libgrpp

a library for the evaluation of molecular integrals of the generalized relativistic pseudopotential operator (GRPP) over Gaussian functions

v 1.0.0

programmer's manual

written by
A. Oleynichenko

Quantum Chemistry Laboratory
Petersburg Nuclear Physics Institute named by B. P. Konstantinov
National Research Centre "Kurchatov Institute"
Gatchina, Russia

August 25, 2023

Contents

1 Introduction

`libgrpp` is a library of subroutines for the evaluation of molecular integrals of the generalized relativistic pseudopotential operator (GRPP) over Gaussian functions [1].

1.1 Generalized relativistic pseudopotentials

Generalized (or Gatchina) relativistic pseudopotentials (GRPPs) of atomic cores imply the use of different semilocal potentials for atomic electronic shells with the same orbital and total momentum quantum numbers but with different principal quantum numbers [2, 3, 4]. GRPPs give rise to accurate and reliable relativistic electronic structure models of atoms, molecules, clusters and solids [5, 6, 7]. GRPPs can readily incorporate the effects of Fermi nuclear charge distribution, Breit electron-electron interactions [8, 9] and quantum electrodynamics effects (electron self-energy and vacuum polarization) [10, 11]. GRPPs give rise to very accurate relativistic Hamiltonians at the moment, allowing one to completely bypass any complicated four-component calculations at a price of a very moderate loss of accuracy [1].

Generalized pseudopotentials developed in the Quantum Chemistry Laboratory (NRC "Kurchatov Institute" – PNPI) in the last three decades are available online at `http://qchem.pnpi.spb.ru/recp`.

1.2 Features of `libgrpp`

Basis functions:

- Cartesian contracted GTOs;
- max angular momentum of basis functions $l_{max} = 10$ (up to n-functions, can be increased by hands).

Pseudopotential integrals:

- scalar-relativistic part: integrals over the local potential (type 1 integrals);
- scalar-relativistic part: integrals with angular projectors (type 2 integrals);
- integrals over the effective spin-orbit (SO) interaction operator;
- integrals over GRPP-specific non-local terms (with projectors onto outercore shells);
- analytic gradients of GRPP integrals.

Other one-electron integrals:

- overlap integrals;
- nuclear attraction integrals.

`libgrpp` provides C and Fortran 90 programming interfaces.

`libgrpp` does not depend on any external libraries. GNU Scientific Library (GSL) comes with LIBGRPP and is not required to be pre-build separately.

1.3 How to compile library and run tests

Intel compilers:

```
mkdir build
cd build
CC=icc FC=ifort cmake ..
make
make test
```

GNU compilers:

```
mkdir build
cd build
CC=gcc FC=gfortran cmake ..
make
make test
```

1.4 Citation

We kindly ask you to acknowledge any use of the `libgrpp` library that results in published material using the following citation:

A. V. Oleynichenko, A. Zaitsevskii, N. S. Mosyagin, A. N. Petrov, E. Eliav, A. V. Titov.
LIBGRPP: A library for the evaluation of molecular integrals of the generalized relativistic pseudopotential operator over Gaussian functions.
***Symmetry*, 15(1), 197 (2023).**
doi: 10.3390/sym15010197

```
@article{libgrpp2023,
    title = {{LIBGRPP}: A library for the evaluation of molecular integrals of
        the generalized relativistic pseudopotential operator over {G}aussian
        functions},
    author = {A. V. Oleynichenko and A. Zaitsevskii and N. S. Mosyagin
        and A. N. Petrov and E. Eliav and A. V. Titov},
    year = {2022},
    journal = {Symmetry},
    volume = {15},
    year = {2023},
    number = {1},
    article-number = {197},
    url = {https://www.mdpi.com/2073-8994/15/1/197},
    doi = {10.3390/sym15010197}
}
```

1.5 Bug report

The authors will be grateful for any comments, bug reports or suggestions:
alexvoleynichenko@gmail.com

2 Definition of the generalized pseudopotential (GRPP) operator

Generalized (Gatchina) relativistic pseudopotential in the spinor representation centered at point $\boldsymbol{C}$ is defined as [4]

$$\begin{aligned}\hat{U}_C^{GRPP} = U_{LJ}(r) \; &+ \sum_{lj} \left[U_{lj}(r) - U_{LJ}(r)\right] P_{lj} \\ &+ \sum_{lj} \sum_{n_c} \{\tilde{P}_{n_c lj} \left[U_{n_c lj}(r) - U_{lj}(r)\right] + \left[U_{n_c lj}(r) - U_{lj}(r)\right] \tilde{P}_{n_c lj}\} \\ &- \sum_{lj} \sum_{n_c n_c'} \tilde{P}_{n_c lj} \left[\frac{U_{n_c lj}(r) + U_{n_c' lj}(r)}{2} - U_{lj}(r)\right] \tilde{P}_{n_c' lj},\end{aligned} \tag{1}$$

where $r = |\boldsymbol{r} - \boldsymbol{C}|$ is the distance to the point $\boldsymbol{C}$ and $\tilde{P}_{n_c lj} = \sum\limits_m |\tilde{\phi}_{n_c ljm}\rangle \langle \tilde{\phi}_{n_c ljm}|$ are projectors onto outercore pseudospinors $\tilde{\phi}_{n_c ljm}$. Projectors $\tilde{P}_{n_c lj}$ depend on r and thus the $\hat{U}^{GRPP}$ operator is non-local. The first and the second terms represent ordinary semi-local potential routinely used in modern quantum chemistry.

The spinor representation (1) is not convenient for practical applications. It is beneficial to convert the $\hat{U}^{GRPP}$ operator to the spin-orbital representation. This approach allows one to treat scalar-relativistic potential and effective spin-orbit interaction separately:

$$\begin{aligned}\hat{U}^{GRPP} = U_L(r) &+ \sum_{l=0}^{L-1} \left[U_l(r) - U_L(r)\right] \; P_l + \sum_{l=1}^{L} \frac{2}{2l+1} U_l^{SO}(r) \; P_l \, \boldsymbol{\ell s} \\ &+ \sum_{l=0}^{L} \sum_{n_c} \hat{U}_{n_c l}^{AREP} \; P_l + \sum_{l=1}^{L} \sum_{n_c} \hat{U}_{n_c l}^{SO} \; P_l \, \boldsymbol{\ell s},\end{aligned} \tag{2}$$

where outercore (and non-local) contributions to scalar-relativistic part $\hat{U}_{n_c l}^{AREP}$ and effective spin-orbit operator $\hat{U}_{n_c l}^{SO}$ are defined via the auxuliary operators $\hat{V}_{n_c lj}$:

$$\begin{aligned}\hat{U}_{n_c l}^{AREP} &= \frac{l+1}{2l+1} \hat{V}_{n_c, l+1/2} + \frac{l}{2l+1} \hat{V}_{n_c, l-1/2} \\ \hat{U}_{n_c l}^{SO} &= \frac{2}{2l+1} \left[\hat{V}_{n_c, l+1/2} - \hat{V}_{n_c, l-1/2}\right]\end{aligned} \tag{3}$$

$$\hat{V}_{n_c lj} = \; (U_{n_c lj} - U_{lj}) \tilde{P}_{n_c lj} + \tilde{P}_{n_c lj} (U_{n_c lj} - U_{lj}) \; - \sum_{n_c'} \tilde{P}_{n_c lj} \left[\frac{U_{n_c lj} + U_{n_c' lj}}{2} - U_{lj}\right] \tilde{P}_{n_c' lj} \tag{4}$$

The first three terms in Eq. (2) correspond to the semi-local part of the GRPP operator. The methods of evaluation of molecular integrals over these terms are well-established and implemented in many other program packages [12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. Current implementation employs the McMurchie-Davidson scheme for type 2 [12] and spin-orbit PP [13] integrals. To avoid numerical instabilities inherent for the recursive analytic scheme [22], radial integrals are evaluated numerically on a grid [23, 15]. For type 1 integrals we use the fully analytic approach [1] based on the classical McMurchie-Davidson scheme [24] for the nuclear attraction integrals and integrals over the $\frac{1}{r^2}$ operator [25, 26] (this algorithm was extended to treat the $\frac{e^{-\zeta r^2}}{r}$ and $\frac{e^{-\zeta r^2}}{r^2}$ operators).

The fourth and fifth terms depending on outercore potentials and projectors onto outercore shells are specific for GRPPs [3, 4]. It was shown that these integrals can be expressed in terms of integrals over radially-local ("type 1") potentials and overlap integrals [1].

3 libgrpp programming interface

3.1 Basis functions: libgrpp_shell_t

Atom-centered Gaussian basis functions are the most widely used in modern molecular electronic structure theory; a detailed discussion can be found in the monograph [27]. Here we will discuss only Cartesian basis sets; transformation to the spherical basis can be easily performed if necessary.

libgrpp programming interface is designed to work with shell pairs of basis functions instead of Cartesian primitives. Each Cartesian shell with angular momentum L consists of $\frac{(L+1)(L+2)}{2}$ Cartesian primitives. Contracted basis function centered at point $\boldsymbol{A}$ is constructed from normalized Cartesian primitive Gaussians with exponential parameters α_i:

$$\phi_A(\boldsymbol{r}) = \sum_i c_i \, N_i \, x_A^{n_A} y_A^{l_A} z_A^{m_A} e^{-\alpha_i (\boldsymbol{r}-\boldsymbol{A})^2}, \tag{5}$$

where $x_A = x - A_x$ (the same for y_A and z_A), c_i stands for the contraction coefficients and the normalization constants are given by

$$N_i = \frac{2\alpha_i}{\pi}^{3/4} (4\alpha_i)^{(n_A+l_A+m_A)/2}. \tag{6}$$

The orbital angular momentum of such a contracted function is formally equal to $L_A = n_A + l_A + m_A$.

To represent these shells in the code the C interface provides special structure libgrpp_shell_t:

```
typedef struct {
    int L;                // angular momentum of the basis function
    int cart_size;        // number of Cartesian primitives
    int *cart_list;       // list of powers (n,l,m) for each Cartesian primitive
    int num_primitives;   // number of Gaussian primitives
    double *coeffs;       // contraction coeff-s (do not include norm factors)
    double *alpha;        // exponential parameters of Gaussians primitives
    double origin[3];     // 3D point at which the shell is centered
} libgrpp_shell_t;
```

It is recommended to use the pair of two special routines to construct and deallocate objects of type libgrpp_shell_t:

```
libgrpp_shell_t *libgrpp_new_shell(
        double *origin,
        int L,
        int num_primitives,
        double *coeffs,
        double *alpha
);

void libgrpp_delete_shell(libgrpp_shell_t *shell);
```

The order of Cartesian primitives inside a shell differs from one quantum chemistry package to another. By default libgrpp provides the ordering adopted in DIRAC [28] (xx, xy, xz, yy, yz, zz, etc).

All libgrpp subroutines used to evaluate molecular integrals store the resulting matrices as one-dimensional arrays of type double. The order of matrix elements is illustrated on Fig. 1.

3.2 Parametrization of pseudopotentials: libgrpp_potential_t

For further use in molecular applications radial parts of GRPP components $U_L(r)$, $\Delta U_l(r)$, $U_l^{SO}(r)$ and $\Delta U_{n_c lj}(r) = U_{n_c lj}(r) - U_{lj}(r)$ (see Eq. (2)) are expressed as linear combinations of radial Gaussian functions,

$$U(r) = \sum_k d_k r^{n_k - 2} e^{-\zeta_k r^2}, \tag{7}$$

	f_{xxx}	f_{xxy}	f_{xxz}	f_{xyy}	f_{xyz}	f_{xzz}	f_{yyy}	f_{yyz}	f_{yzz}	f_{zzz}
d_{xx}	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
d_{xy}	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]	[20]
d_{xz}	[21]	[22]	[23]	[24]	[25]	[26]	[27]	[28]	[29]	[30]
d_{yy}	[31]	[32]	[33]	[34]	[35]	[36]	[37]	[38]	[39]	[40]
d_{yz}	[41]	[42]	[43]	[44]	[45]	[46]	[47]	[48]	[49]	[50]
d_{zz}	[51]	[52]	[53]	[54]	[55]	[56]	[57]	[58]	[59]	[60]

Figure 1: *Molecular integrals calculated by libgrpp for the shell pair are stored in a one-dimensional array (here the special case of the d-f shell pair is shown).*

where r stands for the distance from the point $\boldsymbol{C}$ at which the RPP is centered, $r = |\boldsymbol{r} - \boldsymbol{C}|$. The GRPPs for chemical elements from hydrogen to element 123 were derived from Dirac-Fock(-Breit) atomic calculations in 1995-2022; the parameters n_k, d_k and ζ_k were tabulated and can be found in [29].

The parametrization (7) is represented in `libgrpp` by the structure `libgrpp_potential_t`:

```
typedef struct {
    int L;                // spatial angular momentum of the PP operator
    int J;                // total angular momentum of the PP operator
    int num_primitives;   // number of Gaussian primitives in the PP expansion
    int *powers;          // powers 'n' for each PP primitive
    double *coeffs;       // coefficients of the PP expansion
    double *alpha;        // exponents of the PP expansion
} libgrpp_potential_t;
```

Constructor and destructor for the `libgrpp_potential_t` object are defined as follows:

```
libgrpp_potential_t *libgrpp_new_potential(
    int L,
    int J,
    int num_primitives,
    int *powers,
    double *coeffs,
    double *alpha
);

void libgrpp_delete_potential(libgrpp_potential_t *potential);
```

3.3 Full GRPP object: `libgrpp_grpp_t`

The `libgrpp_grpp_t` structure represents the whole GRPP operator and includes all terms of the expansion (2):

```
typedef struct {
    int n_arep;           // number of semilocal potentials in the scal-rel part
    int n_esop;           // number of potentials in the semilocal SO part
    int n_oc_shells;      // number of outercore shells/potentials
    libgrpp_potential_t *U_L;      // local potential U_L(r)
    libgrpp_potential_t **U_arep;  // list of scal-rel difference potentials
    libgrpp_potential_t **U_esop;  // list of effective SO potentials
    libgrpp_potential_t **U_oc;    // list of outercore potentials U_{n_c,lj}
    libgrpp_shell_t **oc_shells;   // list of outercore shells (for projectors)
} libgrpp_grpp_t;
```

Objects of type `libgrpp_grpp_t` are most conveniently constructed using the function:

```
// constructor of the GRPP object
libgrpp_grpp_t *libgrpp_new_grpp();
```

After the empty GRPP object is created, one can readily bind potentials and outercore shells to it:

```
void libgrpp_grpp_set_local_potential(
    libgrpp_grpp_t *grpp, libgrpp_potential_t *pot
);

void libgrpp_grpp_add_averaged_potential(
    libgrpp_grpp_t *grpp, libgrpp_potential_t *pot
);

void libgrpp_grpp_add_spin_orbit_potential(
    libgrpp_grpp_t *grpp, libgrpp_potential_t *pot
);

// adds the pair (U_nlj, \phi_nlj) to GRPP
void libgrpp_grpp_add_outercore_potential(
    libgrpp_grpp_t *grpp, libgrpp_potential_t *pot, libgrpp_shell_t *oc_shell
);
```

To delete the `libgrpp_grpp_t` object and deallocate memory the destructor is provided:

```
// destructor
void libgrpp_delete_grpp(libgrpp_grpp_t *grpp);
```

Please note that for the `libgrpp_grpp_t` object the semi-local spin-orbit potentials $U_l^{SO}(r)$ are multiplied by $\frac{2}{2l+1}$ *inside* the `libgrpp` subroutines. This factor should not be accounted for during the construction of the `libgrpp_grpp_t` object!

3.4 Scalar-relativistic part: local potential integrals (type 1 integrals)

Integrals over the local potential (type 1 integrals) are defined as:

$$\langle \phi_A | U_L(r_C) | \phi_B \rangle \tag{8}$$

Calculated matrix elements are stored in a one-dimensional array of type `double`, as it was shown on Fig. 1. The array `matrix` must be pre-allocated.

C interface:

```
void libgrpp_type1_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    double *rpp_origin,
    libgrpp_potential_t *potential,
    double *matrix
);
```

Fortran 90 interface:

```
subroutine libgrpp_type1_integrals(                                &
    origin_A, L_A, num_primitives_A, coeffs_A, alpha_A,            &
    origin_B, L_B, num_primitives_B, coeffs_B, alpha_B,            &
    rpp_origin, rpp_nprim, rpp_powers, rpp_coeffs, rpp_alpha,      &
    matrix                                                         &
)
```

```
! shell centered on atom A
real(8)     :: origin_A(*)       ! point at which the basis function is centered
integer(4)  :: L_A               ! angular momentum of the basis function
integer(4)  :: num_primitives_A  ! number of Gaussian primitives
real(8)     :: coeffs_A(*)       ! contraction coefficients
real(8)     :: alpha_A(*)        ! exponents of Gaussian primitives

! shell centered on atom B
real(8)     :: origin_B(*)
integer(4)  :: L_B
integer(4)  :: num_primitives_B
real(8)     :: coeffs_B(*)
real(8)     :: alpha_B(*)

! pseudopotential expansion
real(8)     :: rpp_origin(*) ! point at which the PP is centered
integer(4)  :: rpp_nprim     ! number of Gaussian primitives in the PP expansion
integer(4)  :: rpp_powers(*) ! powers 'n' for each PP primitive
real(8)     :: rpp_coeffs(*) ! coefficients of the PP expansion
real(8)     :: rpp_alpha(*)  ! exponents of the PP expansion

! output: PP matrix elements
real(8)     :: matrix(*)
```

3.5 Scalar-relativistic part: integrals with angular projectors (type 2 integrals)

Integrals with angular projections (type 2 integrals) are defined as:

$$\langle \phi_A | \Delta U_l(r_C) P_l | \phi_B \rangle \,. \tag{9}$$

The `matrix` array used to return matrix elements must be pre-allocated.

C interface:

```
void libgrpp_type2_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    double *rpp_origin,
    libgrpp_potential_t *potential,
    double *matrix
);
```

Fortran 90 interface:

```
subroutine libgrpp_type2_integrals(                                    &
    origin_A, L_A, num_primitives_A, coeffs_A, alpha_A,                &
    origin_B, L_B, num_primitives_B, coeffs_B, alpha_B,                &
    rpp_origin, rpp_L, rpp_nprim, rpp_powers, rpp_coeffs, rpp_alpha, &
    matrix                                                             &
)

! shell centered on atom A
real(8)     :: origin_A(*)       ! point at which the basis function is centered
integer(4)  :: L_A               ! angular momentum of the basis function
integer(4)  :: num_primitives_A  ! number of Gaussian primitives
real(8)     :: coeffs_A(*)       ! contraction coefficients
real(8)     :: alpha_A(*)        ! exponents of Gaussian primitives
```

```
! shell centered on atom B
real(8)     :: origin_B(*)
integer(4)  :: L_B
integer(4)  :: num_primitives_B
real(8)     :: coeffs_B(*)
real(8)     :: alpha_B(*)

! pseudopotential expansion
real(8)     :: rpp_origin(*) ! point at which the PP is centered
integer(4)  :: rpp_L         ! angular momentum of the projector P_l
integer(4)  :: rpp_nprim     ! number of Gaussian primitives in the PP expansion
integer(4)  :: rpp_powers(*) ! powers 'n' for each PP primitive
real(8)     :: rpp_coeffs(*) ! coefficients of the PP expansion
real(8)     :: rpp_alpha(*)  ! exponents of the PP expansion

! output: PP matrix elements
real(8)     :: matrix(*)
```

3.6 Effective spin-orbit interaction integrals

Integrals over the effective spin-orbit interaction operator are defined as:

$$\langle \phi_A | U_l^{SO}(r_C) P_l l_\eta | \phi_B \rangle \,, \qquad \eta = x, y, z. \tag{10}$$

The `so_x_matrix`, `so_y_matrix` and `so_z_matrix` arrays used to return matrix elements must be pre-allocated. Note that the spin-orbit part in (2) also includes multiplication by the $\boldsymbol{s} = \frac{1}{2}\boldsymbol{\sigma}$ operator. The factor $\frac{1}{2}$ is not accounted for by the `libgrpp` library. Multiplication by 2×2 Pauli matrices $\boldsymbol{\sigma}$ should be performed at the stage of the construction of the Hamiltonian matrix (see, for example, [30] for the detailed discussion).

C interface:

```
void libgrpp_spin_orbit_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    double *rpp_origin,
    libgrpp_potential_t *potential,
    double *so_x_matrix,
    double *so_y_matrix,
    double *so_z_matrix
);
```

Fortran 90 interface:

```
subroutine libgrpp_spin_orbit_integrals(                                  &
    origin_A, L_A, num_primitives_A, coeffs_A, alpha_A,                   &
    origin_B, L_B, num_primitives_B, coeffs_B, alpha_B,                   &
    rpp_origin, rpp_L, rpp_nprim, rpp_powers, rpp_coeffs, rpp_alpha,      &
    so_x_matrix, so_y_matrix, so_z_matrix                                 &
)

! shell centered on atom A
real(8)     :: origin_A(*)       ! point at which the basis function is centered
integer(4)  :: L_A               ! angular momentum of the basis function
integer(4)  :: num_primitives_A  ! number of Gaussian primitives
real(8)     :: coeffs_A(*)       ! contraction coefficients
real(8)     :: alpha_A(*)        ! exponents of Gaussian primitives

! shell centered on atom B
```

```
real(8)     :: origin_B(*)
integer(4)  :: L_B
integer(4)  :: num_primitives_B
real(8)     :: coeffs_B(*)
real(8)     :: alpha_B(*)

! pseudopotential expansion
real(8)     :: rpp_origin(*) ! point at which the PP is centered
integer(4)  :: rpp_L         ! angular momentum of the projector P_l
integer(4)  :: rpp_nprim     ! number of Gaussian primitives in the PP expansion
integer(4)  :: rpp_powers(*) ! powers 'n' for each PP primitive
real(8)     :: rpp_coef(*)   ! coefficients of the PP expansion
real(8)     :: rpp_alpha(*)  ! exponents of the PP expansion

! output: PP matrix elements for the X, Y, Z components
real(8)     :: so_x_matrix(*)
real(8)     :: so_y_matrix(*)
real(8)     :: so_z_matrix(*)
```

3.7 GRPP-specific non-local part: integrals with projectors onto outercore shells

Non-local GRPP-specific contributions to scalar-relativistic integrals

$$\langle\phi_A|\sum_{l=0}^{L}\sum_{n_c}\hat{U}_{n_c l}^{AREP}\,P_l|\phi_B\rangle \tag{11}$$

and effective spin-orbit interaction

$$\langle\phi_A|\sum_{l=1}^{L}\sum_{n_c}\hat{U}_{n_c l}^{SO}\,P_l\,l_\eta|\phi_B\rangle\,,\qquad \eta=x,y,z \tag{12}$$

are calculated in `libgrpp` simultaneously. To construct these integrals via the formulas (3) – (4) one has to pass the list of outercore (difference) potentials $\Delta U_{n_c lj}(r)=U_{n_c lj}(r)-U_{lj}(r)$ and outercore pseudospinors $\tilde{\phi}_{n_c lj}$ to the integrator. Note the presence of the factor $\frac{2}{2l+1}$ in the formula (3); thus outercore potentials $\Delta U_{n_c lj}(r)$ are used directly and must not be pre-multiplied by this factor outside of `libgrpp`. Outercore shells $\tilde{\phi}_{n_c lj}$ are centered at the same point $\boldsymbol{C}$ as the GRPP operator.

C interface:

```
void libgrpp_outercore_potential_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    double *rpp_origin,
    int num_oc_shells,
    libgrpp_potential_t **oc_potentials,
    libgrpp_shell_t **oc_shells,
    double *arep_matrix,
    double *so_x_matrix,
    double *so_y_matrix,
    double *so_z_matrix
);
```

Fortran 90 interface:

```
subroutine libgrpp_outercore_potential_integrals(        &
    origin_A, L_A, num_primitives_A, coeffs_A, alpha_A,  &
```

```
    origin_B, L_B, num_primitives_B, coeffs_B, alpha_B,  &
    rpp_origin, num_oc_shells, oc_shells_L, oc_shells_J, &
    rpp_nprim, rpp_powers, rpp_coeffs, rpp_alpha,        &
    oc_shells_nprim, oc_shells_coeffs, oc_shells_alpha,  &
    arep_matrix, so_x_matrix, so_y_matrix, so_z_matrix   &
)

! shell centered on atom A
real(8)    :: origin_A(*)       ! point at which the basis function is centered
integer(4) :: L_A               ! angular momentum of the basis function
integer(4) :: num_primitives_A  ! number of Gaussian primitives
real(8)    :: coeffs_A(*)       ! contraction coefficients
real(8)    :: alpha_A(*)        ! exponents of Gaussian primitives

! shell centered on atom B
real(8)    :: origin_B(*)
integer(4) :: L_B
integer(4) :: num_primitives_B
real(8)    :: coeffs_B(*)
real(8)    :: alpha_B(*)

! pseudopotential expansion
real(8)    :: rpp_origin(*)    ! point at which the GRPP is centered
integer(4) :: num_oc_shells    ! total number of outercore potentials U_{n_c,lj}
integer(4) :: oc_shells_L(:)   ! angular momenta L of each outercore potential
integer(4) :: oc_shells_J(:)   ! total momenta J of each outercore potential
integer(4) :: rpp_nprim(:)     ! number of primitives for each OC potential
integer(4) :: rpp_powers(:, :) ! array of powers 'n' for each OC potential
real(8)    :: rpp_coeffs(:, :) ! expansion coeff-s for each OC potential
real(8)    :: rpp_alpha(:, :)  ! exponential parameters for each OC potential

! outercore shells used to construct projectors
integer(4) :: oc_shells_nprim(:)     ! number of primitives for each OC shell
real(8)    :: oc_shells_coeffs(:, :) ! contraction coeff-s for each OC shell
real(8)    :: oc_shells_alpha(:, :)  ! exponents of primitives for each OC shell

! output: PP matrix elements
! scalar-relativistic and spin-orbit parts are constructed simultaneously
real(8)    :: arep_matrix(*)
real(8)    :: so_x_matrix(*)
real(8)    :: so_y_matrix(*)
real(8)    :: so_z_matrix(*)
```

3.8 Integrals over the full GRPP operator

As it was described in Sect. 3.3, within the C interface the GRPP operator can be represented as an object of type `libgrpp_grpp_t`. Such an object can be passed to the routine `libgrpp_full_grpp_integrals()`:

C interface:

```
void libgrpp_full_grpp_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    libgrpp_grpp_t *grpp_operator,
    double *grpp_origin,
    double *arep_matrix,
```

```
    double *so_x_matrix,
    double *so_y_matrix,
    double *so_z_matrix
);
```

In this case both the scalar-relativistic parts and effective spin-orbit operator are integrated simultaneously.

3.9 Analytic gradients of GRPP integrals

`libgrpp` provides the possibility of calculation of analytic first derivatives of GRPP integrals with respect to nuclear coordinates. The algorithm was described in details in [1]. It is based on the property of translation invariance of GRPP integrals and is exactly the same as in the case of ordinary semi-local pseudopotentials [31, 32, 33, 34, 35]. Currently analytic derivatives of GRPP integrals can be calculated only via the C interface. The corresponding routine employs the `libgrpp_grpp_t` object:

```
void libgrpp_full_grpp_integrals_gradient(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    libgrpp_grpp_t *grpp_operator,
    double *grpp_origin,
    double *point_3d,
    double **grad_arep,
    double **grad_so_x,
    double **grad_so_y,
    double **grad_so_z
);
```

Differentiation is performed with respect to the 3D point `point_3d`. Gradients with respect to each of the three spatial coordinates x, y, z are stored in the `grad_*` arrays. For example, `grad_arep` contains pointers to three arrays:

$$\frac{\partial}{\partial P_x}\langle\phi_A|\hat{U}_C^{GRPP}|\phi_B\rangle\,,\qquad \frac{\partial}{\partial P_y}\langle\phi_A|\hat{U}_C^{GRPP}|\phi_B\rangle\,,\qquad \frac{\partial}{\partial P_z}\langle\phi_A|\hat{U}_C^{GRPP}|\phi_B\rangle\,, \tag{13}$$

where $\mathbf{P} = \{P_x, P_y, P_z\}$ stands for the point with respect to the coordinates of which differentiation is performed (the same for the SO part). Overall 12 arrays of partial derivatives of matrix elements are returned by the routine. Note that the `grad_*` arrays must be pre-allocated before the invocation of the subroutine.

4 Other one-electron integrals

4.1 Overlap integrals

$$\langle\phi_A|\phi_B\rangle \tag{14}$$

C interface:

```
void libgrpp_overlap_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    double *overlap_matrix
);
```

Fortran 90 interface:

```
subroutine libgrpp_overlap_integrals(                    &
    origin_A, L_A, num_primitives_A, coeffs_A, alpha_A,  &
    origin_B, L_B, num_primitives_B, coeffs_B, alpha_B,  &
    overlap_matrix                                       &
)

! shell centered on atom A
real(8)    :: origin_A(*)       ! point at which the basis function is centered
integer(4) :: L_A               ! angular momentum of the basis function
integer(4) :: num_primitives_A  ! number of Gaussian primitives
real(8)    :: coeffs_A(*)       ! contraction coefficients
real(8)    :: alpha_A(*)        ! exponents of Gaussian primitives

! shell centered on atom B
real(8)    :: origin_B(*)
integer(4) :: L_B
integer(4) :: num_primitives_B
real(8)    :: coeffs_B(*)
real(8)    :: alpha_B(*)

! output
real(8)    :: overlap_matrix(*)
```

4.2 Kinetic-energy integrals

$$\langle \phi_A | -\frac{1}{2}\Delta | \phi_B \rangle \tag{15}$$

C interface:

```
void libgrpp_kinetic_energy_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    double *kinetic_matrix
);
```

Fortran 90 interface:

```
subroutine libgrpp_kinetic_energy_integrals(             &
    origin_A, L_A, num_primitives_A, coeffs_A, alpha_A,  &
    origin_B, L_B, num_primitives_B, coeffs_B, alpha_B,  &
    kinetic_matrix                                       &
)

! shell centered on atom A
real(8)    :: origin_A(*)       ! point at which the basis function is centered
integer(4) :: L_A               ! angular momentum of the basis function
integer(4) :: num_primitives_A  ! number of Gaussian primitives
real(8)    :: coeffs_A(*)       ! contraction coefficients
real(8)    :: alpha_A(*)        ! exponents of Gaussian primitives

! shell centered on atom B
real(8)    :: origin_B(*)
integer(4) :: L_B
integer(4) :: num_primitives_B
real(8)    :: coeffs_B(*)
```

```
real(8)     :: alpha_B(*)

! output
real(8)     :: kinetic_matrix(*)
```

4.3 Momentum integrals

$$\langle \phi_A | -i\nabla | \phi_B \rangle = -i \left(\langle \phi_A | \frac{\partial}{\partial x} | \phi_B \rangle , \langle \phi_A | \frac{\partial}{\partial y} | \phi_B \rangle , \langle \phi_A | \frac{\partial}{\partial z} | \phi_B \rangle \right) \tag{16}$$

Subroutines return imaginary parts on momentum integrals, the “minus” sign is accounted for.
C interface:

```
void libgrpp_momentum_integrals(
    libgrpp_shell_t *shell_A,
    libgrpp_shell_t *shell_B,
    double *px_matrix,
    double *py_matrix,
    double *pz_matrix
);
```

Fortran 90 interface:

```
subroutine libgrpp_momentum_integrals(                    &
    origin_A, L_A, num_primitives_A, coeffs_A, alpha_A,   &
    origin_B, L_B, num_primitives_B, coeffs_B, alpha_B,   &
    px_matrix, py_matrix, pz_matrix                       &
)

! shell centered on atom A
real(8)     :: origin_A(*)       ! point at which the basis function is centered
integer(4)  :: L_A               ! angular momentum of the basis function
integer(4)  :: num_primitives_A  ! number of Gaussian primitives
real(8)     :: coeffs_A(*)       ! contraction coefficients
real(8)     :: alpha_A(*)        ! exponents of Gaussian primitives

! shell centered on atom B
real(8)     :: origin_B(*)
integer(4)  :: L_B
integer(4)  :: num_primitives_B
real(8)     :: coeffs_B(*)
real(8)     :: alpha_B(*)

! output
real(8)     :: px_matrix(*)
real(8)     :: py_matrix(*)
real(8)     :: pz_matrix(*)
```

5 Programming examples

`libgrpp` provides two simple demonstration programs, `test_libgrpp_c` and `test_libgrpp_f90`. These programs illustrate how to call the `libgrpp` routines. Here we present the most important code samples in both C and Fortran languages. Note that `libgrpp` should not be initialized before the use; this initialization is implicit and is performed the first time when the library is accessed from the client code.

5.1 Example: C

The following listing shows how to construct integrals over the full GRPP operator for the given list of several basis shells.

```c
// ... some other useful headers ...
#include "../libgrpp/libgrpp.h"

#define MAX_BUF 10000

/**
 * evaluates matrix elements of the GRPP operator
 */
void evaluate_grpp_integrals(
    int num_shells, libgrpp_shell_t **shell_list,
    molecule_t *molecule, libgrpp_grpp_t **grpp_list,
    double *arep_matrix,
    double *so_x_matrix,
    double *so_y_matrix,
    double *so_z_matrix
)
{
    // buffers used to store matrix elements for a shell pair
    double buf_arep[MAX_BUF];
    double buf_spin_orbit[3][MAX_BUF];

    int dim = calculate_basis_dim(shell_list, num_shells);

    // square matrices used to store matrix elements for the whole molecule
    memset(arep_matrix, 0, sizeof(double) * dim * dim);
    memset(so_x_matrix, 0, sizeof(double) * dim * dim);
    memset(so_y_matrix, 0, sizeof(double) * dim * dim);
    memset(so_z_matrix, 0, sizeof(double) * dim * dim);

    // loop over bra shells
    int ioffset = 0;
    for (int ishell = 0; ishell < num_shells; ishell++) {

        libgrpp_shell_t *bra = shell_list[ishell];
        int bra_dim = libgrpp_get_shell_size(bra);

        // loop over ket shells
        int joffset = 0;
        for (int jshell = 0; jshell < num_shells; jshell++) {

            libgrpp_shell_t *ket = shell_list[jshell];
            int ket_dim = libgrpp_get_shell_size(ket);
```

```
            // loop over atoms; GRPPs are provided only for the part of atoms
            for (int iatom = 0; iatom < molecule->n_atoms; iatom++) {
                int z = molecule->charges[iatom];
                libgrpp_grpp_t *grpp = grpp_list[z];
                if (grpp == NULL) {
                    continue;
                }

                double ecp_origin[3];
                ecp_origin[0] = molecule->coord_x[iatom];
                ecp_origin[1] = molecule->coord_y[iatom];
                ecp_origin[2] = molecule->coord_z[iatom];

                evaluate_grpp_integrals_shell_pair(
                    bra, ket, grpp, ecp_origin, buf_arep,
                    buf_spin_orbit[0], buf_spin_orbit[1], buf_spin_orbit[2]
                );

                // the rectangular block of matrix elements is added to the
                // resulting square matrix
                add_block_to_matrix(dim, dim, arep_matrix, bra_dim, ket_dim,
                    buf_arep, ioffset, joffset, 1.0);
                add_block_to_matrix(dim, dim, so_x_matrix, bra_dim, ket_dim,
                    buf_spin_orbit[0], ioffset, joffset, 1.0);
                add_block_to_matrix(dim, dim, so_y_matrix, bra_dim, ket_dim,
                    buf_spin_orbit[1], ioffset, joffset, 1.0);
                add_block_to_matrix(dim, dim, so_z_matrix, bra_dim, ket_dim,
                    buf_spin_orbit[2], ioffset, joffset, 1.0);
            }

            joffset += ket_dim;
        }

        ioffset += bra_dim;
    }
}
```

The subroutine `evaluate_grpp_integrals_shell_pair()` evaluating GRPP matrix elements for the pair of shells is given by the following code:

```
void evaluate_grpp_integrals_shell_pair(
        libgrpp_shell_t *shell_A,
        libgrpp_shell_t *shell_B,
        libgrpp_grpp_t *grpp_operator,
        double *grpp_origin,
        double *arep_matrix,
        double *so_x_matrix,
        double *so_y_matrix,
        double *so_z_matrix
)
{
    size_t size = shell_A->cart_size * shell_B->cart_size;
    double *buf_arep = (double *) calloc(size, sizeof(double));
    double *buf_so_x = (double *) calloc(size, sizeof(double));
    double *buf_so_y = (double *) calloc(size, sizeof(double));
    double *buf_so_z = (double *) calloc(size, sizeof(double));
```

```
    memset(arep_matrix, 0, sizeof(double) * size);
    memset(so_x_matrix, 0, sizeof(double) * size);
    memset(so_y_matrix, 0, sizeof(double) * size);
    memset(so_z_matrix, 0, sizeof(double) * size);

    /*
     * radially-local ("type-1") integrals
     */
    libgrpp_type1_integrals(
        shell_A, shell_B, grpp_origin, grpp_operator->U_L, buf_arep
    );
    update_vector(size, arep_matrix, 1.0, buf_arep);

    /*
     * semilocal AREP ("type-2") integrals
     */
    for (int L = 0; L < grpp_operator->n_arep; L++) {
        libgrpp_type2_integrals(
            shell_A, shell_B, grpp_origin, grpp_operator->U_arep[L], buf_arep
        );
        update_vector(size, arep_matrix, 1.0, buf_arep);
    }

    /*
     * semilocal SO ("type-3") integrals
     */
    for (int L = 1; L < grpp_operator->n_esop; L++) {
        libgrpp_spin_orbit_integrals(
            shell_A, shell_B, grpp_origin, grpp_operator->U_esop[L],
            buf_so_x, buf_so_y, buf_so_z
        );

        update_vector(size, so_x_matrix, 2.0 / (2 * L + 1), buf_so_x);
        update_vector(size, so_y_matrix, 2.0 / (2 * L + 1), buf_so_y);
        update_vector(size, so_z_matrix, 2.0 / (2 * L + 1), buf_so_z);
    }

    /*
     * integrals over outercore non-local potentials,
     * the part specific for GRPP.
     *
     * note that proper pre-factors for the SO part are calculated inside
     * the libgrpp_outercore_potential_integrals() procedure.
     */
    libgrpp_outercore_potential_integrals(
        shell_A, shell_B, grpp_origin, grpp_operator->n_oc_shells,
        grpp_operator->U_oc, grpp_operator->oc_shells,
        buf_arep, buf_so_x, buf_so_y, buf_so_z
    );

    update_vector(size, arep_matrix, 1.0, buf_arep);
    update_vector(size, so_x_matrix, 1.0, buf_so_x);
    update_vector(size, so_y_matrix, 1.0, buf_so_y);
    update_vector(size, so_z_matrix, 1.0, buf_so_z);
```

```
    /*
     * cleanup
     */
    free(buf_arep);
    free(buf_so_x);
    free(buf_so_y);
    free(buf_so_z);
}
```

5.2 Example: Fortran 90

The code in Fortran 90 is more cumbersome, since the object-oriented features of modern Fortran were abandoned in order to preserve the interoperability with codes written in Fortran 77 (widely used in quantum chemistry software). The Fortran 90 interface of `libgrpp` is purely procedural.

The following code snippets are taken from the source code of the `test_ligrpp_f90` sample program (see file `evalints.f90`). Consider, for example, the loop over shell pairs with the invocation of the subroutine `libgrpp_spin_orbit_integrals()`:

```
! initialization of the resulting matrices
arep_matrix = 0.0d0
so_matrices = 0.0d0

ioffs = 0
ishell = 0

! loop over shell pairs
do iatom1 = 1, natoms
  z1 = charges(iatom1) ! atomic charge is used to get the basis set
  do iblock1 = 1, num_blocks(z1)
    do ifun1 = 1, block_num_contr(z1, iblock1)
      call generate_cartesians(iblock1 - 1, cart_list_1, ncart1)
      ishell = ishell + 1

      joffs = 0
      jshell = 0
      do iatom2 = 1, natoms
        z2 = charges(iatom2)
        do iblock2 = 1, num_blocks(z2)
          do ifun2 = 1, block_num_contr(z2, iblock2)
            call generate_cartesians(iblock2 - 1, cart_list_2, ncart2)
            jshell = jshell + 1

            L_A = iblock1 - 1
            L_B = iblock2 - 1
            nprim_A = block_num_prim(z1, iblock1)
            nprim_B = block_num_prim(z2, iblock2)

            ! sum over atoms with pseudopotentials
            do ic = 1, natoms
              zc = charges(ic)
              if (n_arep(zc) == 0) then   ! no RPP for this atom => skip it
                  cycle
              end if

              ! radially-local part (type-1 integrals)
```

```
          ! ... the code ...

          ! semilocal averaged part (type-2 integrals)
          ! ... the code ...

          ! semilocal spin-orbit part

          do iarep = 2, n_arep(zc)
            L = iarep - 1
            call libgrpp_spin_orbit_integrals( &
              coord(iatom1, :), L_A, nprim_A,  &            ! basis shell 1
              coeffs(z1, iblock1, ifun1, :), exponents(z1, iblock1, :), &
              coord(iatom2, :), L_B, nprim_B,  &            ! basis shell 2
              coeffs(z2, iblock2, ifun2, :), exponents(z2, iblock2, :), &
              coord(ic, :), L, ecp_num_prim(zc, iarep), &   ! potential
              ecp_powers(zc, iarep, :),  esop(zc, iarep, :), &
              ecp_alpha(zc, iarep, :), buf_x, buf_y, buf_z   &
            )

            ! contributions to the SO interaction must be scaled
            ! since these factors are not accounted for in the input
            ! file format for GRPPs
            buf_x = buf_x * 2.0 / (2.0 * L + 1)
            buf_y = buf_y * 2.0 / (2.0 * L + 1)
            buf_z = buf_z * 2.0 / (2.0 * L + 1)
            call update_matrix_part(so_matrices(1,:,:), ioffs, joffs, &
              buf_x, ncart1, ncart2)
            call update_matrix_part(so_matrices(2,:,:), ioffs, joffs, &
              buf_y, ncart1, ncart2)
            call update_matrix_part(so_matrices(3,:,:), ioffs, joffs, &
              buf_z, ncart1, ncart2)
          end do

          ! non-local part
          ! ... the code ...

          ! add the contribution of the given shell pair
          ! to the overall square RPP matrices
          call update_matrix_part(so_matrices(1,:,:), ioffs, joffs, &
            buf_x, ncart1, ncart2)
          call update_matrix_part(so_matrices(2,:,:), ioffs, joffs, &
            buf_y, ncart1, ncart2)
          call update_matrix_part(so_matrices(3,:,:), ioffs, joffs, &
            buf_z, ncart1, ncart2)

        end do

        joffs = joffs + ncart2

      end do
    end do
  end do

  ioffs = ioffs + ncart1
```

```
    end do
  end do
end do
```

This code is rather straightforward. It relies on parameters of RPPs and basis sets available in the global namespace (using the mechanism of Fortran 90 modules). Parameters of the basis sets are stored in several arrays defined in `basis.f90`:

```
integer, parameter :: MAXL = 10
integer, parameter :: MAX_CNTRCT_LEN = 30

! basis set data
integer :: num_blocks(N_ELEMENTS)
integer :: block_num_prim(N_ELEMENTS, MAXL)
integer :: block_num_contr(N_ELEMENTS, MAXL)
real(8) :: exponents(N_ELEMENTS, MAXL, MAX_CNTRCT_LEN)
real(8) :: coeffs(N_ELEMENTS, MAXL, MAX_CNTRCT_LEN, MAX_CNTRCT_LEN)
```

Quite analogous structures are defined in `rpp.f90` to store parameters of (G)RPPs:

```
integer, parameter :: ECP_MAXL = 10
integer, parameter :: ECP_MAX_CNTRCT_LEN = 50
integer, parameter :: ECP_MAX_OC_SHELLS = 20

! pseudopotential data
! number of partial-wave potentials for each element
integer :: n_arep(N_ELEMENTS)
integer :: n_esop(N_ELEMENTS)
integer :: n_oc_shells(N_ELEMENTS, ECP_MAXL)
integer :: ecp_num_prim(N_ELEMENTS, ECP_MAXL)

! parameters of RPP Gaussian expansions
real(8) :: ecp_alpha(N_ELEMENTS, ECP_MAXL, ECP_MAX_CNTRCT_LEN)
integer :: ecp_powers(N_ELEMENTS, ECP_MAXL, ECP_MAX_CNTRCT_LEN)
real(8) :: arep(N_ELEMENTS, ECP_MAXL, ECP_MAX_CNTRCT_LEN)
real(8) :: esop(N_ELEMENTS, ECP_MAXL, ECP_MAX_CNTRCT_LEN)
real(8) :: ocpot(N_ELEMENTS, ECP_MAXL, ECP_MAX_OC_SHELLS, ECP_MAX_CNTRCT_LEN)

! outercore basis set used to construct outercore projectors
! (GRPP-specific part)
integer :: ocbas_num_prim(N_ELEMENTS, ECP_MAXL)
integer :: ocbas_num_contr(N_ELEMENTS, ECP_MAXL)
real(8) :: ocbas_alpha(N_ELEMENTS, ECP_MAXL, ECP_MAX_CNTRCT_LEN)
real(8) :: ocbas_coeffs(N_ELEMENTS, ECP_MAXL, ECP_MAX_CNTRCT_LEN, &
  ECP_MAX_CNTRCT_LEN)
```

Of course, basis sets and pseudopotentials can be stored in any other way, since `libgrpp` subroutines expect only pointers to arrays containing these data.

References

[1] A. V. Oleynichenko, A. Zaitsevskii, N. S. Mosyagin, A. N. Petrov, E. Eliav, and A. V. Titov. LIBGRPP: A library for the evaluation of molecular integrals of the generalized relativistic pseudopotential operator over Gaussian functions. *Symmetry*, 15(1):197, 2022.

[2] A. V. Titov, A. O. Mitrushenkov, and I. I. Tupitsyn. Effective core potential for pseudo-orbitals with nodes. *Chem. Phys. Lett.*, 185:330–334, 1991.

[3] N. S. Mosyagin, A. V. Titov, and Z. Latajka. Generalized relativistic effective core potential: Gaussian expansions of potentials and pseudospinors for atoms Hg through Rn. *Int. J. Quantum Chem.*, 63(6):1107–1122, 1997.

[4] A. V. Titov and N. S. Mosyagin. Generalized relativistic effective core potential: Theoretical grounds. *Int. J. Quantum Chem.*, 71(5):359–401, 1999.

[5] Y. V. Lomachuk, D. A. Maltsev, N. S. Mosyagin, L. V. Skripnikov, R. V. Bogdanov, and A. V. Titov. Compound-tunable embedding potential: which oxidation state of uranium and thorium as point defects in xenotime is favorable? *Phys. Chem. Chem. Phys.*, 22(32):17922–17931, 2020.

[6] D. A. Maltsev, Yu. V. Lomachuk, V. M. Shakhova, N. S. Mosyagin, L. V. Skripnikov, and A. V. Titov. Compound-tunable embedding potential method and its application to calcium niobate crystal $CaNb_2O_6$ with point defects containing tantalum and uranium. *Phys. Rev. B*, 103:205105, 2021.

[7] V. M. Shakhova, D. A. Maltsev, Y. V. Lomachuk, N. S. Mosyagin, L. V. Skripnikov, and A. V. Titov. Compound-tunable embedding potential method: analysis of pseudopotentials for Yb in YbF_2, YbF_3, $YbCl_2$ and $YbCl_3$ crystals. *Phys. Chem. Chem. Phys.*, 24:19333–19345, 2022.

[8] A. N. Petrov, N. S. Mosyagin, A. V. Titov, and I. I. Tupitsyn. Accounting for the Breit interaction in relativistic effective core potential calculations of actinides. *J. Phys. B*, 37(23):4621–4637, 2004.

[9] N. S. Mosyagin, A. N. Petrov, A. V. Titov, and I. I. Tupitsyn. Generalized RECP accounting for Breit effects: uranium, plutonium and superheavy elements 112, 113, 114. In *Recent Advances in the Theory of Chemical and Physical Systems*, volume 15, pages 229–251. Kluwer Academic Publishers, 2006.

[10] V. M. Shabaev, I. I. Tupitsyn, and V. A. Yerokhin. Model operator approach to the Lamb shift calculations in relativistic many-electron atoms. *Phys. Rev. A*, 88:012513, Jul 2013.

[11] A. Zaitsevskii, N. S. Mosyagin, A. V. Oleynichenko, and E. Eliav. Generalized relativistic small-core pseudopotentials accounting for quantum electrodynamic effects: Construction and pilot applications. *Int. J. Quantum Chem.*, 123:e27077, 2022.

[12] L. E. McMurchie and E. R. Davidson. Calculation of integrals over ab initio pseudopotentials. *J. Comput. Phys.*, 44(2):289–301, 1981.

[13] R. M. Pitzer and N. W. Winter. Spin-orbit (core) and core potential integrals. *Int. J. Quantum Chem.*, 40(6):773–780, 1991.

[14] A. V. Mitin and C. van Wüllen. Two-component relativistic density-functional calculations of the dimers of the halogens from bromine through element 117 using effective core potential and all-electron methods. *J. Chem. Phys.*, 124(6):064305, 2006.

[15] R. Flores-Moreno, R. J. Alvarez-Mendez, A. Vela, and A. M. Köster. Half-numerical evaluation of pseudopotential integrals. *J. Comput. Chem.*, 27(9):1009–1019, 2006.

[16] M. Dolg and X. Cao. Relativistic pseudopotentials: their development and scope of applications. *Chem. Rev.*, 112:403–480, 2012.

[17] Y.-C. Park and Y.-S. Lee. Two-component spin-orbit effective core potential calculations with an all-electron relativistic program DIRAC. *Bull. Korean Chem. Soc.*, 33:803–808, 2012.

[18] R. A. Shaw and J. G. Hill. Prescreening and efficiency in the evaluation of integrals over ab initio effective core potentials. *J. Chem. Phys.*, 147(7):074108, 2017.

[19] S. C. McKenzie, E. Epifanovsky, G. M. J. Barca, A. T. B. Gilbert, and P. M. W. Gill. Efficient method for calculating effective core potential integrals. *J. Phys. Chem. A*, 122(11):3066–3075, 2018.

[20] S. C. McKenzie. *Efficient computation of integrals in modern correlated methods*. PhD thesis, Faculty of Science, University of Sydney, 2020.

[21] R. A. Shaw and J. G. Hill. libecpint: A C++ library for the efficient evaluation of integrals over effective core potentials. *J. Open Source Softw.*, 6(60):3039.

[22] C. van Wüllen. Numerical instabilities in the computation of pseudopotential matrix elements. *J. Comput. Chem.*, 27(2):135–141, 2005.

[23] C.-K. Skylaris, L. Gagliardi, N. C. Handy, A. G. Ioannou, S. Spencer, A. Willetts, and A. M. Simper. An efficient method for calculating effective core potential integrals which involve projection operators. *Chem. Phys. Lett.*, 296(5):445–451, 1998.

[24] L. E. McMurchie and E. R. Davidson. One- and two-electron integrals over Cartesian Gaussian functions. *J. Comput. Phys.*, 26(2):218–231, 1978.

[25] J. O. Jensen, A. H. Carrieri, C. P. Vlahacos, D. Zeroka, H. F. Hameka, and C. N. Merrow. Evaluation of one-electron integrals for arbitrary operators $V(r)$ over Cartesian Gaussians: Application to inverse-square distance and Yukawa operators. *J. Comput. Chem.*, 14(8):986–994, 1993.

[26] B. Gao, A. J. Thorvaldsen, and K. Ruud. GEN1INT: A unified procedure for the evaluation of one-electron integrals over Gaussian basis functions and their geometric derivatives. *Int. J. Quantum Chem.*, 111(4):858–872, 2011.

[27] T. Helgaker, P. Jørgensen, and J. Olsen. *Molecular Electronic-Structure Theory*. Wiley, 2000.

[28] T. Saue, R. Bast, A. S. P. Gomes, H. J. Aa. Jensen, L. Visscher, I. A. Aucar, R. Di Remigio, K. G. Dyall, E. Eliav, E. Fasshauer, T. Fleig, L. Halbert, E. D. Hedegård, B. Helmich-Paris, M. Iliaš, C. R. Jacob, S. Knecht, J. K. Laerdahl, M. L. Vidal, M. K. Nayak, M. Olejniczak, J. M. H. Olsen, M. Pernpointner, B. Senjean, A. Shee, A. Sunaga, and J. N. P. van Stralen. The DIRAC code for relativistic molecular calculations. *J. Chem. Phys.*, 152(20):204104, 2020.

[29] Generalized relativistic pseudopotentials (see http://qchem.pnpi.spb.ru/recp).

[30] C. van Wüllen. A quasirelativistic two-component density functional and Hartree-Fock program. *Z. Phys. Chem.*, 224(3-4):413–426, 2010.

[31] J. Breidung, W. Thiel, and A. Komornicki. Analytical second derivatives for effective core potentials. *Chem. Phys. Lett.*, 153(1):76–81, 1988.

[32] T. V. Russo, R. L. Martin, P. J. Hay, and A. K. Rappe. Vibrational frequencies of transition metal chloride and oxo compounds using effective core potential analytic second derivatives. *J. Chem. Phys.*, 102(23):9315–9321, 1995.

[33] Q. Cui, D. G. Musaev, M. Svensson, and K. Morokuma. Analytical second derivatives for effective core potential. Application to transition structures of $Cp_2Ru_2(\mu\text{-}H)_4$ and to the mechanism of reaction Cu + CH_2N_2. *J. Phys. Chem.*, 100(26):10936–10944, 1996.

[34] B. M. Bode and M. S. Gordon. Fast computation of analytical second derivatives with effective core potentials: Application to Si_8C_{12}, Ge_8C_{12}, and Sn_8C_{12}. *J. Chem. Phys.*, 111(19):8778–8784, 1999.

[35] C. Song, L.-P. Wang, T. Sachse, J. Preiss, M. Presselt, and T. J. Martinez. Efficient implementation of effective core potential integrals and gradients on graphical processing units. *J. Chem. Phys.*, 143(1):014114, 2015.