

# Molecular symmetry

---

Symmetrization of molecules and molecular orbitals

**Author: Marcus Johansson**

Supervisors: Valera Veryazov, Per-Olof Widmark

Department of Theoretical Chemistry, Lund University

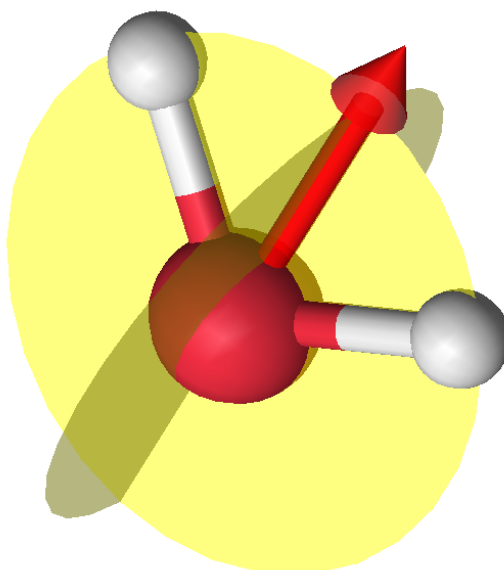
## Abstract

A new code for automatic detection of point groups and for symmetrization of molecular geometry and wavefunctions is presented. Performance and accuracy improvements to a previously designed algorithm for point group determination and an algorithm for symmetrization of said point groups, using linear transformations onto predefined coordinates is described. The new algorithm can find the 120 symmetrized operations of  $I_h$  in  $C_{720}$  in under 30 ms. An algorithm for symmetrization of molecules using projections onto an  $n$ -dimensional basis is described, as well as an algorithm for determining the projection operators and subspaces of the irreducible representations for the symmetry adapted linear combinations of atomic orbitals. An algorithm for symmetrization of molecular orbitals is also described. Code was implemented and integrated into the Molcas quantum chemistry software [1], as well as the Luscus graphical molecular modeling software [2].

## Introduction

Symmetry is something we grow up with. The bilateral symmetry of the human face is one of the first things we see, and symmetry is everywhere in nature. Just like most animals, we have a sense of up and down. This is due to gravity, and all parts of your body want to fall down towards the earth. If we had two left feet we would not keep our balance as well giving a clear evolutionary advantage to people with a mirror plane through their nose. In physics and chemistry symmetry can be just as important as having a left and a right foot.

In general symmetry is a property of an object or function, which means that it is invariant under a transformation and a symmetry operation is one of these transformations. When some point remains stationary under all symmetry operations in a group this is called a point group [3]. Take water as an example as shown in Figure 1. The two mirror planes and rotation by half a turn will keep the molecule looking the same, and the center of mass stationary. In fact there is no easy way of differentiating the two hydrogens - they are symmetrically equivalent.



**Figure 1: Water molecule and symmetry elements**

If some property of the above water molecule, such as the wavefunction describing the electron orbitals, is calculated without symmetry, the hydrogens will not be considered equivalent. This means that any slight deviation between the two will be used, and these deviations will be present in the final result. This

kind of deviation is not uncommon when dealing with data from measurements or arithmetic with limited precision.

The wavefunction is often largely determined by the symmetry of the nuclei [4] and functions, just like objects can have symmetry.

Consider the function shown in Figure 2:

$$\int_{-a}^a x^2 \sin(x)$$

This function has symmetry in two dimensions,  $x^2$  is an even function, which can be seen as having a mirror line in the y-axis, and  $\sin(x)$  is an odd function which can be considered inversion in the point  $\{0,0\}$ .

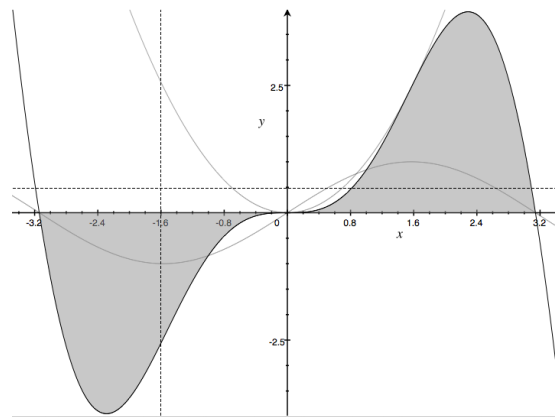


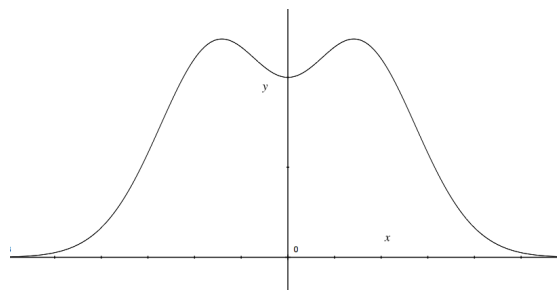
Figure 2:  $\int_{-\pi}^{\pi} x^2 \sin(x)$  in dark grey,  $x^2$  and  $\sin(x)$  in light grey

Rather than calculating the integral above, the symmetry of these two functions can be used to show that such an integral is zero. The same principle can be applied to three or higher dimensions, with a larger set of symmetry operations. If the functions being integrated are constructed in such a way that they are either orthogonal or they belong to a specific representation, then many integrals can simply be set to zero, fewer calculations are required and there is less room for errors resulting from the way computers operate.

Sometimes the symmetry of the wavefunction can be different from the symmetry of the nucleus, e.g. antiferromagnetic ordering of spins [5]. In this case the desired symmetry is a subgroup of the symmetry of the nuclei. In other situations the system is metastable in the symmetric case and this symmetry



should be preserved. Figure 3 is an illustration of an energy curve of the electron configuration for some hypothetical calculation.



**Figure 3: Energy curve for metastable system**

The symmetric solution would lie on the y-axis, whereas the lowest energy is clearly far out to the sides. If the system jumps over one of the humps, it will proceed to break symmetry by falling down one of the sides. Symmetrizing the function during the calculation can prevent this from happening. While the symmetric solution may not always be the desired or indeed correct result, when it is, an algorithm for determining and enforcing this symmetry can be a valuable tool for quantum chemists.

Symmetry can be used in many aspects of quantum chemistry. It can be used to determine the most likely shape of a molecule, to improve the accuracy and speed of calculations or to avoid symmetry breaking when determining molecular orbitals of certain types of molecules.

Symmetrization of a molecule can help get rid of measurement errors before running expensive calculations and determining the subspaces of irreducible representations can give a quick indication of what orbitals will overlap to form molecular bonds. Symmetrization of molecular orbitals can help improve accuracy and avoid symmetry breaking prior to or during a calculation.

## **Aim**

The goal of this project is to allow for the use of full symmetry during molecular quantum calculations. This is to be accomplished by first adapting and refining previously developed symmetry detection software in order to determine the

symmetry. The full symmetry is then to be used for adapting the atomic coordinates and/or molecular orbitals used during calculations.

## Use of symmetry in quantum chemistry

There are several ways to approach the problem of determining the symmetry elements in molecules. Some make use of graph theory and ideal geometries [6] even in an arbitrary number of dimensions [7], others use interatomic distance matrices [8] and geometry determined by moments of inertia [9]. Algorithms often use exhaustive search to determine symmetry elements, making use of some of the above theories as well as the concept of equivalence sets [10] to narrow a search field [11]. This approach often restricts [11] the search to some maximum of all possible symmetry operations as there are infinitely many.

This work makes use of many of the concepts described above for determining symmetry elements. Weighted interatomic distances are used in equivalence set determination, although statistical information and projections are used instead of euclidian degree partitioning [8] allowing for better scaling with large molecules. Moments of inertia and geometry are also used to determine possible symmetry elements. Exhaustive search is used only on a small subset of all symmetry operations in cubic point groups, where they are already known to be present. This allows for the detection of any point group in three dimensions such as  $D_{71h}$  and better scaling with large molecules.

Determining the symmetry elements will lead to the determination of a point group [10], and in some cases subgroups can be detected and used rather than the highest order group [11] [12]. Molecules and point group can then be aligned according to some predefined axes if so desired [12].

This work uses point group alignment information for both symmetrization of the point group itself and to allow for setting another point group instead of the highest order group detected. This allows for the choosing of a lower order group, as well as a higher order group, in case full symmetry detection fails. It also allows for alignment of molecule and point group.

There are also several ways to symmetrize molecules, such as choosing symmetry unique atoms and generating the rest [11].

This work uses projection into a totally symmetric irreducible representation for symmetrization of molecules. This solution is not as fast as generating the elements, but gives the closest possible symmetric version of the molecule.

Constructing symmetry adapted orbitals can e.g. be accomplished by successive application of a projection operator on atomic orbitals [10] using character tables.

This work also uses predefined character tables to construct a projection operator. A vector representation of polynomials describing atomic orbitals is used to construct the symmetry operations for the atomic orbitals in an equivalence set. This is a non-iterative approach, where the same symmetry operation can be applied to all atomic orbitals of the same azimuthal quantum number ( $l$ ).

Visualization of symmetry elements can also be accomplished in several ways. Some have built in libraries of molecules and symmetry elements used to convey the different point groups for educational purposes [13] [14].

This work provides an interactive visualization of symmetry elements, as well as generating and symmetrizing molecules of an arbitrary point group.

## Symmetry operations

All symmetry operations belonging to a point group will leave the molecule seemingly unchanged and the corresponding symmetry elements will intersect in at least one point [3]. The types of possible symmetry operations are proper rotation ( $\hat{C}_n$ ) with the corresponding symmetry element  $\vec{C}_n$ , reflection ( $\hat{\sigma}$ ) for which the symmetry element is a plane represented by its normal vector  $\vec{\sigma}$ , inversion ( $\hat{i}$ ) through a point and improper rotation ( $\hat{S}$ ) which also has a vector  $\vec{S}_n$  representing the symmetry element [15]. A graphical representation of the symmetry operations is shown in Figure 4. Future mentions of symmetry operations will refer to one of the above unless explicitly stated otherwise.

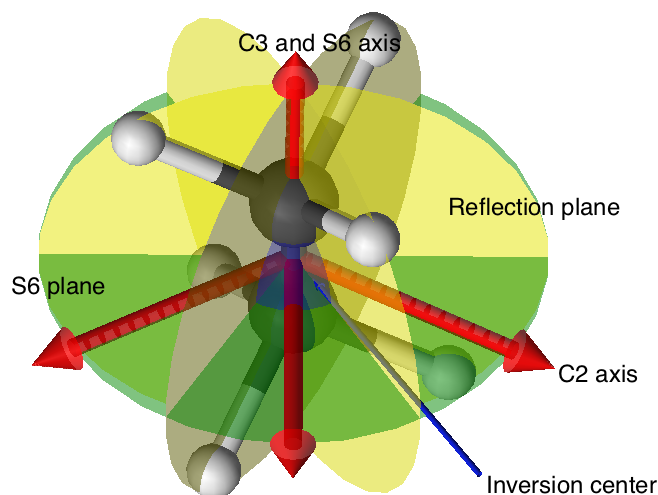


Figure 4: Staggered Ethane, point group  $D_{3d}$

## Equivalence sets

A symmetry operation cannot change interatomic distances, the distance from the center of mass or the type of element being operated on. In order to handle large molecules, it is split into equivalence sets prior to determining any symmetry operations. The algorithm for determining equivalence uses interatomic weighted distances, and well as plane and spherical projections in order to determine what elements are considered equal in subsequent calculations.

If we construct a matrix with the interatomic distances  $\mathbf{D}$  it will be invariant under any symmetry operation [8], where  $\mathbf{D}_{ij} = |\vec{\mathbf{d}}_{ij}|$ ,  $\vec{\mathbf{d}}_{ij} = \vec{\mathbf{a}}_i - \vec{\mathbf{a}}_j$ . This means that we can classify atoms into disjoint equivalence sets  $\mathbf{E}_k$ , where all atoms belong to exactly one set [15]:

$$\mathbf{E}_k \subseteq \mathbf{A}; \bigcup_k \mathbf{E}_k = \mathbf{A}; \mathbf{E}_k \cap \mathbf{E}_l = \emptyset, \forall k \neq l$$

Two atoms belong to the same set  $\mathbf{E}_k$  if the rows  $\mathbf{D}_i, \mathbf{D}_j$  corresponding to atoms  $a_i$  and  $a_j$  are permutations of one another.

This matrix will also be invariant under any symmetry operation and instead of storing the matrix we calculate  $\langle |\vec{\mathbf{d}}_{ij}| \rangle$  and  $\langle |\vec{\mathbf{d}}_{ij}|^2 \rangle$  weighted by the reduced mass for each  $i$  from which we can calculate the mean and standard deviation of each row, where the reduced mass  $\mu = \frac{m_i m_j}{m_i + m_j}$ .

All atoms  $a_j \in A \setminus \{a_i\}$ , as well as copy of  $a_i$  at the center of mass, are projected onto a unit sphere around  $a_i$ , scaled with the reduced mass and summed according to [15]:

$$\vec{p}_i^s = \sum_j \mu \frac{\vec{d}_{ij}}{|\vec{d}_{ij}|}, \quad \begin{cases} \vec{d}_{ij} = \vec{a}_j - \vec{a}_i, & i \neq j \\ \vec{d}_{ii} = \vec{v}_{cm} - \vec{a}_i, & i = j \end{cases}$$

They are also projected onto the plane that has  $\vec{a}_i$  as a normal vector, scaled and added in the same manner according to [15]:

$$\vec{p}_i^p = \sum_j \mu \left( \vec{a}_j - \frac{\vec{a}_i \cdot \vec{a}_j}{|\vec{a}_i|^2} \vec{a}_i \right)$$

where  $\vec{v}_{cm} = \vec{0}$ .

$|\vec{p}_i^s|$ ,  $|\vec{p}_i^p|$ ,  $\langle |\vec{d}_{ij}| \rangle$  and  $\langle |\vec{d}_{ij}|^2 \rangle$  form a four dimensional array that can be used to determine equality [15].

Any element in these sets should be symmetry unique, but in asymmetric molecules where high thresholds are needed, the above algorithm may not be adequate to determine this partitioning. After point group determination, these sets are split by a second equivalence set algorithm, with point group information available. This algorithm simply checks the permutations generated from a symmetry operation as splits the set into the disjoint sets resulting from applying the symmetry operations of the group.

## Finding symmetry operations

The symmetry operations are found by first determining the overall geometry of an equivalence set using the eigenvalues, and eigenvectors of the inertial tensor ( $I$ ).

The principal moments of inertia are the eigenvalues ( $\lambda_i, i \in \{0,1,2\}$ ) of  $I$  and the principal axes the corresponding eigenvectors ( $v_i^{prim}, i \in \{0,1,2\}$ ):

$$I = \sum_{a_i \in A} m_{a_i} ((\vec{a}_i \cdot \vec{a}_i) \mathbf{E} - \vec{a}_i \otimes \vec{a}_i)$$

Where  $m_{a_i}$  is the mass of atom  $a_i \in A$ ,  $A$  is the set of all atoms,  $\vec{a}_i$  is the Cartesian coordinate vector of atom  $a_i$  with the origin at the center of mass,  $\mathbf{E}$  is the identity matrix and  $\otimes$  is the outer product [15].

The symmetry operations are then generated based on this geometry and the principal axes, where the geometry based on the eigenvalues are [9]:

$\lambda_0 = \lambda_1 = \lambda_2$	→ spherical (cubic)
$\lambda_0 = 0; \lambda_1 = \lambda_2$	→ linear
$\lambda_0 = \lambda_1; \lambda_2 = \lambda_0 + \lambda_1$	→ planar regular
$\lambda_0 < \lambda_1; \lambda_2 = \lambda_0 + \lambda_1$	→ planar irregular
$\lambda_0 = \lambda_1 < \lambda_2$	→ polyhedral oblate
$\lambda_0 < \lambda_1 = \lambda_2$	→ polyhedral prolate
$\lambda_0 < \lambda_1 < \lambda_2$	→ asymmetric polyhedron

There are a limited number of symmetry operations for each type of geometry depending on the number of elements in each set.

For cubic point groups the eigenvalues are triply degenerate and the eigenvectors give no indication of any axes for symmetry operations. The algorithm instead looks for  $\hat{C}_2$ ,  $\hat{C}_4$  or  $\hat{\sigma}$  operations on equidistant pairs of elements. Once one operation of the above types has been found, only pairs with this distance are checked. The remaining operations can be generated by looking at angles between these symmetry operations.

## Equivalence set intersection

After the symmetry operations of each equivalence set have been determined, they need to be reduced into one set of operations that are present in all groups. The reduction algorithm constructs, adds and removes symmetry operations based on previously detected operations intersected with any new set. The adding and construction of new symmetry operations are only done when there are  $\hat{C}_\infty$  axes present. This, since any linear sets cannot be represented by their infinite number of symmetry elements but rather only with the  $\hat{C}_\infty$  axis. Any other symmetry operation needs to be present in all sets in order to be kept.

## Point groups

The method to determine the point group from a small set of symmetry elements is well established [3] and this procedure is followed after the symmetry elements have been found.

## Symmetrizing point groups

The symmetry operations generated when determining the symmetry of a molecule are only symmetrical to within some specified threshold. In order to make accurate calculations using this point group, the symmetry elements themselves need to be symmetrized. This is done by first determining a primary and secondary axis for symmetrization. These axes are those of the largest abelian subgroup if it exists, or the primary axis of rotation, and a  $\vec{C}_2/\vec{\sigma}$  otherwise.

For icosahedral symmetry the secondary axis is also determined by its proximity to the  $\vec{C}_5$  axes in the plane for which the primary axis is a normal vector.

The transformation matrix for aligning these to axes with the z and x axes is calculated using:

$$\mathbf{T} = \mathbf{E} + \mathbf{M}_s + \left( \frac{1 - \vec{v} \cdot \vec{w}}{|\vec{v} \times \vec{w}|^2} \right) \mathbf{M}_s^2$$

where  $\mathbf{T}$  is the transformation matrix,  $\mathbf{M}_s$  is the skew symmetric matrix of  $\vec{v} \times \vec{w}$ ,  $\vec{v}$  is the normalized vector to align, and  $\vec{w}$  is the normalized vector to align to. If  $\vec{v}$  and  $\vec{w}$  are parallel  $\mathbf{T}$  is the just  $\mathbf{E}$  or a rotation of  $\pi$  around any axis in  $\vec{w}^\perp$ .

All symmetry operations for the point group are then generated, and aligned using the inverse of the above transformation. This also means that the requirement for e.g. five perpendicular  $\vec{C}_2$  axes in a  $D_{5h}$  can be relaxed, since any missing  $\vec{C}_2$  axes will be generated during symmetrization.

## Point group generation

Each symmetry operation in the point group needs to be generated in order for this to be a group. The symmetry detection algorithm does not e.g. include any powers of symmetry operations, and may fail to find all of them. The starting point is the minimal set of generators for this group, constructed from the subgroups. All powers of the rotations are then generated using

$$\hat{C}_n^k = \hat{C}_{n'}^{k'}$$

where  $n' = \frac{n}{gcd(k,n)}$ ,  $k' = \frac{k \bmod n}{gcd(k,n)}$ ,  $k < n$  otherwise  $k = k \bmod n$  and  $\hat{C}_n^0 = \hat{E}$

After this, any implied improper rotations are generated using reflection planes and rotations. The powers of improper rotations are generated using:

$$\begin{aligned}
\hat{S}_n^k &= \hat{E} && \text{when } k_{C'} = 0 \text{ and } n \text{ is even} \\
\hat{S}_n^k &= \hat{\sigma} && \text{when } k_{C'} = 0 \text{ and } n \text{ is odd} \\
\hat{S}_n^k &= \hat{S}_{n_{C'}}^{(k_{C'}+n) \bmod 2n_{C'}} && \text{when } k > n, k \text{ is odd and } n \text{ is odd} \\
\hat{S}_n^k &= \hat{S}_{n_{C'}}^{k_{C'}} && \text{when } k \text{ is odd} \\
\hat{S}_n^k &= \hat{C}_{n_{C'}}^{k_{C'}} && \text{when } k \text{ is even} \\
\hat{S}_2^k &= \hat{i} && \text{from resulting calculations}
\end{aligned}$$

where  $n_{C'}$  and  $k_{C'}$  are the order and exponent of the resulting power of the proper rotation as above and  $k < 2n$  otherwise  $k = k \bmod 2n$

Powers of  $\hat{i}$  and  $\hat{\sigma}$  are simply  $\hat{i}$  and  $\hat{\sigma}$  respectively if the exponent is odd, or  $\hat{E}$  if even.

Finally, applying proper rotations to all symmetry elements generates the remaining operations.

## Matrix representation of symmetry operations

A symmetry operation can be represented in matrix form. The most familiar representation would be a transformation of the three dimensional Cartesian coordinate system, where a rotation of angle theta around the z axis is:

$$\hat{O} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This representation uses the basis of the Cartesian coordinate system:  $\{\vec{i} \vec{j} \vec{k}\}$  where  $\vec{i} = [1 \ 0 \ 0]$ ;  $\vec{j} = [0 \ 1 \ 0]$ ;  $\vec{k} = [0 \ 0 \ 1]$  and any vector resulting from this transformation will be a linear combination of the original basis.

This same rotation can be represented in any other basis. One such basis is the 1s-orbitals of hydrogens in ammonia. The nitrogen will be ignored for now, as its s-orbitals are not transformed by any of the symmetry operations in the point group  $C_{3v}$  to which it belongs, and p-orbitals require transformation of the functions themselves.

If we set the angle  $\theta = \frac{2\pi}{3}$  this rotation can be represented as a permutation matrix:



$$\hat{O} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where the basis is now  $\{s_{H_1} s_{H_2} s_{H_3}\}$ .

Similarly any symmetry operation ( $\hat{O}$ ) can be represented as a transformation matrix of a linearly independent basis where the resulting vectors will span the same space.

## Irreducible representations

An irreducible representation is a decomposition of an operation into a direct sum, such that no further reductions can be performed. In the above case the symmetry operations can be represented as:

$$\hat{O} = \hat{O}^{(1)} \oplus \hat{O}^{(2)}$$

where  $\oplus$  is a direct sum. A similarity transform resulting in such a decomposition for all symmetry operations yields the irreducible representations of the entire group.

Using the example of ammonia above, a new basis constructed by a linear combination:

$$s_1 = \frac{s_{H_1} + s_{H_2} + s_{H_3}}{\sqrt{3}}; s_2 = \frac{2s_{H_1} - s_{H_2} - s_{H_3}}{\sqrt{6}}; s_3 = \frac{s_{H_2} - s_{H_3}}{\sqrt{2}}$$

Since one of the irreducible representations is degenerate, there are infinitely many valid linear combinations for  $s_2$  and  $s_3$ . The normalized basis above is illustrated in Figure 5. All symmetry operations in this basis are in block diagonal form, i.e. they can be represented as a direct sum of a one-dimensional and a two-dimensional representation, which are called  $A_1$  and  $E$  respectively. The  $E$  representation cannot be reduced further, and so this is an irreducible representation of  $C_{3v}$  in this basis.

The names  $A_1$  and  $E$  are standardized names for types of representations, where the dimensionalities of irreducible representations  $A, B, E, T, G$  and  $H$  are 1, 1, 2, 3, 4 and 5.

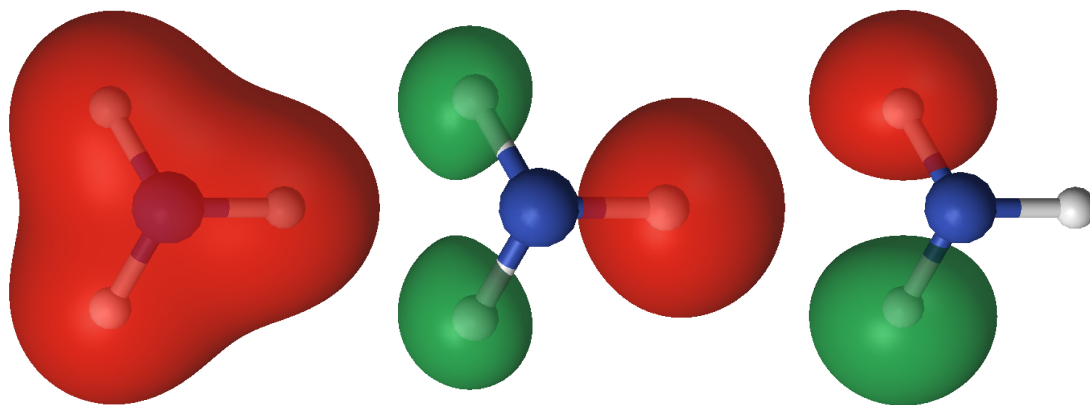


Figure 5: Normalized symmetry adapted basis of 1s orbitals on hydrogen in ammonia.

This new basis, can be represented as a set of vectors that span some subspaces  $V_{\Gamma_i}$  of the entire space. In the example above there are three dimensions, so they can be thought of as our standard three dimensions spanned by  $\{s_{H_1}, s_{H_2}, s_{H_3}\}$ . In the irreducible representation, the same space is spanned by  $\{s_1, s_2, s_3\}$ . The  $A_1$  representation is a one-dimensional subspace  $V_{\Gamma_{A_1}}$  so can be thought of as a line, while the E representation is two-dimensional subspace  $V_{\Gamma_E}$ , so can be thought of as a plane in three dimensions. Any symmetry operation only transforms these functions within the subspace they span e.g.  $\hat{C}_3$  will take any vector in  $V_{\Gamma_{A_1}}$  to another vector (it will be the same vector) in  $V_{\Gamma_{A_1}}$  whereas the same rotation applied to a vector in  $V_{\Gamma_E}$  will result in any vector in  $V_{\Gamma_E}$  i.e. in the same plane. Figure 6 shows a more familiar 3-dimensional representation of these subspaces. It is the same basis as in Figure 5 only represented in a different way.

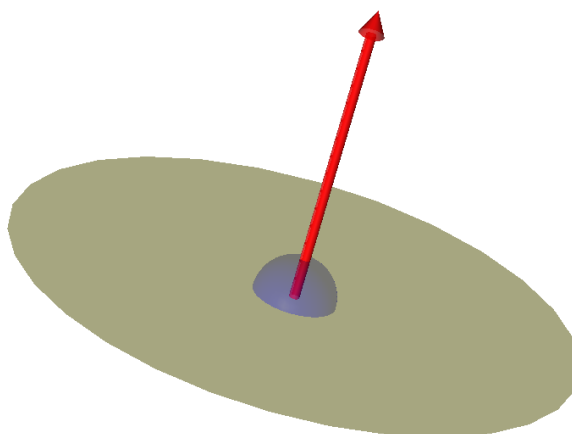


Figure 6: Symmetry adapted 1s orbitals of ammonia represented as vector and plane

With higher dimensions this is harder to visualize, but the principle is the same.

## Character of symmetry operations

The character ( $\chi$ ) of a symmetry operation is the trace of the matrix when represented in the basis that spans an irreducible representation. This character is invariant under a similarity transformation [3] and can therefore be placed in table form in order to generate projection operations for the irreducible representations. These tables are called character tables, and the  $C_{3v}$  character table, the point group of ammonia, is shown in Table 1:

**Table 1: Character table for the  $C_{3v}$  point group**

$C_{3v}$	$1\hat{E}$	$2\hat{C}_3$	$3\hat{\sigma}_v$
$A_1$	1	1	1
$A_2$	1	1	-1
E	2	-1	0

Where the rows are the irreducible representations and the columns the symmetry operations, with the prefixed number representing the number of operations in this class.

## Symmetry operation classes

Symmetry operations  $\hat{O}$  and  $\hat{O}'$  belong to the same class if there exists an operation  $\hat{T}$  in the group such that:

$$\hat{O}' = \hat{T}^{-1}\hat{O}\hat{T}$$

Since this is a similarity transform,  $\hat{O}'$  and  $\hat{O}$  are similar matrices. All symmetry operations of the same class have the same character [3]. The class can be calculated using the above equation or using the permutations of the symmetry operations.

## Projection operator

The projection operator for projecting into the subspace  $V_{\Gamma_i}$  can be constructed using [3]:

$$P^{\Gamma_i} = \frac{d_{\Gamma_i}}{h} \sum_{\hat{\sigma}} \chi^{\Gamma_i}(\hat{\sigma}) \hat{\sigma}$$

Where  $\hat{\sigma}$  is the symmetry operation represented in the bases being transformed,  $h$  is the order of the group  $d_{\Gamma_i}$  is the dimension and  $\chi^{\Gamma_i}(\hat{\sigma})$  is the character of  $\hat{\sigma}$  in the irreducible representation  $\Gamma_i$ . Since the symmetry operations are represented as matrices and,  $d_{\Gamma_i}$ ,  $h$  and  $\chi^{\Gamma_i}(\hat{\sigma})$  are all known from the character tables this is simply a sum of weighted matrices. This equation also implies that a projection operator can be constructed to project into any of the irreducible representations regardless what basis is being used, as long as the symmetry operations are constructed for this basis. The same principle would apply to symmetrizing a molecule, molecular orbitals or vibrational modes, but not the point group itself, as it defines the operations.

## Permutations

The permutations of each symmetry operation on an equivalence set is required in order to build the projection operator. The simplest way to find the permutation is simply to apply the symmetry operation to each atom in the set and compare the resulting vectors. Since the equivalence sets are already limited in size, this is the approach used.

A representative permutation for the entire point group can be found in a similar fashion, by applying each symmetry operation to all others. The closure property of the group guarantees that the result is also member of the point group, which provides a consistency check for the point group generation.

*The below algorithm for dealing with alignment of orbitals in different equivalence sets has yet to be implemented.*

These permutations can be used to classify the symmetry operations, as well as finding bijections or surjections between elements or cycles respectively for each equivalence set. E.g. in 1,1,1-Trifluoroethane a reflection plane may have the form  $(H_1)(H_2 H_3)$  for the hydrogens and  $(F_2)(F_1 F_3)$  for the fluorines, and we

can make a bijection  $H_1 \rightarrow F_2$ , and similarly for the remaining atoms using the other symmetry operations. If we use the permutation of the point group as a domain we can similarly make a surjection in this case.

If the size of the equivalence set is equal to the order of the point group there are several possible bijections. One way to solve this is to look the lengths of the combined vectors that form a cycle (of length 2 in the reflection example).

## Symmetrizing molecules

There are several way to symmetrize a molecule, and more than the final version were implemented. One approach is to simply choose an atom from each equivalence set and generate the rest. This requires that atoms are first projected onto axes or planes that they are close to [11]. The choice of starting atom may affect the result in this approach. The final choice was to project the molecule into the totally symmetric space spanned by the Cartesian coordinates on each atom. This is in effect a least squares approximation of a line in  $n$ -dimensional space. The permutations are already available, so the additional calculations required are not computationally intensive. The component of the set not in the totally symmetric space can be obtained during the calculation, providing a good indication of measurement errors.

In case of linear molecules, there is technically an infinite set of symmetry operations, which means that the projection takes the form of a projection onto the  $\vec{C}_\infty$  axis in addition to the projection based on the other symmetry operations.

## Atomic orbitals

Atomic orbitals are one-electron wavefunctions that are the solution to the Schroedinger equation for an electron around a nucleus. These generally come in the form [3]:

$$R_{nl}Y_{lm_l}$$

where  $R_{nl}$  are radial functions and  $Y_{lm_l}$  are the spherical harmonics. For the purpose of this topic, only the real components of linear combinations of the spherical harmonics are considered:

$$\varphi = P_{lm_l}f(r) = \begin{cases} P_{l0}(z)f(r), & m_l = 0 \\ P_{l|m_l|}(z)((x + iy)^{|m_l|} + (x - iy)^{|m_l|})f(r), & m_l > 0 \\ -iP_{l|m_l|}(z)((x + iy)^{|m_l|} - (x - iy)^{|m_l|})f(r), & m_l < 0 \end{cases}$$

where  $P_{lm_l}$  is the resulting polynomial  $P_{l|m_l|}(z)$  is related to the associated Legendre polynomial [16] and  $f(r)$  is a radial function.

The radial part of the wavefunction can in principle be ignored, since any calculation will be run on one equivalence set and orbitals with the same principal and azimuthal quantum numbers. The result is a representation of the  $l$  and  $m_l$  quantum numbers as a polynomial  $P_{lm_l}$  in  $\{x y z\}$  and  $n$  as a simple index, as any symmetry operation will only transform an  $n, l, m_l$  orbital into a linear combination of  $n, l, m_l'$  orbitals.

This set of orbitals form the linearly independent basis for symmetrization.

## Subspace generation for irreducible representations

In order to obtain the projection operator for an irreducible representation, the symmetry operations for the basis need to be determined. This projection operator can then be used find the vectors that span  $V_{\Gamma_i}$ .

### Transformation of atomic orbital functions

The symmetry operations for purely radial wavefunctions are simply the permutation matrix, since any symmetry operation applied to the function leaves it identical. The symmetry operations for any wavefunction with  $l > 0$  however requires a transformation of the function itself, in addition to the permutation. This transformation is obtained using:

$$\hat{O}_l = AS_p^T \hat{O}_3^{\otimes l} S_p$$

Where  $\hat{O}_l$  is the symmetry operation for azimuthal  $l$  on a central atom,  $S_p$  is a matrix whose column vectors  $(\vec{p}_i)$  span a  $2l + 1$  dimensional subspace  $V_p \in \mathbb{R}^{3^l}$  and are the coefficients in  $P_{lm_l}$  of the monomials in the trinomial expansion of  $(x + y + z)^l$ ,  $\hat{O}_3$  is the symmetry operation matrix in the standard Cartesian basis,  $\otimes l$  indicated the  $l$ th Kronecker power, and  $A = (S_p^T S_p)^{-1}$ .

E.g. for  $d$ -orbitals the polynomials  $\vec{p}_{z^2}$  and  $\vec{p}_{xy}$  would be:

$$\vec{p}_{z^2} = \frac{1}{2\sqrt{3}} \begin{bmatrix} \underbrace{-1}_{x^2} & \underbrace{0}_{xy} & \underbrace{0}_{xz} & \underbrace{0}_{yx} & \underbrace{-1}_{y^2} & \underbrace{0}_{yz} & \underbrace{0}_{zx} & \underbrace{0}_{zy} & \underbrace{2}_{z^2} \end{bmatrix}$$

$$\vec{p}_{xy} = \begin{bmatrix} \underbrace{0}_{x^2} & \underbrace{0.5}_{xy} & \underbrace{0}_{xz} & \underbrace{0.5}_{yx} & \underbrace{0}_{y^2} & \underbrace{0}_{yz} & \underbrace{0}_{zx} & \underbrace{0}_{zy} & \underbrace{0}_{z^2} \end{bmatrix}$$

When  $\vec{p}_i$  are orthogonal:

$$A = \text{diag}\left(\frac{1}{|\vec{p}_i|^2}\right)$$

where *diag* indicates a diagonal matrix. In the case of *s*, *p* and *d*-orbitals  $\vec{p}_i$  form an orthonormal basis scaled by a constant factor, so can be normalized, making the  $A = E$ . Only orbitals in one equivalence set need to be considered, since no symmetry operation can transform an orbital outside this space (they can however belong to the same irreducible representation and overlap in the molecular orbitals).

The transformation matrix then becomes:

$$\hat{O} = \hat{P} \otimes \hat{O}_l$$

Where  $\hat{P}$  is the permutation matrix for the symmetry operation when the basis is the coordinate vectors of the elements in the equivalence set.

### Determination of subspaces

The projection operator can be written in the form:

$$S_{\Gamma_i} S_{\Gamma_i}^T$$

where  $S_{\Gamma_i}$  is the  $n \times d$  matrix whose column vectors are the orthonormalized vectors that span the subspace  $V_{\Gamma_i}$  that  $P^{\Gamma_i}$  projects into.

This means that one can obtain the vectors that span all irreducible subspaces by orthogonalizing all  $P^{\Gamma_i}$ . This is achieved using an altered version of Modified Gram Schmidt [17]. Less memory is required in large calculations, when working with this representation.

Which irreducible representations are spanned by these orbitals can be calculated using:

$$\text{rank}(P^{\Gamma_i}) = \frac{d_{\Gamma_i}}{h} \sum_{\hat{o}} \chi^{\Gamma_i}(\hat{o}) \text{tr}(\hat{O})$$

This information is used as a sanity check of the orthogonalization procedure, as these use floating point arithmetic. These subspaces should then be split into

their respective components (see: Future improvements: Symmetrizing degenerate irreducible representations)

### Orthogonalization

The Gram-Schmidt algorithm implemented is the same as Modified Gram-Schmidt with two exceptions, the vectors are only orthogonalized, not normalized, since this is done at a later stage, and the vectors spanning all projection operators are orthogonalized rather than one single matrix, using indexing for each irreducible representation. No weight or metric is used during orthogonalization, since only orbitals of the same type are considered which would result in an overlap with the same symmetry as the nuclei.

### Symmetrizing orbitals

In order for an orbital to be symmetric, it can only have components in one irreducible representation. The choice was made to filter out all but the largest component of any supplied orbital. The projection onto the subspace  $V_{\Gamma_i}$  spanned by an irreducible representation can be represented as a sum, rather than a matrix product:

$$\sum_{j=1}^d (\vec{v}_\varphi \cdot \vec{v}_{\Gamma_{ij}}) \vec{v}_{\Gamma_{ij}}$$

where  $\vec{v}_\varphi$  is the vector of coefficients for any orbital  $\varphi$ ,  $d$  is the dimension of the  $V_{\Gamma_i}$  subspace and  $\vec{v}_{\Gamma_{ij}}$  is the  $j$ th orthonormalized vector spanning  $V_{\Gamma_i}$ . This means that there is no need to reconstruct  $P^{\Gamma_i}$  from  $V_{\Gamma_i}$  calculated earlier.

The algorithm verifies that the number of orbitals projected into each of the irreducible representations match the span of the irreducible representations.

### Implementation

The library was implemented in ANSI C-99. The API uses an opaque context, and requires (and enforces) no memory management from the user. Integration code for the Molcas quantum chemistry software was written in Fortran. The implementation is about 7000 lines of active C code not including whitespace or



comments, about 3500 lines of discarded algorithms, and 2000 lines of integration and test code written in C, Fortran, Julia, Erlang, and Objective-C. The library allows the user to find the equivalence sets and symmetry elements of a molecule and also allows the user to set the point group and alignment axes/transform if previously known, and perform symmetrization using this point group. This allows for the calculation of the point group of a subset of the molecule, acquisition of the alignment transform, and application of a sub/parent group with the same orientation on a larger molecule. It also allows the user to determine symmetrized linear combinations of atomic orbitals and the symmetrization of orbitals according to this symmetry for a subset of all point groups and orbital angular momentums.

### Elements

The library will attempt to fill in missing information about elements, such as mass, nuclear charge and name if not supplied, but will not replace this information if already present. This means that it is possible to supply, e.g. an element called X, set the mass to 18 and the nuclear charge to any number unique for the molecule and replacing water molecules with this element. This would then treat water as a single element resulting in partial symmetry in cases where water molecules are arbitrarily rotated.

### Building molecules

The API allows for the generation of elements based on a point group and a set of starting elements. One application of this is to generate the hydrogens on symmetric proteins. Instead of estimating the position of all hydrogens, only symmetry unique hydrogens are estimated, while the remaining hydrogens can be generated from the point group.

### Thresholds

There are several thresholds used during symmetry detection and symmetrization. This allows for more fine-grained control, and a broader range of detected symmetry elements, but is not as general as what could be used with a search oriented algorithm [11]. Thresholds for equivalence set detection, geometry determination, angles and permutations are relative, whereas the

zero- and orthogonality-thresholds are absolute. The equivalence-threshold is used to determine the equivalence sets prior to symmetry operation detection and the geometry-threshold is used when determining the geometry based on the eigenvalues of the inertial tensor. The angle threshold is used for angles, and the permutation-threshold is used when determining the permutation of a symmetry operation and when generating atoms. The zero-threshold is used to determine the proximity to center of mass, and the orthogonality threshold is used to avoid floating point errors during the Gram Schmidt process to determine orthogonality.

### Orbitals

Orbitals can be assigned to each element and the subspaces for the irreducible representations computed. These components can be acquired in matrix form, sorted by irreducible representation and equivalence set. The library will determine the index of the coefficients for each component based on memory alignment. Any square matrix of components can then be supplied for symmetrization, according to the above algorithm. At this stage, symmetrization means projection into the subspace with the largest component, and as such the implementation cannot make sure the components in degenerate orbitals are equal.

### Limitations

While most algorithms generate all required information during the calculation, there are two parts that are hard-coded. These are the polynomials for the atomic orbitals, and the character tables. This means that currently the only supported orbitals types are s, p and d, and the point groups  $C_{2h}$ ,  $C_{3v}$ ,  $C_{4v}$ ,  $D_{2h}$ ,  $D_{4h}$ ,  $T_d$  and  $I_h$ . This limitation only applies to orbital calculations, symmetry detection and symmetrization of molecules can handle any point group in three dimensions, and test have been run on point groups ranging from  $D_{71h}$  to  $C_2$  as well as all cubic point groups. In order to add more orbitals, the polynomials should either be auto generated or added for each  $l$  and  $m_l$ . In order to add more point groups, the tables should either be auto generated or added to a header file as well as a function table if required. A more detailed description of this procedure can be found in Appendix III.

## Error handling

Every function in the API returns an enumeration error code, where `MSYM_SUCCESS` is no error. A string representation of the error code can be acquired using the `msymErrorString` function, and a detailed description can be acquired using `msymGetErrorDetails`, where the latter is context independent, so will be overwritten regardless of context, and is cleared on calling it. There are currently 94 different error details descriptions, most of which have dynamic information embedded. An example being: *“Elements in symmetric polyhedron do not lie on two parallel circles  $X > Y$  (angle threshold)”*, where  $X$  and  $Y$  are real numbers. This error would either indicate that the angle threshold is too low, or that this equivalence set is not in fact a symmetric polyhedron, meaning that the equivalence set partitioning was incorrect or the geometry thresholds should be changed.

## Integration

A graphical representation of the symmetry elements was implemented in Luscus, with the ability to find, set, and generate point groups, as well as symmetrization and alignment to the xyz axes. The library was integrated to the Molcas software [18]. Example code of how to use the library is included in Appendix I. The wrapper functions for the Molcas software can be seen in Appendix II.

## Results

All of symmetry operations and orbitals shown in this paper have been generated fully or in part using the software described above, with the exception of Figure 17, which is a calculation run without symmetry.

## Molecules

Even though it uses a much slower full permutation check for each symmetry operation, the new cubic point group symmetry detection algorithm is 50-100 times faster than the previous version, much more accurate and can handle sets larger than 120 elements, when proper equivalence set determination fails. Finding and symmetrizing the 120 symmetry operations of a  $C_{720}$  Buckminster fullerene in under 30 ms.

The algorithm can handle relatively large errors as shown in Figure 7 and Figure 8, which show a skewed Benzene molecule, prior to and after symmetrization respectively, as well as the detected symmetry operations of  $D_{6h}$ .

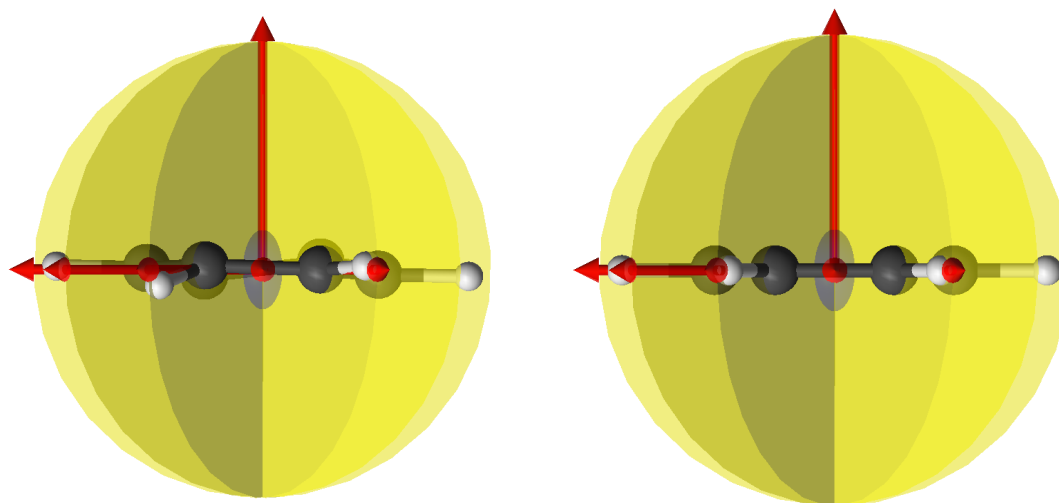


Figure 7: Skew Benzene molecule, with  $D_{6h}$  symmetry elements

Figure 8: Symmetrized Benzene molecule, with  $D_{6h}$  symmetry elements

There is of course a dependency on the shape and size of the molecule, for how large these errors can be. If e.g. there are over 15000 atoms and 1200 equivalence sets in a molecule as seen in Figure 10, the difference between some of the sets will likely be smaller than an average molecule, and if there are 100 symmetry operations, most of which are in a plane, as shown in Figure 9, the angles between axes will be smaller than e.g.  $D_{6h}$  symmetry.

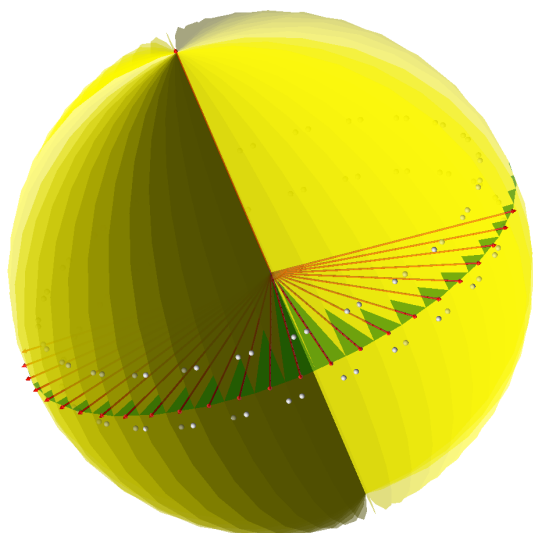


Figure 9: Generated  $D_{25d}$  molecule, with 100 symmetry elements

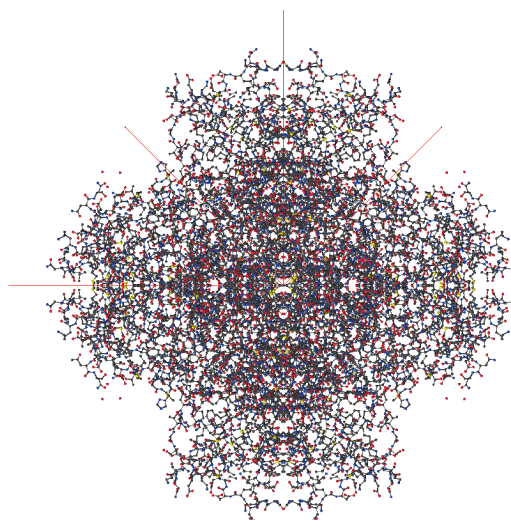


Figure 10: Crystal structure of the type II 3-dehydroquinase, and symmetry elements.

The amino acid phenylalanine has no symmetry, but the phenyl group has approximate symmetry. The entire phenyl group has  $C_{2v}$  symmetry whereas the carbons have  $D_{6h}$  symmetry. Figure 11 and Figure 12 show the amino acid phenylalanine and the detected symmetry operations of the carbons of the phenyl group and the entire phenyl group respectively. These elements can then be symmetrized if so desired.

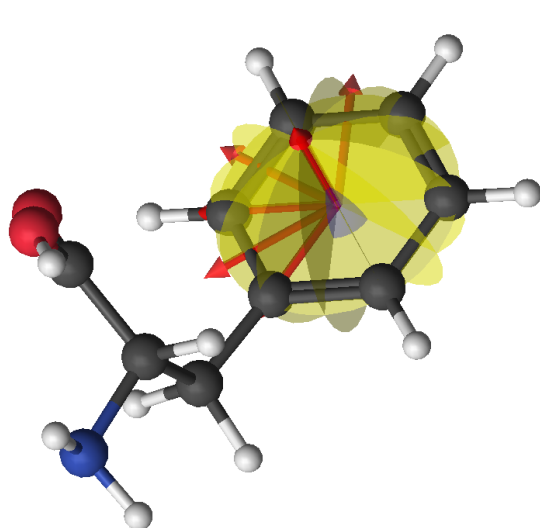


Figure 11: Phenylalanine with  $D_{6h}$  symmetry elements.

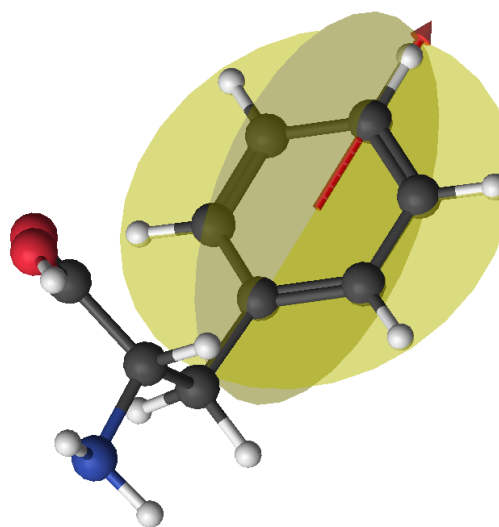


Figure 12: Phenylalanine with  $C_{2h}$  symmetry elements

The thresholds required to detect symmetry can vary significantly, and there are often some intermediate thresholds that will fail to detect any symmetry if there are more than one possible point group. Figure 13 and Figure 14 show two possible point groups for a twisted ferrocene molecule. In Figure 13, the point

group  $D_5$  was detected by setting all thresholds to  $10^{-3}$ , whereas in Figure 14 the point group  $D_{5h}$  was detected by increasing the angle threshold to  $10^{-2}$  and the permutation threshold to  $10^{-0.8}$

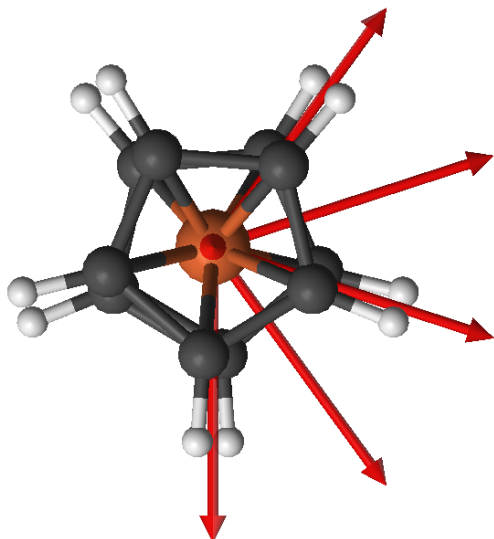


Figure 13: Ferrocene, with detected  $D_5$  symmetry elements

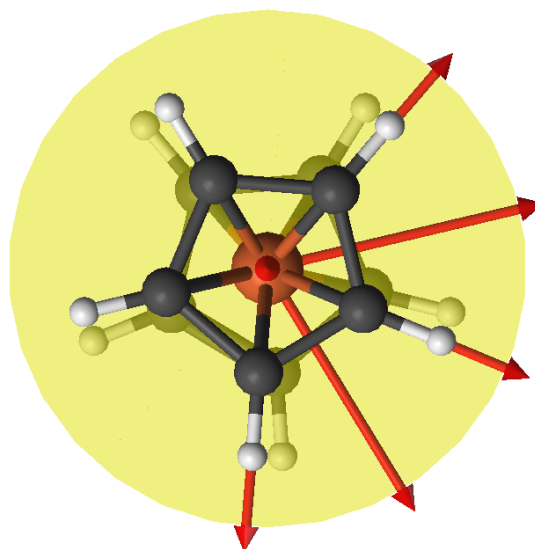


Figure 14: Ferrocene, with detected  $D_{5h}$  symmetry elements

In general, the permutation threshold should be larger than the other thresholds if the molecule is asymmetric, but the library will present information on what part of detection has failed using the `msymGetErrorDetails` API. In the example above, setting the permutation threshold to  $10^{-1}$  will result in the error: “Unable to determine permutation for symmetry operation  $R$ ”, where  $R$  is a  $\hat{\sigma}$  operation. There is no way of generalizing how thresholds should be set for every molecule if they are symmetry broken. In some cases (as above) the angle threshold needs to be adjusted, whereas in Figure 7, all thresholds need to be in the order of  $10^{-1}$ . For ferrocene, the equivalence test will detect the three equivalence sets with a low threshold, but the angles between the hydrogens or carbons when projected onto the plane are clearly above 0.

### Orbitals

It takes about 0.18 seconds to generate the 300 orbital subspaces of a  $C_{60}$  Buckminster fullerene with minimal basis. Since all elements in this molecule belong to one equivalence set, it gives an indication of performance of the algorithm. Had there been several equivalence sets, the subspace generation would be faster since it is only run on one equivalence set at a time, but this

would be a bad test case. Symmetrization of orbitals in the same  $C_{60}$  molecule takes about 0.14 seconds. Figure 15 and Figure 16 are visual representations of one of the vectors in  $A_g$  the vector in  $A_u$  in a  $C_{60}$  molecule respectively with minimal basis.

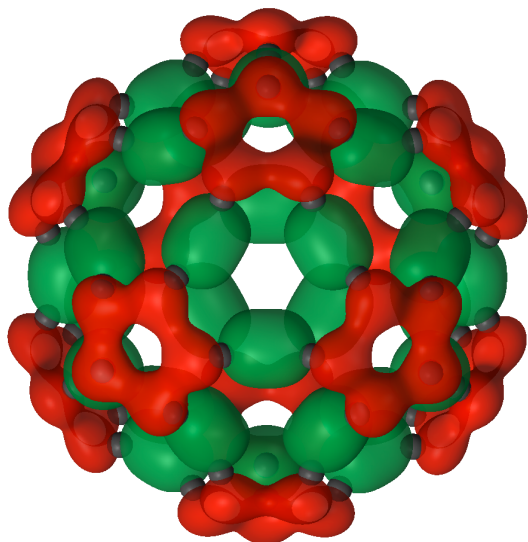


Figure 15: One of the symmetry adapted orbitals in the  $A_g$  irreducible representation formed by the 2p-orbitals in  $C_{60}$  Buckminster fullerene

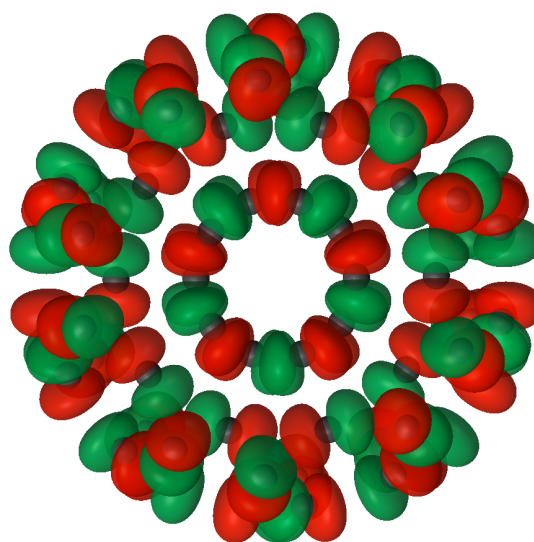


Figure 16: The symmetry adapted orbitals in the  $A_u$  irreducible representation formed by the 2p-orbitals in  $C_{60}$  Buckminster fullerene

Orbital symmetrization was used during restricted Hartree-Fock calculations on dinitrogen tetroxide belonging to the  $D_{2h}$  point group, with symmetry broken input orbitals. Since the calculation will eventually converge on a symmetric solution only one iteration was run in order to test the symmetrization algorithm. Table 2 shows the Mulliken charges per center and basis function type without orbital symmetrization, and Table 3 the corresponding table with orbital symmetrization, after the first iteration. Figure 17 and Figure 18 show one of the resulting orbitals without and with symmetrization respectively. If continued, the calculations will converge in 6 iterations with symmetrization and 10 without.

Table 2: Mulliken charges per center and basis function, without orbital symmetrization

	N1	N2	O3	O4	O5	O6
<b>1s</b>	2.0000	2.0000	2.0003	2.0003	2.0003	2.0003
<b>2s</b>	1.3369	1.3361	1.8595	1.8585	1.8589	1.8589
<b>2px</b>	1.1675	1.1701	1.6764	1.6817	1.6823	1.6809
<b>2pz</b>	1.1464	1.1471	1.4301	1.4235	1.4254	1.4275
<b>2py</b>	0.9900	0.9894	1.2127	1.2148	1.2123	1.2121
<b>Total</b>	6.6408	6.6427	8.1789	8.1788	8.1792	8.1797
<b>N-E</b>	0.3592	0.3573	-0.1789	-0.1788	-0.1792	-0.1797



Table 3: Mulliken charges per center and basis function, with orbital symmetrization

	N1	N2	O3	O4	O5	O6
1s	2.0000	2.0000	2.0003	2.0003	2.0003	2.0003
2s	1.3365	1.3365	1.8590	1.8590	1.8590	1.8590
2px	1.1688	1.1688	1.6803	1.6803	1.6803	1.6803
2pz	1.1468	1.1468	1.4266	1.4266	1.4266	1.4266
2py	0.9897	0.9897	1.2130	1.2130	1.2130	1.2130
Total	6.6418	6.6418	8.1791	8.1791	8.1791	8.1791
N-E	0.3582	0.3582	-0.1791	-0.1791	-0.1791	-0.1791

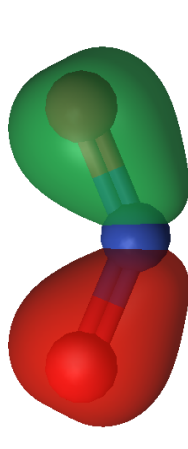


Figure 17: Dinitrogen tetroxide orbital without symmetrization

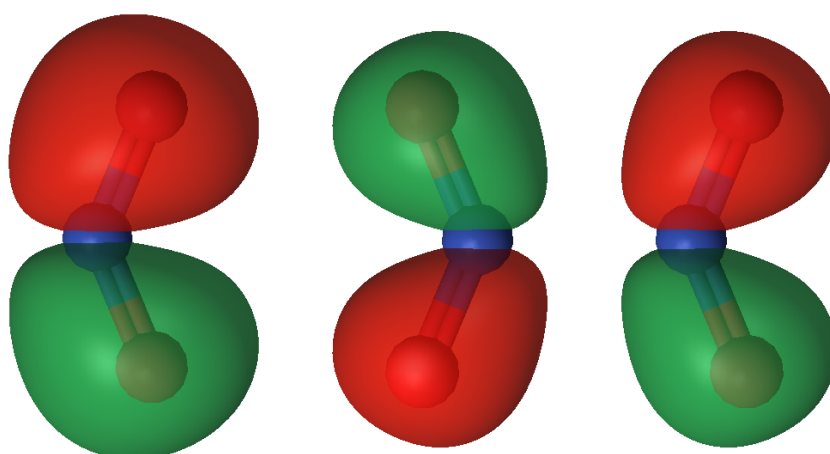


Figure 18: Dinitrogen tetroxide orbital with symmetrization

Figure 17 shows the show the full valence calculation for an  $\text{Fe}_2\text{F}_5$  molecule without symmetrization turned on, and Table 5 the same calculation using symmetry. Without symmetry this calculation does not converge.

Table 4:  $\text{Fe}_2\text{F}_5$  full valence calculation, without symmetry

	F1	FE2	F3	F4	F5	F6	FE7
Charge	-0.5237	1.4803	-0.6035	-0.6075	-0.6035	-0.6075	1.4655
Covalence	0.6340	1.6376	0.6267	0.6217	0.6267	0.6217	1.6236
Valence	0.9291	2.5105	0.9934	0.9933	0.9934	0.9933	2.4871

Table 5:  $\text{Fe}_2\text{F}_5$  full valence calculation, with symmetry

	F1	FE2	F3	F4	F5	F6	FE7
Charge	-0.5233	1.4678	-0.6031	-0.6031	-0.6031	-0.6031	1.4678
Covalence	0.6357	1.6381	0.6277	0.6277	0.6277	0.6277	1.6381
Valence	0.9301	2.4999	0.9937	0.9937	0.9937	0.9937	2.4999



## Future improvements

### Symmetry detection

When detecting e.g. a  $D_{5h}$  molecule as seen in Figure 14, the  $C_2$  axes could be created using averaging instead of relying on the mirror plane when thresholds are high. This would allow for a lower permutation threshold at the later stages of the algorithm

### Character tables

The algorithm cannot currently handle complex characters in character tables, this should either be implemented, or linear combinations of the components could be generated so as to avoid using complex numbers.

The character tables and atomic orbital polynomials should be generated rather than constant, so it can handle any molecule like the rest of the library. If auto generation of character tables is not implemented it should be redesigned to reduce the memory used by the current implementation, since while easy to read it is not memory efficient.

### Orbital projection

Instead of just verifying the span during orbital symmetrization, the algorithm could try to project the orbitals into the same span, by calculating which projection would result in the largest components as a whole.

### Symmetrizing degenerate irreducible representations

*Some of the below algorithms have been partially implemented, but none of the implementations were complete and/or general enough to include in this version.*

The components of degenerate irreducible representations need to be equal, which requires the linear transforms of the subspaces to be equal as well. A symmetry operation  $\hat{O}^{\Gamma_i}$  can be represented in the basis of the column space that spans the irreducible representation (this works for a symmetry operation, not an arbitrary operation):

$$\hat{O}^{\Gamma_{i'}} = AS_{\Gamma_i}^T \hat{O}^{\Gamma_i} S_{\Gamma_i}$$

$$A = (S_{\Gamma_i}^T S_{\Gamma_i})^{-1} = \text{diag}\left(\frac{1}{|v_i|^2}\right)$$

where  $S_{\Gamma_i}$  is the matrix whose column vectors span  $V_{\Gamma_i}$  and where  $v_i$  is the  $i$ th vector of  $S_{\Gamma_i}$ . In this case  $A$  is the identity matrix since the spanning vectors can be normalized.

One can then construct an operation ( $\hat{O}'$ ) to represent the symmetry operation  $\hat{O}$ , and find the change of basis matrix ( $T$ ) that satisfies the similarity transform:

$$\hat{O}' = T^{-1}\hat{O}T$$

$T$  can be found using:

$$B_{\hat{O}}B_{\hat{O}'}^{-1}$$

where  $B_{\hat{O}}$  are the eigenbasis of  $\hat{O}$  and  $B_{\hat{O}'}$  the eigenbasis of  $\hat{O}'$

This poses a problem for degeneracy larger than two as the eigenvectors of the representative symmetry operations are complex. This is not a problem in general since  $T$  is a real matrix, just with the fact that the current implementation does not handle complex eigen-factorization, nor finding similarity transforms without eigenfactorization.

Another approach is to use:

$$\text{kern}(\hat{O}'^T \otimes I - I \otimes \hat{O})$$

and a third to look for the block diagonal Jordan form of  $\hat{O}$  since a Jordan form of  $\hat{O}'$  can be generated beforehand.

While this may give us aligned components of each irreducible representation, it is not however enough information in order to average the components of a set of arbitrary orbitals projected onto this subspace, which is required if symmetry is not to be broken.

In order to average the components a similarity transform that produces the irreducible representations of each symmetry operation in the basis of the vectors that span the irreducible representation for the entire equivalence set is required. I.e. the E representation spanned by the  $p$ -orbitals on the hydrogens in ammonia when using a Valence Double-Zeta plus Polarization (VDZP) basis, will be 6-dimensional composed of 3 separate 2-dimensional subspaces, and should be reduced to these 3 representations meaning each symmetry operation in the basis of the E representation should be block diagonal with at most 2 dimensional blocks.

One approach to find these is to look at the orbitals on each atom, and project it onto the irreducible representations of the components on a single atom. In the example above this would split the 6-dimensional space into one 2- and one 4-dimensional space. The  $p_z$ -orbitals forming the components of the 2-dimensional space (assuming the molecule is aligned to the z-axis).

Once the subspaces are acquired, the basis of the incoming orbitals in  $\Gamma_E$  ( $Q_{\Gamma_E}$ ) is changed to that of the vectors spanning  $V_{\Gamma_E}$ :

$$S_{\Gamma_E}^T Q_{\Gamma_E}$$

The  $d \times d$  ( $2 \times 2$  in example above) block is then orthonormalized, scaled back up to the average of the individual vectors in that block, and changed back to the original basis. This approach has the drawback that it requires the incoming orbitals to be grouped in order of degeneracy. E.g. there cannot be two subsequent orbitals that have the same eigenvalue for  $\hat{\sigma}$  (assuming they are eigenfunctions to that reflection plane).

This should pose less of a problem if the incoming orbitals are close to symmetric, since they will have close to identical energies and can be ordered by energy, but symmetrizing arbitrary orbitals will not work.

Yet another approach is to project onto a subgroup with no degeneracy if such a subgroup exists, and use the resulting subspaces to split the n-dimensional subspaces. This will of course be limited to groups that have non-degenerate subgroups.

### Implementation

Parameters should be declared as constant and/or static when appropriate in order to allow for better compiler optimization, and some naming inconsistencies should be fixed.

### Alternative use cases

Just like symmetrizing molecules or orbitals, it's possible to symmetrize some linear transformations that are applied to a symmetry unique element or orbital. For instance, applying a translation to a symmetry unique element implies applying a translation to all elements in the same equivalence set in order not to break symmetry, this could be a useful feature for geometry optimization,

molecular modeling or simply for educational purposes, when it comes to understanding the concept of a symmetry unique element. Using the same principle for orbitals could be another approach to enforcing symmetry during a calculation.

## Conclusion

In this work improvements to the algorithms for detection of symmetry elements and point groups are described. The algorithms perform better than the previous implementations [15] in terms of time complexity and accuracy. The time required for detection of the  $I_h$  point group of a  $C_{720}$  molecule has been reduced from 400 to 30 ms. The algorithms for symmetrization of the point group elements and molecules is also described. The results show that these algorithms can symmetrize molecules that are visibly asymmetric. Algorithms described in this work for symmetrization of molecular orbitals include the generation of subspaces for irreducible representations, using the basis of atomic orbitals. The symmetrization algorithm uses projection in order to symmetrize molecular orbitals, as well as the span of the irreducible representations for verification. The results show that this method is sufficient for symmetrizing non-degenerate irreducible representations but is not sufficient when dealing with higher dimensions, due to the requirement of equality between components. Proposals for improvements in this area are also described. Code was implemented and integrated into the Molcas quantum chemistry software, as well as the Luscus graphical molecular modeling software, and can be used for real calculations as well as for educational purposes.

## Acknowledgements

I'd like to thank my supervisors Valera Veryazov and Per-Olof Widmark for giving me the opportunity to work on such an interesting subject, and for the guidance throughout this project. I'd like to thank Maria Yakovleva for supporting me during this time. Thank you to the department of theoretical chemistry and everyone who helped me on this project.

## References

- [1] F. Aquilante et al., "MOLCAS 7: the next generation," *Journal of Computational Chemistry*, vol. 31, pp. 224-247, 2010.
- [2] G. Kovacevič and V. Veryazov. Luscus. [Online].  
<http://luscus.sourceforge.net/>
- [3] P. Atkins and R. Friedman, *Molecular Quantum Mechanics*, 5th ed.: Oxford University Press, 2011.
- [4] R. McWeeny, *Methods of Molecular Quantum Mechanics*, 2nd ed., 1992.
- [5] N. W. Achcroft and I. Merlin, *Solid State Physics*.: W. B. Saunders Company, 1976.
- [6] J. Ivanov and G. Schüürmann, "Simple Algorithms for Determining the Molecular Symmetry," *J. Chem. Inf. Comput. Sci.*, pp. 728-737, 1999.
- [7] M. J. Lawrence. symfind: Symmetry Detection Software. [Online].  
<http://www.gang.umass.edu/~lawrence/symfind-documentation/>
- [8] K. Balasubramanian, "Computer Perception of Molecular Symmetry," *J. Chem. Inf. Comput. Sci.*, vol. 35, no. 4, pp. 761-770, 1995.
- [9] O. Beruski and L. N. Vidal, "Algorithms for computer detection of symmetry elements in molecular systems," *Journal of Computational Chemistry*, vol. 35, no. 4, pp. 290-299, 2013.
- [10] C. Leforestier and O. Kahn, "A computer program to determine the molecular point group and the symmetry adapted orbitals," *Computers & Chemistry*, vol. 1, no. 1, pp. 13-19, 1976.
- [11] R. J. Largent, W. F. Polik, and J. R. Schmidt, "Symmetrizer: Algorithmic determination of point groups in nearly symmetric molecules," *Journal of Computational Chemistry*, vol. 33, no. 19, pp. 1637-1642, 2012.
- [12] Gaussian Inc. Gaussian 09 user's reference: Symmetry. [Online].  
[http://www.gaussian.com/g\\_tech/g\\_ur/k\\_symmetry.htm](http://www.gaussian.com/g_tech/g_ur/k_symmetry.htm)
- [13] D. H. Johnston. Symmetry @ Otterbein. [Online].  
<http://symmetry.otterbein.edu/gallery/index.html>
- [14] Molwave. (2008) 3DMolSym: Interactive Visualization of Symmetry Elements and Operations. [Online].  
<http://www.molwave.com/software/3dmolsym/3dmolsym.htm>
- [15] M. Johansson. (2014, September) LUP Student Papers. [Online].  
<http://lup.lub.lu.se/student-papers/record/4645536>
- [16] E.W. Weisstein. Associated Legendre Polynomial. [Online].  
<http://mathworld.wolfram.com/AssociatedLegendrePolynomial.html>
- [17] E.W. Weisstein. Gram-Schmidt Orthonormalization. [Online].  
<http://mathworld.wolfram.com/Gram-SchmidtOrthonormalization.html>
- [18] V. Veryazov, P.-O. Widmark, L. Serrano-Andres, R. Lindh, and B.O. Roos, "MOLCAS as a development platform for quantum chemistry software," *International journal of Quantum Chemistry*, vol. 100, pp. 626-635, 2004.

## Appendix I

```
int example(const char* in_file){
    msym_error_t ret = MSYM_SUCCESS;
    msym_element_t *elements = NULL;
    msym_orbital_t *orbitals = NULL, **porbitals = NULL;

    const char *error = NULL;
    char point_group[6];
    double cm[3], radius = 0.0, symerr = 0.0;

    /* Do not free these variables */
    msym_element_t *melements = NULL;
    msym_symmetry_operation_t *msops = NULL;
    int msopsl = 0, mlength = 0;

    /* This function reads xyz files.
     * It initializes an array of msym_element_t to 0,
     * then sets the coordinates and name of the elements */
    int length = read_xyz(in_file, &elements);
    if(length <= 0) return -1;

    double (*coefficients)[length] = NULL;

    /* Allocate and initialize memory for orbitals */
    orbitals = calloc(length, sizeof(msym_orbital_t));
    porbitals = calloc(length, sizeof(msym_orbital_t*));

    /* Add a 1s orbital to each atom */
    for(int i = 0; i < length; i++){
        /* You can also just set orbitals[i].n = 1 */
        sprintf(orbitals[i].name, sizeof(orbitals[i].name), "1s");
        porbitals[i] = &orbitals[i];
        elements[i].ao = &porbitals[i];
        elements[i].aol = 1;
    }

    /* Create a context */
    msym_context ctx = msymCreateContext();

    /* Use default thresholds otherwise call:
     * msymSetThresholds(msym_context ctx, msym_thresholds_t *thresholds); */

    /* Set elements and orbitals */
    if(MSYM_SUCCESS != (ret = msymSetElements(ctx, length, elements))) goto err;

    /* These are no longer needed, internal versions of these are kept in the context,
     * They are indexed in the same way that they have been allocated.
     * I.e. during orbital symmetrization or when getting the symmetrized LCAO,
     * the coefficients will correspond to the same indexing as "orbitals",
     * this is the main reason for the two levels of indirection */
    free(elements); elements = NULL;
    free(orbitals); orbitals = NULL;
    free(porbitals); porbitals = NULL;

    /* Some trivial information */
    if(MSYM_SUCCESS != (ret = msymGetCenterOfMass(ctx, cm))) goto err;
    if(MSYM_SUCCESS != (ret = msymGetRadius(ctx, &radius))) goto err;

    printf("Molecule has center of mass [%lf; %lf; %lf] "
           "and a radius of %lf\n", cm[0], cm[1], cm[2], radius);

    /* Find molecular symmetry */
    if(MSYM_SUCCESS != (ret = msymFindSymmetry(ctx))) goto err;

    /* Get the point group name */
    if(MSYM_SUCCESS != (ret = msymGetPointGroup(ctx, sizeof(char[6]), point_group))) goto err;
    printf("Found point group %s\n", point_group);

    /* Set pointgroup to the D2h subgroup if it has Th symmetry
     * using the same alignment as the original.
     * If specific axes are wanted the alignment axes can be set instead
     * And of course you can keep Th if you want =D */
    if(0 == strcmp(point_group, "Th", 2)){
        double transform[3][3];
        if(MSYM_SUCCESS != (ret = msymGetAlignmentTransform(ctx, transform))) goto err;
        if(MSYM_SUCCESS != (ret = msymSetPointGroup(ctx, "D2h"))) goto err;
        if(MSYM_SUCCESS != (ret = msymSetAlignmentTransform(ctx, transform))) goto err;
        if(MSYM_SUCCESS != (ret = msymFindSymmetry(ctx))) goto err;
        if(MSYM_SUCCESS != (ret = msymGetPointGroup(ctx, sizeof(char[6]), point_group))) goto err;
    }

    /* Retrieve the symmetry operations */
    if(MSYM_SUCCESS != (ret = msymGetSymmetryOperations(ctx, &msopsl, &msops))) goto err;
}
```

```

for(int i = 0; i < msopsl;i++){
    if(msops[i].type == PROPER_ROTATION && msops[i].order == 3 && msops[i].power == 1){
        printf("Found a C3^1 axis, YEE!\n");
    }
}

coefficients = malloc(sizeof(double[length][length]));

/* Get the subspaces for this point group (in order of irreducible representation) */
if(MSYM_SUCCESS != (ret = msymGetOrbitalSubspaces(ctx,length,coefficients))) goto err;

/* Mess them up a bit */
for(int i = 0;i < length;i++){
    for(int j = 0;j < length;j++){
        coefficients[i][j] += 0.0001*i - 0.0001*j;
    }
}

/* And symmetrize them */
if(MSYM_SUCCESS != (ret = msymSymmetrizeOrbitals(ctx,length,coefficients))) goto err;
free(coefficients); coefficients = NULL;

/* Symmetrize the molecule.
 * You can do this before orbital symmetrization as well,
 * but the permutations are already built, so you don't need to */
if(MSYM_SUCCESS != (ret = msymSymmetrizeMolecule(ctx, &symerr))) goto err;

printf("Molecule has been symmetrized to point group %s "
       "with an error of %lf\n",point_group, symerr);

if(MSYM_SUCCESS != (ret = msymGetElements(ctx, &mlength, &melements))) goto err;
if(mlength != length){ printf("Not possible!\n"); goto err;}

/* Aligning axes prior to orbital symmetrization will
 * change the orientation of orbitals with l >= 1 */
if(MSYM_SUCCESS != (ret = msymAlignAxes(ctx))) goto err;

printf("New element coordinates:\n%d\n\n",mlength);
for(int i = 0;i < mlength;i++){
    printf("%s %lf %lf %lf\n",
           melements[i].name,
           melements[i].v[0],
           melements[i].v[1],
           melements[i].v[2]);
}

/* Make a new element with the same type as the first one we read */
msym_element_t myelement;
memset(&myelement,0,sizeof(msym_element_t));
myelement.n = melements[0].n;
myelement.v[0] = melements[0].v[0];
myelement.v[1] = melements[0].v[1];
myelement.v[2] = melements[0].v[2];

/* Generate some new elements of the same point group */
if(MSYM_SUCCESS != (ret = msymGenerateElements(ctx,1,&myelement))) goto err;

/* This is not a memory leak, context keeps track of melements,
 * and it should never be freed, msymReleaseContext does this. */
if(MSYM_SUCCESS != (ret = msymGetElements(ctx, &mlength, &melements))) goto err;

printf("Generated element coordinates:\n%d\n\n",mlength);
for(int i = 0;i < mlength;i++){
    printf("%s %lf %lf %lf\n",
           melements[i].name,
           melements[i].v[0],
           melements[i].v[1],
           melements[i].v[2]);
}

msymReleaseContext(ctx);
printf("We're done!\n");

return ret;
err:
free(elements);
free(orbitals);
free(porbitals);
free(coefficients);
error = msymErrorString(ret);
fprintf(stderr,"Error %s: ",error);
error = msymGetErrorDetails();
fprintf(stderr,"%s\n",error);
return ret;
}

```

## Appendix II

- \* All subroutines below will end program execution on error
- \* Create context (must be called before rest of API)  
Call `fmsym_create_context(ctx)`
- \* Set elements and orbitals from runfile  
Call `fmsym_set_elements(ctx)`
- \* Find the symmetry  
Call `fmsym_find_symmetry(ctx)`
- \* Symmetrize molecule. Results will be in `$WorkDir/$Project$SubProject.msym`  
Call `fmsym_symmetrize_molecule(ctx)`
- \* Generate Orbital subspaces. Results will be in `$WorkDir/$Project$SubProject.MSymAOrb`  
Call `fmsym_generate_orbital_subspaces(ctx)`
- \* Symmetrize orbfile. Results will be in `$WorkDir/$Project$SubProject.MSymMOrb`  
Call `fmsym_symmetrize_orb_file(ctx,'INPORB')`
- \* Symmetrize orbitals. Result will override, contents of CMO, row order  
Call `fmsym_symmetrize_orbitals(ctx,CMO)`
- \* Release context (must be called to free memory)  
Call `fmsym_release_context(ctx)`



## Appendix III

In order to add new polynomials for angular momentum (without auto generating them), there is one function and one header file to consider:

```
msym_error_t orbitalPolynomial(int l, int m, double *poly)
```

in orbital.c sets the orbital polynomial, and a case for  $l > 2$  needs to be added.

The polynomials are defined in orbital.h and should be added from  $-l$  to  $l$ . E.g. the polynomial for p orbitals looks like so:

```
static double ppolynomial [[3]] = {
    [0] = {0,1,0}, //Py
    [1] = {0,0,1}, //Pz
    [2] = {1,0,0} //Px
};
```

In order to add a new character table (without auto generating them) there are three files to consider: character\_table.h, character\_table.c and point\_group.c

The function:

```
msym_error_t findCharacterTable(msym_point_group_t *pg)
```

in point\_group.c contains a function map which maps a point group type to a character table function. If the point group type e.g. POINT\_GROUP\_S2n currently maps to characterTableUnknown a proper function should be supplied. If a function pointer is already supplied, a case for the order of the highest rotation should be added in that function, e.g. for  $D_{6h}$  a case for  $n = 6$  is required. The functions are implemented like so:

```
msym_error_t characterTableIh(int n, CharacterTable *ct){
    ct->l = sizeof(IhIrrep)/sizeof(IhIrrep[0]);
    ct->irrep = malloc(ct->l*sizeof(IrreducibleRepresentation));
    for(int i = 0; i < ct->l;i++){
        enum IrreducibleRepresentationEnum irrep = IhIrrep[i];
        ct->irrep[i].name = IrreducibleRepresentationName[irrep];
        ct->irrep[i].v = IhTable[irrep];
        ct->irrep[i].d = Degeneracy[irrep];
        ct->irrep[i].l = sizeof(IhTable[irrep])/sizeof(IhTable[irrep][0]);
    }
    return MSYM_SUCCESS;
}
```

which means that character\_table.h should contain the IhIrrep and IhTable structures, as well as the names and degeneracy of the irreducible representation type if not already present. The characters of the irreducible should match the class of the symmetry operations in the point group. The class for a symmetry operation can be acquired from msym\_symmetry\_operation\_t.class. There is also a function void printSymmetryOperation(msym\_symmetry\_operation\_t \*sop) which will print a symmetry operation including class.