gau2grid Documentation

Release v2.0.6+4.gcd0e8d0.dirty

Daniel G. A. Smith

Jan 11, 2021

GETTING STARTED

gau2grid is a python-generated C library for vectorized computation of grid to gaussian collocation matrices

The core of gau2grid is generating the collocation matrices between a real space grid and a gaussian basis set expanded to a given angular momenta. Where a simple gaussian can be represented with the cartesian form as:

$$\phi(\mathbf{r}) = x^l y^m z^n e^{-\alpha r^2}$$

where for a given angular momenta ℓ, a gaussian basis has all possible combinations of l, m, n that satisfy $l+m+n = \ell$. These gaussians can also take a spherical harmonic form of:

$$\phi(\mathbf{r}) = Y_\ell^m(\hat{\mathbf{r}}) e^{-\alpha r^2}$$

where m ranges from $+\ell$ to $-\ell$. The spherical form offers a more compact representation at higher angular momenta, but is more difficult to work with when examining cartesian derivates.

In quantum chemistry, an individual basis is often represented as a sum of several gaussian with different exponents and coefficients together:

$$\phi(\mathbf{r}) = Y_\ell^m(\hat{\mathbf{r}}) \sum_i c_i e^{-\alpha_i r^2}$$

Collocation matrices between a single basis set and multiple grid points can then be represented as follows:

$$\phi_{mp} = Y_\ell^m(\mathbf{r}_p \widehat{-\mathbf{r}_{\text{center}}}) \sum_i c_i e^{-\alpha_i (\mathbf{r}_{\text{center}} - \mathbf{r}_p)^2}$$

where the basis is evaluated at every point p for every component of the basis i.e. basis function m. The ϕ_{mp} matrices are the primary focus on the gau2grid library.

CHAPTER

ONE

INDEX

Getting Started

1.1 Python installation

You can install gau2grid with `conda` or by installing from source.

1.1.1 Conda

You can update gau2grid using conda:

```
conda install pygau2grid -c psi4
```

This installs gau2grid and the NumPy dependancy.

1.1.2 Install from Source

To install gau2grid from source, clone the repository from github:

```
git clone https://github.com/dgasmith/gau2grid.git
cd gau2grid
python setup.py install
```

1.1.3 Test

Test gau2grid with `py.test`:

```
cd gau2grid
py.test
```

1.2 C installation

You can install gau2grid with `conda` or by installing from source.

1.2.1 Conda

You can update gau2grid using conda:

```
conda install gau2grid -c psi4
```

This installs the gau2grid library.

1.2.2 Install from Source

Gau2grid uses the CMake build system to compile and configure options. To begin, clone the repository:

```
git clone https://github.com/dgasmith/gau2grid.git
cd gau2grid
```

A basic CMake build can then be executed with:

```
cmake -H. -Bobjdir
cd objdir
make
make install
```

1.2.3 CMake Options

Gau2grid can be compiled with the following CMake options:

- `CMAKE_INSTALL_PREFIX` - The path to install the library to (default, `/usr/local`)
- `CMAKE_INSTALL_LIBDIR` - Directory to which libraries installed
- `MAX_AM` - The maximum gaussian angular momentum to compile (default, `8`)
- `CMAKE_BUILD_TYPE` - Build type (Release or Debug) (default, `Release`)
- `ENABLE_XHOST` - Enables processor-specific optimization (default, `ON`)
- `BUILD_FPIC` - Libraries will be compiled with position independent code (default, `ON`)
- `BUILD_SHARED_LIBS` - Build final library as shared, not static (default, `ON`)
- `ENABLE_GENERIC` - Enables mostly static linking of system libraries for shared library (default, `OFF`)

CMake options should be prefixed with `-D`, for example:

```
cmake -H. -Bobjdir -DCMAKE_INSTALL_PREFIX=~/installs
```

1.3 Gaussian Component Orders

The order of the individual components can vary between use cases. gau2grid can produce any resulting order that a user requires. The `C` version of the code must be compiled to a given order. The currently supported orders are as follows.

1.3.1 Cartesian Order

gau2grid currently supports both the `cca` and `molden` orders. The number of components per angular momentum can be computed as:

$$N_{\text{cartesian}} = (\ell + 1)(\ell + 2)/2$$

Row Order

The `cca` order iterates over the upper triangular hyper diagonal and has the following pattern:

- `S` ($\ell = 0$): *1*
- `P` ($\ell = 1$): `X`, `Y`, `Z`
- `D` ($\ell = 2$): `XX`, `XY`, `XZ`, `YY`, `YZ`, `ZZ`
- `F` ($\ell = 3$): `XXX`, `XXY`, `XXZ`, `XYY`, `XYZ`, `XZZ`, `YYY`, `YYZ`, `YZZ`, `ZZZ`

Molden Order

The `molden` order is primarily found in a Molden format and only has a determined values for $0 \leq \ell < 4$.

- `S` ($\ell = 0$): *1*
- `P` ($\ell = 1$): `X`, `Y`, `Z`
- `D` ($\ell = 2$): `XX`, `YY`, `ZZ`, `XY`, `XZ`, `YZ`
- `F` ($\ell = 3$): `XXX`, `YYY`, `ZZZ`, `XYY`, `XXY`, `XXZ`, `XZZ`, `YZZ`, `YYZ`, `XYZ`

1.3.2 Spherical Order

gau2grid currently supports both the `CCA` and `gaussian` orders. The numnber of components per angular momentum can be computed as:

$$N_{\text{spherical}} = 2\ell + 1$$

CCA Order

An industry standard order known as the Common Component Architecture:

- `S` ($\ell = 0$): Y_0^0
- `P` ($\ell = 1$): Y_1^{-1}, Y_1^0, Y_1^{+1},
- `D` ($\ell = 2$): Y_2^{-2}, Y_2^{-1}, Y_2^0, Y_2^{+1}, Y_2^{+2}

Gaussian Order

The `gaussian` order as used by the Gaussian program:

- `S` ($\ell = 0$): Y_0^0
- `P` ($\ell = 1$): Y_1^0, Y_1^{+1}, Y_1^{-1},
- `D` ($\ell = 2$): Y_2^0, Y_2^{+1}, Y_2^{-1}, Y_2^{+2}, Y_2^{-2}

Python Interface

- *Collocation Example*
- *API Reference*

1.4 API Reference

`gau2grid.`**`collocation`**(*xyz*, *L*, *coeffs*, *exponents*, *center*, *grad=0*, *spherical=True*, *out=None*, *cartesian_order='cca'*, *spherical_order='cca'*)

Computes the collocation matrix for a given gaussian basis of the form:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Where for a given angular momentum ℓ, components m range from $+\ell$ to $-\ell$ for each grid point p.

This function uses a optimized C library as a backend.

Parameters

- **`xyz`** (*array_like*) – The `(3, N)` cartesian points to compute the grid on
- **`L`** (*int*) – The angular momentum of the gaussian
- **`coeffs`** (*array_like*) – The coefficients of the gaussian
- **`exponents`** (*array_like*) – The exponents of the gaussian
- **`center`** (*array_like*) – The cartesian center of the gaussian
- **`grad`** (*int, optional (default: 0)*) – Can return cartesian gradient and Hessian per point if requested.
- **`spherical`** (*bool, optional (default: True)*) – Transform the resulting cartesian gaussian to spherical
- **`out`** (*dict, optional*) – A dictionary of output NumPy arrays to write the data to.

Returns Returns a dictionary containing the requested arrays (`PHI`, `PHI_X`, `PHI_XX`, etc). Where each matrix is of shape `(ngaussian_basis x npoints)`

Return type dict of array_like

`gau2grid.`**`collocation_basis`**(*xyz*, *basis*, *grad=0*, *spherical=True*, *out=None*, *cartesian_order='cca'*, *spherical_order='cca'*)

Computes the collocation matrix for a given gaussian basis of the form:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Where for a given angular momentum ℓ, components m range from $+\ell$ to $-\ell$ for each grid point p.

This function uses a optimized C library as a backend.

xyz [array_like] The `(3, N)` cartesian points to compute the grid on

basis [list of dicts] Each dict should contain the following keys (L, coeffs, exponents, center).

L [int] The angular momentum of the gaussian

coeffs [array_like] The coefficients of the gaussian

exponents [array_like] The exponents of the gaussian

center [array_like] The cartesian center of the gaussian

grad [int, default=0] Can return cartesian gradient and Hessian per point if requested.

spherical [bool, default=True] Transform the resulting cartesian gaussian to spherical

out [dict, optional] A dictionary of output NumPy arrays to write the data to.

Returns Returns a dictionary containing the requested arrays (`PHI`, `PHI_X`, `PHI_XX`, etc). Where each matrix is of shape (ngaussian_basis x npoints)

Return type dict of array_like

gau2grid.**orbital**(*orbs*, *xyz*, *L*, *coeffs*, *exponents*, *center*, *spherical=True*, *out=None*, *cartesian_order='cca'*, *spherical_order='cca'*)

Computes a array of a given orbital on a grid for a given gaussian basis of the form:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Where for a given angular momentum ℓ, components m range from $+\ell$ to $-\ell$ for each grid point p.

This function uses a optimized C library as a backend.

Parameters

- **orbitals** (*array_like*) – The `(norb, nval)` section of orbitals.
- **xyz** (*array_like*) – The `(3, N)` cartesian points to compute the grid on
- **L** (*int*) – The angular momentum of the gaussian
- **coeffs** (*array_like*) – The coefficients of the gaussian
- **exponents** (*array_like*) – The exponents of the gaussian
- **center** (*array_like*) – The cartesian center of the gaussian
- **spherical** (*bool, optional (default: True)*) – Transform the resulting cartesian gaussian to spherical
- **out** (*dict, optional*) – A dictionary of output NumPy arrays to write the data to.

Returns Returns a `(norb, N)` array of the orbitals on a grid.

Return type array_like

gau2grid.**orbital_basis**(*orbs*, *xyz*, *basis*, *spherical=True*, *out=None*, *cartesian_order='cca'*, *spherical_order='cca'*)

Computes a array of a given orbital on a grid for a given gaussian basis of the form:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Where for a given angular momentum ℓ, components m range from $+\ell$ to $-\ell$ for each grid point p.

orbital [array_line] A `(norb, nao)` orbital array aligned to the orbitals basis

xyz [array_like] The (3, N) cartesian points to compute the grid on

basis [list of dicts] Each dict should contain the following keys (L, coeffs, exponents, center).

L [int] The angular momentum of the gaussian

coeffs [array_like] The coefficients of the gaussian

exponents [array_like] The exponents of the gaussian

center [array_like] The cartesian center of the gaussian

spherical [bool, default=True] Transform the resulting cartesian gaussian to spherical

out [dict, optional] A dictionary of output NumPy arrays to write the data to.

Returns Returns a `(norb, N)` array of the orbitals on a grid.

Return type array_like

1.5 Collocation Example

1.5.1 Single Collocation

A collocation grid between a single basis and a Cartesian grid can be computed with the *collocation()* function. For example, we will use a grid starting at the origin along the `z` axis:

```
>>> import gau2grid
>>> import numpy as np
>>> xyz = np.zeros((3, 5))
>>> xyz[2] = np.arange(5)
```

We can then create a gaussian with only a single coefficient and exponent of 1 centered on the origin:

```
>>> L = 0
>>> coef = [1]
>>> exp = [1]
>>> center = [0, 0, 0]
```

The collocation grid can then be computed as:

```
>>> ret = gau2grid.collocation(xyz, L, coef, exp, center)
>>> ret["PHI"]
[[  1.00000e+00   3.67879e-01   1.83156e-02   1.23409e-04   1.12535e-07]]
```

The `p` gaussian can be also be computed. Note that since our grid points are along the `z` axis, the `x` and `y` components are orthogonal and thus zero.

```
>>> L = 1
>>> ret = gau2grid.collocation(xyz, L, coef, exp, center, spherical=False, grad=1)
>>> ret["PHI"]
[[  0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00]  # P_x
 [  0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00]  # P_y
 [  0.00000e+00   3.67879e-01   3.66312e-02   3.70229e-04   4.50140e-07]] # P_z
```

As the previous execution used `grad=1`, the `X`, `Y`, and `Z` cartesian gradients are also available and can be accessed as:

```
>>> ret["PHI_Z"]
[[  0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00]
 [  0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00]
 [  1.00000e+00  -3.67879e-01  -1.28209e-01  -2.09797e-03  -3.48859e-06]]
```

1.5.2 Basis Collocation

Often it is beneficial to compute the collocation matrix between several basis functions and a set of grid points at once the *collocation_basis()* helper function provides this functionality. To begin, a set of basis sets can be constructed with the following form:

```
>>> basis = [{
    'center': [0., 0., 0.],
    'exp': [38, 6, 1],
    'coef': [0.4, 0.6, 0.7],
    'am': 0
}, {
    'center': [0., 0., 0.],
    'exp': [0.3],
    'coef': [0.3],
    'am': 1
}]
```

Execution of this basis results in a collocation matrix where basis results are vertically stacked on top of each other:

```
>>> ret = gau2grid.collocation_basis(xyz, basis, spherical=False)
>>> ret["PHI"]
[[  1.70000e+00   2.59003e-01   1.28209e-02   8.63869e-05   7.87746e-08]  # S
 [  0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00]  # P_x
 [  0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00]  # P_y
 [  0.00000e+00   2.22245e-01   1.80717e-01   6.04850e-02   9.87570e-03]] # P_z
```

C Interface

- *Collocation Example*
- *API Reference*

1.6 API Reference

1.6.1 Helper Functions

A collection of function ment to provide information and the gau2grid library.

int **gg_max_L**();
Returns the maximum compiled angular momentum

int **gg_ncomponents**(**const** int *L*, **const** int *spherical*)
Returns the number of components for a given angular momentum.

Parameters

- **L** – The angular momentum of the basis function.
- **spherical** – Boolean that returns spherical (1) or cartesian (0) basis representations.

The following enums are also specified:

- `GG_SPHERICAL_CCA` - CCA spherical output.
- `GG_SPHERICAL_GAUSSIAN` - Gaussian spherical output.
- `GG_CARTESIAN_CCA` - CCA cartesian output.
- `GG_CARTESIAN_MOLDEN` - Molden cartesian output.

1.6.2 Transpose Functions

Transposes matrices if input or output order is incorrect.

void gg_naive_transpose(unsigned long n, unsigned long m, const double* PRAGMA_RESTRICT inp
Transposes a matrix using a simple for loop.

Parameters

- **n** – The number of rows in the input matrix.
- **m** – The number of rows in the output matrix.
- **input** – The `(n x m)` input matrix.
- **output** – The `(m x n)` output matrix.

void gg_fast_transpose(unsigned long n, unsigned long m, const double* PRAGMA_RESTRICT inpu
Transposes a matrix using a small on-cache temporary array. Is usually faster than `gg_naive_transpose()`.

Parameters

- **n** – The number of rows in the input matrix.
- **m** – The number of rows in the output matrix.
- **input** – The `(n x m)` input matrix.
- **output** – The `(m x n)` output matrix.

1.6.3 Orbital Functions

Computes orbitals on a grid.

void gg_orbitals(int L, const double* PRAGMA_RESTRICT C, const unsigned long norbitals, con
Computes orbital a section on a grid. This function performs the following contraction inplace.

$$C_{im}\phi_{mp} \rightarrow ret_{ip}$$

This is often more efficient than generating ϕ_{mp} and then contracting with the orbitals C as there is greater cache locality.

Parameters

- **L** – The angular momentum of the basis function.
- **C** – A `(norbitals, ncomponents)` matrix of orbital coefficients.
- **norbitals** – The number of orbs to compute.
- **npoints** – The number of grid points to compute.
- **xyz** – A `(npoints, 3)` or (npoints, n) array of the xyz coordinates.

- **xyz_stride** – The stride of the xyz input array. 1 for `xx..., yy..., zz...` style input, 3 for `xyz, xyz, xyz, ...` style input.
- **nprim** – The number of primitives (exponents and coefficients) in the basis set
- **coeffs** – A `(nprim, )` array of coefficients (c).
- **exponents** – A `(nprim, )` array of exponents (α).
- **center** – A `(3, )` array of x, y, z coordinate of the basis center.
- **order** – Enum that specifies the output order.
- **orbital_out** – `(norbitals, npoints)` array of orbitals on the grid.

1.6.4 Collocation Functions

Creates collocation matrices between a gaussian function and a set of grid points.

void gg_collocation(int L, const unsigned long npoints, const double* PRAGMA_RESTRICT xyz,

Computes the collocation array:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Parameters

- **L** – The angular momentum of the basis function.
- **npoints** – The number of grid points to compute.
- **xyz** – A `(npoints, 3)` or (npoints, n) array of the xyz coordinates.
- **xyz_stride** – The stride of the xyz input array. 1 for `xx..., yy..., zz...` style input, 3 for `xyz, xyz, xyz, ...` style input.
- **nprim** – The number of primitives (exponents and coefficients) in the basis set
- **coeffs** – A `(nprim, )` array of coefficients (c).
- **exponents** – A `(nprim, )` array of exponents (α).
- **center** – A `(3, )` array of x, y, z coordinate of the basis center.
- **order** – Enum that specifies the output order.
- **phi_out** – `(ncomponents, npoints)` collocation array.

void gg_collocation_deriv1(int L, const unsigned long npoints, const double* PRAGMA_RESTRIC

Computes the collocation array and the corresponding first cartesian derivatives:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Parameters

- **L** – The angular momentum of the basis function.
- **npoints** – The number of grid points to compute.
- **xyz** – A `(npoints, 3)` or (npoints, n) array of the xyz coordinates.
- **xyz_stride** – The stride of the xyz input array. 1 for `xx..., yy..., zz...` style input, 3 for `xyz, xyz, xyz, ...` style input.
- **nprim** – The number of primitives (exponents and coefficients) in the basis set

- **coeffs** – A `(nprim, )` array of coefficients (c).
- **exponents** – A `(nprim, )` array of exponents (α).
- **center** – A `(3, )` array of x, y, z coordinate of the basis center.
- **order** – Enum that specifies the output order.
- **phi_out** – `(ncomponents, npoints)` collocation array.
- **phi_x_out** – `(ncomponents, npoints)` collocation derivative with respect to `x`.
- **phi_y_out** – `(ncomponents, npoints)` collocation derivative with respect to `y`.
- **phi_z_out** – `(ncomponents, npoints)` collocation derivative with respect to `z`.

void gg_collocation_deriv2(int L, const unsigned long npoints, const double* PRAGMA_RESTRIC

Computes the collocation array and the corresponding first and second cartesian derivatives:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Parameters

- **L** – The angular momentum of the basis function.
- **npoints** – The number of grid points to compute.
- **xyz** – A `(npoints, 3)` or (npoints, n) array of the xyz coordinates.
- **xyz_stride** – The stride of the xyz input array. 1 for `xx..., yy..., zz...` style input, 3 for `xyz, xyz, xyz, ...` style input.
- **nprim** – The number of primitives (exponents and coefficients) in the basis set
- **coeffs** – A `(nprim, )` array of coefficients (c).
- **exponents** – A `(nprim, )` array of exponents (α).
- **center** – A `(3, )` array of x, y, z coordinate of the basis center.
- **order** – Enum that specifies the output order.
- **phi_out** – `(ncomponents, npoints)` collocation array.
- **phi_x_out** – `(ncomponents, npoints)` collocation derivative with respect to `x`.
- **phi_y_out** – `(ncomponents, npoints)` collocation derivative with respect to `y`.
- **phi_z_out** – `(ncomponents, npoints)` collocation derivative with respect to `z`.
- **phi_xx_out** – `(ncomponents, npoints)` collocation derivative with respect to `xx`.
- **phi_xy_out** – `(ncomponents, npoints)` collocation derivative with respect to `xy`.
- **phi_xz_out** – `(ncomponents, npoints)` collocation derivative with respect to `xz`.
- **phi_yy_out** – `(ncomponents, npoints)` collocation derivative with respect to `yy`.
- **phi_yz_out** – `(ncomponents, npoints)` collocation derivative with respect to `yz`.
- **phi_zz_out** – `(ncomponents, npoints)` collocation derivative with respect to `zz`.

void gg_collocation_deriv3(int L, const unsigned long npoints, const double* PRAGMA_RESTRIC

Computes the collocation array and the corresponding first, second, and third cartesian derivatives:

$$\phi_{mp} = Y_\ell^m \sum_i c_i e^{-\alpha_i |\phi_{\text{center}} - p|^2}$$

Parameters

- **L** – The angular momentum of the basis function.
- **npoints** – The number of grid points to compute.
- **xyz** – A `(npoints, 3)` or (npoints, n) array of the xyz coordinates.
- **xyz_stride** – The stride of the xyz input array. 1 for `xx..., yy..., zz...` style input, 3 for `xyz, xyz, xyz, ...` style input.
- **nprim** – The number of primitives (exponents and coefficients) in the basis set
- **coeffs** – A `(nprim, )` array of coefficients (c).
- **exponents** – A `(nprim, )` array of exponents (α).
- **center** – A `(3, )` array of x, y, z coordinate of the basis center.
- **order** – Enum that specifies the output order.
- **phi_out** – `(ncomponents, npoints)` collocation array.
- **phi_x_out** – `(ncomponents, npoints)` collocation derivative with respect to `x`.
- **phi_y_out** – `(ncomponents, npoints)` collocation derivative with respect to `y`.
- **phi_z_out** – `(ncomponents, npoints)` collocation derivative with respect to `z`.
- **phi_xx_out** – `(ncomponents, npoints)` collocation derivative with respect to `xx`.
- **phi_xy_out** – `(ncomponents, npoints)` collocation derivative with respect to `xy`.
- **phi_xz_out** – `(ncomponents, npoints)` collocation derivative with respect to `xz`.
- **phi_yy_out** – `(ncomponents, npoints)` collocation derivative with respect to `yy`.
- **phi_yz_out** – `(ncomponents, npoints)` collocation derivative with respect to `yz`.
- **phi_zz_out** – `(ncomponents, npoints)` collocation derivative with respect to `zz`.
- **phi_xxx_out** – `(ncomponents, npoints)` collocation derivative with respect to `xxx`.
- **phi_xxy_out** – `(ncomponents, npoints)` collocation derivative with respect to `xxy`.
- **phi_xxz_out** – `(ncomponents, npoints)` collocation derivative with respect to `xxz`.
- **phi_xyy_out** – `(ncomponents, npoints)` collocation derivative with respect to `xyy`.
- **phi_xyz_out** – `(ncomponents, npoints)` collocation derivative with respect to `xyz`.
- **phi_xzz_out** – `(ncomponents, npoints)` collocation derivative with respect to `xzz`.
- **phi_yyy_out** – `(ncomponents, npoints)` collocation derivative with respect to `yyy`.
- **phi_yyz_out** – `(ncomponents, npoints)` collocation derivative with respect to `yyz`.
- **phi_yzz_out** – `(ncomponents, npoints)` collocation derivative with respect to `yzz`.

- **phi_zzz_out** – (ncomponents, npoints) collocation derivative with respect to zzz.

1.7 Collocation Example

1.7.1 Single Basis Functions

A collocation grid between a single basis and a Cartesian grid can be computed with the `gg_collocation()` function. For example, we will use a grid starting at the origin along the `z` axis and a `S` shell at the origin:

```
#include <stdio.h>
#include "gau2grid.h"

int main() {
    // Generate grid
    long int npoints = 5;
    double xyz[15] = {0, 0, 0, 0, 0, // x components
                      0, 0, 0, 0, 0}; // y components
                      0, 1, 2, 3, 4}; // z components
    long int xyz_stride = 1; // This is a contiguous format

    // Gaussian data
    int nprim = 1;
    double coef[1] = {1};
    double exp[1] = {1};
    double center[3] = {0, 0, 0};
    int order = GG_CARTESIAN_CCA; // Use cartesian components

    double s_output[5] = {0};
    gg_collocation(0,                                   // The angular momentum
                   npoints, xyz, xyz_stride,            // Grid data
                   nprim, coef, exp, center, order,     // Gaussian data
                   s_output);                           // Output

    // Print output to stdout
    for (int i = 0; i < npoints; i += 1) {
        printf("%lf  ", s_output[i]);
    }
    printf("\n");
}
```

The resulting output should be:

```
1.000000  0.367879  0.018316  0.000123  0.000000
```

For higher angular momentum functions that output size should `ncomponents x npoints` in size. Where each component is on a unique row or the `X` component starts at position `0`, the `Y` component starts at position `5`, and the `Z` component starts at position `10` as out grid is of length `5`. See *Gaussian Component Orders* for more details or order output.

The xyz input shape can either be organized contiguously in each dimension like the above or packed in a xyz, xyz, ... fashion. If the `xyz_stride` is not 1, the shape refers to the strides per row. For example, if the data is packed as xyzw, xyzw, ... (where w could be a DFT grid weight) the `xyz_stride` should be 4.

```
long int xyz_stride = 3;
double xyz[15] = {0, 0, 0,
                  0, 0, 1,
                  0, 0, 2,
                  0, 0, 3,
                  0, 0, 4}; // xyz, xyz, ... format


gg_collocation(0,                                  // The angular momentum
               npoints, xyz, xyz_stride,           // Grid data
               nprim, coef, exp, center, order,    // Gaussian data
               s_output);                          // Output
```

1.7.2 Multiple Basis Functions

Often collocation matrices are computed for multiple basis functions at once. The below is an example of usage:

```
#include <stdio.h>
#include "gau2grid.h"

int main() {
    // Generate grid
    long int npoints = 5;
    double xyz[15] = {0, 0, 0, 0, 0, // x components
                      0, 0, 0, 0, 0}; // y components
                      0, 1, 2, 3, 4}; // z components
    long int xyz_stride = 1;

    // Gaussian data
    int nprim = 1;
    double coef[1] = {1};
    double exp[1] = {1};
    double center[3] = {0, 0, 0};
    int order = GG_SPHERICAL_CCA; // Use cartesian components

    // Size ncomponents * npoints, (1 + 3 + 5) * 5
    double output[45] = {0};
    int row = 0;
    for (int L = 0; L < 3; L++) {
        gg_collocation(L,                                  // The angular momentum
                       npoints, xyz, xyz_stride            // Grid data
                       nprim, coef, exp, center, order,    // Gaussian data
                       output + (row * npoints));          // Output, shift pointer

        row += gg_ncomponents(L, spherical); // Increment rows skipped
    }

    // Print out by row
    for (int i = 0; i < row; i += 1) {
        for (int j = 0; j < npoints; j += 1) {
            printf("%lf  ", output[i * npoints + j]);
        }
        printf("\n");
    }
}
```

The resulting output should be:

```
1.000000  0.367879  0.018316  0.000123  0.000000 // S
0.000000  0.367879  0.036631  0.000370  0.000000 // P_0
0.000000  0.000000  0.000000  0.000000  0.000000 // P^+_0
0.000000  0.000000  0.000000  0.000000  0.000000 // P^-_0
0.000000  0.367879  0.073263  0.001111  0.000002 // D_0
0.000000  0.000000  0.000000  0.000000  0.000000 // D^+_1
0.000000  0.000000  0.000000  0.000000  0.000000 // D^-_1
0.000000  0.000000  0.000000  0.000000  0.000000 // D^+_2
0.000000  0.000000  0.000000  0.000000  0.000000 // D^-_2
```

INDEX