran compilers: g77, Intel, and Portland, all capable of using both the Myrinet and the Ethernet connectivity. Special care must be taken at the time of compilation and execution so that the `PATH` variable points to the proper libraries and the `mpirun` command appropriate for a given connection/compiler combination. Commands and programs can be executed on compute nodes from the login node through the `ssh` utility. It is also possible to `ssh`-login to the compute nodes, although it is not necessary for running pSAPT2K2. The consecutive modules of pSAPT2K2 are launched from within the script `pSAPT.mpich.maui` or `pSAPT.mpich.maui.sep` (described in the following) using the appropriate `mpirun` command, while GAMESS is started using the `ddikick.x` utility of the DDI package. The default source of `ddikick.x`, `ddikick.c`, has to be modified to use `ssh` instead of `rsh`.

In the debugging stage, pSAPT2K2 jobs may be submitted on the four interactive nodes `hnfe01` – `hnfe04` (i.e., on up to 8 processors) using a technique similar to that described above for `samson`. The only practical difference is that the invocation of the `mpirun` command utilizes the `-machinefile` option, which for some reason is not working on `samson`. Consequently, the `PROC` file is only needed by GAMESS while the pSAPT2K2 modules use a file called `calcnodes` in a format similar to

```
hnfe01:2
hnfe02:2
hnfe03:2
```

where `:2` means that two processes will be run on each of the three interactive (2-processor) nodes, a total of 6 processes. In this case, a `PROC` file consistent with `calcnodes`, needed to inform GAMESS where to run, will have the form

```
hnfe01
hnfe01
hnfe02
hnfe02
hnfe03
hnfe03
```

The driver script to be called by `submit` is now `pSAPT.mpich.maui` which differs from `samson`'s `pSAPT.mpich` in the syntax of `mpirun` command as well as in some file management portions (it is assumed here that the scratch directory is set to the common filesystem `/scratch/huinalu`). In order to run GAMESS, `pSAPT.mpich.maui` calls a custom script `runGMS.mpich.maui` which differs from `runGMS.mpich` in details of how the DDI's `HOSTLIST` is constructed and in the use of `ssh` instead of `rsh` for internode communication. Jobs on interactive nodes can only be run using the Ethernet network interface.

In the production stage, pSAPT2K2 jobs have to be run on compute nodes. The legal way to accomplish this is to use the `huinalu`'s queuing system called "Maui Scheduler". The scheduler requires the user to prepare two scripts. The first is the scheduler command script which in turn calls the second script performing the file management tasks and invoking the pSAPT2K2 driver script. The format of the scheduler commands script `MAUI_script` is as follows:

```
# Initial working directory and wallclock time
IWD == "/u/bukowski/tests/ArHF_Tc"
```

```
# Job wall-time limit in seconds
WCLimit == 1800

# Task geometry
Tasks == 8
Nodes == 8
TaskPerNode == 1

# Feature requests
Arch == x86
OS == Linux

# Account
Account == "AFPRD-0102-001"

# MPI Ethernet job; for Myrinet, use gm instead of p4
JobType == "mpi.ch_p4"

# Execute a script file managing files and submitting the SAPT job
Exec == "/u/bukowski/tests/ArHF_Tc/sub_s_sep"

# Optional arguments to the "Exec" script
Args == ""

# Where scheduler output will end up
Output == "/u/bukowski/LOGS/$(MAUI_JOB_ID).out"
Error == "/u/bukowski/LOGS/$(MAUI_JOB_ID).err"
Log == "/u/bukowski/LOGS/$(MAUI_JOB_ID).log"

# Input
Input == "/dev/null"
```

The launch script `sub_s_sep` should be similar to

```
#!/bin/sh

JOB=ArHF                # job name
CURDIR=/u/bukowski/tests/ArHF_Tc
SCRDIR=/scratch/local/bukowski      # local scratch directory
SAPT_SCRIPT=/u/bukowski/psapt2K2/bin/pSAPT.mpich.maui.sep
                        # SAPT driver script

# Set the number of nodes as assigned by the scheduler
NPROC=$MAUI_TASK_COUNT

OUTFILE=/scratch/huinalu/bukowski/$JOB.out_tst.$NPROC

# copy all SAPT input files onto the scratch dir
# on every node assigned by scheduler. With common file-system -
# use just a simple copy

cd $CURDIR

# Create the list of nodes

rm -f calcnodes
for node in `echo $MAUI_JOB_NODES | sed -e 's/:/ /g'` ; do
        echo $node >> calcnodes
done
```

```
cp calcnodes PROC

# Copy input files and lists of nodes to each local scratch directory

for i in `awk '{print $1}' calcnodes`
do
echo Copying input to $i
 ssh $i mkdir $SCRDIR
 scp *inp $i:$SCRDIR
 scp $JOB'P.data' $i:$SCRDIR
 scp  PROC $i:$SCRDIR
 scp calcnodes $i:$SCRDIR
done

# Submit the job in scratch directory

cd $SCRDIR

$SAPT_SCRIPT $JOB nocp $NPROC $SCRDIR > $OUTFILE  2>&1

for i in `awk '{print $1}' calcnodes`
do
echo Cleaning up on $i
 ssh $i rm $SCRDIR/*
done
```

The script `sub_s_sep` launches the pSAPT2K2 calculation using the **local** scratch file-systems `/scratch/local`, where all the necessary data files (including input) are copied using the `scp` command. The number and the list of nodes are retrieved from the environment variables `MAUI_TASK_COUNT` and `MAUI_JOB_NODES` supplied from within the scheduler environment. Due to some file management issues which could not be resolved, it is necessary that only one process is launched on each compute node (i.e., the parameter `TaskPerNode` in `MAUI_script` must be set to 1). The actual pSAPT2K2 run is started using the driver `pSAPT.mpich.maui.sep`, adapted to make use of the local scratch environment. Replacing this driver with `pSAPT.mpich.maui`, changing the scratch directory to the global one, say,
`/scratch/huinalu/bukowski`, and replacing the loop structure with `scp` by a set of simple `cp` commands would result in a pSAPT2K2 calculation utilizing the common scratch directory instead of the local ones. Since pSAPT2K2 jobs are crucially dependent on the efficiency of I/O operations, the local scratch option should be preferred as it allows to avoid competition for bandwidth between the processes.

## 14.5 Input files

The input files for GAMESS and the SAPT suite of codes are constructed in essentially the same way as in the case of the sequential version of the program, SAPT2020, as described in detail in Sec. 10. The only difference is that the options `SAPT0`, `SAPT2`, `SAPT`, and others of this kind introduced in later editions of SAPT codes are currently not supported in the namelist `INPUTCOR`. The theory levels corresponding to these options may of course be recovered by requesting the appropriate SAPT corrections individually. Note also that no SAPT corrections beyond the standard level, like, e.g., the components of $E^{(30)}_{\mathrm{SAPT}}$, Eq. (9), are available in pSAPT2K2, and frozen core is not implemented in the parallel SAPT program.

## 14.6 Memory and disk requirements

Memory requirements of pSAPT2K2 can be estimated based on the total number of basis functions, the numbers of occupied and virtual orbitals, and the number of processors involved. The transformation code `ptran` requires slightly more than $on^3/P$ 8-byte words on each of the $P$ processors, where $n$ is the dimension of the basis set and $o$ is the number of occupied orbitals of the bigger monomer. For the `pcc` and `psapt` parts, memory requirements can be roughly estimated as $3o^2v^2$ words on each processor, where $v$ is the number of virtual orbitals for the larger monomer.

More precise calculation of the memory required for a pSAPT2K2 run can be done using the program `memcalc` (built automatically during installation). In order to use this program, prepare the regular `nameP.data` file with the first (title) line containing, in this order: the total number of DC$^+$BS basis functions, the total number of functions for monomer A (will be different than the previous one in a MC$^+$BS-type run), the number of occupied orbitals for monomer A, and then the same information for monomer B, followed by the number of processors. Then run the program by typing

```
    memcalc < nameP.data > memcalc.out
```

The file `memcalc.out` will contain detailed information about memory requirements of different parts of the code. The variables `MEMTRAN` from namelist `TRN` and `MEMSAPT` from namelist `INPUTCOR` should then be adjusted accordingly. The memory needed to run the `pcc` program is calculated there explicitly, so that no separate namelist variable is needed.

An estimate of the total disk space requirement can be obtained as presented in Sec. 10.7 for a sequential program. It should be noted here that most of the large scratch files are distributed between the processes. The amount of disk space needed on a single processor is thus only a fraction $1/P$ of the total disk requirement. This feature becomes important on Beowulf clusters, where typically only a relatively small scratch area is available on each node.

## 14.7 Electrostatic, dispersion, and induction (EDI) energies from monomer properties

It is well known that the electrostatic component of the interaction energy can be calculated from monomer *charge densities*, which are purely monomer properties. Similarly, the (second-order) induction energy may be expressed in terms of monomer charge densities and the *static* *susceptibility functions*. The same holds also for the dispersion energy, which is given by an integral of the product of two monomer *dynamic susceptibility functions* over (imaginary) frequency, the so-called Casimir-Polder integral. Thus, with the exception of exchange energies, all the components of the interaction are in fact determined exclusively by monomer properties. These properties can be calculated only *once* for each monomer at a high level of (intramonomer) correlation, and then used, after rotating and translating to the new monomer positions, to obtain the three interaction components for *any* dimer configuration. The point is that the expensive parts of the calculation – those of the correlated charge density and susceptibility functions – have to be performed only once for each monomer, and not at each and every dimer geometry. In particular, costly four-index transformations usually needed in highly correlated methods (such as the four-virtual transformations) are done only for a single-point monomer calculation, and avoided during the actual interaction energy calculations.

The electron density of a monomer can be expressed as a combination of $n(n+1)/2$ products of atomic orbitals (AOs), where $n$ denotes the number of these orbitals in the basis set used. Thus, a calculation of the electrostatic energy from two precomputed monomer densities requires of the order of $n^4$ 2-electron integrals (for similar size monomers, so that $n$ is approximately the same for both of them). The monomer susceptibility functions (both static and dynamic) are given as combinations of $ov*(ov+1)/2$ terms (products of four molecular orbitals, two depending on coordinates of electron 1 and the other two on coordinates of electron 2). The time cost of obtaining induction and dispersion components is again dominated by the need to compute $n^4$ 2-electron integrals. This unfavorable scaling can be greatly reduced if both the electron densities and the susceptibility functions are fitted in terms of a suitably chosen auxiliary basis. In most cases, the size of this basis, $m$, can be assumed to be proportional to the size of the original AO basis (but several times larger). The electron density can now be expressed as a combination of $m$ terms, while the susceptibility functions consist of $m(m+1)/2$ terms (products of the auxiliary basis functions). The cost of the calculation of electrostatics, induction, and dispersion, dominated by 2-electron integrals between the auxiliary functions, is now proportional to just $m^2$ instead of $n^4$.

The electron densities and susceptibility functions are fitted by minimizing functionals of the type

$$\Delta = \int [\rho(\boldsymbol{r}_1) - \overline{\rho}(\boldsymbol{r}_1)](1/r_{12})[\rho(\boldsymbol{r}_2) - \overline{\rho}(\boldsymbol{r}_2)]d\boldsymbol{r}_1 d\boldsymbol{r}_2 \tag{12}$$

$$\int \overline{\rho}(\boldsymbol{r}_1)d\boldsymbol{r}_1 = N \text{ or zero.} \tag{13}$$

The quantity $\rho$ denotes here either one of the MBPT contributions to electron density, which is normalized to the number of electrons $N$, or a transition density (a product of an occupied and a virtual orbital), which is normalized to zero. The quantity $\overline{\rho}$ is an approximation to $\rho$ in the form of a linear expansion in terms of auxiliary basis functions. The choice of $1/r_{12}$ as the “weight” in the functional $\Delta$ has been suggested by Dunlap [61]. Minimization of the functional $\Delta$ reduces then to solving a system of linear equations for the expansion coefficients. Only one such equation system needs to be solved for each of the MBPT components of density, whereas fitting of the susceptibility function for each imaginary frequency requires solving $ov$ such systems, one for each pair of the occupied and virtual orbitals.

Presented here is the *beta* version of the suite of codes implementing the ideas discussed above. The codes allow the generation of monomer charge densities (currently at the SCF, MBPT2 and MBPT3 levels, with and without orbital relaxation) and susceptibility functions (currently at the CHF level), generation of an auxiliary basis, and the fitting of the monomer properties in terms of this basis. The charge densities, susceptibility functions, and SCF vectors are then used as input to a program which calculates the electrostatic, induction, and dispersion energies for an arbitrary set of dimer geometries. Two versions of this code exist, one utilizing the exact representation of monomer properties (i.e., in terms of the original AO basis), called `caldisp_gms`, and the other – working with the density-fitted monomer properties, called `caldisp_fit`.

### 14.7.1 The `pEDI` scripts

Most of the the tasks mentioned above are accomplished with the help of the script `pEDI.X`, where `X` stands for the platform at hand. This script works very much like the script `pSAPT.X` used in “regular” pSAPT2K2 calculations. Thus, the SCF runs for the two monomers are performed first, followed by the integral transformation, the CCSD/MBPT, and `psapt` runs. A new element is that the `psapt` module now calculates the MBPT2 and MBPT3 densities and stores them on disk, and the `pcksdisp` program is called to produce the susceptibility functions, currently at the CHF (or RPA) level. In addition to generating the monomer properties, the regular pSAPT2K2 calculation is also performed for one dimer geometry, obtained by shifting the position of monomer B by 10 bohr along the positive $z$ axis with respect to the position specified in the `nameB` file. The CHF dispersion energy is also computed for this geometry by the program `pcksdisp` (the same one that generates the susceptibility functions). The results for this one geometry, reported in the output from the `pEDI.X` run, may then be compared to their counterparts obtained using the transformationless codes, to assess the correctness and accuracy of the latter. The transformationless code itself, `caldisp_gms`, is invoked at the end of `pEDI.X` to compute the electrostatics, induction, and dispersion energies for specified dimer geometries.

The detailed instructions for running `pEDI.X` are as follows:

1. Assuming that the name of your job is `name`, prepare the files `nameA` and `nameB` containing the specifications of the geometries and basis sets of monomers A and B in GAMESS(US) format.

Remember to put blank lines after basis set of each atom, including the blank at the end of the file. Out of these files, the `pEDI.X` script will construct GAMESS input files needed to calculate the SCF vectors and generate integrals. A spherical basis set will be assumed (i.e., `ISPHER=1`), and the name of the job will be used as the comment line. Geometries of the monomers specified in `nameA` and `nameB` should correspond to whatever you consider to be the initial configuration of the given monomer, i.e., the COM (or other characteristic point around which rotations will be performed) should coincide with the origin of the coordinate system, and all Euler angles describing the orientation of the monomer will be assumed zero at this geometry.

2. Prepare the file `nameP.data`, as for the regular (pure) MCBS SAPT calculation (note that in namelist `TRN` the variables `BASIS` and `TAGS` have to be specified for such a calculation). In the namelist `INPUTCOR`, the variables `CHFDISP=T, CHFIND=T, E12=T, E12R=T, E13PL=T, E13PLR=T` have to be specified in order for the MBPT2 and MBPT3 densities to be dumped on disk. Failure to specify any of these corrections will result in the corresponding density missing in the dump files `denaMO.data` and `denbMO.data` (it does not hurt the subsequent calculations except that the quantities requiring the missing densities will be reported as zero). Also, some other corrections may be specified, so that the results from the `Summary Table` may be later compared to what comes out of the transformationless algorithms for the geometry considered in the `pEDI.X` run. These may include `E1TOT=T, E2IND=T, E2INDR=T`, and `E2DSP=T`. The namelist `INPUT` controlling the behavior of the propagator code `pcksdisp` is generated automatically by the `pEDI.X` script (and appended to `file5.dat`), so the user does not have to worry about it. For the sake of completeness we state here that this namelist currently contains the following:

```
ISITCASPOL=T, ISITINDUCT=T,
ISITSOSDISP=F,
ISITPROP=F,
ISITCKS=T, ISITUCKS=F,
ISITPOL=F,ISITC6DISP=F,
USESUMN6=T,
MAKEH1H2=T,
IQUADTYP=1, NQUAD=16,
OMEGA0=0.5
```

The meaning of these parameters is described in Appendix E. In particular, the user may want to alter the length `NQUAD` of the quadrature employed in the calculation of the Casimir-Polder integral.

3. Prepare the file `input.edi`, specifying the dimer geometries for which the interaction energies are to be calculated by `caldisp_gms` out of the precomputed monomer properties. `input.edi` consists of lines (one for each dimer geometry) containing, in this order, the COM-COM separation (in Å), the $\beta$ and $\gamma$ Euler angles (in degrees) for monomer A ($\alpha$ is assumed as zero), followed by the $\alpha, \beta$, and $\gamma$ Euler angles of monomer B. It is assumed that the COM of monomer A coincides with the origin of the coordinate system and that monomer B is shifted in the positive direction of the $z$ axis. For example,

   ```
   5.29177249 0.          0.            0.          0.           0.
   2.38521611 86.80692336 90.00000000 180.00000000 93.37890335 360.00000000
   2.63658901 87.13779711 90.00000000 180.00000000 81.94039222 360.00000000
   2.89422970 87.42491802 90.00000000 180.00000000 75.80434134 359.99999915
   ```

   would be a valid `input.edi` file containing 4 geometries. The first of these geometries is exactly the same as the one for which the interaction energies are computed during the `pEDI.X` run.

4. Run the EDI job the same way a "regular" pSAPT2K2 job would be run. For example, on O2K/O3K platform without a queuing system, to run in /scratch/mydir on 8 processors one would type

   ```
   pEDI.o2k name noscfcp 8 /scratch/mydir > name.out 2>&1 &
   ```

   Note that the `noscfcp` keyword has been used since the supermolecular SCF interaction energy is of no interest here.

A result of running the `pEDI.X` script will be the file `name.out` containing standard output from the whole run. In this file, the SAPT corrections requested in the `nameP.data` file will be reported (in the "Summary Table" section), together with the dispersion and induction energies calculated by the propagator code `pcksdisp` for one specific dimer configuration, described earlier. The last part of `name.out` will be the output from the `caldisp_gms` code, i.e., the electrostatic, induction, and dispersion energies for all geometries specified in `input.edi`. Specifically, the following corrections will be calculated: $E^{(10)}_{\rm elst}, E^{(12)}_{\rm elst}, E^{(12)}_{\rm elst,resp}, E^{(13)}_{\rm elst}, E^{(13)}_{\rm elst,resp}, E^{(20)}_{\rm ind}, E^{(20)}_{\rm ind,resp}, E^{(20)}_{\rm disp}$, and $E^{(2)}_{\rm disp}$(RPA). The last of the corrections mentioned above is calculated from the dynamic susceptibility functions at the CHF level (equivalent to RPA) and currently does not have its counterpart among the corrections calculated in a regular pSAPT2K2 run. It is more accurate in terms of theory level than $E^{(20)}_{\rm disp}$.

The calculated monomer properties will be packed for further use (e.g., for a standalone invocation of `caldisp_gms`) into a file `name.prop.tar.gz`, which, after running

```
%
   gzip -d name.prop.tar.gz
   tar -xvf name.prop.tar
```

will decompress info the following files:

1. `vecta.data` and `vectb.data` – unformatted sequential files with orbital energies and SCF vectors
2. unformatted sequential files `denaMO.data` and `denbMO.data`, containing MBPT2 and MBPT3 densities (relaxed and non-relaxed)
3. `propa.data` and `propb.data` – dynamic susceptibility functions, currently at the CHF level (unformatted sequential files)
4. `prop0a.data`, `prop0b.data` – static susceptibility functions, currently at the CHF level (unformatted sequential files)
5. geometry and basis set info of the monomers will be recorded in formatted files `infoa.data` and `infob.data` in the format readable by the subsequent transformationless codes.

After collecting these files and `input.edi` in one scratch directory, one can run the `caldisp_gms` program independently of the `pEDI.X` script by typing (or submitting through the queuing system) a command similar to

```
mpirun -np 8 caldisp_gms > dimers.out 2>&1
```

modifying, of course, the number of processing nodes, paths, and output file name appropriately. As usual, the executable `caldisp_gms` can be found in `psapt2K2/bin` directory.

### 14.7.2 Calculating electrostatic, induction, and dispersion energies from fitted monomer electron densities and susceptibility functions

A promising alternative route of EDI calculations utilizes approximate representations of monomer properties in terms of the auxiliary basis sets. The accuracy of this method strongly depends on the quality of these sets. The auxiliary basis sets can be obtained using the utility program `make_aux`, realizing the method described in Appendix F, and the property fitting can be accomplished with the help of the program `tdenfit` (the relevant lines in the `compall` script must be commented out if these two programs are to be built during installation). Later, in the work on SAPT(DFT), it was found [14, 15] that the optimized auxiliary basis sets like those from Ref. 62 perform better. Thus, we recommend to use these sets in EDI calculations.

The driver scripts `pEDI.X` can be easily adapted to perform the property fitting by uncommenting the `Generate auxiliary basis` and `Fit propagators and densities using auxiliary` `basis` sections. The invocation of the `tar` command should also be changed so that the fitted representations of monomer properties are included in the `name.prop.tar` file. The `pEDI.X` script modified in this way will ask for two additional input files, `input.aux.A` and `input.aux.B`, specifying parameters needed for the construction of auxiliary bases. These involve the number of pruning cycles and the $\epsilon$ parameter for each of these cycles for each atom. For example, in the case of molecule A consisting of 2 atoms and with 3 pruning cycles required for each of them, the file `input.aux.A` could look like

```
3
0.8
0.9
1.0
3
0.8
0.9
1.0
```

After the run finishes, the auxiliary bases generated by `make_aux` will be placed in files `auxa.data` and `auxb.data`, similar to `infoa.data` and `infob.data`, whereas the fitted properties will be placed in (unformatted sequential) files `auxdena.data`, `auxdenb.data`, `auxprop0a.data`, `auxprop0b.data`, `auxpropa.data`, and `auxpropb.data`. The automatic generation of auxiliary basis sets results in rather large sets if high accuracy of the fits is required.

To perform the interaction energy calculation using density-fitted monomer properties, collect all these files, along with `input.edi`, in a scratch directory (e.g., by uncompressing the `name.prop.tar.gz` file resulting from the modified `pEDI.X` script) and then run the `caldisp_fit` executable by typing a command similar to

```
mpirun -np 8 caldisp_fit > dimers.out 2>&1
```

possibly modifying the number of processing nodes, paths, and output file name. The output file, `dimers.out` in this case, will contain, for each dimer geometry, the following corrections: $E^{(10)}_{\rm elst}$, $E^{(12)}_{\rm elst}$, $E^{(12)}_{\rm elst,resp}$, $E^{(13)}_{\rm elst}$, $E^{(13)}_{\rm elst,resp}$, $E^{(20)}_{\rm ind,resp}$, and $E^{(2)}_{\rm disp}$(RPA).

*Important note*: The number of processors to run `caldisp_gms` and `caldisp_fit` is independent of the number of processors on which the `pEDI.X` script was run. It is, however, important, that all the files used (with an exception of `input.edi`) are obtained in a single `pEDI.X` run. For example, using `vecta.data` and `vectb.data` obtained on a different number of processors (or a different machine) than that used to get `denaMO.data, denbMO.data` or `propa.data, propb.data` may result in nonsensical results if orbital degeneracies are present for one (or both) monomers as it is the case, e.g., for atoms and linear molecules. In such cases, GAMESS can perform arbitrary rotations within the degenerate subspaces. These rotations are dependent on the number of processors and even on the platform where the calculation is run, so it is imperative that all the files are obtained consistently. It is therefore recommended that the monomer properties are always moved around in the form of the compressed files `name.prop.tar.gz` rather than separately.

## 15 SAPT(CC)

A SAPT version of first and second order in $V$ and of infinite order in $W$ at the CCSD level, developed by Korona *et al.* [24–31], is available in SAPT2020. Such calculations require MOLPRO2010.1 since the code for these corrections has a form of a MOLPRO patch. This patch is located in the `misc/patch.ccsapt` subdirectory of regular SAPT2020. In order to run these codes, one must apply the patch by copying all the files from the `misc/patch.ccsapt/src/eom` subdirectory to the `src/eom` subdirectory of an existing MOLPRO2010.1 installation and recompiling MOLPRO. Example inputs for SAPT(CC) runs are provided in the `misc/patch.ccsapt/examples` subdirectory. One should note that the SAPT(CC) patch is independent from the patch required to run regular SAPT with MOLPRO as the integral and SCF front end (Sec. 8.1) and the two patches can, but do not have to, be applied simultaneously. No further information is available on SAPT(CC).

## 16 SAPT(DFT)—SAPT based on coupled Kohn-Sham treatment of monomers

### 16.1 Introduction

SAPT(DFT) is an extension of the SAPT theory. In SAPT(DFT), the monomers are described in terms of Kohn-Sham (KS) orbitals and orbital energies as well as of TD-DFT response functions. A complete description of the theory together with references to the historical developments of the method, as well as numerical examples, can be found in Ref. 14.

SAPT(DFT) consists of two steps. First is the so-called SAPT(KS), where electrostatics, first-order exchange, induction, exchange-induction, dispersion, and exchange-dispersion are obtained by using SAPT terms of the zeroth order in $W$ with SCF orbitals and orbital energies replaced by their KS equivalents. For meaningful results, SAPT(KS) requires asymptotically corrected (AC) Kohn-Sham calculations. SAPT(KS) does not reproduce dispersion correctly. This problem was found to be due to the use of a formula asymptotically related to uncoupled dynamic polarizabilities. Instead, the dispersion (and induction) energies should be calculated from frequency-dependent density susceptibility (FDDS) functions, also referred to as propagators, obtained from TD-DFT, i.e., at the coupled Kohn-Sham (CKS) level. The SAPT(KS) and CKS steps form the complete

SAPT(DFT). The total SAPT(DFT) interaction energy (up to second order in $V$) can be defined as [14]

$$\begin{aligned} E_{\rm int}^{\rm SAPT(DFT)} &= E^{(1)}_{\rm elst}({\rm KS}) + E^{(1)}_{\rm exch}({\rm KS}) + E^{(2)}_{\rm ind}({\rm CKS}) + E^{(2)}_{\rm exch-ind}({\rm CKS}) \\ & \quad + E^{(2)}_{\rm disp}({\rm CKS}) + E^{(2)}_{\rm exch-disp}({\rm CKS}), \end{aligned} \tag{14}$$

where the terms with the CKS label result from the coupled Kohn–Sham approach. The method can be shown to be potentially exact for all major components of the interaction energy (asymptotically for exchange interactions) in the sense that these components would be exact if the DFT description of the monomers were exact. The nominal scaling of SAPT(KS) is $O(N^5)$ and that of SAPT(DFT) is $O(N^6)$, in both cases significantly better than the $O(N^7)$ scaling of the regular SAPT. The scaling of SAPT(DFT) can be lowered to $O(N^5)$ by using density fitting [12, 14, 15, 58, 63]. Furthermore, although the density-fitting transformation still scales as $O(N^5)$, it has a greatly reduced prefactor.

SAPT2020 includes two versions of SAPT(DFT). One of them is our first implementation of the method, and requires DALTON 2.0 [38] to function. The other one is aimed towards computing interaction energies of large dimers (containing more than 100 atoms), and is interfaced with the ORCA program [54].

## 16.2 Installation and usage

SAPT(DFT) requires DALTON 2.0 [38] or ORCA [54] for monomer DFT calculations (in the latter case, only the density-fitting version has been interfaced, because ORCA does not provide two-electron integrals needed for the standard version. The users of SAPT(DFT) need to obtain a separate license and download the package from
`http://www.kjemi.uio.no/software/dalton/dalton.html`. Consult the DALTON manual for installation requirements. Some of the SAPT(DFT) functionality is available via a patch `dalton.diff` distributed together with SAPT2020. First, compile and install the standard DALTON 2.0. After testing the installation, update the path to the DALTON directory in the `patchdalton` script located in the main SAPT2020 directory and run the script. If the patching is successful, compile the patched DALTON. The recompilation should last significantly shorter than the original compilation. If you have not compiled SAPT2020 with DALTON support, set the `DALTON` variable of the `Compall` script to the proper path to the DALTON executable, make sure the variable `SAPTDFT=YES`, and recompile SAPT2020. If the compilation is error-free, the SAPT(DFT) code is ready to use (both the regular and density-fitted versions are created). Before performing calculations for systems of interest to you, check the examples from `examples/saptdft` directory. The SAPT(DFT) run consists of the monomers calculation, transformation, the SAPT(KS) step, and the CKS step.

Notice that some of the errors printed by Dalton are harmless in the SAPT(DFT) calculations. In particular,

```
 --- SEVERE ERROR, PROGRAM WILL BE ABORTED ---
  SIRIUS NORMAL STOP AFTER ORBITAL ORTHONORMALIZATION.
```

is a normal stop after calculating the necessary one-electron integrals. **Warning.** Patched DALTON should not be used for any ather purpose except for SCF/DFT calculations. The effect of the patch on other parts of the DALTON code has not been tested.

The keywords discussed below are for both the regular and density-fitted versions of SAPT(DFT). Some additional keywords for the latter case are described in Sec. 16.4. SAPT(DFT) requires `SAPTKS=T` keyword in the `INPUTCOR` namelist of the `nameP.data` file and proper keywords in the `TRN` namelist prepared like for the regular runs (see also Sec. 9.5). For CKS calculations that are a part of SAPT(DFT), a separate `SAPTDFT` namelist with keywords `CKSDISP=T` and `CKSIND=T` should also be present at the minimum. The complete set of `SAPTDFT` namelist keywords is:

1. `CKSDISP` —$E^{(2)}_{\rm disp}$(CKS)
2. `CKSIND` — $E^{(2)}_{\rm ind}$(CKS)
3. `MAXMEM` — maximum memory in words to be used in the CKS calculation. The program will allocate necessary memory not greater than `MAXMEM`. If the required memory is greater than `MAXMEM`, the program will exit. If not set, up to 100 megawords will be used.
4. `IQUADTYP` : The type of quadrature scheme to be used in performing the $\omega$-integral in the Casimir-Polder dispersion formula:
   - `IQUADTYP=1` sets the Gauss-Legendre quadrature with the transformation of the integral variable $\omega = \omega_0 \frac{(1+t)}{(1-t)}$. This is the default.
   - `IQUADTYP=2` sets the Gauss-Legendre quadrature with the transformation $\omega = \omega_0 \tan(t)$.
   - `IQUADTYP=3` sets the Gauss-Laguerre quadrature scheme.
5. `NQUAD` : The variable `NQUAD` sets the number of quadrature points to be used for the integration. The default is 8.
6. `OMEGA0` and `ALPHA` : The transformation used in the Gauss-Legendre quadrature schemes involves a constant $\omega_0$. The variable `OMEGA0` in the namelist `SAPTDFT` allows you to set this constant. It is typically between 0.3 and 0.5 and the default is 0.5. The Gauss-Laguerre quadrature scheme involves the constant $\alpha$. For the integrals encountered here, one should set `ALPHA=0.0`.

The DALTON patch enables some new options in DALTON which are required for SAPT(DFT) calculations. All the new keywords belong to the `*DFT INPUT` submodule. The following new keywords are implemented:

1. `.DFTAC` — the Fermi-Amaldi asymptotic correction with the Tozer-Handy splicing scheme [64]. The subsequent line should contain 4 numbers: `DFTIPT`, `DFTBR1`, `DFTBR2`, `DELTAIPT`. The first parameter, `DFTIPT`, is the ionization potential of the monomer in atomic units. It is recommended to use an experimental value. The parameters `DFTBR1` and `DFTBR2` are related to the Tozer-Handy switching function and are distances, in Bragg radii, where the switching takes place. Tozer *et al.* recommend values of 3 and 4 [65]. The last parameter, `DELTAIPT`, is reserved for the open-shell program and should be set to zero for all closed-shell systems. For open-shell calculation it is equal to $\mathrm{IP}_\beta - \mathrm{IP}_\alpha$, where $\mathrm{IP}_\beta$ are $\mathrm{IP}_\alpha$ are ionization potentials for $\beta$ and $\alpha$ electrons, respectively.

2. `.CKS` — Calculates TD-DFT integrals for the CKS program. Mandatory if the CKS code is used. By default, the LDA kernel is used in TD-DFT (see point 3 below).

3. `.GGAKER` — Use generalized gradient approximation (GGA) instead of LDA in the TD-DFT kernel. It is significantly slower and is not recommended for large systems. More details and a discussion of accuracy of the LDA kernel is given in Ref. 14.

4. `.GRAC` — Gradient-regulated asymptotic correction (GRAC) of Ref. 37. The subsequent line should contain 4 numbers. The first and last one are identical to the Fermi-Amaldi asymptotic correction (see above) and describe the ionization potential. The second and third numbers describe switching function parameters. Recommended values are 0.5 and 40, respectively.

For the DFT calculations, we recommend the standard Dalton grid or one of the denser grids. For the `.THRESH` parameter in `*HF INPUT`, we recommend values of about $10^{-7}$.

Since SAPT(DFT) currently works with DALTON interface only (and with ORCA in the density-fitted version), a few scripts are located in `misc/daltutil` to help in creating DALTON input files. Script `atmol2dalton` converts ATMOL `name.intinp` input files into `name.mol` required by DALTON, `altergeo.awk` updates the `name.mol` geometry, and with `createinputs` one can construct `name.mol` files using basis sets included in the DALTON distribution. The usage of the scripts is explained in the sources. The `name.dal` files can be taken from the examples.

One precaution when using SAPT(DFT) for systems with large monomer separation is that SAPT(DFT) does not implement strict monomer charge neutrality. This constraint was found to reduce accuracy in typical calculations, but may cause incorrect behavior of electrostatics if monomer separation is on the order of 100 Angstrom. In fact, for intermonomer separation larger than 20 Angstrom, we strongly recommend the use of quadruple precision in the inversion of the auxiliary component $J_{KL}$ (please see sec. 16.4 for further detail). Additionally, we also recommend to remove any midbond functions from the basis set, since in such conditions these functions lead to significant numerical error, without improving the results. Finally, for even longer distances we recommend resorting to calculations based on the asymptotic expansion instead of using SAPT; such calculations are available in the package AutoPES [35].

## 16.3 Terms beyond second order in the interaction operator

The SAPT(DFT) formalism presented above has been implemented up to second order in the operator $V$. For systems with a significant induction component (e.g., the water dimer), some higher-order terms constitute a large fraction of the total interaction energy. Those terms can be estimated from the supermolecular HF-SCF approach, with $\delta E^{\mathrm{HF}}_{\mathrm{int,resp}}$ defined in Eq. (5). Since no such term can be computed from a pure DFT calculation, this term has to be extracted from a separate SAPT/HF-SCF run. For the calculation of $\delta E^{\mathrm{HF}}_{\mathrm{int,resp}}$, no expensive SAPT terms are needed and the scaling is $O(N^5)$. Nevertheless, the time of this calculation would be similar to that of the SAPT(DFT) step since the costly transformation step must be done again. To calculate only $\delta E^{\mathrm{HF}}_{\mathrm{int,resp}}$, one should prepare inputs as for the regular SAPT runs (any interface can be used as long as the geometry and basis sets are the same as in the SAPT(DFT) run), set `DELTASCF=T` in the `INPUTCOR` namelist (cf. Sec. 10.2.3), and use `scfcp` option for the `SAPT` script. The $\delta E^{\mathrm{HF}}_{\mathrm{int,resp}}$ value should then be added to the SAPT(DFT) interaction energy. An example of such a calculation is in the directory `examples/saptdft/H2O-aTZ-MC`.

## 16.4 Density-fitting version of SAPT(DFT)

Density-fitting (DF) method (also known as resolution of identity) has been implemented in SAPT(DFT) [15, 58]. The resulting approach has the scaling reduced from $O(N^6)$ to $O(N^5)$. Substantial savings are also achieved for $O(N^5)$ terms. The code with density fitting is also more memory and disk efficient.

Running DF-SAPT(DFT) requires some modifications of the input files and the creation of a file with an auxiliary basis set. The latter file should be named `name.aux` and contains coefficients of the auxiliary basis function for each atom. The format is the following (see also `examples/dfsaptdft`):

```
O z
charge 0.0 0.0 0.0
coefficients
...
```

```
O z
charge 0.0 0.0 0.0
coefficients
```

where `charge` is the charge of an atom and the coefficients are in the TURBOMOLE format. We recommend basis sets from Ref. 62. These basis sets are available in `turbomole` format at `ftp://ftp.chemie.uni-karlsruhe.de/pub/cbasen/`. The number and ordering of the atoms must be identical to the main basis set used. The quality of the auxiliary basis set is critical for the accuracy of the DF approach and using auxiliary basis set optimized for a different main basis would result in poor accuracy.

In the `TRN` namelist of the `nameP.data`, one should specify `T2EL=F` to suppress the standard two-electron transformation. There is also an additional `DF` namelist. The only parametr used in this memlist is `MEMTRAN=x` keyword, where `x` is equal to either the number of requested memory words or zero. In the latter case, the memory is allocated automatically to the lowest reasonable level. Finally, for running DF-SAPT(DFT), one uses `SAPTdf` and `RunlotDALTONdf` scripts instead of `SAPT` and `RunlotDALTON`, respectively. Example inputs and scripts for running DF-SAPT(DFT) (analogous to non-DF SAPT(DFT) counterparts in `examples/saptdft`) are located in `examples/dfsaptdft`, along with the pertinent outputs. Notice small DF errors when comparing the results of DF and non-DF calculations.

For larger auxiliary basis sets, where the linear dependencies start to emerge, the numerical errors start to amplify and the accuracy of the electrostatic energy is diminished. The problem has been described in Ref. 58. An effective solution is to perform the inversion of the auxiliary $J$ matrix with quadruple precision (QP). For compilers supporting QP, one should use this option to improve the accuracy. The QP code has been tested with `ifort` and `ibm64` platforms. To compile the QP code, one should include `QUADINV=YES` in the `Compall` script. If the compiler does not support quadruple precision, the compilation will fail. Since QP calculations are more time consuming than double precision ones, the QP inversion is not turned on by default. To use it, one needs to also specify `quad` option for `SAPTdf` or `RunlotDALTONdf` scripts. With this option, after the SAPT(DFT) calculation is complete, an extra calculation of the electrostatic energy is performed. This result replaces the standard double precision results in the summary table. By comparing the double and quadruple precision values, one can estimate the severity of the quasi-linear dependence. Additionally, slightly better accuracy of density-fitting of the electrostatic energy is obtained by using basis sets of Ref. 66 rather than those of Ref. [62]. One can include such basis set in `name-elst.aux` and it will be used instead of `name.aux` in the quadruple precision calculation of the electrostatic energy. If the file is missing, the standard `name.aux` will be used. DF-SAPT(DFT) is also partially parallelized. All calculations performed by the (patched) DALTON program can be run in parallel. Also the calculations of the CKS kernel integrals are parallelized.

The rest of the programs is not parallelized but can be used with parallel DALTON. DF-SAPT(DFT) is limited to up to $g$ functions in the auxiliary basis set. This restriction is due to the use of GAMESS integrals. In addition, when using the ORCA interface, the main basis set is currently limited to up to $f$ functions (in practice, both these restrictions are equivalent, because the auxiliary basis set should contain functions with the highest angular momentum $L$ at least by one larger than the main basis set). The density-fitting code contains some atomic integrals code from GAMESS. The GAMESS authors requested that the users of density-fitting SAPT(DFT) should cite GAMESS [22] along with SAPT.

### 16.4.1 Using DALTON for monomer DFT calculations

The DALTON input files required in a DF-SAPT(DFT) run are almost identical with those in regular SAPT(DFT). However, in `name.dal` files, `CKS` keyword should be replaced by `CKSAUX`. This keyword works only with the LDA kernel, therefore, `GGAKER` keyword cannot be used. It it recommended to include `.NOTWO` keyword in `**INTEGRALS` of `nameA.dal` and `nameB.dal` when monomer basis sets are used.

### 16.4.2 Using ORCA for monomer DFT calculations

*Note: currently, most of the functionality described in this subsection has been reworked in the* `fastdf` *program. We recommend the users to use that instead, see Chapter 17.*

ORCA is a quantum chemistry package developed by Neese and coworkers [54]. Its DFT module has very efficient density-fitting techniques and we recommend this front-end for DF-SAPT(DFT) calculations, especially in the case of monomers larger than about a dozen of atoms. The Linux, Windows, and Mac-OS X binaries are available at `http://www.thch.uni-bonn.de/tc/orca`, free of charge for academic users (a registration and acquiring a license is necessary). The ORCA source files are not needed by DF-SAPT(DFT). Version 2.9.0 or later is required, because earlier versions contain errors in the GRAC asymptotic correction code and lead to incorrect SAPT results. The installation of ORCA can be done before or after the installation of SAPT, as both are completely independent, but in the latter case one has to edit the file `vars.cfg` so that the variable `ORCADIR` is set to the directory with the ORCA executables.

To use DF-SAPT(DFT) with ORCA, ISITORCA=T must be set in the `TRN` namelist of the `nameP.data` file. The required monomer and/or dimer ORCA input files follow the same naming pattern (ending with `.data`) as in the case of GAUSSIAN (see Sec. 10.1 for the detailed description). Each such file, besides containing regular ORCA input in the first section, must end with a special second section starting with a line containing the string META as its first four characters. This section always defines the molecular geometry and basis set used in the DFT calculation (even if a basis set is already defined in the first section, it gets overridden here). Additionally, if one of the density-fitting DFT algorithms implemented in ORCA is invoked, the META section *can* be used to override the auxiliary (density fitting) basis set defined before or provided by ORCA as a default. The example shown below can be used to performed the DFT calculation of the first monomer within the water dimer, using the full dimer-centered basis set (file `nameA.data`).

```
! PBE def2-svp def2-svp/J RI TightSCF bohrs

%method
Grid 4 FinalGrid 5
gracLB true
ip 12.62
end

META
  6     0 1
8       0.1088092367800     0.0000000000000    -0.0607755435800   8  T
1      -0.1616146814400     0.0000000000000     1.7553074350400   1  T
1      -1.5793331071000     0.0000000000000    -0.7828987377400   1  T
8      -0.0964834851100     0.0000000000000     6.5351482873200   8  F
1       0.7718678809000     1.4536519623000     7.2451823417100   1  F
1       0.7718678809000    -1.4536519623000     7.2451823417100   1  F

BAS 1
3
 0 3
         13.0107010000      0.0334854882
          1.9622572000      0.2347218706
          0.4445379600      0.8137702843
 0 1
          0.1219496200      1.0000000000
 1 1
          0.8000000000      1.0000000000

BAS 8
6
```

```
0 5
       2266.1767785000     -0.0053893504
        340.8701019100     -0.0402347214
         77.3631351670     -0.1800818421
         21.4796449400     -0.4682885766
          6.6589433124     -0.4469261716
0 1
          0.8097597567      1.0000000000
0 1
          0.2553077223      1.0000000000
1 3
         17.7215043170      0.0626302488
          3.8635505440      0.3333113849
          1.0480920883      0.7414863830
1 1
          0.2764154441      1.0000000000
2 1
          1.2000000000      1.0000000000
```

The first line declares the use of the PBE functional, def2-SVP as the main basis set and def2-SVP/J as the Coulomb-fitting auxiliary basis set, as well as tight convergence criteria for the SCF procedure and the use of bohr units in the geometry section (see the ORCA manual for details). The `gracLB true` setting is essential for any SAPT(DFT) calculation as it turns on the GRAC asymptotic correction of the exchange-correlation functional (the other type of asymptotic correction, Fermi-Amaldi, is currently not implemented in ORCA). The correction requires the ionization potential of the molecule, which *must* be given in electron-volt units (as opposed to the hartree units used in DALTON). The META section starts with a line containing three integers: number of atoms (including ghosts), total charge and multiplicity. For each atom, there is a line containing: basis set identifier, x, y, and z Cartesian coordinates, and atomic number (1 for midbond functions), and a logical value specifying whether it is a ghost atom (T = real, F = ghost). In the above example, the basis identifier is set to the atomic number, but this choice is arbitrary. The main basis set for each element is defined in a block (separated by exactly one empty line from previous blocks) starting with a line containing the string BAS as the first three characters and the basis set identifier. The next line of the BAS block gives the total number of contracted orbitals. For each such orbital, there is a sub-block starting with two integers (angular momentum and number of primitive functions) and then containing, on separate lines, pairs of exponential Gaussian parameters and contraction coefficients. The order of elements (BAS blocks) is arbitrary but all elements contained in the geometry specification must have an explicitly defined basis set (in particular, it can be specified as empty by setting the number of contracted orbitals to zero), even if this definition is redundant with respect to the basis set keyword in the general ORCA input. This requirement stems from the fact that the META section information is used not only by ORCA but also by a separate one-electron integral module, based on routines extracted from the GAMESS(US) code. Currently, only Gaussian basis functions with angular momenta up to $f$ can be used. The EMSL Basis Set Exchange [67] is a good place to download most common basis sets. When doing so, we recommend avoiding the use of Optimized Gaussian Contractions since we've found out that when using large basis sets it may lead to numerical instabilities. These instabilities always lead to nonsensical results that can be easily detected.

The def2-SVP/J auxiliary basis set requested in the ORCA input could be replaced by another basis set, by using in the META section additional blocks, with the keyword BAS replaced by AUX. The order of these blocks is also arbitrary but no empty AUX blocks are allowed (this is an ORCA restriction). Note also that the RI-J Coulomb fitting (requested by the RI keyword in our example) can only be used with pure density functionals. For hybrid functionals, ORCA has two other methods known as RIJCOSX and RI-JK, the latter requiring "/JK" (rather than "/J") auxiliary basis sets. All these DFT fitting basis sets must *not* be confused with auxiliary basis sets required by DF-SAPT(DFT) and defined in the file `name.aux`, which are designed to fit the correlation effects and must be provided by the user, as in the case of DALTON (such basis sets are actually used in ORCA at the MP2 level and denoted by "/C").

It is important to understand that while the main basis set which will be used by Orca has to be provided as the input, the density fitting basis set in Orca will be from Orca repositories. The consistency between the first line of input and basis read in is not checked. Note that the density fitting will be used by Orca by default for nonhybrid density functionals such as PBE. Note also that auxiliary bases used in Orca are in general different from those used in SAPT.

The ORCA route in DF-SAPT(DFT) requires a preparation of one input file not occurring in the DALTON route, called `kernel.data`. This file contains just two lines of the form

```
n_rad n_aug mem
ksi
```

where `n_rad` and `n_aug` are the numbers of radial and angular grid points around each atom in the quadrature of the CKS kernel integral. In general, much looser grids can be used here compared to the main DFT grids without any significant loss of accuracy. Typically, the values of `n_rad` = 50 and `n_ang` = 50 lead to completely negligible errors, while values of `n_rad` = 30 and `n_ang` = 26 introduce errors in the dispersion energy of just a few tenths of one percent. Parameter `mem` specifies the maximum amount of memory (in 8-byte words) to be used in the kernel generation. For larger systems, limiting memory use requires slower multipass calculations. The parameter `ksi` is the fraction of exact exchange of the employed functional (for instance, 0.25 for PBE0). As the last difference compared to the DALTON route, the script dfSAPT.orca rather than dfSAPT is used.

In DF-SAPT(DFT) calculations with large basis sets (close to 1000 orbitals or more), the calculation of the propagator becomes the most time-consuming part. In such cases, we recommend the use of the new, more efficient dispersion energy algorithm described in Ref. 32. It can be selected by including `FASTDISP=T` keyword in the `INPUTCOR` namelist (cf. Sec. 10.2.3). This route is only available with the ORCA interface and pure (*i.e.*, non-hybrid) DFT functionals. Additionally, the $E^{(2)}_{\mathrm{exch-disp}}$(CKS) correction in Eq. (14) is then approximated by scaling the uncoupled quantity, $E^{(2)}_{\mathrm{exch-disp}}$(KS), as described in Ref. 14.

The directory `examples/ORCA/RDX` contains input and output files from SAPT(DFT) calculations for cyclotrimethylene trinitramine (RDX) dimer containing 42 atoms, using the MC$^{+}$BS aug-cc-pVDZ basis set (648 basis functions per monomer) and the PBE0 or PBE functional with the GRAC asymptotic correction (the PBE0 calculation reproduces one of the potential energy points of Ref .[68], where the DALTON interface was used).

## 17 The `fastdf` program

SAPT2020 comes with a new version of the ORCA interface, with renewed algorithms, which we've named `fastdf`. All these algorithms use density-fitting in order to minimize the resources and time needed to perform the computations.

### 17.1 Notes on the installation

As most of the programs included in the SAPT2020 suite, the `fastdf` program is installed via the `Compall` script. In order to get `fastdf` installed, the user needs to set the variable `ORCADIR` to point to the path where ORCA was installed (not including the executable name), and set `FASTDF=YES`.

`libint` support: The `fastdf` program is now compatible with `libint`; this allows for computation of electronic repulsion integrals of arbitrary large $l$ values. In order to use `fastdf` with `libint` (we highly recommend this option), the user must compile a `libint` library with the following options: `enable-eri3=0`, `enable-eri2=0`, `with-max-am=N` (where `N` is equal to the maximum angular momentum value desired; with `N=6` the user will be able to use up to quintuple-z basis sets for dimers with elements of the second row). For inexperienced users, we recommend downloading `libint` from `https://github.com/evaleev/libint/releases/tag/v2.6.0` and following the installation instructions. The user should then define the variable `LIBINT` to point to the directory where it was installed. Since `libint` depends on `Eigen3`, the user should download this library (no need to compile) and define `EIGEN3` to point to where it was downloaded. If the user chooses not to use `libint`, `LIBINT` should be set to `NO`. In this case, the `Compall` script will link a separate program to compute the electronic repulsion integrals, with max $l = 4$; this results in the program going only up to triple-z basis sets.

*Requirements:* The `fastdf` program is compatible with the `ifort` and `gfortran` compilers. It makes use of modern Fortran features, therefore not being compatible with older versions of these compilers. `libint` support requires a working C++11 compiler; since both GNU and Intel provide C++ compilers bundled with the fortran ones, this is not an issue. One small note is that in the case of the Intel compilers, `icpc` (the C++ compiler) requires a recent version of `gfortran` installed in the system. For instance, GCC version 8.2.0 is sufficient.

## 17.2 Using the `fastdf` program

The user should prepare a single input file named `<jobname>.sapt`, where `<jobname>` stands for a name describing the SAPT job. The `fastdf` parser ignores trailing white spaces, so they can be used for prettier formatting. It also ignores capitalization, so for instance `SAPTDFT` is the same as `saptdft` or `sAPTdfT`, and blank lines. A minimal working example `<jobname>.sapt` file looks as follows:

```
method {
    SAPTDFT
    functional PBE
    grac 12.6206 12.6206
}

basis {
    main_basis_set aug-cc-pvtz
}

monomer {
    O   0.1088092367800     0.0000000000000    -0.0607755435800
    H  -0.1616146814400     0.0000000000000     1.7553074350400
    H  -1.5793331071000     0.0000000000000    -0.7828987377400
}

monomer {
    O  -0.0964834851100     0.0000000000000     6.5351482873200
    H   0.7718678809000     1.4536519623000     7.2451823417100
    H   0.7718678809000    -1.4536519623000     7.2451823417100
}
```

The `fastdf` program is linked to the file `bin/sapt-fastdf`. After having prepared the `<jobname>.sapt` file, the user should run the following command:

```
<path_to_sapt>/bin/sapt-fastdf <jobname>
```

If the user adds `<path_to_sapt>/bin` to their `PATH` variable, then

```
sapt-fastdf <jobname>
```

will be enough.

The input file is divided in sections, each of them opened and closed by curly brackets ({}).

### 17.2.1 The `method` section

It describes the method to be used, and provides required parameters for its computation. The first line of the method section should always be the desired method of calculation. Right now the current available methods are:

- SAPTDFT: The SAPT(DFT) method as described in eq. (14).
- SAPTKS: SAPT(KS) method (i.e., second order contributions computed by regular SAPT using Kohn-Sham orbitals). This is not recommended for calculations with density functionals, but can be used to compute SAPT0 interaction energies if the `functional` variable is set to `HF` (Hartree-Fock; see below for further explanation).
- DELTAHFR: This calls for the computation of $\delta^{\rm HF}_{\rm int,r}$
- CUSTOM: The user can ask for whichever SAPT(DFT) corrections they like.

For methods that involve corrections that need the KS orbitals and/or orbital energies, the functional for the SCF calculation must be specified. This can be done with the line `functional <functional>` (where `<functional>` must be replaced for the desired functional). If this line is skipped, by default the PBE functional will be chosen. Also, the KS part must be performed with the asymptotic correction. This correction can be specified by including the line `grac ipA ipB`, where `ipA` and `ipB` are the ionization potentials of monomers A and B in eV. If the user wants to skip the asymptotic correction, they can instead use the line `grac false`, although it is never recommended to skip it. If the CUSTOM method is chosen, an additional line should be included, with the desired corrections. This line should spell `corrections` plus a list of all the desired corrections separated by one or more spaces. The following corrections are available:

- `e1elst`: $E^{(1)}_{\mathrm{elst}}(\mathrm{KS})$
- `e1exch`: $E^{(1)}_{\mathrm{exch}}(\mathrm{KS})$
- `e2ind`: $E^{(2)}_{\mathrm{ind}}(\mathrm{KS})$
- `e2indr`: $E^{(2)}_{\mathrm{ind}}(\mathrm{CKS})$
- `e2exind`: $E^{(2)}_{\mathrm{exch-ind}}(\mathrm{KS})$
- `e2exindr`: $E^{(2)}_{\mathrm{exch-ind}}(\mathrm{CKS})$
- `e2disp`: $E^{(2)}_{\mathrm{disp}}(\mathrm{KS})$
- `e2dispr`: $E^{(2)}_{\mathrm{exch-disp}}(\mathrm{CKS})$
- `e2exdisp`: $E^{(2)}_{\mathrm{exch-disp}}(\mathrm{KS})$
- `e2exdispr`: $E^{(2)}_{\mathrm{exch-disp}}(\mathrm{CKS})$
- `e10elst`: $E^{(10)}_{\mathrm{elst}}$
- `e10exch`: $E^{(10)}_{\mathrm{exch}}$
- `e20ind`: $E^{(20)}_{\mathrm{ind}}$
- `e20indr`: $E^{(20)}_{\mathrm{ind,resp}}$
- `e20exind`: $E^{(20)}_{\mathrm{exch-ind}}$
- `e20exindr`: $E^{(20)}_{\mathrm{exch-ind,resp}}$
- `e20disp`: $E^{(20)}_{\mathrm{disp}}$
- `e20exdisp`: $E^{(20)}_{\mathrm{exch-disp}}$
- `hfint`: $E^{\mathrm{HF}}_{\mathrm{int}}$
- `deltahf`: $\delta E^{\mathrm{HF}}_{\mathrm{int}}$
- `deltahfr`: $\delta E^{\mathrm{HF}}_{\mathrm{int,resp}}$

For example, if the user only wanted to compute $E^{(20)}_{\mathrm{disp}}$ and $E^{(20)}_{\mathrm{ind}}$, the following method block would be valid:

```
method {
    CUSTOM
    memory 10gb
    grac false
    corrections e20disp e20ind
}
```

The keyword `memory` can be specified to control how much memory some of the subroutines can use. As of now, this only has an impact on a few subroutines, but it's important to control the SCF process, so the user should always set this to a reasonable value. It can be written in bytes, kilobytes, megabytes or gigabytes; for example, `memory 104857600` would ask for 100 MB of memory, and also would `memory 100mb`.

### 17.2.2 The `monomer` section

The monomer section is intended to input all the information about each monomer in the dimer. The input file should always have two monomer sections, one for each monomer. For all intended purposes, the first monomer provided is named "A", whereas the second one is named "B". Monomers can be input by lines in the format `<atom> <x> <y> <z>`, where `<atom>` is the atomic symbol, and `x`, `y`, and `z` are the coordinates of the atom. By default the coordinate unit is Bohr, but the user can choose to input coordinates in Ångstrom by adding the line `unit angs`. For instance, the following monomer section would input an ammonia molecule with its coordinates in Ångstrom:

```
monomer {
    N  -1.578718  -0.046611   0.000000
    H  -2.158621   0.136396  -0.809565
    H  -2.158621   0.136396   0.809565
    H  -0.849471   0.658193   0.000000
    unit angs
}
```

The charge of the monomer can be added with the line `charge Z`. By default each monomer's charge is set to 0. The program is also able to read `.xyz` files. To do so, instead of writing the molecule's atoms and coordinates, the line `xyz` will tell it to read the coordinates from file `A.xyz` or `B.xyz` (`.xyz` files should always have their coordinates in Ångstrom). If using the `.xyz` format, the user can still define the charge of the monomer. For example,

```
monomer {
    xyz
    charge -1
}
```

would read the monomer from file `A.xyz` (or `B.xyz`, if it's the second monomer), and set its charge to -1.

### 17.2.3 The `basis` section

The basis section is used to define the basis set(s) used by the program. The input depends on whether the user wants to use a DCBS, DC$^+$BS, MCBS or MC$^+$BS. Both "$^+$" variants will require the input of an additional section, called `midbond`, described later. Basis sets are read from the `fastdf/basis` folders, and are written in the `.gbs` format. The user can add new basis sets to this folder, if needed. When doing so, it is important that the file begins with `****` (all blank lines and lines starting with `!` are not considered) and leave a blank space at the end. Some usual keywords may not be recognized by SAPT.

- DCBS: For users who want to use a DCBS, the line `main_basis_set <basis>` should be enough.
- MCBS: The line `type mcbs` should be included, in addition to the `main_basis_set` line. For example:

  ```
  basis {
      main_basis_set aug_cc_pvtz
      type mcbs
  }
  ```

- DC$^+$BS: In order to use this kind of basis set, the user should choose a main basis set, and the midbond functions. To do this, an additional `midbond` section must be included in the input; it will be described further in this section.
- MC$^+$BS: The MC$^+$BS involves not only midbond functions but also functions located in the atoms of the other monomer (i.e., the basis set of monomer A includes all the basis elements located in the atoms of monomer A and only some of the basis elements located in those of monomer B). If the user just wants to include midbond functions, but no functions located in the atoms of the other monomer, then the MCBS example will work, if an additional `midbond` section is provided (see further explanation). If the user wants to also include basis functions located in the atoms of the other monomer, then the `off_basis_set` should also be

defined. This basis set can be defined by removing some elements from the main basis set. There's two ways of doing this; the first one is automatic, by using the `reduce` command. For example:

```
basis {
    main_basis_set aug_cc_pvtz
    off_basis_set reduce 2
}
```

sets the reduction level to 2; this implies that two layers of atomic orbitals will be removed from the off basis set. So, for instance, if aug-cc-pVTZ is chosen, for the oxygen atom, the main basis set will contain *s, p, d* and *f* orbitals, whereas the off basis set will contain only the *s* and *p* ones. If the reduction level is set too high, at least one layer of *s* orbitals will be kept; so for the oxygen atom in the aug-cc-pVTZ set, `reduce 10` will have the same effect as `reduce 3`.

The auxiliary basis set must also be chosen; this is done with the line `aux_basis_set <aux_basis_set>`. By default, the program tries to find a basis set named `<main_basis_set>-ri`; most basis sets included in SAPT2020 also include their correspoding auxiliary basis sets, but if the user wants to provide their own ones, they should either provide the `-ri` one or specify the auxiliary basis set to be used every time. Also, if midbond functions are used, an additional basis set for them is required. By default it is read from file `aux_midbond.gbs`, located in the folder `fastdf/basis_sets`.

The maximum $\zeta$-value of the basis set depends on the way `fastdf` is compiled. The CKS equations require the so-called kernel integrals $(ij|f_{xc}|K)$, where $f_{xc}$ is the exchange-correlation kernel. `fastdf` uses a separate program that allows computation of integrals involving gaussian orbitals with $\zeta$ up to 6, which allows for the use of basis sets of up to $5\zeta$ for elements in the second row. On the other hand, the $\zeta$ value for the regular electron-repulsion integrals depends on the way `fastdf` is installed. Please refer to 17.1 for more details.

### 17.2.4 The `midbond` section

In this section the user can specify the off-atomic sites and the basis functions located in them. These basis functions are read by default from the file `fastdf/basis/midbond.gbs`, but the user could choose to change it with the line `midbond_basis_set <mbset>` (although it is recommended to just add different sets of midbond functions to the default file). The format for placing the midbond functions is the same as in the `monomer` section, with the difference that here instead of the names of the atoms one should input names associated with sets of basis functions defined in the `midbond.gbs` file. An example midbond section is:

```
midbond {
    M1 0.02360 0.0 4.08770
}
```

Here `M1` stands for a set of *3s3p3d3f* functions. Instead of specifying the coordinates, the user can automatically place the midbond functions in the $1/r^6$-weighted average of the atomic positions. This can be done with the line `<name> r_6`, where `<name>` is the name assigned to the set of functions. For most applications involving small dimers, `M1 r_6` should be a good choice.

### 17.2.5 The `options` section

Finally, the `options` section is used to choose any additional options. Each option chosen has the format `option value`. For options involving logical values, they must be set to either `true` or `false`; for example, if the user wants to add the $\delta E^{\mathrm{HF}}_{\mathrm{int,resp}}$ correction, the line `add_deltahfr true` should be included in the `options` section. The available options and values are listed in tables 3 and 4. A typical `options` section looks as this:

```
    options {
        nprocs 4
        debug_level 2
        orca_ex_ri_method RIJK
        add_deltahfr true
    }
```

The default values for the options are stored in the `fastdf/defaults.wad` file. This file is generated by the compiler, and can be modified by the user.

## 17.3 Controlling the SCF part

The whole SCF part can be controlled through the options starting with `orca_`. When choosing a basis set, SAPT defines it for each atom in the dimer and includes it in the generated ORCA input files by using the command `NewGTO`, so there's no need to specify a basis set specifically for the SCF calculation. Asides from the use of density fitting in the SAPT calculations, ORCA can also use density fitting to speed up the SCF procedure. This can be controlled by using the option `orca_ex_ri_method`, which can be set to either `NORI` (not use density fititng), `RIJ` (only fit Coulomb integrals), `RIJK` (fit both Coulomb and exchange integrals, default) or `RIJCOSX` (same as `RIJK` but with a faster method for the exchange part); these methods are better described in the ORCA manual. When using nonhybrid functionals, no exchange integrals are computed, so `RIJ`, `RIJK` and `RIJCOSX` are equivalent. If density fitting is to be used for the SCF calculation, then

| Option | Default | Description |
|---|---|---|
| `debug_level` | 0 | This number influences how much information is written in the output. The maximum value is 2. |
| `nprocs` | 1 | Sets the amount of threads dedicated to the computations. This option has effect only if threaded BLAS libraries are used and/or the `orca_nprocs` variable is not defined. |
| `cks_quadrature_points` | 8 | Sets how many points will be included in the quadrature for the integration of the FDDSs. |
| `cks_ints_nrad` | 50 | Sets the amount of points to be included in the radial part of the grid used to compute the kernel integrals. |
| `cks_ints_nang` | 50 | Sets the amount of points to be included in the angular part of the grid used to compute the kernel integrals. |
| `mb_cutoff` | 0.8 | When placing the midbond functions by using the `r_6` directive, if they are closer than `mb_cutoff` times the atomic radius of any atom, then they are removed. |
| `chf_ind_tol` | `1.00E-06` | Tolerance for the induction energy computation from the coupled HF equations. |
| `cks_ind_tol` | `1.00E-06` | Tolerance for the induction energy computation from the coupled KS equations. Only relevant if hybrid functionals are used. |
| `exchange_fraction` | 0 | Fraction of HF exchange added to the exchange-correlation potential. For the PBE functional this number should be set to 0, and for PBE0 to 0.25. |
| `e2exdispr_scaling` | `None` | Choose whether to compute $E^{(2)}_{\mathrm{exch-disp}}$(CKS) or to compute $E^{(2)}_{\mathrm{exch-disp}}$(KS) and scale it. The options are **`none`**, **`disp`** and **`alpha`**. See further discussion. |
| `e2exdispr_alpha` | -1 | Scaling factor for $E^{(2)}_{\mathrm{exch-disp}}$(KS). Only relevant when `e2exdispr_scaling` is set to `alpha`. |
| `freeze_core` | `true` | Enables the frozen-core approximation for $E^{(2)}_{\mathrm{disp}}$ and $E^{(2)}_{\mathrm{exch-disp}}$. |
| `print_input_file` | `true` | Prints the input file at the beginning of the program. |

Table 3: Options available for the `options` section of the input file .

| Option | Default | Description |
|---|---|---|
| orca_nprocs | 1 | Sets the number of threads to be used in the ORCA program. If this option is not set, then the nprocs variable will be used instead. |
| orca_grid | 4 | Sets the ORCA variable Grid for numerical integration. |
| orca_finalgrid | 5 | Sets the ORCA variable FinalGrid for numerical integration. |
| orca_convergence | TightSCF | Sets the convergence criterion for the SCF iterations. |
| orca_aux_basis_set | AutoAux | Sets the auxiliary basis set used in ORCA. It can (and should) be different from the one set for the SAPT part of the calculation. By default, SAPT invokes the tt AutoAux procedure, which generates a RI basis set for each calculation. |
| orca_ex_ri_method | RIJK | Sets the density-fitting method for the SCF part. |

Table 4: Options available to control the SCF process through the ORCA program. Please refer to the Orca manual for a definition of the variables involved.

an auxiliary basis set must be chosen for this. This auxiliary basis set is independent of the one chosen for the SAPT calculation with the option aux_basis_set in the basis block, and instead must be chosen in the options block, by setting the orca_aux_basis_set variable. By default, the latter is set to AutoAux, which tells ORCA to create a suitable fitting basis set by default, but instead, the user can choose whichever auxiliary basis set they find suitable, as long as it is available in ORCA (please refer to the ORCA manual to see which sets are available).

## 17.4 Scaling for $E^{(2)}_{\mathrm{exch-disp}}$

The most accurate way to compute $E^{(2)}_{\mathrm{exch-disp}}$ is to do so by solving the CKS equations for the FDDS amplitudes and then contracting them with the one- and two- electron integrals (see Eq. (10) of Ref. [41] for the formula). This can be a very time-consuming process since the solution of the CKS equations and the contraction of the FDDS amplitudes are both $O(N^5)$ processes (if density-fitting is used). Therefore, it is frequent to see that instead the uncoupled amplitudes are used, and the resulting $E^{(2)}_{\mathrm{exch-disp}}(\mathrm{KS})$ is scaled to approximate $E^{(2)}_{\mathrm{exch-disp}}(\mathrm{CKS})$. The fastdf program has two ways of scaling $E^{(2)}_{\mathrm{exch-disp}}(\mathrm{KS})$. One of them is to compute the quotient between KS and CKS dispersion energies:

$$E^{(2)}_{\mathrm{exch-disp}}(\mathrm{CKS}) \approx E^{(2)}_{\mathrm{exch-disp}}(\mathrm{KS}) \times \frac{E^{(2)}_{\mathrm{disp}}(\mathrm{CKS})}{E^{(2)}_{\mathrm{disp}}(\mathrm{KS})}; \tag{15}$$

this can be achieved by setting `e2exdispr_scaling` to `disp`. The other option is to simply use a factor $\alpha$ fitted to benchmark data:

$$E^{(2)}_{\mathrm{exch-disp}}(\mathrm{CKS}) \approx \alpha\, E^{(2)}_{\mathrm{exch-disp}}(\mathrm{KS}), \tag{16}$$

which can be done by setting `e2exdispr_scaling` to `alpha`, and `e2exdispr_alpha` to the desired $\alpha$ value. Heßelmann and Korona have recommended the value $\alpha = 0.686$ [69].

## 17.5 Frozen core approximation

The frozen core approximation in the Infinte Excitation Energy variant [41], has been implemented for $E^{(2)}_{\mathrm{disp}}$ and $E^{(2)}_{\mathrm{exch-disp}}$. This approximation excludes some occupied orbitals from the computation of the dispersion and exchange-dispersion energies, leading not only to time savings, but also to disk space savings by not storing amplitudes related to the frozen orbitals. The resulting approximate dispersion and exchange-dispersion energies are usually very close to the exact ones; therefore this approximation is turned on by default. It can be switched off by setting `frozen_core` to `false` in the `options` block.

## 17.6 Hybrid vs nonhybrid functionals

Functionals that include a fraction of HF exchange in the exchange-correlation potential are called hybrid functionals. Since the HF exchange is non-local, the resulting functional is also non-local, and computational cost increases, with the SCF scaling going from $O(N^3)$ to $O(N^4)$. This also has a negative impact in the solution of the CKS equations, since the inclusion of the HF exchange adds expensive contractions between virtual indices that greatly increase the computational cost and storage requirements.

SAPT2020.1 includes efficient algorithms for the computation of FDDSs, and for $E^{(2)}_{\mathrm{disp}}(\mathrm{CKS})$ and $E^{(2)}_{\mathrm{exch-disp}}(\mathrm{CKS})$ when using non-hybrid functionals, with $O(N^2)$ memory and $O(N^3)$ storage requirements, but not for hybrid ones. For the latter, we've adapted our old algorithms, which are good enough for small systems but not appliable to large ones since the requirements for memory and storage scale as $O(N^3)$ and $O(N^4)$, respectively. Whenever possible, it is advised to use hybrid functionals, especially for the first order corrections (which don't involve FDDSs and therefore are equally costly for hybrid and non-hybrid functionals), since the HF exchange forces better asymptotic behavior of the KS orbitals, but for second-order contributions the user may be forced to use non-hybrid functionals. Fortunately, these have been shown to work well for the description of dispersion and exchange-dispersion contributions [70]. We would like to remind the user that it is not enough to set the functional to PBE0, or whichever functional they want to use, but also the variable `exchange_fraction` must be set accordingly.

Currently, the fastdf program doesn't have a way of using more than one functional in the same calculation, so for instance if one wishes to use PBE0 for first-order contributions and PBE for second-order ones, they should prepare two separate input files, one for the first-order part and one for the second-order part.

### 17.7 Examples

There are some examples of fastdf input files in the `examples/fastdf` folder.

- `C2a`: contains the input files necessary to compute HFDc$^{(1)}$ interaction energies with PBE0 for first-order correlation and PBE for dispersion and exchange-dispersion energies (these are expensive calculations and are expected to take up a large amount of disk space and computational time) for the complex C2a in the S12L [71].
- `rdx`: contains a single input file necessary to run a SAPT(DFT) calculation for the RDX dimer with both the PBE and PBE0 functionals. We recommend the user to compare the efficiency of both approaches (please keep in mind that the calculations will use about 25 and 125 GB for the PBE and PBE0 examples, respectively.
- `nh3-h2o`: contains a single input file necessary to run a SAPT(DFT)+$\delta E^{\mathrm{HF}}_{\mathrm{int,resp}}$ calculation for the $NH_3$-$H_2O$ dimer.
- `ar-h2o`: contains an input file to compute the SAPT(DFT) interaction energy of the Ar-$H_2O$ dimer.

## 18 COM-COM Asymptotic Multipole Expansion Based on DFT

### 18.1 Introduction

Included in SAPT2020 is a set of programs used to compute interaction energies in the asymptotic regime using a COM-COM multipole expansion with coefficients calculated using DFT. This is separate from the set of programs for calculating interaction energies in the asymptotic regime using wavefunction methods.

The interaction energy $E_{\mathrm{int}}(\boldsymbol{R})$ in the large-$R$ region can be approximated by the COM-COM asymptotic multipole expansion [1, 72]

$$\begin{aligned}E_{\mathrm{int}}(\boldsymbol{R}) \approx \sum_n \frac{C_n(\boldsymbol{\Omega})}{R^n} &= \sum_n \frac{C_{n,\mathrm{elst}}(\boldsymbol{\Omega})}{R^n} + \sum_n \frac{C_{n,\mathrm{ind}}(\boldsymbol{\Omega})}{R^n} + \sum_n \frac{C_{n,\mathrm{disp}}(\boldsymbol{\Omega})}{R^n} \\ &= E^{\mathrm{asym}}_{\mathrm{elst}} + E^{\mathrm{asym}}_{\mathrm{ind}} + E^{\mathrm{asym}}_{\mathrm{disp}} \qquad (17)\end{aligned}$$

where the coefficients $C_n$ analytically depend on the angles $\mathbf{\Omega}$ and contain contributions from electrostatic, induction, and dispersion interactions. Each of these components has its own well-defined asymptotic expansion which is seamlessly connected to the corresponding SAPT component. For a detailed form of Eq. (17), see Ref. [35].

## 18.2 Usage

To compute the interaction energy components using the COM-COM asymptotic multipole expansion, first create input files in the same way as for a normal, non-dinsity fitting DFT calculation using the DALTON interface. The inputs must be in the format used for the `Runlot` script, i.e., including the `dimer.cnf` and `geoparm.d` files. One additional file, called `NAME.dispinp`, is required for the asymptotic calculations, where 'NAME' is the system name. It has the following format:

```
&disper
 title = 'Asymptotic calculations',
 groupa = 'CINFV',
 groupb = 'CINFV',
 nmax = 12,
 ngrid = 10,
 nfpola = 1,
 nfpolb = 2,
 itypea = 0,
 itypeb = 0,
 prpol = t,
 prcop = t,
 tabul = t
 &end
```

The symmetry groups 'CINFV' are included for compatibility with older versions of the code and are ignored. The only variable which may need to be changed is 'nmax', which specifies the maximum power of $1/R$ used.

The program is then run with the command

```
./bin/asymp_com/Master_cmplx_asym.sh NAME SAME > Master.log 2>&1
```

where 'NAME' is the name of the system and 'SAME' should be set to 'yes' or 'no', specifying whether or not the monomers are identical.

The file `log` contains the final outputs, including interaction energies. The files `NAME.dalcalmultA` and `NAME.dalcalmultB` contain the full set of expansion coefficients.

Example runs with inputs and outputs are provided in the directory `./examples/asymp_com`.

## 18.3 Program structure

The script `Master_cmplx_asym.sh` performs the following steps

- Call the script `OrunA`, which performs the following steps
  - Call the main SAPT script, which computes the monomers' frequency-dependent density susceptibilities (FDDS)
  - Call the `dalton2rb` program, which extracts basis set information from the DALTON output files
  - Call the `OpreparecalmulA` script, which generates input files for the `dal_calmult` program
  - Call the `dal_calmult` program, which generates the following files:
    - `fortA.1`, which contains tesseral multipoles
    - `fortA.2`, which contains static polarizabilities
    - `fortA.3`, which contains dynamic polarizabilities
    - `NAME.dalcalmultA`, which contains all the information of the above three files in a human-readable format
- In the case that the monomers are not identical, the script `OrunAB` is called instead of `OrunA`, which performs the above steps for both monomers A and B, with the filenames modified appropriately in the case of monomer B.
- Call the program `disper_cmplx`, which computes dispersion coefficients and produces the file `ooo`
- Call the script `col_7_8_cmplx.awk`, which takes as input `ooo`, extracts the values of the dispersion coefficients, and produces the file `coefd.dat`
- Call the program `induct_complex`, which computes induction coefficients and produces the file `ooo`
- Call the script `col_7_8_cmplx.awk`, which takes as input `ooo`, extracts the values of the induction coefficients, and produces the file `coefia-b.dat`
- Perform the above two steps with input files switched to generate the file `coefib-a.dat`
- Call the program `sym_cmplx`, which combines the files `coefia-b.dat` and `coefib-a.dat` into the file `coefi.dat`, taking care of the appropriate phase adjustments
- Call the program `potcal`, which computes the interaction energies and produces the final output file `log`

## References

[1] B. Jeziorski, R. Moszyński, and K. Szalewicz, Chem. Rev. **94**, 1887 (1994).

[2] K. Szalewicz and B. Jeziorski, in *Molecular Interactions — from van der Waals to strongly bound complexes*, edited by S. Scheiner (Wiley, New York, 1997), p. 3.

[3] B. Jeziorski and K. Szalewicz, in *Encyclopedia of Computational Chemistry*, edited by P. von Ragué Schleyer, N. L. Allinger, T. Clark, J. Gasteiger, P. A. Kollman, H. F. Schaefer III, and P. R. Schreiner (Wiley, Chichester, UK, 1998), vol. 2, pp. 1376–1398.

[4] B. Jeziorski and K. Szalewicz, in *Handbook of Molecular Physics and Quantum Chemistry*, edited by S. Wilson (Wiley, 2003), vol. 3, part 2, chap. 9, pp. 232–279.

[5] K. Szalewicz, K. Patkowski, and B. Jeziorski, in *Intermolecular Forces and Clusters*, edited by D. J. Wales (Springer, 2005), vol. 116 of *Structure and Bonding*, pp. 43–117.

[6] K. Szalewicz, R. Bukowski, and B. Jeziorski, in *Theory and Applications of Computational Chemistry: The First 40 Years. A Volume of Technical and Historical Perspectives*, edited by C. E. Dykstra, G. Frenking, K. S. Kim, and G. E. Scuseria (Elsevier, Amsterdam, 2005), chap. 33, p. 919.

[7] K. Szalewicz, Wiley Interdisc. Rev.–Comp. Mol. Sci. **2**, 254 (2012).

[8] B. Jeziorski, R. Moszyński, A. Ratkiewicz, S. Rybak, K. Szalewicz, and H. L. Williams, in *Methods and Techniques in Computational Chemistry: METECC-94*, edited by E. Clementi (STEF, Cagliari, 1993), vol. B, p. 79.

[9] R. Bukowski, W. Cencek, K. Patkowski, P. Jankowski, M. Jeziorska, M. Kolaski, and K. Szalewicz, Mol. Phys. **104**, 2241 (2006).

[10] H. L. Williams and C. F. Chabalowski, J. Phys. Chem. A **105**, 646 (2001).

[11] A. J. Misquitta and K. Szalewicz, Chem. Phys. Lett. **357**, 301 (2002).

[12] A. J. Misquitta, B. Jeziorski, and K. Szalewicz, Phys. Rev. Lett. **91**, 033201 (2003).

[13] A. J. Misquitta and K. Szalewicz, J. Chem. Phys. **122**, 214109 (2005).

[14] A. J. Misquitta, R. Podeszwa, B. Jeziorski, and K. Szalewicz, J. Chem. Phys. **123**, 214103 (2005).

[15] R. Bukowski, R. Podeszwa, and K. Szalewicz, Chem. Phys. Lett. **414**, 111 (2005).

[16] R. Podeszwa, R. Bukowski, and K. Szalewicz, J. Phys. Chem. A **110**, 10345 (2006).

[17] V. Lotrich and K. Szalewicz, J. Chem. Phys. **106**, 9668 (1997).

[18] R. Moszyński, P. Wormer, B. Jeziorski, and A. van der Avoird, J. Chem. Phys. **103**, 8058 (1995), erratum: **107**, 672 (1997).

[19] V. F. Lotrich and K. Szalewicz, J. Chem. Phys. **112**, 112 (2000).

[20] R. Podeszwa and K. Szalewicz, J. Chem. Phys. **126**, 194101 (2007).

[21] P. S. Żuchowski, R. Podeszwa, R. Moszyński, B. Jeziorski, and K. Szalewicz, J. Chem. Phys. **129**, 084101 (2008).

[22] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. Su, et al., J. Comput. Chem. **14**, 1347 (1993).

[23] V. Saunders and M. Guest, ATMOL Program Package, SERC Daresbury Laboratory, Daresbury, Great Britain.

[24] T. Korona, R. Moszyński, and B. Jeziorski, Mol. Phys. **100**, 1723 (2002).

[25] T. Korona and B. Jeziorski, J. Chem. Phys. **125**, 184109 (2006).

[26] T. Korona, M. Przybytek, and B. Jeziorski, Mol. Phys. **104**, 2303 (2006).

[27] T. Korona, Phys. Chem. Chem. Phys. **9**, 6004 (2007).

[28] T. Korona and B. Jeziorski, J. Chem. Phys. **128**, 144107 (2008).

[29] T. Korona, J. Chem. Phys. **128**, 224104 (2008).

[30] T. Korona, Phys. Chem. Chem. Phys. **10**, 6509 (2008).

[31] T. Korona, J. Chem. Theory Comput. **5**, 2663 (2009).

[32] R. Podeszwa, W. Cencek, and K. Szalewicz, J. Chem. Theory Comput. **8**, 1963 (2012).

[33] P. E. S. Wormer and H. Hettema, POLCOR package, University of Nijmegen, 1992.

[34] P. E. S. Wormer and H. Hettema, J. Chem. Phys. **97**, 5592 (1992).

[35] M. P. Metz, K. Piszczatowski, and K. Szalewicz, J. Chem. Theory Comput. (2016).

[36] K. Patkowski, K. Szalewicz, and B. Jeziorski, Theor. Chem. Acc. **127**, 211 (2010).

[37] M. Grüning, O. V. Gritsenko, S. J. A. van Gisbergen, and E. J. Baerends, J. Chem. Phys. **114**, 652 (2001).

[38] DALTON*, a molecular electronic structure program, Release 2.0 (2005), see* `http://www.kjemi.uio.no/software/dalton/dalton.html`.

[39] K. Patkowski, K. Szalewicz, and B. Jeziorski, J. Chem. Phys. **125**, 154107 (2006).

[40] H. L. Williams, K. Szalewicz, R. Moszyński, and B. Jeziorski, J. Chem. Phys. **103**, 4586 (1995).

[41] K. Patkowski and K. Szalewicz, J. Chem. Phys. **127**, 164103 (2007).

[42] MOLPRO*: a package of ab initio programs* designed by H.-J. Werner and P. J. Knowles, version 2002.6, R. D. Amos, A. Bernhardsson, A. Berning, P. Celani, D. L. Cooper, M. J. O. Deegan, A. J. Dobbyn, F. Eckert, C. Hampel, G. Hetzer, P. J. Knowles, T. Korona, R. Lindh, A. W. Lloyd, S. J. McNicholas, F. R. Manby, W. Meyer, M. E. Mura, A. Nicklass, P. Palmieri, R. Pitzer, G. Rauhut, M. Schütz, U. Schumann, H. Stoll, A. J. Stone, R. Tarroni, T. Thorsteinsson, and H.-J. Werner.

[43] M. J. Frisch *et al.*, *Gaussian 09, Revision A.1*, Gaussian, Inc., Wallingford, CT, 2009.

[44] P. Pulay, J. Comp. Chem. **3**, 556 (1982).

[45] J. O. Hirschfelder, Chem. Phys. Lett. **1**, 325 (1967).

[46] R. J. Wheatley, unpublished.

[47] R. Moszyński, B. Jeziorski, S. Rybak, K. Szalewicz, and H. L. Williams, J. Chem. Phys. **100**, 5080 (1994).

[48] M. Jeziorska, B. Jeziorski, and J. Cizek, Int. J. Quantum Chem. **32**, 149 (1987).

[49] S. Rybak, B. Jeziorski, and K. Szalewicz, J. Chem. Phys. **95**, 6579 (1991).

[50] R. Moszyński, B. Jeziorski, A. Ratkiewicz, and S. Rybak, J. Chem. Phys. **99**, 8856 (1993).

[51] E. M. Mas, K. Szalewicz, R. Bukowski, and B. Jeziorski, J. Chem. Phys. **107**, 4207 (1997).

[52] R. Bukowski, J. Sadlej, B. Jeziorski, P. Jankowski, K. Szalewicz, S. A. Kucharski, H. L. Williams, and B. S. Rice, J. Chem. Phys. **110**, 3785 (1999).

[53] CADPAC*: The Cambridge Analytic Derivatives Package Issue 6,* Cambridge, 1995. A suite of quantum chemistry programs developed by R. D. Amos with contributions from I. L. Alberts *et al.*

[54] F. Neese, ORCA, An Ab Initio, DFT and Semiempirical electronic structure package, with contributions from U. Becker, D. Ganyushin, A. Hansen, D. Liakos, C. Kollmar, S. Kossmann, T. Petrenko, C. Reimann, C. Riplinger, K. Sivalingam, B. Wezisla, and F. Wennmohs.

[55] H. L. Williams, E. M. Mas, K. Szalewicz, and B. Jeziorski, J. Chem. Phys. **103**, 7374 (1995).

[56] A. J. Stone, *The Theory of Intermolecular Forces* (Clarendon Press, Oxford, 1996), p. 8.

[57] O. Akin-Ojo, R. Bukowski, and K. Szalewicz, J. Chem. Phys. **119**, 8379 (2003).

[58] R. Podeszwa, R. Bukowski, and K. Szalewicz, J. Chem. Theory Comput. **2**, 400 (2006).

[59] V. F. Lotrich, H. L. Williams, K. Szalewicz, B. Jeziorski, R. Moszynski, P. E. S. Wormer, and A. van der Avoird, J. Chem. Phys. **103**, 6076 (1995).

[60] K. A. Peterson, D. Figgen, E. Goll, H. Stoll, and M. Dolg, J. Chem. Phys. **119**, 11113 (2003).

[61] B. I. Dunlap, Phys. Chem. Chem. Phys. **2**, 2113 (2000).

[62] F. Weigend, A. Köhn, and C. Hättig, J. Chem. Phys. **116**, 3175 (2002).

[63] A. Hesselmann, G. Jansen, and M. Schütz, J. Chem. Phys. **122**, 014103 (2005).

[64] D. J. Tozer and N. C. Handy, J. Chem. Phys. **109**, 10180 (1998).

[65] D. J. Tozer, R. D. Amos, N. C. Handy, B. O. Roos, and L. Serrano-Andrés, Mol. Phys. **97**, 859 (1999).

[66] F. Weigend, Phys. Chem. Chem. Phys. **4**, 4285 (2002).

[67] B. P. Pritchard, D. Altarawy, B. Didier, T. D. Gibson, and T. L. Windus, Journal of Chemical Information and Modeling **59**, 4814 (2019), URL `https://doi.org/10.1021%2Facs.jcim.9b00725`.

[68] R. Podeszwa, R. Bukowski, B. M. Rice, and K. Szalewicz, Phys. Chem. Chem. Phys. **9**, 5561 (2007).

[69] A. Heßelmann and T. Korona, J. Chem. Phys. **141**, 094107 (2014).

[70] J. Garcia and K. Szalewicz, The Journal of Physical Chemistry A (2020), URL `https://doi.org/10.1021%2Facs.jpca.9b11900`.

[71] S. Grimme, Chemistry - A European Journal **18**, 9955 (2012), URL `https://doi.org/10.1002%2Fchem.201200497`.

[72] A. J. Stone, *The Theory of Intermolecular Forces* (Clarendon Press, Oxford, 2013), 2nd ed.

[73] A. J. Misquitta, Ph.D. thesis, University of Delaware (2004).

## A Porting SAPT2020 to different platforms

If one wants to port the SAPT codes to an architecture that is not supported in the official release, there are three main fragments of the codes that are strongly architecture-dependent and should be taken care of. Note that the issues described below have to be resolved for each program (e.g., `tran`, `cc`, and `sapt.x`) separately, as the programs do not currently use any common library with the system-dependent routines.

1. The memory allocation routines. Each of the programs in the SAPT suite uses a single REAL∗8 array which is allocated when this program starts. All matrices, both real and integer, used by the program are then defined within the allocated core array so that no further calls to any architecture-dependent allocation routines are needed (the programs assume that the integer variables are 4-byte by default). Note that whereas the CC program automatically determines the core size needed and allocates that much memory, all other programs need to have the requested core size declared in an appropriate namelist unless the default value of 40 000 000 words is sufficient (see Sec. 10.2 for details). Currently, the SAPT codes use the following mechanisms to allocate the core arrays:

   - The Fortran 90 `ALLOCATE` routine for SUN, HPUX, and Linux with the PGF90 or Intel compilers,
   - The Fortran `malloc` intrinsic for SGI and Linux with the PGF77 compiler,
   - A C function `memget` supplied by SAPT for the G77 compiler under Linux and for IBM AIX.

   For a new platform, a proper way of handling memory allocation must be chosen. The Fortran 90 `ALLOCATE` is recommended where available.

2. The timing routines. The routines `timit` in `tran/main.F`, `second` in `cc/whole.F`, `timing` in `e2d/ccbits.F`, and `timt` in `sapt/m.F` return time (in seconds) elapsed since some fixed moment and are called throughout the program. These routines call system timing routines which are architecture-dependent:

   - the `etime` routine for SUN and HPUX,
   - the `mclock` routine for SGI, IBM, and Linux with Portland or Intel compilers (note that this one returns time in hundredths of seconds),
   - the `second` routine for Linux and the G77 compiler.

   Again, a proper system timing routine needs to be chosen for a new architecture.

3. The packing/unpacking routines which manipulate the integral indices. As the current basis set size limit for SAPT is 1023 functions, 40 bits are needed to store the four orbital indices for a transformed two-electron integral. These indices are stored on disk packed into one 4-byte integer and one 1-byte integer, and the routines that take care of the integral packing and unpacking are located in `tran/unpack.F`, `cc/unpack.F`, `e2d/ccbits.F`, and `sapt/unpack10.F`. These routines have the following functions:

   - `unpack10`: (INTEGER∗4,INTEGER∗1)→4 orbital indices
   - `pack10`: 4 orbital indices→(INTEGER∗4,INTEGER∗1)
   - `unpack10a`: (INTEGER∗4,INTEGER∗1)→2 pair indices
   - `pack10a`: 2 pair indices→(INTEGER∗4,INTEGER∗1)
   - `spltindx`: INTEGER∗8→(INTEGER∗4,INTEGER∗1)
   - `joinindx`: (INTEGER∗4,INTEGER∗1)→INTEGER∗8

   The packing/unpacking routines are implemented using the intrinsic functions for bitwise operations: `ishft`, `ibits`, and `iand`. The efficiency of packing and unpacking influences the total calculation time quite notably, and the implementation through bitwise operations has been found to be optimal for several architectures. Note, however, that subtle differences in the syntax of these intrinsics exist for different architectures and some of the packing/unpacking routines are platform-dependent. Apart from the set of routines described above, the `tran` program needs another set of unpacking procedures to access two-electron integral indices written by various integral and SCF front-ends, including GAUSSIAN, GAMESS, and DALTON (see the `tran/unpack.F` file for details). These routines are highly architecture-dependent since the structure of the integral indices depends on the endianness (big-endian or little-endian) of the machine (for GAUSSIAN) and on the options selected when compiling GAUSSIAN or GAMESS [see Sec. 8.1 (GAUSSIAN) and Sec. 9.4 (GAMESS) for more on this subject]. When porting to a new architecture, one must make sure that both the internal SAPT packing/unpacking routines listed above and the routines used to unpack integral indices written by a particular front-end program work correctly.

## B Integral/SCF interfacing

The SAPT group of codes can be interfaced with virtually any integral/SCF program. A short description of the elements of the interfacing process is presented here. The existing interfaces can be used as a template for the creation of new interfaces. A short listing of what the `tran` program needs follows.

- One-electron integrals.
    1. Overlap
    2. Hamiltonian
    3. Kinetic
    4. Potential
- Two-electron integrals.
- SCF eigenvalues.
- SCF eigenvectors.
- Information about the run.
    1. Basis set size.
    2. Number of occupied orbitals.
    3. Number of virtual orbitals.
    4. Charge on each atom.
    5. Geometry of the monomers.

Most of the information is read by a small interface program from the integral/SCF program files and then rewritten into a simple form that the next stages of the process can easily understand. Modifications to the transformation program must be made to read in the two-electron integrals (in **`trans.f`**, **`atmtr.f`**, and **`trnn.f`** modules). Notice also the common block `SCFPACK` which contains logical variables indicating which integral/SCF program is used. This common block must be changed throughout the program to include a logical variable for any new integral/SCF program. Finally, in the driver subroutine **`trans.F`**, make sure that there is a correct setting of the logical input record length for the new integral/SCF program.

## C List of subroutines

This appendix contains the list of SAPT2020 subroutines with a short description of their functions.

Table 5: List of subroutines - `tran`.

| Module | Subroutine | Comments |
|---|---|---|
| trans.F | metatrans | Allocate memory and call main driver. |
| trans.F | trans | Main driver routine. |
| trans.F | mkoffset | Prepare auxiliary arrays for MC$^+$BS. |
| trans.F | lvalue | Number of orbitals for a given shell. |
| trans.F | cpvc | Copy eigenvectors for a MC$^+$BS transformation. |
| trans.F | whichint | Flag integral types needed. |
| trans.F | pread | Read GAMESS integral file. |
| trans.F | pread1–pread2 | Variants of pread. |
| atmtr.F | atmtr | Driver for the four-virtual transformation. |
| atmtr.F | index4 | Perform four-virtual transformation. |
| atmtr.F | sort1 | First sort for the four-virtual transformation. |
| atmtr.F | wrda | Write intermediates to a direct access file. |
| atmtr.F | calc1 | Transform first two indices for four-virtual integrals. |
| atmtr.F | sort2 | Second sort for the four-virtual transformation. |
| atmtr.F | calc2 | Transform last two indices for four-virtual integrals. |
| atmtr.F | peswrec | Write final transformed integrals to file. |
| atmtr.F | writem | Print matrix for debugging. |
| atmtr.F | chksum1 | Check control sum for debugging. |
| atmtr.F | get_cadp | Manipulate CADPAC integral buffer. |
| atmtr.F | find_cadp | Read CADPAC integral buffer. |
| io.F | daopen | Open direct access files. |
| io.F | dawrit | Write to direct access files. |
| io.F | daread | Read from direct access files. |
| io.F | daclos | Close direct access files. |
| io.F | r8zero | Zero a REAL∗8 array indexed by an INTEGER∗4 variable. |
| io.F | r8zero8 | Zero a REAL∗8 array indexed by an INTEGER∗8 variable. |
| io.F | i4zero | Zero an INTEGER∗4 array. |
| io.F | putrec | Put a record on disk. |
| io.F | ropen | Open input file. |
| io.F | seopen | Open other sequential files. |
| io.F | closeall | Close all files. |
| io.F | seqopn | Open GAMESS integral file. |
| io.F | nr2asc | Integer→string conversion for constructing filenames. |
| io.F | inittwoeldalt | Initialize DALTON integral file. |
| io.F | readtwoel | Read integrals from DALTON. |
| io.F | findlab | Find DALTON labels. |
| io.F | readmolpro | Read integrals from MOLPRO. |
| main.F | timit | Read elapsed time. |
| main.F | prsq | Print square matrix. |
| main.F | rdvc | Read eigenvectors. |
| main.F | ifa | Initialize table lookup values. |
| main.F | flag1 | Choose either eigenvectors |
| main.F | flag2 | of monomer A or B. |
| main.F | alarmx | Abnormal ending. |

Table 5: List of subroutines - `tran` (part 2).

| Module | Subroutine | Comments |
|---|---|---|
| memory.F | memory | Partition core memory. |
| mono.F | chkitype | Analyze indices for a MC$^+$BS transformation. |
| mono.F | permut | Perform index permutation. |
| mono.F | sameinds | Compare index quadruplets. |
| tonel.F | onel | One electron transformation driver. |
| tonel.F | tr1e | Actual transformation of one-electron integrals. |
| tran.F | tran | Driver for in-core two-electron transformation. |
| tran.F | igetlda | Get dimension of the eigenvector matrix. |
| tranw.F | tranw | Driver for out-of-core two-electron transformation. |
| trnn.F | intowp | Calculate space for an integer array. |
| trnn.F | inread | Read integrals from GAUSSIAN. |
| trnn.F | labscf90 | Unpack record label for GAUSSIAN90 and later. |
| trnn.F | unpacka | Unpack ACES indices. |
| trnn.F | seeka | Search for ACES labels. |
| trnn.F | namemain | Get names of ATMOL1024 mainfile parts. |
| trnn.F | atmini | Initialize ATMOL two-electron integral file. |
| trnn.F | atmininew | Initialize ATMOL1024 two-electron integral file. |
| trnn.F | tr1 | First step of in-core and out-of-core transformation. |
| trnn.F | canon | Return indices in canonical order. |
| trnn.F | tr2 | Second step of in-core transformation. |
| trnn.F | loop2lims | Set loop limits for tr2 and tr2w. |
| trnn.F | tr3 | Third step of in-core transformation. |
| trnn.F | loop3lims | Set loop limits for tr3 and tr4w. |
| trnnw.F | tr2w | Second step of out-of-core transformation. |
| trnnw.F | tr3w | Third step of out-of-core transformation. |
| trnnw.F | tr4w | Fourth step of out-of-core transformation. |
| trnnw.F | looptr3w | Set loop limits for tr3w. |
| unpack.F | packg | Pack four 8-bit numbers into a 32-bit integer. |
| unpack.F | unpackg | Do the reverse of packg. |
| unpack.F | packg64 | Pack four 16-bit numbers into two 32-bit integers. |
| unpack.F | unpackg64 | Do the reverse of packg64. |
| unpack.F | unpack64to32 | Unpack a 64-bit integer into two 32-bit ones. |
| unpack.F | unpack10 | Unpack an (INTEGER$*$4,INTEGER$*$1) pair into four indices. |
| unpack.F | unpack10a | Unpack an (INTEGER$*$4,INTEGER$*$1) pair into two indices. |
| unpack.F | pack10 | Do the reverse of unpack10. |
| unpack.F | pack10a | Do the reverse of unpack10a. |
| unpack.F | spltindx | Split an index into an (INTEGER$*$4,INTEGER$*$1) pair. |
| unpack.F | itobyte | Integer$\rightarrow$byte conversion. |
| unpack.F | unpckgms | Unpack GAMESS integral labels. |
| unpack.F | unpckdlt | Unpack DALTON integral labels. |

Table 6: List of subroutines - `cc`[a].

| Module | Subroutine | Comments |
|---|---|---|
| mcc.F | mccsd | Main driver. |
| mcc.F | mono | Perform CC for a given monomer. |
| ccio.F | getamp, getampn | Get amplitudes from disk (obsolete, replaced by newget). |
| ccio.F | putamp | Write amplitudes to disk (obsolete, replaced by newput). |
| ccio.F | ampinf, mapda | Auxiliary routines for getamp and putamp (obsolete). |
| ccio.F | getbuf | Read one packet (of the size $\mathrm{ov}^2$) of ovvv integrals. |
| ccio.F | readen | Read orbital energies. |
| ccio.F | lrecl | Adjust buffer sizes for sorting 2el integrals. |
| ccio.F | save | Omit small values in an array. |
| ccio.F | desym1 | <oo\|vv> array: symmetry unique elements → full |
| ccio.F | desym2 | <oo\|oo> array: symmetry unique elements → full |
| ccio.F | desym3 | <oo\|ov> array: symmetry unique elements → full |
| ccio.F | getampa | Add disk amplitudes to array (obsolete, replaced by newgeta). |
| ccm1.F | izero | Zero an INTEGER∗4 array. |
| ccm1.F | r8zero | Zero a REAL∗8 array indexed by an INTEGER∗4 variable. |
| ccm1.F | r8zero8 | Zero a REAL∗8 array indexed by an INTEGER∗8 variable. |
| ccm1.F | xdata | Initialize some arrays. |
| ccm1.F | i8zero | Zero an INTEGER∗8 array. |
| ccm2.F | result | Print final energies. |
| data.F | zdata | Get user input data. |
| data.F | howmany | Determine number of iterations. |
| data.F | params | Initialize some parameters depending on o and v. |
| data.F | getinf | Read the numbers of occupied and virtual orbitals (o and v). |
| diis.F | diis_drv | Driver for DIIS. |
| diis.F | diis | Perform DIIS extrapolation after an iteration. |
| diis.F | writemat | Write a matrix to disk. |
| diis.F | readmat | Read a matrix from disk. |
| diis.F | closediis | Cleanup of DIIS stuff (close files). |
| double.F | d0a–d2a | Calculates $t2$ amplitude (CCSD). |
| double.F | d1at2b | Compute the most expensive ('ladder') diagram of $t2$ amplitude (obsolete, replaced by d1prep and daxpys). |
| double.F | energy | Calculate CC energy after each iteration. |
| double.F | vda | Calculate single terms contributing to double amplitude. |
| double.F | ampcnv | Convergence check for amplitudes. |
| int1.F | iq0a | Calculate $\tau$ intermediate. |
| int1.F | iq1a–iq6a | Calculate two-index $\chi$ intermediates and $\chi(i,j,k,l)$. |
| int1.F | fi0a,fi2a,fi3a | Calculate $f$ intermediates. |
| int2.F | iq7a–iq9a | Calculate four-index $\chi$ intermediates. |
| mem.F | ccmem | Memory partitioning. |
| mpi.F | mpion, mpioff | Initialize and close MPI environment |
| mpi.F | mpiar1, mpiar | Wrappers to MPI_ALL_REDUCE routine. |
| mpi.F | dist | Assign task numbers to all processes. |
| mpi.F | sy2124 | $x(i,j,k,l) \leftarrow 2x(i,j,k,l) - x(i,l,k,j)$. |
| mpi.F | de2124 | Do the reverse of sy2124. |
| mpi.F | nr2asc | Integer→string conversion for constructing filenames. |
| mpi.F | wcread, wcwrit | Simplified versions of newget and newput. |
| mpi.F | newput | Write amplitudes to disk. |
| mpi.F | newget | Read amplitudes from disk. |
| mpi.F | newgeta | Add amplitudes from disk to array. |
| mpi.F | inwc14 | $x(i,j,k,l) \leftarrow x(l,j,k,i)$. |

Table 6: List of subroutines - `cc` (continued)[a].

| Module | Subroutine | Comments |
|---|---|---|
| mpi.F | iveccp | Copy an integer vector. |
| mpi.F | ioset | Establish I/O channel numbers. |
| mpi.F | opens | Open integral, amplitude, and intermediate files. |
| mpi.F | estim | Calculate core size needed for CC. |
| newr.F | symtr | $x(i,a,j,b) \leftarrow x(i,a,j,b) + x(j,b,i,a)$. |
| newr.F | vecmul | Multiply vector by a constant. |
| newr.F | matmulsk | Wrapper to DGEMM matrix multiplier. |
| newr.F | matmula | Perform $C \leftarrow A \cdot B$ for one column of $C$. |
| newr.F | symt21 | $x(i,j,k,l) \leftarrow 2x(i,j,k,l) + x(i',j',k',l')$ (two indices switched). |
| newr.F | desm21 | Do the reverse of symt21. |
| newr.F | insi12 | $x(i,j,k) \leftarrow x(j,i,k)$. |
| newr.F | insi13 | $x(i,j,k) \leftarrow x(k,j,i)$. |
| newr.F | insi23 | $x(i,j,k) \leftarrow x(i,k,j)$. |
| newr.F | insitu | Permute two indices of a 4-dimensional array. |
| newr.F | transq | Transpose a square array. |
| newr.F | veccop, vecadd | Copy (add) one vector to another. |
| newr.F | tranmd | permute two indices (same size) of a 4-dimensional array. |
| newr.F | trt1 | Transpose a rectangular array. |
| newr.F | vadd21 | $x(i,a,j,b) \leftarrow 2x(i,a,j,b) - y(i,j,a,b)$. |
| newr.F | trnsp1 | $x(a,i,b,j) \leftarrow y(i,a,j,b)$. |
| newr.F | trnsp2 | $x(i,a,j,k) \leftarrow x(k,j,i,a)$. |
| newr.F | filli | Fill an INTEGER∗4 array with natural numbers. |
| newr.F | wt2 | Calculate second-order energy. |
| newr.F | vminus | Multiply a vector by $-1$. |
| rpamono.F | | RPA stuff (currently not functional). |
| single.F | sda | Calculate single amplitude terms depending on $\tau$ intermediates. |
| single.F | vsta | Calculate single amplitude terms depending on $\chi'$ intermediates. |
| single.F | ssa | Calculate single amplitude terms depending on $\tau$ intermediates. |
| single.F | t1ft2 | Calculate single ampl. terms depending on integrals and singles. |
| tpdrvn.F | tpdrvn | Loop over vvvv integrals in the 'ladder' diagram. |
| tpdrvn.F | d1prep | Store indices involved in the 'ladder' diagram. |
| tpdrvn.F | daxpys | Perform the 'ladder' diagram for one vvvv integral batch. |
| tpdrvn.F | fort_daxpy | Explicitly calculate DAXPY (usually faster than BLAS1). |
| tpdrvn.F | sort2 | Sort indices from d1prep to optimize cache use. |
| tpdrvn.F | resort | Re-sort four-virtual integrals for faster access. |
| tpdrvn.F | vvvvsort4 | Evaluate four-virtual diagram. |
| tpdrvn.F | sort4v | Sort four-virtual integrals with 2-fold desymmetrization. |
| tpdrvn.F | wrtbu | Write sorted integrals to disk. |
| tpdrvn.F | getbufv4mod | Read sorted integrals from disk. |
| tpdrvn.F | getbufseq | Read re-sorted integrals from disk. |
| triple.F | totsamp, totamp | Sum single and double amplitudes. |
| unpack.F | unpack10 | Unpack an (INTEGER∗4,INTEGER∗1) pair into four indices. |
| whole.F | errorx,errorx8 | Error exit from program. |
| whole.F | get2el | Get integrals from disk (obsolete, replaced by newget). |
| whole.F | timing | Gather and print timing info from subroutines. |
| whole.F | nmfind | Locate a subroutine in a list. |
| whole.F | indxsm | Calculate an index in a triangular matrix. |
| whole.F | bdaxpy,bdnrm2 | Wrappers to BLAS routines DAXPY and DNRM2. |
| whole.F | *other* | Auxiliary routines for get2el (obsolete). |

[a] A good reference for this program is the paper by M. Urban, I. Černušák, V. Kellö, and J. Noga in *Methods in Computational Chemistry* edited by S. Wilson, Plenum Press, 1987, page 117.

Table 7: Comments on selected subroutines - `sapt.x` (part 1).

| Module | Subroutine | Comments |
|---|---|---|
| m.F | driver | Main SAPT driver. |
| m.F | theta | Calculate $\Theta$ intermediate for monomer A. |
| m.F | thetb | Same as theta for monomer B. |
| m.F | veta | Calculate $v$ intermediate for monomer A. |
| m.F | vetb | Same as veta for monomer B. |
| m.F | omaov | Calculate $\Omega$ (occ,vir) intermediate for monomer A. |
| m.F | ombov | Same as omaov for monomer B. |
| m.F | tsa | Evaluate singles amplitudes for monomer A. |
| m.F | tsb | As tsa for monomer B. |
| m.F | copy | Copy vector A to vector B. |
| m.F | rbfov | Read into core 1el integrals of (occ,vir) type. |
| m.F | timt | Architecture-dependent function to read elapsed time. |
| m.F | omaoo | Computes $\Omega$ A (occ,occ) intermediate for A. |
| m.F | omavv | (vir,vir) for monomer A. |
| m.F | omboo | (occ,occ) for monomer B. |
| m.F | ombvv | (vir,vir) for monomer B. |
| m.F | rbfoo | Read into core 1-el integrals and store (occ,occ). |
| m.F | rbfvv | . . . and store (vir,vir). |
| m.F | save | Omit small values in a table. |
| m.F | nr2asc | Integer$\rightarrow$string conversion for constructing filenames. |
| b.F | report | Print a table with subroutine timings. |
| b.F | izero | Zero an integer array. |
| b.F | r8zerobig | Zero a REAL$*$8 array indexed by an INTEGER$*$8 variable. |
| b.F | r8zero | Zero a REAL$*$8 array indexed by an INTEGER$*$4 variable. |
| b.F | saptbd | Initialization block data for the program. |
| b.F | alarm0 | Error exit with an INTEGER$*$8 message code. |
| b.F | alarm0a | Error exit with an INTEGER$*$4 message code. |
| b.F | ienter | Keep track of subroutines entered for timing purposes. |
| b.F | ndx0 | Indexing function with entries ndx$nn$ for various 2el. integrals. |
| b.F | readin | Read 2el integrals, special version for $E^{(10)}$. |
| b.F | readon | Read 1el integrals, store only (occ,occ). |
| b.F | readov | Read 1el integrals of a given type. |
| b.F | readbf | Read 2el integrals, most general version. |
| b.F | readvv | Read 1el integrals divided by number of electrons. |
| b.F | readen | Read HF orbital energies from disk. |
| b.F | getbuf | Get sorted 2el integral buffer from disk. |
| b.F | wrtbu | Write sorted 2el buffer. |
| b.F | iexit | Keep track of subroutines exited for timing purposes. |
| b.F | lrecl | Adjust buffer sizes for sorting 2el integrals. |
| b.F | rbfab | Read first-order dispersion amplitudes. |
| b.F | getampr11 | Reads amplitudes written by the e2disp program. |
| b.F | getr11 | Opens an appropriate file and calls getampr11. |
| chf.F | setchf | Coupled Hartree-Fock routine driver. |
| chf.F | solvea | Linear eq. solver for monomer A. |
| chf.F | solvea_ooc | Out-of-core version of the above. |
| chf.F | solveb | Linear eq. solver for monomer B. |
| chf.F | solveb_ooc | Out-of-core version of the above. |
| chf.F | quit | Exit routine if no convergence. |
| chf.F | putchf | Write computed CHF coefficients onto disk. |
| chf.F | getchf | Get computed CHF coefficients back. |

Table 7: Comments on selected subroutines - `sapt.x` (part 2).

| Module | Subroutine | Comments |
|---|---|---|
| getamp.F | getamp | Retrieve monomer CC amplitudes from disk. |
| getamp.F | gampoovv | Like getamp but with permuted indices. |
| getamp.F | gampvovo | ” |
| getamp.F | gampvvoo | ” |
| getamp.F | newget | Actual reading of amplitudes. |
| getamp.F | newoovv | Like newget but with permuted indices. |
| getamp.F | newvovo | ” |
| getamp.F | newvvoo | ” |
| getamp.F | ampopen | Open files with monomer CC amplitudes. |
| memreq.F | memreq | Calculate memory needed for different corrections. |
| unpack10.F | unpack10 | Unpack an (INTEGER∗4,INTEGER∗1) pair into four indices. |
| unpack10.F | pack10 | Do the reverse of unpack10. |
| unpack10.F | unpack10a | Unpack an (INTEGER∗4,INTEGER∗1) pair into two indices. |
| unpack10.F | pack10a | Do the reverse of unpack10a. |
| unpack10.F | spltindx | Split an index into an (INTEGER∗4,INTEGER∗1) pair. |
| unpack10.F | joinindx | Do the reverse of spltindx. |
| e1.F | first | $E^{(10)}_{\rm elst}$ and $E^{(10)}_{\rm exch}$ driver. |
| e1.F | delta | Special form of Kronecker delta function. |
| e1.F | inv | Calculate inverse matrix. |
| e1.F | pmat | Prepare the $P$ matrix for inversion. |
| e1xs2.F | e1xs2 | Driver routine for $E^{(10)}_{\rm exch}(S^2)$. |
| e1xs2.F | e1s2k | Actual calculation of $E^{(10)}_{\rm exch}(S^2)$. |
| e12.F | srt12 | $E^{(120)}_{\rm elst}$ and $E^{(102)}_{\rm elst}$ driver routine. |
| e12.F | sort12 | Presort of 2el integrals for $E^{(12)}_{\rm elst}$. |
| e12.F | e120pl | Compute $E^{(120)}_{\rm elst}$. |
| e12.F | e102pl | Compute $E^{(102)}_{\rm elst}$. |
| e13.F | e13 | A wrapper for the $E^{(13)}_{\rm elst}$ driver. |
| e13.F | e13drv | Actual driver for $E^{(130)}_{\rm elst}$ and $E^{(103)}_{\rm elst}$. |
| e13.F | e13pl1–e13pl7 | Compute components of $E^{(13)}_{\rm elst}$. |
| e13.F | komaov | Compute (occ,vir) electrostatic potential for A–different version. |
| e13.F | kombov | As komaov but for monomer B. |
| e13.F | komaoo | As komaov but form (occ,occ) matrix. |
| e13.F | komavv | As komaov but form (vir,vir) matrix. |
| e13.F | komboo | As komaoo but for monomer B. |
| e13.F | kombvv | As komavv but for monomer B. |
| e11x.F | e11ex | $E^{(11)}_{\rm exch}$ driver. |
| e11x.F | e11x | Actual calculation of $E^{(11)}_{\rm exch}$. |
| e11x.F | transpov | Transpose the overlap matrix. |
| e11x.F | e11x1–e11x4 | Compute components of $E^{(11)}_{\rm exch}$. |
| e1x.F | e111e | $E^{(111)}_{\rm exch}$ driver. |
| e1x.F | e111exh | Calculate $E^{(111)}_{\rm exch}$. |
| e1x.F | mkg24 | Construct $g$ intermediate. |
| e1x.F | readbfshrink | Read two-electron integrals omitting the core ones. |

Table 7: Comments on selected subroutines - `sapt.x` (part 3).

| Module | Subroutine | Comments |
|---|---|---|
| e12x.F | e12ex | Main $E^{(120)}_{\rm exch}$ and $E^{(102)}_{\rm exch}$ driver. |
| e12x.F | k2f1 | Driver for the $K^2_{\rm f}$ part of $E^{(120)}_{\rm exch}$. |
| e12x.F | k2fa | Calculate the $K^2_{\rm f}$ part of $E^{(120)}_{\rm exch}$. |
| e12x.F | k2f2 | Driver for the $K^2_{\rm f}$ part of $E^{(102)}_{\rm exch}$. |
| e12x.F | k2fb | Calculate the $K^2_{\rm f}$ part of $E^{(102)}_{\rm exch}$. |
| e12x.F | add1a | Prepare the one-electron component for k2fa. |
| e12x.F | add2a | Prepare the one-electron component for k2fb. |
| k11u.F | k11u | Driver for the $K^{11}_{\rm u}$ part of $E^{(12)}_{\rm exch}$. |
| k11u.F | k11u1–k11u32 | Calculate components of $K^{11}_{\rm u}$. |
| k11u.F | writemat | Write a matrix to a temporary file. |
| k11u.F | getmat | Read a matrix from a temporary file. |
| k11u.F | getmat2 | Another version of getmat. |
| e2.F | e02 | MBPT2 monomer energies calculation. |
| e2.F | e200d | $E^{(20)}_{\rm disp}$ driver. |
| e2.F | e200disp | Calculate $E^{(20)}_{\rm disp}$. |
| e2.F | eind | $E^{(20)}_{\rm ind}$ driver. |
| e2.F | eindab | Calculate $E^{(20)}_{\rm ind}$. |
| e2.F | e21 | $E^{(21)}_{\rm disp}$ driver. |
| e2.F | prep210 | Prepare matrices for $E^{(210)}_{\rm disp}$. |
| e2.F | prep201 | Prepare matrices for $E^{(201)}_{\rm disp}$. |
| e2.F | e21d | Actual calculation of $E^{(210)}_{\rm disp}$ or $E^{(201)}_{\rm disp}$. |
| e2.F | e21d1–e21d3 | Compute components of $E^{(21)}_{\rm disp}$. |
| e4i.F | e22i0 | $E^{(22)}_{\rm ind}$ driver. |
| e4i.F | e22its | Compute triple excitation part of $E^{(22)}_{\rm ind}$. |
| e4i.F | e22is | Compute single excitation part of $E^{(22)}_{\rm ind}$. |
| e4i.F | e22irl | Compute the ring-ladder diagram of $E^{(22)}_{\rm ind}$. |
| e4i.F | e22ib | Compute the remainder of $E^{(22)}_{\rm ind}$. |
| e4.F | e22d0 | $E^{(22)}_{\rm disp}$ driver. |
| e4.F | srt220 | Sort three-virtual integrals for $E^{(220)}_{\rm disp}$. |
| e4.F | srt202 | As above but for $E^{(202)}_{\rm disp}$. |
| e4.F | e211a | Calculate the first part of $E^{(211)}_{\rm disp}$. |
| e4.F | e211b | Calculate the remaining terms in $E^{(211)}_{\rm disp}$. |
| e4.F | e22ds | Compute $E^{(220)}_{\rm disp}$(S) or $E^{(202)}_{\rm disp}$(S). |
| e4.F | e22dr | The ring contribution to $E^{(220)}_{\rm disp}$(D)/$E^{(202)}_{\rm disp}$(D). |
| e4.F | e22rl | The ring-ladder contribution to $E^{(220)}_{\rm disp}$(D)/$E^{(202)}_{\rm disp}$(D). |
| e4.F | e22da | First part of $E^{(220)}_{\rm disp}$(Q)/$E^{(202)}_{\rm disp}$(Q). |
| e4.F | e22db | Second part of $E^{(220)}_{\rm disp}$(Q)/$E^{(202)}_{\rm disp}$(Q). |
| e4.F | e220dso | Compute singles term for $E^{(220)}_{\rm disp}$(CCD+ST(CCD)). |
| e4.F | e202dso | Compute singles term for $E^{(202)}_{\rm disp}$(CCD+ST(CCD)). |
| e4.F | prntime1 | Print extended timings for $E^{(22)}_{\rm disp}$. |

Table 7: Comments on selected subroutines - `sapt.x` (part 4).

| Module | Subroutine | Comments |
|---|---|---|
| e22t94.F | e22t94 | $E^{(220)}_{\rm disp}(\mathrm{T})/E^{(202)}_{\rm disp}(\mathrm{T})$ driver. |
| e22t94.F | eq98 | Outer loops for $E^{(22)}_{\rm disp}(\mathrm{T})$. |
| e22t94.F | eq99 | Inner loops for $E^{(22)}_{\rm disp}(\mathrm{T})$. |
| e22t94.F | invndx | Calculate orbital indices from the amplitude index. |
| e22t94.F | srt22t94 | Sort three-virtual integrals for $E^{(22)}_{\rm disp}(\mathrm{T})$. |
| e22t94.F | getvvb | Special version of b.F/getbuf. |
| e22t94.F | getvva | ” |
| e22t94.F | rbfabx | Special version of b.F/rbfab. |
| e22t94.F | pack2 | Pack two integers into one INTEGER$*$4 word. |
| e22t94.F | unpack2 | Undo pack2. |
| e22t94.F | getr11x | Special version of b.F/getr11. |
| e22t94.F | getampx | Special version of b.F/getampr11. |
| e2ex.F | e2ex | Main driver for $E^{(20)}_{\rm exch}$. |
| e2ex.F | exia | Driver for $E^{(20)}_{\rm exch-ind}(A \leftarrow B)$. |
| e2ex.F | e2iba | Calculate $E^{(20)}_{\rm exch-ind}(A \leftarrow B)$. |
| e2ex.F | exib | Driver for $E^{(20)}_{\rm exch-ind}(B \leftarrow A)$. |
| e2ex.F | e2iab | Calculate $E^{(20)}_{\rm exch-ind}(B \leftarrow A)$. |
| e2ex.F | ex2d | Driver for the in-core version of $E^{(20)}_{\rm exch-disp}$. |
| e2ex.F | exd2 | Actual (in-core) calculation of $E^{(20)}_{\rm exch-disp}$. |
| e2ex.F | ex2d1a–ex2d1c | Calculate components of $E^{(20)}_{\rm exch-disp}$. |
| e2ex.F | ex2d2–ex2d8 | ” |
| e2ex.F | transp13 | Transpose an intermediate matrix. |
| e2xdooc.F | ex2dsemiooc | Driver for the out-of-core version of $E^{(20)}_{\rm exch-disp}$. |
| e2xdooc.F | wrseq | Write an intermediate matrix to disk. |
| e2xdooc.F | rdseq | Read an intermediate matrix from disk. |
| e2xdooc.F | writechunk | Write a chunk of the intermediate matrix. |
| e2xdooc.F | readchunk | Read a chunk of the intermediate matrix. |
| e2xdooc.F | wrooo | Special version of wrseq. |
| e2xdooc.F | exd2ooc | Actual (out-of-core) calculation of $E^{(20)}_{\rm exch-disp}$. |
| e2xdooc.F | exd2semiooc | Actual (semi out-of-core) calculation of $E^{(20)}_{\rm exch-disp}$. |
| e2xdooc.F | oex2d1a–oex2d1c | Out-of-core-versions of e2ex.F/ex2d1a–ex2d1c. |
| e2xdooc.F | preoex2d2 | Prepare matrices for oex2d2. |
| e2xdooc.F | oex2d2 | Out-of-core-version of e2ex.F/ex2d2. |
| e2xdooc.F | preoex2d3 | Prepare matrices for oex2d3. |
| e2xdooc.F | oex2d3–oex2d4 | Out-of-core-versions of e2ex.F/ex2d3–ex2d4. |
| e2xdooc.F | preoex2d5 | Prepare matrices for oex2d5. |
| e2xdooc.F | oex2d5 | Out-of-core-version of e2ex.F/ex2d5. |
| e2xdooc.F | preoex2d6 | Prepare matrices for oex2d6. |
| e2xdooc.F | oex2d6–oex2d8 | Out-of-core-versions of e2ex.F/ex2d6–ex2d8. |
| e2xdooc.F | srte2xd | Sort integrals for out-of-core $E^{(20)}_{\rm exch-disp}$. |
| e2xdooc.F | getbufshrink | Like b.F/getbuf but omit core integrals. |
| e2xdooc.F | readbfs14 | Read integrals for a fixed index. |
| e2xdooc.F | readbfs15 | ” |
| e2xdooc.F | readbfs48 | ” |

Table 7: Comments on selected subroutines - `sapt.x` (part 5).

| Module | Subroutine | Comments |
|---|---|---|
| e3.F | srt3d0 | Driver routine for $E^{(30)}_{\rm disp}$. |
| e3.F | sort3d | Sorting routine for above. |
| e3.F | e3dsp | Calculate above. |
| e3.F | dspin0 | Driver routine for $E^{(30)}_{\rm ind-disp}$. |
| e3.F | srtind | Sorting routine for above. |
| e3.F | dspin1a–dspin1b | Compute first part of $E^{(30)}_{\rm ind-disp}$. |
| e3.F | dspin2 | Compute second part of $E^{(30)}_{\rm ind-disp}$. |
| e3.F | eind3 | Driver for $E^{(30)}_{\rm ind}$. |
| e3.F | e3ind | Actual calculation of $E^{(30)}_{\rm ind}$. |
| e3.F | readbfsh2 | Special version of e1x.F/readbfshrink. |
| e3.F | readbfsh3 | ” |
| e3.F | readbfsh4 | ” |
| e3x.F | e30exi | Driver routine for $E^{(30)}_{\rm exch-ind}$. |
| e3x.F | e3x1 | Prepare amplitudes for $E^{(30)}_{\rm exch-ind}(10)$ and (01). |
| e3x.F | e3x3a | Calculate $E^{(30)}_{\rm exch-ind}(20)$. |
| e3x.F | e3x3b | Calculate $E^{(30)}_{\rm exch-ind}(02)$. |
| e3x.F | e30exdi | Driver routine for $E^{(30)}_{\rm exch-ind-disp}$. |
| e3x.F | e3xid1a | Prepare amplitudes for $E^{(30)}_{\rm exch-ind-disp}(10)$. |
| e3x.F | e3xid1b | Prepare amplitudes for $E^{(30)}_{\rm exch-ind-disp}(01)$. |
| e3x.F | e3xid2a–e3xid2b | Prepare amplitudes for $E^{(30)}_{\rm exch-ind-disp}(11)$. |
| e3x.F | e3xid4a–e3xid6a | Calculate $E^{(30)}_{\rm exch-ind-disp}(21)$. |
| e3x.F | e3xid4b–e3xid6b | Calculate $E^{(30)}_{\rm exch-ind-disp}(12)$. |
| e3x.F | e3x11 | Calculate the (11) part of $E^{(30)}_{\rm exch}$. |
| e3x.F | ex2d3ind | Version of e2ex.F/ex2d3 suitable for $E^{(30)}_{\rm exch}(11)$. |
| e3x.F | ex2d4ind | Version of e2ex.F/ex2d4 suitable for $E^{(30)}_{\rm exch}(11)$. |
| e3x.F | prep11i | Prepare amplitudes for $E^{(30)}_{\rm exch-ind}(11)$. |
| e3xd.F | e30exd | Driver routine for $E^{(30)}_{\rm exch-disp}$. |
| e3xd.F | e3xd2a–e3xd2c | Prepare amplitudes for $E^{(30)}_{\rm exch-disp}(11)$. |
| e3xd.F | e3xd4a–e3xd4c | Calculate $E^{(30)}_{\rm exch-disp}(20)$. |
| e3xd.F | e3xd4d–e3xd4f | Calculate $E^{(30)}_{\rm exch-disp}(02)$. |
| e3xd.F | e3xd5ab | Calculate $E^{(30)}_{\rm exch-disp}(21)$. |
| e3xd.F | e3xd5cd | Calculate $E^{(30)}_{\rm exch-disp}(12)$. |
| e3xd.F | e3xd6 | Calculate $E^{(30)}_{\rm exch-disp}(22)$. |
| e3xd.F | prep11d | Prepare amplitudes for $E^{(30)}_{\rm exch-disp}(11)$. |
| direct.F | directe3d | Driver for the semi-AO-based $E^{(30)}_{\rm disp}$. |
| direct.F | directe3xd | Driver for the semi-AO-based $E^{(30)}_{\rm exch-disp}$. |
| direct.F | eeo | Calculate the four-virtual diagram in AO basis. |
| direct.F | de3dsp | Actual (semi-AO-based) calculation of $E^{(30)}_{\rm disp}$. |
| direct.F | de30exd | Actual (semi-AO-based) calculation of $E^{(30)}_{\rm exch-disp}$. |
| direct.F | de3xd2a | Version of e3xd.F/e3xd2a with amplitudes computed in AOs. |
| direct.F | namemain | Get the name of the ATMOL1024 integral file. |
| direct.F | search | Manipulate the ATMOL1024 integral file. |
| direct.F | find | ” |
| direct.F | rdsam | ” |

## D Summary table from output for the example BER ($Be_2$)

```
========================================================================
                          Summary Table
========================================================================
               Mono A:     2 occupied    8 virtual   10 total
               Mono B:     2 occupied    8 virtual   10 total
               ----------- Molecule  A       4 Electron(s)
               ATOM       XX            YY             ZZ        Charge
                  1    0.000000000    0.000000000    0.000000000  4.0

               ----------- Molecule  B       4 Electron(s)
                  2    0.000000000    7.000000000    0.000000000  4.0
E^{HF}_{AB}                -21.5579794623086016   hartrees
E^{HF}_{A}                 -10.7773391618701009   hartrees
E^{HF}_{B}                 -10.7773391618701009   hartrees
------------------------------------------------------------------------
Correction                   mHartree        Kcal/mol            1/cm
----------                   --------        --------            ----
                                     --- SCF (SAPT_super) ---
E^{HF}_{int}              -3.301138568      -2.07149746       -724.5162
E^{(10)}_{elst}           -3.687513408      -2.31395154       -809.3156
E^{(10)}_{exch}            3.149525953       1.97635903        691.2410
E^{(10)}_{exch}{S^2}       3.135991881       1.96786627        688.2707
E^{(10)}_{exch}-S^2        0.013534072       0.00849277          2.9704
E^{(20)}_{ind}            -4.659057533      -2.92360519      -1022.5449
E^{(20)}_{ind,resp}       -6.900439657      -4.33009489      -1514.4714
E^{(20)}_{ex-ind}          4.109230703       2.57858336        901.8719
E^{(20)}_{ex-ind,r}        6.033628681       3.78616233       1324.2284
SAPT SCF ^a               -1.087814286      -0.68261434       -238.7476
SAPT SCF_{resp} ^b        -1.404798432      -0.88152506       -308.3176
\delta^{HF}_{int}         -2.213324283      -1.38888312       -485.7685
\delta^{HF}_{int,r}       -1.896340137      -1.18997240       -416.1985

                                     --- CORRELATION ---
E^{(12)}_{elst}            0.616739822       0.38701041        135.3587
E^{(13)}_{elst}            0.811463861       0.50920169        178.0957
\eps^{(1)}_{elst}(k)       1.428203683       0.89621209        313.4545
E^{(12)}_{elst,resp}       0.748858176       0.46991599        164.3554
E^{(13)}_{elst,resp}       0.764997350       0.48004349        167.8975
\eps^{(1)}_{elst,r}(k)     1.513855526       0.94995948        332.2529
E^{(11)}_{exch}            0.640864839       0.40214909        140.6536
E^{(12)}_{exch}           -0.125792341      -0.07893595        -27.6082
\eps^{(1)}_{exch}(k)       0.515072498       0.32321314        113.0453
\eps^{(1)}_{exch}(CCSD)   -0.784465905      -0.49226020       -172.1704
^tE^{(22)}_{ind}           1.081861737       0.67887906        237.4412
^tE^{(22)}_{ex-ind}*      -0.945961754      -0.59360046       -207.6146
E^{(20)}_{disp}           -1.195076004      -0.74992214       -262.2889
E^{(21)}_{disp}           -0.203325820      -0.12758899        -44.6249
E^{(22)}_{disp}            0.473822197       0.29732817        103.9920
\eps^{(2)}_{disp}(k)       0.270496376       0.16973918         59.3671
E^{(2)}_{disp}(k)         -0.924579627      -0.58018296       -202.9218
E^{(20)}_{exch-disp}       0.069731391       0.04375715         15.3043
SAPT_{corr}               -0.075210475      -0.04719533        -16.5068
SAPT_{corr,resp}           0.010441367       0.00655206          2.2916

                                     --- TOTAL (hybrid) ---
```

```
SCF+SAPT_{corr}           -3.376349044      -2.11869279        -741.0230
SCF+SAPT_{corr,resp}      -3.290697201      -2.06494540        -722.2246
==================================================================
```

## E Capabilities of pcksdisp program

This Appendix contains a more detailed description of the program `pcksdisp`. In the `pEDI.X` scripts, this program is used to calculate the CHF static and dynamic susceptibility functions of the monomers. In addition, both these objects can also be calculated at the UCHF level. The CHF and UCHF induction and dispersion energies are also reported. The uncoupled (UCHF) dispersion and induction energies are equivalent to $E^{(20)}_{\rm disp}$ and $E^{(20)}_{\rm ind}$ SAPT corrections, respectively. At the CHF level, the induction energy is equal to $E^{(20)}_{\rm ind,resp}$. The CHF dispersion is equivalent to the so-called RPA dispersion (see Ref. 40 for examples). The RPA dispersion energy is currently not computed by the regular (non-parallel) SAPT algorithms. Two other quantities that can be obtained using `pcksdisp` are the static/dynamic dipole-dipole polarizability tensor and the isotropic $C_6$ dispersion asymptotic coefficient (for the interaction of identical monomers), both at the UCHF and CHF levels.

The `pcksdisp` program uses the transformed intra- and intermonomer integrals (i.e., the MO representation) as generated by the `ptran` module. Therefore, within a script like `pEDI.X`, it should be run after the SCF and transformation programs. Note that currently `pcksdisp` assumes all integrals to be located in a single file `f2e.000.001`. Thus, all such files have to be properly merged after a parallel `ptran` run. This task is accomplished with the help of the program `tmerge`. To ensure that `ptran` generates all the integrals necessary for `pcksdisp`, the namelist `INPUTCOR` in the `nameP.data` file should contain the directives

```
CHFDISP=T, CHFIND=T
```

The actual control parameters for `pcksdisp` are collected in the namelist `INPUT`, which should be appended to `nameP.data` (the scripts `pEDI.X` do this automatically) and look similar to

```
&INPUT
   ISITCASPOL=T, ISITINDUCT=T,
   ISITSOSDISP=T,
   ISITPROP=F,
   ISITCKS=F, ISITUCKS=T,
   ISITPOL=F,ISITC6DISP=F,
   USESUMN6=T,
   MAKEH1H2=T,
   IQUADTYP=1, NQUAD=10,
   OMEGA0=0.5,
   DEBUG1=T, DEBUG2=F, DEBUG3=F, DEBUG4=F, DEBUG5=F, DEBUG6=F, DEBUG7=F,
   DEBUG8=F
 &END
```

The meaning of the options is as follows:

- **Main Control flags:**
  - `ISITCASPOL` : (`T/F`) Set if this is a Casimir-Polder dispersion calculation.
  - `ISITINDUCT` : (`T/F`) Set if this is an induction calculation.
  - `ISITPROP` : (`T/F`) Set if a properties calculation needs to be performed.
  - `ISITSOSDISP` : (`T/F`) Set to perform the regular sum-over-states (SOS) $E^{(20)}_{\rm disp}$ calculation.
- **Propagator Type (only one can be selected):**
  - `ISITCKS` : (`T/F`) Set to `T` if the propagator is to be computed in the CHF approximation, otherwise – set to `F`.
  - `ISITUCKS` : (`T/F`) Set to `T` if the UCHF approximation is to be used, otherwise – set to `F`.
- **Properties calculation options:**
  - `ISITPOL` : (`T/F`) Set if the frequency-dependent dipole polarizability tensor $\alpha_{xy}(\omega)$ is to be computed. The frequencies at which the computation will be made are set by the input keywords `NUMFREQ` and `FREQ<#>` (see below). The dipole integrals are needed for this calculation.
  - `ISITC6DISP` : (`T/F`) Set if the $C_6$ dispersion coefficient is to be computed. At the moment the program outputs only the isotropic coefficient.
- `USESUMN6` : (`T/F`) Sets the $o^3v^3$ summation algorithm in a Casimir-Polder dispersion calculation. Set this to `T` unless you want to use the old $o^4v^4$ summation algorithm.
- `MAKEH1H2` : (`T/F`) Set to enable construction of the Electric and Magnetic Hessians (the $H^{(1)}$ and $H^{(2)}$ matrices) in the CHF approximation. Transformed integrals of certain types are required for this option. If this option is set to `F`, then the $H^{(1)}$ and $H^{(2)}$ matrices must be read in from a file in either the CHF or CKS approximation.
- `NUMFREQ` and `FREQ1, FREQ2,...,FREQ8` : `NUMFREQ` is the number of frequencies at which a frequency-dependent polarization calculation is to be made. This should be less than or

equal to 8. The variables `FREQ1, FREQ2,...,FREQ8` contain the frequencies. If complex frequencies are required, set a negative frequency. These variables override the quadrature scheme described below.

- `IQUADTYP` and `NQUAD` : The type of quadrature scheme to be used in performing the $\omega$-integral in the Casimir-Polder dispersion calculation and in the calculation of the $C_6$ dispersion coefficient:

  - `IQUADTYP=1` sets the Gauss-Legendre quadrature with the transformation $\omega = \omega_0 \frac{(1+t)}{(1-t)}$.
  - `IQUADTYP=2` sets the Gauss-Legendre quadrature with the transformation $\omega = \omega_0 \tan(t)$.
  - `IQUADTYP=3` sets the Gauss-Laguerre quadrature scheme.

  The variable `NQUAD` sets the number of quadrature points to be used for the integration.

- `OMEGA0` and `ALPHA` : The transformation used in the Gauss-Legendre quadrature schemes involves a constant $\omega_0$. The namelist variable `OMEGA0` allows one to set this constant. It is typically between 0.3 and 0.5. The Gauss-Laguerre quadrature scheme involves the constant $\alpha$. For the integrals encountered here, one should set `ALPHA=0.0`.

- **Debugging levels** : There are many debugging levels built into the code. These can be activated by setting the relevant combination of debugging flags to `T`. Nota bene: large matrices may be printed out at certain levels. Here is a list of current debugging levels:

  - `DEBUG1` : (`T/F`) Test the quadrature grid.
  - `DEBUG2` : (`T/F`) Use the un-coupled propagator in the coupled propagator route. This is very useful when testing the code as the dispersion energy obtained this way should be the same as the value of $E^{(20)}_{\text{disp}}$ obtained from the SOS formula.
  - `DEBUG3` : (`T/F`) Print out all matrices in the construction of the coupled propagator.
  - `DEBUG4` : (`T/F`) Print out all 2-electron integrals read in.
  - `DEBUG5` : (`T/F`) Print out information in the $N^6$ summation algorithm (subroutine `SUMN6`).
  - `DEBUG6` : (`T/F`) Print out information in the `PROPERTIES` and `C6DISP` routines.
  - `DEBUG7` : (`T/F`) Print out integrals, etc. Used in computing the $H^{(1)}$ and $H^{(2)}$ matrices in the CHF approximation.
  - `DEBUG8` : (`T/F`) Print out information in the induction module. Integrals and intermediates are printed. This can generate a very large output.

# F Generation of auxiliary basis

This Appendix is adopted from the Ph.D. Thesis by Alston Misquitta [73].

The following procedure can be applied to construct an auxiliary basis set for each atom in the dimer under consideration. Denoted by $\mathcal{M}$ is the decontracted basis set used in calculations for given dimer. The auxiliary basis, denoted by $\mathcal{X}$, is developed as follows:

1. Construct an auxiliary basis as the tensor product of $\mathcal{M}$ with itself. That is,

$$\mathcal{X} = \mathcal{M} \otimes \mathcal{M}. \tag{18}$$

   If $G^{\mathcal{M}}(l_i, \alpha_i)$ and $G^{\mathcal{M}}(l_j, \alpha_j)$ are two basis functions of $\mathcal{M}$ centered at the same point, where $l_i$ and $l_j$ are the angular quantum numbers and $\alpha_i$ and $\alpha_j$ are the exponents, then the product is a basis function belonging to $\mathcal{X}$ centered at the same point and given by $G^{\mathcal{X}}(l_k, \alpha_k)$ where $l_k = l_i + l_j$ and $\alpha_k = \alpha_i + \alpha_j$. The resulting basis $\mathcal{X}$ is a (large) basis including high symmetry functions compared to the original basis. All the products involving basis functions from different centers are rejected.

2. Within each angular symmetry of $\mathcal{X}$, a reduction is performed in the number of basis functions as follows. Given an $\epsilon$, if there are $n$ basis functions for which $\log \alpha_{k_1}$, $\log \alpha_{k_2}$,...,$\log \alpha_{k_n}$ are in an $\epsilon$ neighborhood, then these $n$ functions are replaced by one function with the exponent $\beta = (\alpha_{k_1} \alpha_{k_2} ... \alpha_{k_n})^{1/n}$. Perform this reduction for the whole basis set.

3. If necessary, the resulting basis can be reduced even further with the previous step repeated on the reduced basis with a different value of $\epsilon$.

4. Further pruning of the reduced basis $\mathcal{X}$ can be done as follows:

   (a) Reject all functions of $g$-symmetry and higher. This is necessary if CADPAC is used to obtain integrals.

   (b) The 'large' exponents, particularly the ones of high symmetries, can generally be removed. The criterion for this removal is best found by trial and error and by monitoring the constraint conditions.

We have found the optimal values of $\epsilon$ used in the pruning process to be between 0.3 and 0.5. This procedure typically results in an auxiliary basis $\mathcal{X}$ that is 2 to 3 times larger than the basis used to obtain molecular orbitals and eigenvalues. While in general the auxiliary basis for a molecule should be centered also on sites between pairs of atoms in addition to atomic sites, it is our experience that the use of only atomic centers is adequate. This is also true for the optimized auxiliary basis sets [14].