Contents

1 Mandate

- summary
- audience
  - Developers in quantum chemistry
  - Theory people looking for quick implementation of ideas
  - ChemTools
- features
- software
  - Pure Python
  - Multiplatform (Windows, MacOS, Linux)
  - NumPy and SciPy are the only dependencies
  - Parallizeable (to the extend that numpy is parallizeable)
  - Open-source

2 Gaussian Primitive

A Cartesian Gaussian primitive is

$$g_i(\mathbf{r}|\mathbf{R}_A, \mathbf{a}) = N(\alpha_i, \mathbf{a})(x - X_A)^{a_x}(y - Y_A)^{a_y}(z - Z_A)^{a_z} \exp\left(-\alpha_i|\mathbf{r} - \mathbf{R}_A|^2\right) \tag{1}$$

where $\mathbf{r} = (x, y, z)$, $\mathbf{R}_A$ is the center of the primitive, $\mathbf{a} = (a_x, a_y, a_z)$ is the Cartesian components of the angular momentum, $\ell = a_x + a_y + a_z$, and

$$\begin{aligned} N(\alpha_i, \mathbf{a}) &= \sqrt{\left(\frac{2\alpha_i}{\pi}\right)^{\frac{3}{2}} \frac{(4\alpha_i)^{a_x+a_y+a_z}}{(2a_x-1)!!(2a_y-1)!!(2a_z-1)!!}} \\ &= \sqrt{\left(\frac{2\alpha_i}{\pi}\right)^{\frac{3}{2}} (4\alpha_i)^{\ell}} \sqrt{\frac{1}{(2a_x-1)!!(2a_y-1)!!(2a_z-1)!!}} \\ &= N_1(\alpha_i, \ell) N_2(\mathbf{a}) \end{aligned} \tag{2}$$

is the normalization constant of the primitive. In this module, the primitives are normalized.

A spherical Gaussian primitive is

$$g_i^s(r|\mathbf{R}_A, m, \ell) = N_s(\alpha_i, \ell) Y_{\ell m}(\phi, \theta) r^{\ell} \exp\left(-\alpha_i r^2\right) \tag{3}$$

where ℓ is the angular momentum, m is the z component of the angular momentum, $r = |\mathbf{r} - \mathbf{R}_A|$ is the distance from the center $\mathbf{R}_A$, $Y_{\ell m}$ is a spherical harmonic, and

$$N_s(\alpha_i, \ell) = \sqrt{\left(\frac{2\alpha_i}{\pi}\right)^{\frac{3}{2}} \frac{(4\alpha_i)^{\ell}}{(2\ell-1)!!}} \tag{4}$$

In this module, we treat all spherical harmonics to be real.

The "solid harmonic" (Helgaker 6.4.2), $Y_{\ell m}(\phi,\theta)r^\ell$, can be transformed into the corresponding Cartesian expression, $(x-X_A)^{a_x}(y-Y_A)^{a_y}(z-Z_A)^{a_z}$. Real-valued solid harmonics are denoted with $S_{\ell m}$.

$$g_i^s(r|\mathbf{R}_A,m,\ell) = N_s(\alpha_i,\ell)S_{\ell m}(r,\phi,\theta)\exp\left(-\alpha_i r^2\right) \tag{5}$$

In this module, we strictly utilize Cartesian Gaussian primitives at the lower level and transform to the spherical form whenever needed. Therefore, primitives will refer to Cartesian primitives unless otherwise stated.

2.1 Transformation of Spherical Primitives to Cartesian Primitives

To transform the spherical primitives to Cartesian primitives, the (real) solid harmonics are first transformed into the Cartesian expressions. Using the equations from Helgaker 6.4.4,

$$S_{\ell m} = N_{\ell m}^S \sum_{t=0}^{\lfloor\frac{\ell-|m|}{2}\rfloor}\sum_{u=0}^{t}\sum_{v=0}^{\lfloor\frac{|m|}{2}\rfloor} C_{tuv}^{lm} x^{2t+|m|-2(u+v_m)} y^{2(u+v_m)} z^{\ell-2t-|m|} \tag{6}$$

where

$$v_m = \begin{cases} v & \text{if } m \geq 0 \\ v+\frac{1}{2} & \text{if } m < 0 \end{cases} \tag{7}$$

$$C_{tuv}^{lm} = (-1)^{t+v}\left(\frac{1}{4}\right)^t \binom{\ell}{t}\binom{\ell-t}{|m|+t}\binom{t}{u}\binom{|m|}{2v_m} \tag{8}$$

$$N_{\ell m}^S = \frac{1}{2^{|m|}\ell!}\sqrt{\frac{2(\ell+|m|)!(\ell-|m|)!}{2^{\delta_{0m}}}} \tag{9}$$

Then, the normalization constant of the Cartesian primitive must be replaced with the spherical primitive.

$$\begin{aligned}\frac{N(\alpha_i,\mathbf{a})}{N_s(\alpha_i,\ell)} &= \frac{\sqrt{\left(\frac{2\alpha_i}{\pi}\right)^{\frac{3}{2}}\frac{(4\alpha_i)^\ell}{(2\ell-1)!!}}}{\sqrt{\left(\frac{2\alpha_i}{\pi}\right)^{\frac{3}{2}}(4\alpha_i)^\ell}\sqrt{\frac{1}{(2a_x-1)!!(2a_y-1)!!(2a_z-1)!!}}} \\ &= \sqrt{\frac{(2a_x-1)!!(2a_y-1)!!(2a_z-1)!!}{(2\ell-1)!!}}\end{aligned} \tag{10}$$

3 Contractions

Cartesian contractions are linear combinations of Cartesian primitives.

$$\begin{aligned}\phi(\mathbf{r}|\mathbf{R}_A,\mathbf{a},\mathbf{d},\boldsymbol{\alpha}) &= N_c(\mathbf{R}_A,\mathbf{a},\mathbf{d},\boldsymbol{\alpha})\sum_i d_i g_i(\mathbf{r}|\mathbf{R}_A,\mathbf{a}) \\ &= N_c(\mathbf{R}_A,\mathbf{a},\mathbf{d},\boldsymbol{\alpha})N_2(\mathbf{a})(x-X_A)^{a_x}(y-Y_A)^{a_y}(z-Z_A)^{a_z}\sum_i N_1(\alpha_i,\ell)\exp\left(-\alpha_i|\mathbf{r}-\mathbf{R}_A|^2\right)\end{aligned} \tag{11}$$

where $\mathbf{d}$ is the contraction coefficient, $\mathbf{a}$ are the exponents of the primitive, and

$$\begin{aligned} N_c(\mathbf{R}_A, \mathbf{a}, \mathbf{d}, \boldsymbol{\alpha}) &= \left(\int \left(\sum_i d_i g_i(\mathbf{r}|\mathbf{R}_A, \mathbf{a}) \right) \left(\sum_j d_j g_j(\mathbf{r}|\mathbf{R}_A, \mathbf{a}) \right) d\mathbf{r} \right)^{-\frac{1}{2}} \\ &= \left(\sum_i \sum_j d_i d_j \int g_i(\mathbf{r}|\mathbf{R}_A, \mathbf{a}) g_j(\mathbf{r}|\mathbf{R}_A, \mathbf{a}) d\mathbf{r} \right)^{-\frac{1}{2}} \end{aligned} \tag{12}$$

is the normalization constant of the contraction. In this module, the contractions are normalized.

Since the Cartesian expression, $(x - X_A)^{a_x}(y - Y_A)^{a_y}(z - Z_A)^{a_z}$, separates out from the rest of the primitives, the spherical contractions can be created from Cartesian contractions in the same way that the spherical primitives are constructed. In fact, we can group together many contractions of the same angular momentum and transform the contractions at the same time. In many cases, it is economical to group together contractions that share the same properties. We will denote these groups as shells.

3.1 Segmented Contractions

In order to group transform the Cartesian contractions into spherical contractions, all of the Cartesian components are needed, i.e. all combinations of nonnegative integers a_x, a_y, and a_z that adds up to ℓ. Segmented contractions is a group of contractions with the same angular momentum (ℓ), center ($\mathbf{R}_A$), contraction coefficients ($\mathbf{b}$) and exponents ($\boldsymbol{\alpha}$):

$$\{\phi(\mathbf{r}|\mathbf{R}_A, \mathbf{a}_j, \mathbf{d}, \boldsymbol{\alpha})|(a_j)_x + (a_j)_y + (a_j)_z = \ell\} \tag{13}$$

To avoid confusion with the term contraction, we use the term "shell of segmented contractions".

3.2 Generalized Contractions

Generalized contractions are a set of contractions that have the same center ($\mathbf{R}_A$) and exponents ($\boldsymbol{\alpha}$).

$$\{\phi(\mathbf{r}|\mathbf{R}_A, \mathbf{a}_{j\ell}, \mathbf{d}_k, \boldsymbol{\alpha}_k)|(a_{j\ell})_x + (a_{j\ell})_y + (a_{j\ell})_z = \ell\} \tag{14}$$

In this module, we do not group together contractions that differ in angular momentum. We use the term "shell of generalized contractions" to refer to the set of contractions with the same center ($\mathbf{R}_A$), same angular momentum (ℓ) and exponents ($\boldsymbol{\alpha}$):

$$\{\phi(\mathbf{r}|\mathbf{R}_A, \mathbf{a}_j, \mathbf{d}_k, \boldsymbol{\alpha}_k)|(a_j)_x + (a_j)_y + (a_j)_z = \ell\} \tag{15}$$

We can think of shell of generalized contractions as a union of shells of segmented contractions that differ only by the contraction coefficients, i.e. they use the same set of primitives.

4 Basis Set

In this module, basis set is defined to be a list of shells of generalized contractions.

4.1 Loading basis sets

Basis set information is often stored in text format. In `gbasis`, the Gaussian94 format (.gbs) and the NWChem format (.nwchem) are supported.

`gbasis` also interfaces to the module `iodata`, which handles the inputs and outputs for different quantum chemistry formats, such as Gaussian formatted checkpoint files (.fchk) and AIM wavefunction files (.wfn and .wfx). It also interfaces to `pyscf`, which is an ab initio computational chemistry program.

Usage: Supposed we have the following system:

```
atoms = ["H", "H"]
coords = np.array([[0, 0, 0], [0, 0, 1]])
```

- To build a basis set from gbs file for the given atoms and coordinates,

  ```
  from gbasis.parsers import parse_gbs, make_contractions

  with open("from_basissetexchange.gbs", "r")as f:
      all_basis = f.read()
  all_basis_dict = parse_gbs(all_basis)
  basis = make_contractions(all_basis_dict, atoms, coords)
  ```

- To build a basis set from nwchem file for the given atoms and coordinates,

  ```
  from gbasis.parsers import parse_nwchem, make_contractions

  with open("from_basissetexchange.nwchem", "r")as f:
      all_basis = f.read()
  all_basis_dict = parse_gbs(all_basis)
  basis = make_contractions(all_basis_dict, atoms, coords)
  ```

Using `iodata`, suppose we have the following system:

```
from iodata import load_one

mol = load_one("molecule.fchk")
```

To build a basis set,

```
from gbasis.wrappers import from_iodata

basis = from_iodata(mol)
```

Using `pyscf`, suppose we have the following system:

```
from pyscf import gto
mol = gto.Mole()
mol.build(
     atom = "O 0 0 0; H  0 1 0; H 0 0 1",
     basis = "sto-3g"
)
```

To build a basis set,

```
from gbasis.wrappers import from_pyscf

basis = from_pyscf(mol)
```

4.2 Types of coordinate systems used by basis functions

In `gbasis`, user can provide the coordinate system used by each shell of generalized contractions. All of the higher level functions have the keyword argument `coord_type` to specify the coordinate systems used by the basis. If `coord_type="spherical"`, all of the shells are treated to be spherical. If `coord_type="cartesian"`, all of the shells are treated to be Cartesian. If different shells correspond to different coordinate system, then a list/tuple of the same length as the basis set must be provided with each entry being `"spherical"` or `"cartesian"` to specify the coordinate system of the corresponding shell.

Usage:

- To treat all contractions to be spherical

  ```
  output = high_level_function(basis, coord_type="spherical")
  ```

- To treat all contractions to be Cartesian

  ```
  output = high_level_function(basis, coord_type="cartesian")
  ```

- To treat first shell of generalized contractions to be Cartesian and second shell to be spherical

  ```
  output = high_level_function(basis, coord_type=["cartesian", "spherical"])
  ```

  In this case, the basis set must consist of exactly two shells of generalized contractions. Otherwise, an error will raised.

4.3 Linear transformations of basis functions

In `gbasis`, user can linearly transform the basis functions before computing the desired properties. All of the higher level functions have the keyword argument `transform` to specify the matrix that transforms the basis set. The transformation is applied to the left, i.e.

$$\psi_i = \sum_j T_{ij} \phi_j \tag{16}$$

where ϕ_j is a basis function before transformation and ψ_i is a basis function after transformation. The number of basis functions depends on the coordinate systems specified for each shell. Note that the transformation matrix assumes that the basis functions are ordered according to Section 4.4.

Usage:

```
output = high_level_function(basis, transform=matrix)
```

where `matrix` is the transformation matrix.

4.4 Ordering of basis functions

Since a shell of generalized contractions is a set of contractions, they must be unpacked. When unpacked, the basis functions are first ordered by the shells, then by the segmented contraction, and then by the angular momentum component. For example, suppose the basis set consists of three shells of generalized contractions, G_1, G_2, and G_3:

$$\begin{aligned} G_1 &= \{\phi(\mathbf{r}|\mathbf{R}_A, \mathbf{a}_{1j}, \mathbf{d}_{1k}, \boldsymbol{\alpha}_1) | (a_{1j})_x + (a_{1j})_y + (a_{1j})_z = 1\} \\ G_2 &= \{\phi(\mathbf{r}|\mathbf{R}_B, \mathbf{a}_{2j}, \mathbf{d}_{2k}, \boldsymbol{\alpha}_2) | (a_{2j})_x + (a_{2j})_y + (a_{2j})_z = 2\} \\ G_3 &= \{\phi(\mathbf{r}|\mathbf{R}_A, \mathbf{a}_{3j}, \mathbf{d}_{3k}, \boldsymbol{\alpha}_3) | (a_{3j})_x + (a_{3j})_y + (a_{3j})_z = 2\} \end{aligned} \tag{17}$$

where $\mathbf{d}_1$ corresponds to two sets of contraction coefficients, $\mathbf{d}_2$ corresponds to three sets of contraction coefficients, and $\mathbf{d}_3$ corresponds to one set of contraction coefficients. Then, the basis functions in the Cartesian form will be ordered as follows:

$$\begin{aligned}
&\phi(\mathbf{r}|\mathbf{R}_A, (1,0,0), \mathbf{d}_{11}, \boldsymbol{\alpha}_1), \phi(\mathbf{r}|\mathbf{R}_A, (0,1,0), \mathbf{d}_{11}, \boldsymbol{\alpha}_1), \phi(\mathbf{r}|\mathbf{R}_A, (0,0,1), \mathbf{d}_{11}, \boldsymbol{\alpha}_1), \\
&- \\
&\phi(\mathbf{r}|\mathbf{R}_A, (1,0,0), \mathbf{d}_{12}, \boldsymbol{\alpha}_1), \phi(\mathbf{r}|\mathbf{R}_A, (0,1,0), \mathbf{d}_{12}, \boldsymbol{\alpha}_1), \phi(\mathbf{r}|\mathbf{R}_A, (0,0,1), \mathbf{d}_{12}, \boldsymbol{\alpha}_1), \\
&= \\
&\phi(\mathbf{r}|\mathbf{R}_B, (2,0,0), \mathbf{d}_{21}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (1,1,0), \mathbf{d}_{21}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (1,0,1), \mathbf{d}_{21}, \boldsymbol{\alpha}_2), \\
&\phi(\mathbf{r}|\mathbf{R}_B, (0,2,0), \mathbf{d}_{21}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (0,1,1), \mathbf{d}_{21}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (0,0,2), \mathbf{d}_{21}, \boldsymbol{\alpha}_2), \\
&- \\
&\phi(\mathbf{r}|\mathbf{R}_B, (2,0,0), \mathbf{d}_{22}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (1,1,0), \mathbf{d}_{22}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (1,0,1), \mathbf{d}_{22}, \boldsymbol{\alpha}_2), \\
&\phi(\mathbf{r}|\mathbf{R}_B, (0,2,0), \mathbf{d}_{22}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (0,1,1), \mathbf{d}_{22}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (0,0,2), \mathbf{d}_{22}, \boldsymbol{\alpha}_2), \\
&- \\
&\phi(\mathbf{r}|\mathbf{R}_B, (2,0,0), \mathbf{d}_{23}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (1,1,0), \mathbf{d}_{23}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (1,0,1), \mathbf{d}_{23}, \boldsymbol{\alpha}_2), \\
&\phi(\mathbf{r}|\mathbf{R}_B, (0,2,0), \mathbf{d}_{23}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (0,1,1), \mathbf{d}_{23}, \boldsymbol{\alpha}_2), \phi(\mathbf{r}|\mathbf{R}_B, (0,0,2), \mathbf{d}_{23}, \boldsymbol{\alpha}_2), \\
&= \\
&\phi(\mathbf{r}|\mathbf{R}_A, (2,0,0), \mathbf{d}_{31}, \boldsymbol{\alpha}_3), \phi(\mathbf{r}|\mathbf{R}_A, (1,1,0), \mathbf{d}_{31}, \boldsymbol{\alpha}_3), \phi(\mathbf{r}|\mathbf{R}_A, (1,0,1), \mathbf{d}_{31}, \boldsymbol{\alpha}_3), \\
&\phi(\mathbf{r}|\mathbf{R}_A, (0,2,0), \mathbf{d}_{31}, \boldsymbol{\alpha}_3), \phi(\mathbf{r}|\mathbf{R}_A, (0,1,1), \mathbf{d}_{31}, \boldsymbol{\alpha}_3), \phi(\mathbf{r}|\mathbf{R}_A, (0,0,2), \mathbf{d}_{31}, \boldsymbol{\alpha}_3)
\end{aligned} \tag{18}$$

where , delimits the contractions within a shell of segmented contractions, $-$ delimits the shells of segmented contractions within a shell of generalized contractions, and $=$ delimits the shells of generalized contractions.

The basis functions in the spherical form will be ordered as follows:

$$
\begin{aligned}
&\phi^s(\mathbf{r}|\mathbf{R}_A,-1,1,\mathbf{d}_{11},\boldsymbol{\alpha}_1),\phi^s(\mathbf{r}|\mathbf{R}_A,0,1,\mathbf{d}_{11},\boldsymbol{\alpha}_1),\phi^s(\mathbf{r}|\mathbf{R}_A,1,1,\mathbf{d}_{11},\boldsymbol{\alpha}_1),\\
&--\\
&\phi^s(\mathbf{r}|\mathbf{R}_A,-1,1,\mathbf{d}_{12},\boldsymbol{\alpha}_1),\phi^s(\mathbf{r}|\mathbf{R}_A,0,1,\mathbf{d}_{12},\boldsymbol{\alpha}_1),\phi^s(\mathbf{r}|\mathbf{R}_A,1,1,\mathbf{d}_{12},\boldsymbol{\alpha}_1),\\
&==\\
&\phi^s(\mathbf{r}|\mathbf{R}_B,-2,2,\mathbf{d}_{21},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,-1,2,\mathbf{d}_{21},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,0,2,\mathbf{d}_{21},\boldsymbol{\alpha}_2),\\
&\phi^s(\mathbf{r}|\mathbf{R}_B,1,2,\mathbf{d}_{21},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,2,2,\mathbf{d}_{21},\boldsymbol{\alpha}_2),\\
&--\\
&\phi^s(\mathbf{r}|\mathbf{R}_B,-2,2,\mathbf{d}_{22},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,-1,2,\mathbf{d}_{22},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,0,2,\mathbf{d}_{22},\boldsymbol{\alpha}_2),\\
&\phi^s(\mathbf{r}|\mathbf{R}_B,1,2,\mathbf{d}_{22},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,2,2,\mathbf{d}_{22},\boldsymbol{\alpha}_2),\\
&--\\
&\phi^s(\mathbf{r}|\mathbf{R}_B,-2,2,\mathbf{d}_{23},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,-1,2,\mathbf{d}_{23},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,0,2,\mathbf{d}_{23},\boldsymbol{\alpha}_2),\\
&\phi^s(\mathbf{r}|\mathbf{R}_B,1,2,\mathbf{d}_{23},\boldsymbol{\alpha}_2),\phi^s(\mathbf{r}|\mathbf{R}_B,2,2,\mathbf{d}_{23},\boldsymbol{\alpha}_2),\\
&==\\
&\phi^s(\mathbf{r}|\mathbf{R}_A,-2,2,\mathbf{d}_{31},\boldsymbol{\alpha}_3),\phi^s(\mathbf{r}|\mathbf{R}_A,-1,2,\mathbf{d}_{31},\boldsymbol{\alpha}_3),\phi^s(\mathbf{r}|\mathbf{R}_A,0,2,\mathbf{d}_{31},\boldsymbol{\alpha}_3),\\
&\phi^s(\mathbf{r}|\mathbf{R}_A,1,2,\mathbf{d}_{31},\boldsymbol{\alpha}_3),\phi^s(\mathbf{r}|\mathbf{R}_A,2,2,\mathbf{d}_{31},\boldsymbol{\alpha}_3)
\end{aligned}
\tag{19}
$$

where each spherical contraction has the form $\phi^s(\mathbf{r}|\mathbf{R}_A,m,\ell,\mathbf{d},\boldsymbol{\alpha})$.

The specific ordering of the angular momentum components in the Cartesian and spherical form is determined by the properties

`gbasis.contractions.GeneralizedContractionShell.angmom_components_cart`

and

`gbasis.contractions.GeneralizedContractionShell.angmom_components_sph`

respectively. To change the ordering, make a child of `GeneralizedContractionShell` and overwrite these properties with the desired ordering.

5 Evaluations

For the examples, suppose we have the following set of points:

```
import numpy as np

grid_1d = np.linspace(-2, 2, num=10)
grid_x, grid_y, grid_z = np.meshgrid(grid_1d, grid_1d, grid_1d)
grid_3d = np.vstack([grid_x.ravel(), grid_y.ravel(), grid_z.ravel()]).T
```

5.1 Evaluation of contractions

The functions in module `gbasis.eval` return the evaluations of the contractions at different coordinates:

$$\phi(\mathbf{r}_n|\mathbf{R}_A,\mathbf{a}_j,\mathbf{d}_k,\boldsymbol{\alpha}_k) \tag{20}$$

The returned value is an array whose rows corresponds to the basis function and columns corresponds to the coordinate, $\mathbf{r}_n$.

These functions can be used to find the values of the orbitals at various points, such as a grid.

Usage:

- To evaluate the atomic orbitals,

```
from gbasis.eval import evaluate_basis

output = evaluate_basis(basis, grid_3d, coord_type="spherical")
```

- To evaluate the molecular orbitals,

```
from gbasis.eval import evaluate_basis

output = evaluate_basis(basis, grid_3d, transform=transform_mo_ao, coord_type="spherical")
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.2 Evaluation of derivatives of contractions

In `gbasis`, contractions can be derivatized to arbitrary orders. The functions in module `gbasis.eval_deriv` return the evaluations of the given derivative of the contractions at different coordinates.

$$\frac{\partial^{m_x+m_y+m_z}}{\partial x^{m_x}\partial y^{m_y}\partial z^{m_z}}\phi(\mathbf{r}_n|\mathbf{R}_A,\mathbf{a}_j,\mathbf{d}_k,\boldsymbol{\alpha}_k) \tag{21}$$

The returned value is an array whose rows corresponds to the basis function and columns corresponds to the coordinate, $\mathbf{r}_n$.

Usage: Suppose the following derivative of the contraction is desired:

$$\frac{\partial^3}{\partial x\partial y^2} \tag{22}$$

- To evaluate the derivatives of the atomic orbitals,

```
from gbasis.eval_deriv import evaluate_deriv_basis

output = evaluate_deriv_basis(basis, grid_3d, np.array([1, 2, 0]), coord_type="spherical")
```

- To evaluate the derivatives of the molecular orbitals,

```
from gbasis.eval import evaluate_basis

output = evaluate_basis(
    basis, grid_3d, np.array([1, 2, 0]), transform=transform_mo_ao, coord_type="spherical"
)
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.3 Evaluations of density related properties

The functions in module `gbasis.density` return the evaluations of the density and its derivatives.

Suppose the `one_dm` is the one-electron density matrix.

5.3.1 Density

$$\rho(\mathbf{r}_n) = \sum_{ij} \gamma_{ij} \phi_i(\mathbf{r}_n) \phi_j(\mathbf{r}_n) \tag{23}$$

Usage:

- To evaluate the density using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_density

output = evaluate_density(one_dm, basis, grid_3d, coord_type="spherical")
```

- To evaluate the density using density matrix expressed with respect to molecular orbitals,

```
from gbasis.density import evaluate_density

output = evaluate_density(one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spheri
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.3.2 Arbitrary derivatives of density

$$\frac{\partial^{L_x+L_y+L_z}}{\partial x^{L_x} \partial y^{L_y} \partial z^{L_z}} \rho(\mathbf{r}_n) = \sum_{l_x=0}^{L_x} \sum_{l_y=0}^{L_y} \sum_{l_z=0}^{L_z} \binom{L_x}{l_x} \binom{L_y}{l_y} \binom{L_z}{l_z} \sum_{ij} \gamma_{ij} \frac{\partial^{l_x+l_y+l_z} \rho(\mathbf{r}_n)}{\partial x^{l_x} \partial y^{l_y} \partial z^{l_z}} \frac{\partial^{L_x+L_y+L_z-l_x-l_y-l_z} \rho(\mathbf{r}_n)}{\partial x^{L_x-l_x} \partial y^{L_y-l_y} \partial z^{L_z-l_z}} \tag{24}$$

Usage: Suppose the following derivative of the density is desired:

$$\frac{\partial^3}{\partial x \partial y^2} \tag{25}$$

- To evaluate the derivative of the density using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_deriv_density

output = evaluate_deriv_density(np.array([1, 2, 0]), one_dm, basis, grid_3d, coord_type="spheri
```

- To evaluate the derivative of the density using density matrix expressed with respect to molecular orbitals,

```
from gbasis.density import evaluate_deriv_density

output = evaluate_deriv_density(
    np.array([1, 2, 0]), one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spheric
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.3.3 Gradient of density

$$\nabla\rho(\mathbf{r}_n) = \begin{bmatrix} \frac{\partial}{\partial x}\rho(\mathbf{r}_n) \\ \frac{\partial}{\partial y}\rho(\mathbf{r}_n) \\ \frac{\partial}{\partial z}\rho(\mathbf{r}_n) \end{bmatrix} \tag{26}$$

Usage:

- To evaluate the gradient of the density using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_density_gradient

output = evaluate_density_gradient(one_dm, basis, grid_3d, coord_type="spherical")
```

- To evaluate the gradient of the density using density matrix expressed with respect to molecular orbitals,

```
from gbasis.density import evaluate_density_gradient

output = evaluate_density_gradient(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.3.4 Laplacian of density

$$\nabla^2\rho(\mathbf{r}_n) = \frac{\partial^2}{\partial x^2}\rho(\mathbf{r}_n) + \frac{\partial^2}{\partial y^2}\rho(\mathbf{r}_n) + \frac{\partial^2}{\partial z^2}\rho(\mathbf{r}_n) \tag{27}$$

Usage:

- To evaluate the laplacian of the density using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_density_laplacian

output = evaluate_density_laplacian(one_dm, basis, grid_3d, coord_type="spherical")
```

where

- To evaluate the laplacian of the density using density matrix expressed with respect to molecular orbitals,

```
from gbasis.density import evaluate_density_laplacian

output = evaluate_density_laplacian(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.3.5 Hessian of density

$$H[\rho(\mathbf{r}_n)] = \begin{bmatrix} \frac{\partial^2}{\partial x^2}\rho(\mathbf{r}_n) & \frac{\partial^2}{\partial x \partial y}\rho(\mathbf{r}_n) & \frac{\partial^2}{\partial x \partial z}\rho(\mathbf{r}_n) \\ \frac{\partial^2}{\partial x \partial y}\rho(\mathbf{r}_n) & \frac{\partial^2}{\partial y^2}\rho(\mathbf{r}_n) & \frac{\partial^2}{\partial y \partial z}\rho(\mathbf{r}_n) \\ \frac{\partial^2}{\partial x \partial z}\rho(\mathbf{r}_n) & \frac{\partial^2}{\partial z^2}\rho(\mathbf{r}_n) & \frac{\partial^2}{\partial x \partial z}\rho(\mathbf{r}_n) \end{bmatrix} \tag{28}$$

Usage:

- To evaluate the Hessian of the density using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_density_hessian

output = evaluate_density_hessian(one_dm, basis, grid_3d, coord_type="spherical")
```

where

- To evaluate the Hessian of the density using density matrix expressed with respect to molecular orbitals,

```
from gbasis.density import evaluate_density_hessian

output = evaluate_density_hessian(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.4 Evaluations of density matrix related properties

Given the density matrix,

$$\gamma(\mathbf{r}_1, \mathbf{r}_2) = \sum_{ij} \gamma_{ij} \phi_i(\mathbf{r}_1) \phi_j(\mathbf{r}_2) \tag{29}$$

many properties can be defined by evaluating the derivatives of the density matrix at the same coordinate:

$$\frac{\partial^{p_x+p_y+p_z}}{\partial x_1^{p_x} \partial y_1^{p_y} \partial z_1^{p_z}} \frac{\partial^{q_x+q_y+q_z}}{\partial x_2^{q_x} \partial y_2^{q_y} \partial z_2^{q_z}} \gamma(\mathbf{r}_1, \mathbf{r}_2)\Big|_{\mathbf{r}_1=\mathbf{r}_2=\mathbf{r}_n} = \sum_{ij} \gamma_{ij} \left. \frac{\partial^{p_x+p_y+p_z}}{\partial x_1^{p_x} \partial y_1^{p_y} \partial z_1^{p_z}} \phi_i(\mathbf{r}_1) \right|_{\mathbf{r}_1=\mathbf{r}_n} \left. \frac{\partial^{q_x+q_y+q_z}}{\partial x_2^{q_x} \partial y_2^{q_y} \partial z_2^{q_z}} \phi_j(\mathbf{r}_2) \right|_{\mathbf{r}_1=\mathbf{r}_n} \tag{30}$$

where $\mathbf{r}_1$ is the first coordinate, $\mathbf{r}_2$ is the second coordinate, and $\mathbf{r}_n$ is the coordinate at which the derivative is evaluated.

Since γ_{ij} is symmetric,

$$\frac{\partial^{p_x+p_y+p_z}}{\partial x_1^{p_x} \partial y_1^{p_y} \partial z_1^{p_z}} \frac{\partial^{q_x+q_y+q_z}}{\partial x_2^{q_x} \partial y_2^{q_y} \partial z_2^{q_z}} \gamma(\mathbf{r}_1, \mathbf{r}_2)\Big|_{\mathbf{r}_1=\mathbf{r}_2=\mathbf{r}_n} = \frac{\partial^{q_x+q_y+q_z}}{\partial x_1^{q_x} \partial y_1^{q_y} \partial z_1^{q_z}} \frac{\partial^{p_x+p_y+p_z}}{\partial x_2^{p_x} \partial y_2^{p_y} \partial z_2^{p_z}} \gamma(\mathbf{r}_1, \mathbf{r}_2)\Big|_{\mathbf{r}_1=\mathbf{r}_2=\mathbf{r}_n} \tag{31}$$

Again, suppose `one_dm` is the one-electron density matrix.

5.4.1 Stress tensor

$$\begin{aligned}
\boldsymbol{\sigma}_{ij}(\mathbf{r}_n|\alpha,\beta) =& -\frac{1}{2}\alpha \left(\frac{\partial^2}{\partial r_i \partial r'_j} \gamma(\mathbf{r}, \mathbf{r}') + \frac{\partial^2}{\partial r_j \partial r'_i} \gamma(\mathbf{r}, \mathbf{r}') \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&+ \frac{1}{2}(1-\alpha) \left(\frac{\partial^2}{\partial r_i \partial r_j} \gamma(\mathbf{r}, \mathbf{r}) + \frac{\partial^2}{\partial r'_i \partial r'_j} \gamma(\mathbf{r}, \mathbf{r}') \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&- \frac{1}{2}\delta_{ij}\beta\nabla^2\rho(\mathbf{r}_n) \\
=& -\alpha \left. \frac{\partial^2}{\partial r_i \partial r'_j} \gamma(\mathbf{r}, \mathbf{r}') \right|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} + (1-\alpha) \left. \frac{\partial^2}{\partial r_i \partial r_j} \gamma(\mathbf{r}, \mathbf{r}) \right|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} - \frac{1}{2}\delta_{ij}\beta\nabla^2\rho(\mathbf{r}_n)
\end{aligned} \tag{32}$$

Usage:

- To evaluate the stress tensor ($\alpha = 1$ and $\beta = 0$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.stress_tensor import evaluate_stress_tensor

output = evaluate_stress_tensor(one_dm, basis, grid_3d, coord_type="spherical")
```

- To evaluate the stress tensor ($\alpha = 0.5$ and $\beta = 1$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.stress_tensor import evaluate_stress_tensor

output = evaluate_stress_tensor(one_dm, basis, grid_3d, alpha=0.5, beta=1, coord_type="spherica
```

- To evaluate the stress tensor ($\alpha = 1$ and $\beta = 0$) using density matrix expressed with respect to molecular orbitals,

```
from gbasis.stress_tensor import evaluate_stress_tensor

output = evaluate_stress_tensor(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.4.2 Ehrenfest force

Ehrenfest force is defined as the divergence of the stress tensor

$$
\begin{aligned}
F_j(\mathbf{r}_n|\alpha,\beta) =& \sum_i \frac{\partial}{\partial r_i}\boldsymbol{\sigma}_{ij} \\
=& -\alpha \sum_i \frac{\partial^3}{\partial r_i^2 \partial r'_j}\gamma(\mathbf{r},\mathbf{r}')\bigg|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} - \alpha \sum_i \frac{\partial^3}{\partial r_i \partial r'_i \partial r'_j}\gamma(\mathbf{r},\mathbf{r}')\bigg|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&+ (1-\alpha)\sum_i \frac{\partial^3}{\partial r_i^2 \partial r_j}\gamma(\mathbf{r},\mathbf{r})\bigg|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} + (1-\alpha)\sum_i \frac{\partial^3}{\partial r_i \partial r_j \partial r'_i}\gamma(\mathbf{r},\mathbf{r})\bigg|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} - \frac{1}{2}\sum_i \delta_{ij}\beta \frac{\partial}{\partial r_i}\nabla^2\rho(\mathbf{r}_n) \\
=& -\alpha \sum_i \frac{\partial^3}{\partial r_i^2 \partial r'_j}\gamma(\mathbf{r},\mathbf{r}')\bigg|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&+ (1-\alpha)\sum_i \frac{\partial^3}{\partial r_i^2 \partial r_j}\gamma(\mathbf{r},\mathbf{r})\bigg|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} + (1-2\alpha)\sum_i \frac{\partial^3}{\partial r_i \partial r_j \partial r'_i}\gamma(\mathbf{r},\mathbf{r})\bigg|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} - \frac{1}{2}\sum_i \delta_{ij}\beta \frac{\partial}{\partial r_i}\nabla^2\rho(\mathbf{r}_n)
\end{aligned}
\tag{33}
$$

Usage:

- To evaluate the Ehrenfest force ($\alpha = 1$ and $\beta = 0$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.stress_tensor import evaluate_ehrenfest_force

output = evaluate_ehrenfest_force(one_dm, basis, grid_3d, coord_type="spherical")
```

- To evaluate the Ehrenfest force ($\alpha = 0.5$ and $\beta = 1$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.stress_tensor import evaluate_ehrenfest_force

output = evaluate_ehrenfest_force(one_dm, basis, grid_3d, alpha=0.5, beta=1, coord_type="spheri
```

- To evaluate the Ehrenfest force ($\alpha = 1$ and $\beta = 0$) using density matrix expressed with respect to molecular orbitals,

```
from gbasis.stress_tensor import evaluate_ehrenfest_force

output = evaluate_ehrenfest_force(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.4.3 Ehrenfest Hessian

$$
\begin{aligned}
H_{jk}(\mathbf{r}_n|\alpha,\beta) =& \frac{\partial}{\partial r_k} F_j(\mathbf{r}_n|\alpha,\beta) \\
=& -\alpha \sum_i \left(\frac{\partial^4}{\partial r_i^2 \partial r_k \partial r'_j} \gamma(\mathbf{r},\mathbf{r}') + \frac{\partial^4}{\partial r_i^2 \partial r'_j \partial r'_k} \gamma(\mathbf{r},\mathbf{r}') \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&+ (1-\alpha) \sum_i \left(\frac{\partial^4}{\partial r_i^2 \partial r_j \partial r_k} \gamma(\mathbf{r},\mathbf{r}) + \frac{\partial^4}{\partial r_i^2 \partial r_j \partial r'_k} \gamma(\mathbf{r},\mathbf{r}) \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&+ (1-2\alpha) \sum_i \left(\frac{\partial^4}{\partial r_i \partial r_j \partial r_k \partial r'_i} \gamma(\mathbf{r},\mathbf{r}) + \frac{\partial^4}{\partial r_i \partial r_j \partial r'_i \partial r'_k} \gamma(\mathbf{r},\mathbf{r}) \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&- \frac{1}{2} \sum_i \delta_{ij} \beta \frac{\partial^2}{\partial r_i \partial r_k} \nabla^2 \rho(\mathbf{r}_n)
\end{aligned}
\tag{34}
$$

Usage:

- To evaluate the Ehrenfest Hessian ($\alpha = 1$ and $\beta = 0$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.stress_tensor import evaluate_ehrenfest_hessian

output = evaluate_ehrenfest_hessian(one_dm, basis, grid_3d, coord_type="spherical")
```

- To evaluate the Ehrenfest Hessian ($\alpha = 0.5$ and $\beta = 1$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.stress_tensor import evaluate_ehrenfest_hessian

output = evaluate_ehrenfest_hessian(
    one_dm, basis, grid_3d, alpha=0.5, beta=1, coord_type="spherical"
)
```

- To evaluate the Ehrenfest Hessian ($\alpha = 1$ and $\beta = 0$) using density matrix expressed with respect to molecular orbitals,

```
from gbasis.stress_tensor import evaluate_ehrenfest_hessian

output = evaluate_ehrenfest_hessian(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.4.4 Positive-definite kinetic energy density

$$\begin{aligned} t_+(\mathbf{r}_n) &= \frac{1}{2} \nabla_{\mathbf{r}} \cdot \nabla_{\mathbf{r}'} \gamma(\mathbf{r}, \mathbf{r}')|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\ &= \frac{1}{2} \left(\frac{\partial^2}{\partial x \partial x'} \gamma(\mathbf{r}, \mathbf{r}') + \frac{\partial^2}{\partial y \partial y'} \gamma(\mathbf{r}, \mathbf{r}') + \frac{\partial^2}{\partial z \partial z'} \gamma(\mathbf{r}, \mathbf{r}') \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \end{aligned} \tag{35}$$

Usage:

- To evaluate the positive-definite kinetic energy density using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_posdef_kinetic_energy_density

output = evaluate_posdef_kinetic_energy_density(one_dm, basis, grid_3d, coord_type="spherical")
```

- To evaluate the positive-definite kinetic energy density using density matrix expressed with respect to molecular orbitals,

```
from gbasis.density import evaluate_posdef_kinetic_energy_density

output = evaluate_posdef_kinetic_energy_density(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

5.4.5 General form of kinetic energy density

$$t_\alpha(\mathbf{r}_n) = t_+(\mathbf{r}_n) + \alpha \nabla^2 \rho(\mathbf{r}_n) \tag{36}$$

Usage:

- To evaluate the general form of the kinetic energy density ($\alpha = 1$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_general_kinetic_energy_density

output = evaluate_general_kinetic_energy_density(one_dm, basis, grid_3d, coord_type="spherical"
```

- To evaluate the general form of the kinetic energy density ($\alpha = 0.5$) using density matrix expressed with respect to atomic orbitals,

```
from gbasis.density import evaluate_general_kinetic_energy_density

output = evaluate_general_kinetic_energy_density(
    one_dm, basis, grid_3d, alpha=0.5, coord_type="spherical"
)
```

- To evaluate the general form of the kinetic energy density ($\alpha = 1$) using density matrix expressed with respect to molecular orbitals,

```
from gbasis.density import evaluate_general_kinetic_energy_density

output = evaluate_general_kinetic_energy_density(
    one_dm, basis, grid_3d, transform=transform_mo_ao, coord_type="spherical"
)
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6 Integrals

6.1 Overlap integral

$$\int \phi_a(\mathbf{r})\phi_b(\mathbf{r})d\mathbf{r} \tag{37}$$

Usage:

- To compute the overlap of a set of atomic orbitals

```
from gbasis.overlap import overlap_integral

output = overlap_integral(basis, coord_type="spherical")
```

- To compute the overlap of a set of molecular orbitals

```
from gbasis.overlap import overlap_integral

output = overlap_integral(basis, transform=transform_mo_ao, coord_type="spherical")
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.1.1 Overlap integral between two different basis sets

Overlap integrals between two different basis sets are supported i.e. ϕ_a belongs to one basis set and ϕ_b belongs to another.

TODO: usage, example

6.2 Multipole moment integral

Multipole moment integral can be obtained for arbitrary moments.

$$\int \phi_a(\mathbf{r})(x - X_C)^{c_x}(y - Y_C)^{c_y}(z - Z_C)^{c_z}\phi_b(\mathbf{r})d\mathbf{r} \tag{38}$$

Usage: Suppose the integral of the following moments is desired:

$$(x - 1.5)^2(y - 2.5)^3(z - 3.5) \tag{39}$$

$$(x - 1.5)(y - 2.5)^2(z - 3.5)^3 \tag{40}$$

- To compute the moment of a set of atomic orbitals

```
from gbasis.moment import moment_integral

output = moment_integral(
    basis, np.array([1.5, 2.5, 3.5]), np.array([[2, 3, 1], [1, 2, 3]]),
coord_type="spherical"
)
```

- To compute the overlap of a set of molecular orbitals

```
from gbasis.moment import moment_integral

output = moment_integral(
    basis,
    np.array([1.5, 2.5, 3.5]),
    np.array([[2, 3, 1], [1, 2, 3]]),
    transform=transform_mo_ao,
    coord_type="spherical",
)
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.3 Integrals over differential operator

Integrals over arbitrary differential operator (for Cartesian coordinates) are supported.

$$\int \phi_a(\mathbf{r})\frac{\partial^{e+f+g}}{\partial x^e \partial y^f \partial z^g}\phi_b(\mathbf{r})d\mathbf{r} \tag{41}$$

6.3.1 Kinetic energy integral

$$\begin{aligned}\left\langle \hat{T} \right\rangle &= \int \phi_a(\mathbf{r}) \left(-\frac{1}{2}\nabla^2\right) \phi_b(\mathbf{r})d\mathbf{r} \\ &= -\frac{1}{2}\left(\int \phi_a(\mathbf{r})\frac{\partial^2}{\partial x^2}\phi_b(\mathbf{r})d\mathbf{r} + \int \phi_a(\mathbf{r})\frac{\partial^2}{\partial y^2}\phi_b(\mathbf{r})d\mathbf{r} + \int \phi_a(\mathbf{r})\frac{\partial^2}{\partial z^2}\phi_b(\mathbf{r})d\mathbf{r}\right)\end{aligned} \tag{42}$$

Usage:

- To compute the kinetic energy integral of a set of atomic orbitals

  ```
  from gbasis.kinetic_energy import kinetic_energy_integral

  output = kinetic_energy_integral(basis, coord_type="spherical")
  ```

- To compute the kinetic energy integral of a set of molecular orbitals

  ```
  from gbasis.kinetic_energy import kinetic_energy_integral

  output = kinetic_energy_integral(basis, transform=transform_mo_ao, coord_type="spherical")
  ```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.3.2 Momentum integral

$$\begin{aligned}\langle \hat{\mathbf{p}} \rangle &= \int \phi_a(\mathbf{r}) \left(-i\nabla\right) \phi_b(\mathbf{r}) d\mathbf{r} \\ &= -i \begin{bmatrix} \int \phi_a(\mathbf{r}) \frac{\partial}{\partial x} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) \frac{\partial}{\partial y} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) \frac{\partial}{\partial z} \phi_b(\mathbf{r}) d\mathbf{r} \end{bmatrix}\end{aligned} \tag{43}$$

Usage:

- To compute the momentum integral of a set of atomic orbitals

  ```
  from gbasis.momentum import momentum_integral

  output = momentum_integral(basis, coord_type="spherical")
  ```

- To compute the momentum integral of a set of molecular orbitals

  ```
  from gbasis.momentum import momentum_integral

  output = momentum_integral(basis, transform=transform_mo_ao, coord_type="spherical")
  ```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.3.3 Angular momentum integral

$$\begin{aligned}\left\langle \hat{\mathbf{L}} \right\rangle &= \int \phi_a(\mathbf{r}) \left(-i\mathbf{r} \times \nabla\right) \phi_b(\mathbf{r}) d\mathbf{r} \\ &= -i \begin{bmatrix} \int \phi_a(\mathbf{r}) y \frac{\partial}{\partial z} \phi_b(\mathbf{r}) d\mathbf{r} - \int \phi_a(\mathbf{r}) z \frac{\partial}{\partial y} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) z \frac{\partial}{\partial x} \phi_b(\mathbf{r}) d\mathbf{r} - \int \phi_a(\mathbf{r}) x \frac{\partial}{\partial z} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) x \frac{\partial}{\partial y} \phi_b(\mathbf{r}) d\mathbf{r} - \int \phi_a(\mathbf{r}) y \frac{\partial}{\partial x} \phi_b(\mathbf{r}) d\mathbf{r} \end{bmatrix}\end{aligned} \tag{44}$$

Usage:

- To compute the angular momentum integral of a set of atomic orbitals

```
from gbasis.angular_momentum import angular_momentum_integral

output = angular_momentum_integral(basis, coord_type="spherical")
```

- To compute the angular momentum integral of a set of molecular orbitals

```
from gbasis.angular_momentum import angular_momentum_integral

output = angular_momentum_integral(basis, transform=transform_mo_ao, coord_type="spherical")
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.4 Integral for interaction with point-charge

$$\int \phi_a(\mathbf{r}) \frac{1}{|\mathbf{r} - \mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} \tag{45}$$

Usage: Suppose there are two point charges: -3 charge at $(0, 1, 2)$ and 5 charge at $(3, 4, 6)$

- To compute the integral for interaction between these point-charges and the set of atomic orbital

```
from gbasis.point_charge import point_charge_integral

output = point_charge_integral(
    basis, np.array([[0, 1, 2], [3, 4, 6]]), np.array([-3, 5]), coord_type="spherical"
)
```

- To compute the integral for interaction between these point-charges and the set of molecular orbital

```
from gbasis.point_charge import point_charge_integral

output = point_charge_integral(
    basis,
    np.array([[0, 1, 2], [3, 4, 6]]),
    np.array([-3, 5]),
    transform=transform_mo_ao,
    coord_type="spherical",
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.4.1 Nuclear-electron attraction integral

$$\int \phi_a(\mathbf{r}) \frac{-Z_c}{|\mathbf{r}-\mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} = -Z_C \int \phi_a(\mathbf{r}) \frac{1}{|\mathbf{r}-\mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} \tag{46}$$

Usage: Suppose there are two nuclei: He at $(0, 1, 2)$ and Al at $(3, 4, 6)$

- To compute the nuclear-electron attraction integral of the set of atomic orbital

```
from gbasis.nuclear_electron_attraction import nuclear_electron_attraction_integral

output = nuclear_electron_attraction_integral(
    basis, np.array([[0, 1, 2], [3, 4, 6]]), np.array([2, 13]), coord_type="spherical"
)
```

- To compute the nuclear-electron attraction integral of the set of molecular orbital

```
from gbasis.nuclear_electron_attraction import nuclear_electron_attraction_integral

output = nuclear_electron_attraction_integral(
    basis,
    np.array([[0, 1, 2], [3, 4, 6]]),
    np.array([2, 13]),
    transform=transform_mo_ao,
    coord_type="spherical",
)
```

where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.4.2 Electrostatic potential

$$-\left(-\sum_A \frac{Z_A}{|\mathbf{R}_C-\mathbf{R}_A|} + \sum_{ab} \gamma_{ab} \int \phi_a(\mathbf{r}) \frac{-1}{|\mathbf{r}-\mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r}\right) = \sum_A \frac{Z_A}{|\mathbf{R}_C-\mathbf{R}_A|} - \sum_{ab} \gamma_{ab} \int \phi_a(\mathbf{r}) \frac{1}{|\mathbf{r}-\mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} \tag{47}$$

Usage: Suppose there are two nuclei, He at $(0, 1, 2)$ and Al at $(3, 4, 6)$ $(0, 1, 2)$ and $(3, 4, 6)$, the electrostatic potential is measured at points $(0.5, 1.5, 2.5)$ and $(2.5, 3.5, 5.5)$, and the one-electron density matrix is given by `one_dm`.

- To compute the electrostatic potential using density matrix expressed with respect to atomic orbitals

```
from gbasis.electrostatic_potential import electrostatic_potential

output = electrostatic_potential(
    basis,
    one_dm,
    np.array([[0.5, 1.5, 2.5], [2.5, 3.5, 5.5]]),
    np.array([[0, 1, 2], [3, 4, 6]]),
    np.array([2, 13]),
    coord_type="spherical",
)
```

- To compute the electrostatic potential using density matrix expressed with respect to molecular orbitals

```
from gbasis.electrostatic_potential import electrostatic_potential

output = electrostatic_potential(
    basis,
    one_dm,
    np.array([[0.5, 1.5, 2.5], [2.5, 3.5, 5.5]]),
    np.array([[0, 1, 2], [3, 4, 6]]),
    np.array([2, 13]),
    transform=transform_mo_ao,
    coord_type="spherical"
)
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

6.5 Electron-electron repulsion integral

In the Chemists' notation,

$$\int \phi_a^*(\mathbf{r}_1)\phi_b(\mathbf{r}_1)\frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}\phi_c^*(\mathbf{r}_2)\phi_d(\mathbf{r}_2)d\mathbf{r} \tag{48}$$

In the Physicists' notation

$$\int \phi_a^*(\mathbf{r}_1)\phi_b^*(\mathbf{r}_2)\frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|}\phi_c(\mathbf{r}_1)\phi_d(\mathbf{r}_2)d\mathbf{r} \tag{49}$$

Though both conventions are supported at the higher level, lower level code uses the Chemists' notation.

Usage:

- To compute the electron repulsion integral of a set of atomic orbitals in Physicists' notation

```
from gbasis.electron_repulsion import electron_repulsion_integral

output = electron_repulsion_integral(basis, coord_type="spherical")
```

- To compute the electron repulsion integral of a set of molecular orbitals in Physicists' notation

```
from gbasis.electron_repulsion import electron_repulsion_integral

output = electron_repulsion_integral(basis, transform=transform_mo_ao, coord_type="spherical")
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

- To compute the electron repulsion integral of a set of molecular orbitals in Chemists' notation

```
from gbasis.electron_repulsion import electron_repulsion_integral

output = electron_repulsion_integral(
    basis, transform=transform_mo_ao, coord_type="spherical", notation="chemist"
)
```

  where `transform_mo_ao` is the transformation matrix from atomic orbitals to molecular orbitals.

7 Future Development

- screening (overlap)
- screening (two electron integrals)
- improve performance in two electron integral
- zeroth order regular approximation (zora) for relativistic effects
- density fitting
- derivative of contractions with respect to the center
- derivative of integrals with respect to center
- periodic boundary condition
- damped two electron integral