

Contents

1	Mandate	2
2	Gaussian Primitive	2
2.1	Transformation of Spherical Primitives to Cartesian Primitives	3
3	Contractions	4
3.1	Segmented Contractions	4
3.2	Generalized Contractions	4
4	Basis Set	5
4.1	Loading basis sets	5
4.2	Types of coordinate systems used by basis functions	6
4.3	Linear transformations of basis functions	6
4.4	Ordering of basis functions	7
4.5	Conventions for ordering Cartesian and spherical components	8
5	Evaluations	9
5.1	Evaluation of contractions	10
5.2	Evaluation of derivatives of contractions	10
5.3	Evaluations of density related properties	11
5.3.1	Density	11
5.3.2	Arbitrary derivatives of density	11
5.3.3	Gradient of density	12
5.3.4	Laplacian of density	12
5.3.5	Hessian of density	13
5.4	Evaluations of density matrix related properties	13
5.4.1	Stress tensor	14
5.4.2	Ehrenfest force	15
5.4.3	Ehrenfest Hessian	16
5.4.4	Positive-definite kinetic energy density	16
5.4.5	General form of kinetic energy density	17
6	Integrals	18
6.1	Overlap integral	18
6.1.1	Overlap integral between two different basis sets	18
6.2	Multipole moment integral	18
6.3	Integrals over differential operator	19
6.3.1	Kinetic energy integral	19
6.3.2	Momentum integral	20
6.3.3	Angular momentum integral	20
6.4	Integral for interaction with point-charge	21
6.4.1	Nuclear-electron attraction integral	21
6.4.2	Electrostatic potential	22
6.5	Electron-electron repulsion integral	22

7	Future Development	23
8	Not sure where to put this	23

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

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

where $\mathbf{r} = (x, y, z)$, $\mathbf{R}_A = (X_A, Y_A, Z_A)$ 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

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

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

A spherical Gaussian primitive is

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

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

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

is the normalization constant of the spherical primitive. 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 $S_{\ell m}$.

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

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,

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

where

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

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

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

By linearly transforming the all of primitives with different Cartesian expressions, $(x - X_A)^{a_x}(y - Y_A)^{a_y}(z - Z_A)^{a_z}$, for the given angular momentum using Equation 6, spherical primitives can be obtained, though they are normalized with the Cartesian normalization constants.

Then, the normalization constant of the Cartesian primitive must be replaced with the spherical primitive by multiplying the newly formed spherical primitives with the following factor.

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

3 Contractions

Cartesian contractions are linear combinations of Cartesian primitives.

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

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 L2 normalization constant of the contraction. In this module, the contractions are L2 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 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 ($\boldsymbol{\alpha}$).

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

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 ($\boldsymbol{\alpha}$):

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

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.

Example: Supposed we have the following system:

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

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

```
1 from gbasis.parsers import parse_gbs, make_contractions
2
3 all_basis_dict = parse_gbs("./path/to/from_basissetexchange.gbs")
4 basis = make_contractions(all_basis_dict, atoms, coords)
```

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

```
1 from gbasis.parsers import parse_nwchem, make_contractions
2
3 all_basis_dict = parse_nwchem("./path/to/from_basissetexchange.nwchem")
4 basis = make_contractions(all_basis_dict, atoms, coords)
```

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).

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

```
1 from iodata import load_one
2
3 mol = load_one("molecule.fchk")
```

To build a basis set,

```
1 from gbasis.wrappers import from_iodata
2
3 basis = from_iodata(mol)
```

gbasis also interfaces to **pyscf**, which is an ab initio computational chemistry program.

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

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

```
6 | )
```

To build a basis set,

```
1 from gbasis.wrappers import from_pyscf
2
3 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. The function **make_contractions** has 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.

Example:

- To treat all contractions to be spherical

```
1 basis = make_contractions(all_basis_dict, atoms, coords, coord_type="spherical")
```

- To treat all contractions to be Cartesian

```
1 basis = make_contractions(all_basis_dict, atoms, coords, coord_type="cartesian")
```

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

```
1 basis = make_contractions(
2     all_basis_dict, ["H", "H"], coords, coord_type=["cartesian", "spherical"]
3 )
```

In this case, each hydrogen atom must contribute exactly one shell of generalized contractions. Otherwise, an error will be 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.

$$\psi_i = \sum_j T_{ij} \phi_j \quad (16)$$

where $\{\phi_j\}$ is the basis set before transformation and $\{\psi_i\}$ is the 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.

Example:

```
1 output = high_level_function(basis, transform=transform_mo_ao)
```

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 :

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

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), \\ &\text{-----} \\ &\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), \\ &\text{-----} \\ &\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), \\ &\text{-----} \\ &\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} \quad (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), \\
& \text{-----} \\
& \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), \\
& \text{=====} \\
& \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), \\
& \text{-----} \\
& \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), \quad (19) \\
& \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), \\
& \text{-----} \\
& \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), \\
& \text{=====} \\
& \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}$$

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

```
1 gbasis.contractions.GeneralizedContractionShell.angmom_components_cart
```

and

```
1 gbasis.contractions.GeneralizedContractionShell.angmom_components_sph
```

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

4.5 Conventions for ordering Cartesian and spherical components

Cartesian components A shell of generalized contractions of angular momentum ℓ is composed of $(\ell+1)(\ell+2)$ Cartesian components of the form $x^{a_x}y^{a_y}z^{a_z}$ such that $a_x + a_y + a_z = \ell$. In `gbasis`, the ordering of Cartesian components is stored as an $\ell \times 3$ `numpy` array, where each entry (i, j) corresponds to the exponent a_j for the i^{th} component. In other programs, like `HORTON2`, a different notation is used, where the components are written as strings of X, Y, and Z, where the number of times each letter appears corresponds to the exponent a for that letter. These notations are easily convertible, for example, the `gbasis` notation for the component XXXZZ is (3, 0, 2)

In `gbasis`, all components are sorted by descending a_x , then by descending a_y , as generated by the following list comprehension:

```
1 [(x, y, self.angmom - x - y)
2   for x in range(self.angmom, -1, -1)
3   for y in range(self.angmom - x, -1, -1)]
```

For example, for $\ell = 3$, the default ordering is:


```

1  [
2      (3, 0, 0),
3      (2, 1, 0),
4      (2, 0, 1),
5      (1, 2, 0),
6      (1, 1, 1),
7      (1, 0, 2),
8      (0, 3, 0),
9      (0, 2, 1),
10     (0, 1, 2),
11     (0, 0, 3),
12 ]

```

Spherical (pure) components A shell of generalized contractions of angular momentum ℓ is composed of $2\ell + 1$ spherical components that take the form of cosine- and sine-like real regular solid harmonics $C_{\ell,m}$ and $S_{\ell,m}$. We can define these cosine- and sine-like functions using the regular solid harmonics $R_{\ell,m}$ such that

$$\begin{aligned} R_{\ell,m} &= C_{\ell,m} & m = 0, \dots, \ell \\ R_{\ell,-m} &= S_{\ell,m} & m = 1, \dots, \ell \end{aligned} \quad (20)$$

In **gbasis**, we use these definitions to represent spherical components as strings of the form **c{m}** and **s{m}**, where m is the magnetic quantum number for the given angular momentum. N.B.: for sine-like functions, **s{m}** corresponds to $-m$, as per the definition in equation 20. This ordering is stored as a list of these strings, consistent with the notation found in **IOData**.

Some quantum chemistry programs, such as **ORCA**, use conventions that require changing the signs of specific spherical (pure) components. Therefore, the sign of each component must also be encoded in the ordering. In **gbasis**, this is done by prefixing a negative sign to the string for that component, e.g. **-s3**.

The standard convention for ordering spherical components in **gbasis** is from $-m$ to m . The ordering is generated by the following list comprehension:

```

1  ['s{}'.format(m) for m in range(self.angmom, 0, -1)]
2  + ['c{}'.format(m) for m in range(self.angmom + 1)]

```

For example, for $\ell = 3$, the default ordering is:

```

1  ['s3', 's2', 's1', 'c0', 'c1', 'c2', 'c3']

```

In **ORCA** Molden files, the convention for $\ell = 3$, illustrating the use of the negative prefix, is:

```

1  ['c0', 'c1', 's1', 'c2', 's2', '-c3', '-s3']

```

5 Evaluations

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

```

1  import numpy as np
2
3  grid_1d = np.linspace(-2, 2, num=10)

```

```

4 grid_x, grid_y, grid_z = np.meshgrid(grid_1d, grid_1d, grid_1d)
5 grid_3d = np.vstack([grid_x.ravel(), grid_y.ravel(), grid_z.ravel()]).T

```

5.1 Evaluation of contractions

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

$$\phi(\mathbf{r}_n | \mathbf{R}_A, \mathbf{a}_j, \mathbf{d}_k, \boldsymbol{\alpha}) \quad (21)$$

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.

Example:

- To evaluate the atomic orbitals,

```

1 from gbasis.evals.eval import evaluate_basis
2
3 output = evaluate_basis(basis, grid_3d)

```

- To evaluate the molecular orbitals,

```

1 from gbasis.evals.eval import evaluate_basis
2
3 output = evaluate_basis(basis, grid_3d, transform=transform_mo_ao)

```

5.2 Evaluation of derivatives of contractions

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

$$\frac{\partial^{m_x+m_y+m_z}}{\partial x^{m_x} \partial y^{m_y} \partial z^{m_z}} \phi(\mathbf{r}_n | \mathbf{R}_A, \mathbf{a}_j, \mathbf{d}_k, \boldsymbol{\alpha}) \quad (22)$$

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

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

$$\frac{\partial^3}{\partial x \partial y^2} \quad (23)$$

- To evaluate the derivatives of the atomic orbitals,

```

1 from gbasis.evals.eval_deriv import evaluate_deriv_basis
2
3 output = evaluate_deriv_basis(basis, grid_3d, np.array([1, 2, 0]))

```

- To evaluate the derivatives of the molecular orbitals,

```

1 from gbasis.evals.eval_deriv import evaluate_deriv_basis
2
3 output = evaluate_deriv_basis(
4     basis, grid_3d, np.array([1, 2, 0]), transform=transform_mo_ao
5 )

```

5.3 Evaluations of density related properties

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

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

5.3.1 Density

$$\rho(\mathbf{r}_n) = \sum_{ij} \gamma_{ij} \phi_i(\mathbf{r}_n) \phi_j(\mathbf{r}_n) \quad (24)$$

Example:

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

```
1 from gbasis.evals.density import evaluate_density
2
3 output = evaluate_density(one_dm, basis, grid_3d)
```

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

```
1 from gbasis.evals.density import evaluate_density
2
3 output = evaluate_density(one_dm, basis, grid_3d, transform=transform_mo_ao)
```

5.3.2 Arbitrary derivatives of density

$$\frac{\partial^{L_x+L_y+L_z}}{\partial x^{L_x} \partial y^{L_y} \partial z^{L_z}} \rho(\mathbf{r}_n) = \sum_{l_x=0}^{L_x} \sum_{l_y=0}^{L_y} \sum_{l_z=0}^{L_z} \binom{L_x}{l_x} \binom{L_y}{l_y} \binom{L_z}{l_z} \sum_{ij} \gamma_{ij} \frac{\partial^{l_x+l_y+l_z} \rho(\mathbf{r}_n)}{\partial x^{l_x} \partial y^{l_y} \partial z^{l_z}} \frac{\partial^{L_x+L_y+L_z-l_x-l_y-l_z} \rho(\mathbf{r}_n)}{\partial x^{L_x-l_x} \partial y^{L_y-l_y} \partial z^{L_z-l_z}} \quad (25)$$

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

$$\frac{\partial^3}{\partial x \partial y^2} \quad (26)$$

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

```
1 from gbasis.evals.density import evaluate_deriv_density
2
3 output = evaluate_deriv_density(
4     np.array([1, 2, 0]), one_dm, basis, grid_3d, deriv_type="general"
5 )
```

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

```
1 from gbasis.evals.density import evaluate_deriv_density
2
3 output = evaluate_deriv_density(
4     np.array([1, 2, 0]), one_dm, basis, grid_3d, transform=transform_mo_ao,
5     deriv_type="general")
```

6 |)

Notice the use of the keyword argument `deriv_type`, which allows to choose between a general or a specific function derivative procedure based on the order of derivation. If `deriv_type="general"`, an implementation valid for any order of derivative of the density function is used. However, for first and second order derivatives of the density an optimized procedure can be utilized by setting `deriv_type="direct"` which makes the computation faster.

5.3.3 Gradient of density

$$\nabla\rho(\mathbf{r}_n) = \begin{bmatrix} \frac{\partial}{\partial x}\rho(\mathbf{r}_n) \\ \frac{\partial}{\partial y}\rho(\mathbf{r}_n) \\ \frac{\partial}{\partial z}\rho(\mathbf{r}_n) \end{bmatrix} \quad (27)$$

Example:

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

```
1 from gbasis.evals.density import evaluate_density_gradient
2
3 output = evaluate_density_gradient(one_dm, basis, grid_3d, deriv_type="general")
```

- To evaluate the gradient of the density using the specific procedure for up to second order derivatives and the density matrix expressed with respect to atomic orbitals,

```
1 from gbasis.evals.density import evaluate_density_gradient
2
3 output = evaluate_density_gradient(one_dm, basis, grid_3d, deriv_type="direct")
```

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

```
1 from gbasis.evals.density import evaluate_density_gradient
2
3 output = evaluate_density_gradient(
4     one_dm, basis, grid_3d, transform=transform_mo_ao, deriv_type="general"
5 )
```

5.3.4 Laplacian of density

$$\nabla^2\rho(\mathbf{r}_n) = \frac{\partial^2}{\partial x^2}\rho(\mathbf{r}_n) + \frac{\partial^2}{\partial y^2}\rho(\mathbf{r}_n) + \frac{\partial^2}{\partial z^2}\rho(\mathbf{r}_n) \quad (28)$$

Example:

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

```

1 from gbasis.evals.density import evaluate_density_laplacian
2
3 output = evaluate_density_laplacian(one_dm, basis, grid_3d, deriv_type="general")

```

where

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

```

1 from gbasis.evals.density import evaluate_density_laplacian
2
3 output = evaluate_density_laplacian(
4     one_dm, basis, grid_3d, transform=transform_mo_ao, deriv_type="general"
5 )

```

5.3.5 Hessian of density

$$H[\rho(\mathbf{r}_n)] = \begin{bmatrix} \frac{\partial^2}{\partial x^2} \rho(\mathbf{r}_n) & \frac{\partial^2}{\partial x \partial y} \rho(\mathbf{r}_n) & \frac{\partial^2}{\partial x \partial z} \rho(\mathbf{r}_n) \\ \frac{\partial^2}{\partial x \partial y} \rho(\mathbf{r}_n) & \frac{\partial^2}{\partial y^2} \rho(\mathbf{r}_n) & \frac{\partial^2}{\partial y \partial z} \rho(\mathbf{r}_n) \\ \frac{\partial^2}{\partial x \partial z} \rho(\mathbf{r}_n) & \frac{\partial^2}{\partial z^2} \rho(\mathbf{r}_n) & \frac{\partial^2}{\partial x \partial z} \rho(\mathbf{r}_n) \end{bmatrix} \quad (29)$$

Example:

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

```

1 from gbasis.evals.density import evaluate_density_hessian
2
3 output = evaluate_density_hessian(one_dm, basis, grid_3d, deriv_type="general")

```

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

```

1 from gbasis.evals.density import evaluate_density_hessian
2
3 output = evaluate_density_hessian(
4     one_dm, basis, grid_3d, transform=transform_mo_ao, deriv_type="general"
5 )

```

5.4 Evaluations of density matrix related properties

Given the density matrix,

$$\gamma(\mathbf{r}_1, \mathbf{r}_2) = \sum_{ij} \gamma_{ij} \phi_i(\mathbf{r}_1) \phi_j(\mathbf{r}_2) \quad (30)$$

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

$$\left. \frac{\partial^{p_x+p_y+p_z}}{\partial x_1^{p_x} \partial y_1^{p_y} \partial z_1^{p_z}} \frac{\partial^{q_x+q_y+q_z}}{\partial x_2^{q_x} \partial y_2^{q_y} \partial z_2^{q_z}} \gamma(\mathbf{r}_1, \mathbf{r}_2) \right|_{\mathbf{r}_1=\mathbf{r}_2=\mathbf{r}_n} = \sum_{ij} \gamma_{ij} \left. \frac{\partial^{p_x+p_y+p_z}}{\partial x_1^{p_x} \partial y_1^{p_y} \partial z_1^{p_z}} \phi_i(\mathbf{r}_1) \right|_{\mathbf{r}_1=\mathbf{r}_n} \left. \frac{\partial^{q_x+q_y+q_z}}{\partial x_2^{q_x} \partial y_2^{q_y} \partial z_2^{q_z}} \phi_j(\mathbf{r}_2) \right|_{\mathbf{r}_1=\mathbf{r}_n} \quad (31)$$

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,

$$\frac{\partial^{p_x+p_y+p_z}}{\partial x_1^{p_x} \partial y_1^{p_y} \partial z_1^{p_z}} \frac{\partial^{q_x+q_y+q_z}}{\partial x_2^{q_x} \partial y_2^{q_y} \partial z_2^{q_z}} \gamma(\mathbf{r}_1, \mathbf{r}_2) \Big|_{\mathbf{r}_1=\mathbf{r}_2=\mathbf{r}_n} = \frac{\partial^{q_x+q_y+q_z}}{\partial x_1^{q_x} \partial y_1^{q_y} \partial z_1^{q_z}} \frac{\partial^{p_x+p_y+p_z}}{\partial x_2^{p_x} \partial y_2^{p_y} \partial z_2^{p_z}} \gamma(\mathbf{r}_1, \mathbf{r}_2) \Big|_{\mathbf{r}_1=\mathbf{r}_2=\mathbf{r}_n} \quad (32)$$

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

5.4.1 Stress tensor

$$\begin{aligned} \sigma_{ij}(\mathbf{r}_n | \alpha, \beta) &= -\frac{1}{2} \alpha \left(\frac{\partial^2}{\partial r_i \partial r'_j} \gamma(\mathbf{r}, \mathbf{r}') + \frac{\partial^2}{\partial r_j \partial r'_i} \gamma(\mathbf{r}, \mathbf{r}') \right) \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\ &\quad + \frac{1}{2} (1 - \alpha) \left(\frac{\partial^2}{\partial r_i \partial r_j} \gamma(\mathbf{r}, \mathbf{r}) + \frac{\partial^2}{\partial r'_i \partial r'_j} \gamma(\mathbf{r}, \mathbf{r}') \right) \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\ &\quad - \frac{1}{2} \delta_{ij} \beta \nabla^2 \rho(\mathbf{r}_n) \\ &= -\alpha \frac{\partial^2}{\partial r_i \partial r'_j} \gamma(\mathbf{r}, \mathbf{r}') \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} + (1 - \alpha) \frac{\partial^2}{\partial r_i \partial r_j} \gamma(\mathbf{r}, \mathbf{r}) \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} - \frac{1}{2} \delta_{ij} \beta \nabla^2 \rho(\mathbf{r}_n) \end{aligned} \quad (33)$$

Example:

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

```
1 from gbasis.evals.stress_tensor import evaluate_stress_tensor
2
3 output = evaluate_stress_tensor(one_dm, basis, grid_3d)
```

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

```
1 from gbasis.evals.stress_tensor import evaluate_stress_tensor
2
3 output = evaluate_stress_tensor(one_dm, basis, grid_3d, alpha=0.5, beta=1)
```

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

```
1 from gbasis.evals.stress_tensor import evaluate_stress_tensor
2
3 output = evaluate_stress_tensor(
4     one_dm, basis, grid_3d, transform=transform_mo_ao
5 )
```

5.4.2 Ehrenfest force

Ehrenfest force is defined as the negative of the divergence of the stress tensor

$$\begin{aligned}
F_j(\mathbf{r}_n|\alpha, \beta) &= - \sum_i \frac{\partial}{\partial r_i} \sigma_{ij} \\
&= \alpha \sum_i \frac{\partial^3}{\partial r_i^2 \partial r_j'} \gamma(\mathbf{r}, \mathbf{r}') \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} + \alpha \sum_i \frac{\partial^3}{\partial r_i \partial r_i' \partial r_j'} \gamma(\mathbf{r}, \mathbf{r}') \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&\quad - (1-\alpha) \sum_i \frac{\partial^3}{\partial r_i^2 \partial r_j} \gamma(\mathbf{r}, \mathbf{r}) \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} - (1-\alpha) \sum_i \frac{\partial^3}{\partial r_i \partial r_j \partial r_i'} \gamma(\mathbf{r}, \mathbf{r}) \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} + \frac{1}{2} \sum_i \delta_{ij} \beta \frac{\partial}{\partial r_i} \nabla^2 \rho(\mathbf{r}_n) \\
&= \alpha \sum_i \frac{\partial^3}{\partial r_i^2 \partial r_j'} \gamma(\mathbf{r}, \mathbf{r}') \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&\quad - (1-\alpha) \sum_i \frac{\partial^3}{\partial r_i^2 \partial r_j} \gamma(\mathbf{r}, \mathbf{r}) \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} - (1-2\alpha) \sum_i \frac{\partial^3}{\partial r_i \partial r_j \partial r_i'} \gamma(\mathbf{r}, \mathbf{r}) \Big|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} + \frac{1}{2} \sum_i \delta_{ij} \beta \frac{\partial}{\partial r_i} \nabla^2 \rho(\mathbf{r}_n)
\end{aligned} \tag{34}$$

Example:

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

```

1 from gbasis.evals.stress_tensor import evaluate_ehrenfest_force
2
3 output = evaluate_ehrenfest_force(one_dm, basis, grid_3d)

```

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

```

1 from gbasis.evals.stress_tensor import evaluate_ehrenfest_force
2
3 output = evaluate_ehrenfest_force(one_dm, basis, grid_3d, alpha=0.5, beta=1)

```

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

```

1 from gbasis.evals.stress_tensor import evaluate_ehrenfest_force
2
3 output = evaluate_ehrenfest_force(
4     one_dm, basis, grid_3d, transform=transform_mo_ao
5 )

```

5.4.3 Ehrenfest Hessian

$$\begin{aligned}
H_{jk}(\mathbf{r}_n|\alpha, \beta) &= -\frac{\partial}{\partial r_k} F_j(\mathbf{r}_n|\alpha, \beta) \\
&= \alpha \sum_i \left(\frac{\partial^4}{\partial r_i^2 \partial r_j \partial r_k} \gamma(\mathbf{r}, \mathbf{r}') + \frac{\partial^4}{\partial r_i^2 \partial r_j' \partial r_k'} \gamma(\mathbf{r}, \mathbf{r}') \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&\quad - (1-\alpha) \sum_i \left(\frac{\partial^4}{\partial r_i^2 \partial r_j \partial r_k} \gamma(\mathbf{r}, \mathbf{r}) + \frac{\partial^4}{\partial r_i^2 \partial r_j \partial r_k'} \gamma(\mathbf{r}, \mathbf{r}) \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&\quad - (1-2\alpha) \sum_i \left(\frac{\partial^4}{\partial r_i \partial r_j \partial r_k \partial r_i'} \gamma(\mathbf{r}, \mathbf{r}) + \frac{\partial^4}{\partial r_i \partial r_j \partial r_i' \partial r_k'} \gamma(\mathbf{r}, \mathbf{r}) \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&\quad + \frac{1}{2} \sum_i \delta_{ij} \beta \frac{\partial^2}{\partial r_i \partial r_k} \nabla^2 \rho(\mathbf{r}_n)
\end{aligned} \tag{35}$$

Example:

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

```

1 from gbasis.evals.stress_tensor import evaluate_ehrenfest_hessian
2
3 output = evaluate_ehrenfest_hessian(one_dm, basis, grid_3d)

```

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

```

1 from gbasis.evals.stress_tensor import evaluate_ehrenfest_hessian
2
3 output = evaluate_ehrenfest_hessian(
4     one_dm, basis, grid_3d, alpha=0.5, beta=1
5 )

```

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

```

1 from gbasis.evals.stress_tensor import evaluate_ehrenfest_hessian
2
3 output = evaluate_ehrenfest_hessian(
4     one_dm, basis, grid_3d, transform=transform_mo_ao
5 )

```

5.4.4 Positive-definite kinetic energy density

$$\begin{aligned}
t_+(\mathbf{r}_n) &= \frac{1}{2} \nabla_{\mathbf{r}} \cdot \nabla_{\mathbf{r}'} \gamma(\mathbf{r}, \mathbf{r}')|_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n} \\
&= \frac{1}{2} \left(\frac{\partial^2}{\partial x \partial x'} \gamma(\mathbf{r}, \mathbf{r}') + \frac{\partial^2}{\partial y \partial y'} \gamma(\mathbf{r}, \mathbf{r}') + \frac{\partial^2}{\partial z \partial z'} \gamma(\mathbf{r}, \mathbf{r}') \right)_{\mathbf{r}=\mathbf{r}'=\mathbf{r}_n}
\end{aligned} \tag{36}$$

Example:

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

```

1 from gbasis.evals.density import evaluate_posdef_kinetic_energy_density
2
3 output = evaluate_posdef_kinetic_energy_density(
4     one_dm, basis, grid_3d, deriv_type="general"
5 )

```

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

```

1 from gbasis.evals.density import evaluate_posdef_kinetic_energy_density
2
3 output = evaluate_posdef_kinetic_energy_density(
4     one_dm, basis, grid_3d, transform=transform_mo_ao, deriv_type="general"
5 )

```

5.4.5 General form of kinetic energy density

$$t_{\alpha}(\mathbf{r}_n) = t_{+}(\mathbf{r}_n) + \alpha \nabla^2 \rho(\mathbf{r}_n) \quad (37)$$

Example:

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

```

1 from gbasis.evals.density import evaluate_general_kinetic_energy_density
2
3 output = evaluate_general_kinetic_energy_density(
4     one_dm, basis, grid_3d, deriv_type="general"
5 )

```

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

```

1 from gbasis.evals.density import evaluate_general_kinetic_energy_density
2
3 output = evaluate_general_kinetic_energy_density(
4     one_dm, basis, grid_3d, alpha=0.5, deriv_type="general"
5 )

```

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

```

1 from gbasis.evals.density import evaluate_general_kinetic_energy_density
2
3 output = evaluate_general_kinetic_energy_density(
4     one_dm, basis, grid_3d, transform=transform_mo_ao, deriv_type="general"
5 )

```

6 Integrals

6.1 Overlap integral

$$\int \phi_a(\mathbf{r})\phi_b(\mathbf{r})d\mathbf{r} \quad (38)$$

Example:

- To compute the overlap of a set of atomic orbitals

```
1 from gbasis.integrals.overlap import overlap_integral
2
3 output = overlap_integral(basis)
```

- To compute the overlap of a set of molecular orbitals

```
1 from gbasis.integrals.overlap import overlap_integral
2
3 output = overlap_integral(basis, transform=transform_mo_ao)
```

6.1.1 Overlap integral between two different basis sets

Overlap integrals between two different basis sets are supported i.e.

$$\int \phi_a(\mathbf{r})\psi_b(\mathbf{r})d\mathbf{r} \quad (39)$$

Example:

- To compute the overlap between two sets of atomic orbitals

```
1 from gbasis.integrals.overlap_asymm import overlap_integral_asymmetric
2
3 output = overlap_integral_asymmetric(basis_one, basis_two)
```

- To compute the overlap between a set of molecular orbitals and a set of atomic orbitals

```
1 from gbasis.integrals.overlap_asymm import overlap_integral_asymmetric
2
3 output = overlap_integral_asymmetric(
4     basis_one, basis_two, transform_one=transform_ao_mo
5 )
```

6.2 Multipole moment integral

Multipole moment integral can be obtained for arbitrary moments.

$$\int \phi_a(\mathbf{r})(x - X_C)^{c_x}(y - Y_C)^{c_y}(z - Z_C)^{c_z}\phi_b(\mathbf{r})d\mathbf{r} \quad (40)$$

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

$$(x - 1.5)^2(y - 2.5)^3(z - 3.5) \quad (41)$$

$$(x - 1.5)(y - 2.5)^2(z - 3.5)^3 \quad (42)$$

- To compute the moment of a set of atomic orbitals

```

1 from gbasis.integrals.moment import moment_integral
2
3 output = moment_integral(
4     basis, np.array([1.5, 2.5, 3.5]), np.array([[2, 3, 1], [1, 2, 3]])
5 )

```

- To compute the overlap of a set of molecular orbitals

```

1 from gbasis.integrals.moment import moment_integral
2
3 output = moment_integral(
4     basis,
5     np.array([1.5, 2.5, 3.5]),
6     np.array([[2, 3, 1], [1, 2, 3]]),
7     transform=transform_mo_ao
8 )

```

6.3 Integrals over differential operator

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

$$\int \phi_a(\mathbf{r}) \frac{\partial^{e+f+g}}{\partial x^e \partial y^f \partial z^g} \phi_b(\mathbf{r}) d\mathbf{r} \quad (43)$$

6.3.1 Kinetic energy integral

$$\begin{aligned} \langle \hat{T} \rangle &= \int \phi_a(\mathbf{r}) \left(-\frac{1}{2} \nabla^2 \right) \phi_b(\mathbf{r}) d\mathbf{r} \\ &= -\frac{1}{2} \left(\int \phi_a(\mathbf{r}) \frac{\partial^2}{\partial x^2} \phi_b(\mathbf{r}) d\mathbf{r} + \int \phi_a(\mathbf{r}) \frac{\partial^2}{\partial y^2} \phi_b(\mathbf{r}) d\mathbf{r} + \int \phi_a(\mathbf{r}) \frac{\partial^2}{\partial z^2} \phi_b(\mathbf{r}) d\mathbf{r} \right) \end{aligned} \quad (44)$$

Example:

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

```

1 from gbasis.integrals.kinetic_energy import kinetic_energy_integral
2
3 output = kinetic_energy_integral(basis)

```

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

```

1 from gbasis.integrals.kinetic_energy import kinetic_energy_integral
2
3 output = kinetic_energy_integral(basis, transform=transform_mo_ao)

```

6.3.2 Momentum integral

$$\begin{aligned}\langle \hat{\mathbf{p}} \rangle &= \int \phi_a(\mathbf{r}) (-i\nabla) \phi_b(\mathbf{r}) d\mathbf{r} \\ &= -i \begin{bmatrix} \int \phi_a(\mathbf{r}) \frac{\partial}{\partial x} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) \frac{\partial}{\partial y} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) \frac{\partial}{\partial z} \phi_b(\mathbf{r}) d\mathbf{r} \end{bmatrix}\end{aligned}\quad (45)$$

Example:

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

```
1 from gbasis.integrals.momentum import momentum_integral
2
3 output = momentum_integral(basis)
```

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

```
1 from gbasis.integrals.momentum import momentum_integral
2
3 output = momentum_integral(basis, transform=transform_mo_ao)
```

6.3.3 Angular momentum integral

$$\begin{aligned}\langle \hat{\mathbf{L}} \rangle &= \int \phi_a(\mathbf{r}) (-i\mathbf{r} \times \nabla) \phi_b(\mathbf{r}) d\mathbf{r} \\ &= -i \begin{bmatrix} \int \phi_a(\mathbf{r}) y \frac{\partial}{\partial z} \phi_b(\mathbf{r}) d\mathbf{r} - \int \phi_a(\mathbf{r}) z \frac{\partial}{\partial y} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) z \frac{\partial}{\partial x} \phi_b(\mathbf{r}) d\mathbf{r} - \int \phi_a(\mathbf{r}) x \frac{\partial}{\partial z} \phi_b(\mathbf{r}) d\mathbf{r} \\ \int \phi_a(\mathbf{r}) x \frac{\partial}{\partial y} \phi_b(\mathbf{r}) d\mathbf{r} - \int \phi_a(\mathbf{r}) y \frac{\partial}{\partial x} \phi_b(\mathbf{r}) d\mathbf{r} \end{bmatrix}\end{aligned}\quad (46)$$

Example:

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

```
1 from gbasis.integrals.angular_momentum import angular_momentum_integral
2
3 output = angular_momentum_integral(basis)
```

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

```
1 from gbasis.integrals.angular_momentum import angular_momentum_integral
2
3 output = angular_momentum_integral(basis, transform=transform_mo_ao)
```

6.4 Integral for interaction with point-charge

$$\int \phi_a(\mathbf{r}) \frac{1}{|\mathbf{r} - \mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} \quad (47)$$

Example: 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

```
1 from gbasis.integrals.point_charge import point_charge_integral
2
3 output = point_charge_integral(
4     basis, np.array([[0, 1, 2], [3, 4, 6]]), np.array([-3, 5])
5 )
```

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

```
1 from gbasis.integrals.point_charge import point_charge_integral
2
3 output = point_charge_integral(
4     basis,
5     np.array([[0, 1, 2], [3, 4, 6]]),
6     np.array([-3, 5]),
7     transform=transform.mo_ao
8 )
```

6.4.1 Nuclear-electron attraction integral

$$\int \phi_a(\mathbf{r}) \frac{-Z_c}{|\mathbf{r} - \mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} = -Z_C \int \phi_a(\mathbf{r}) \frac{1}{|\mathbf{r} - \mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} \quad (48)$$

Example: 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

```
1 from gbasis.integrals.nuclear_electron_attraction import
   nuclear_electron_attraction_integral
2
3 output = nuclear_electron_attraction_integral(
4     basis, np.array([[0, 1, 2], [3, 4, 6]]), np.array([2, 13])
5 )
```

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

```
1 from gbasis.integrals.nuclear_electron_attraction import
   nuclear_electron_attraction_integral
2
3 output = nuclear_electron_attraction_integral(
4     basis,
5     np.array([[0, 1, 2], [3, 4, 6]]),
6     np.array([2, 13]),
7     transform=transform.mo_ao
8 )
```

6.4.2 Electrostatic potential

$$-\left(-\sum_A \frac{Z_A}{|\mathbf{R}_C - \mathbf{R}_A|} + \sum_{ab} \gamma_{ab} \int \phi_a(\mathbf{r}) \frac{-1}{|\mathbf{r} - \mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r}\right) = \sum_A \frac{Z_A}{|\mathbf{R}_C - \mathbf{R}_A|} - \sum_{ab} \gamma_{ab} \int \phi_a(\mathbf{r}) \frac{1}{|\mathbf{r} - \mathbf{R}_C|} \phi_b(\mathbf{r}) d\mathbf{r} \quad (49)$$

Example: 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

```
1 from gbasis.evals.electrostatic_potential import electrostatic_potential
2
3 output = electrostatic_potential(
4     basis,
5     one_dm,
6     np.array([[0.5, 1.5, 2.5], [2.5, 3.5, 5.5]]),
7     np.array([[0, 1, 2], [3, 4, 6]]),
8     np.array([2, 13])
9 )
```

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

```
1 from gbasis.evals.electrostatic_potential import electrostatic_potential
2
3 output = electrostatic_potential(
4     basis,
5     one_dm,
6     np.array([[0.5, 1.5, 2.5], [2.5, 3.5, 5.5]]),
7     np.array([[0, 1, 2], [3, 4, 6]]),
8     np.array([2, 13]),
9     transform=transform.mo_ao
10 )
```

6.5 Electron-electron repulsion integral

In the Chemists' notation,

$$\int \phi_a^*(\mathbf{r}_1) \phi_b(\mathbf{r}_1) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \phi_c^*(\mathbf{r}_2) \phi_d(\mathbf{r}_2) d\mathbf{r} \quad (50)$$

In the Physicists' notation

$$\int \phi_a^*(\mathbf{r}_1) \phi_b^*(\mathbf{r}_2) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_2|} \phi_c(\mathbf{r}_1) \phi_d(\mathbf{r}_2) d\mathbf{r} \quad (51)$$

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

Example:

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

```

1 from gbasis.integrals.electron_repulsion import electron_repulsion_integral
2
3 output = electron_repulsion_integral(basis)

```

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

```

1 from gbasis.integrals.electron_repulsion import electron_repulsion_integral
2
3 output = electron_repulsion_integral(basis, transform=transform_mo_ao)

```

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

```

1 from gbasis.integrals.electron_repulsion import electron_repulsion_integral
2
3 output = electron_repulsion_integral(
4     basis, transform=transform_mo_ao, notation="chemist"
5 )

```

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

8 Not sure where to put this

- Feature requests via github
- Contributions are welcome