Rust INTDER Manual

Brent R. Westbrook

INTDER is a program for performing coordinate transformations between simple-/symmetry-internal coordinates and Cartesian coordinates. It was originally written in Fortran by Wesley D. Allen and coworkers but has since been translated to Rust by Brent R. Westbrook. This documentation corresponds to the Rust version only.

1 Internal Coordinates

At their most basic, internal coordinates are those defined with reference to other atoms in the same molecule (or system), as opposed to an external set of axes or some other frame of reference. Simple examples of internal coordinates are distances between pairs of atoms and angles between sets of three atoms. A major benefit of using internal coordinates over an external coordinate system like Cartesian coordinates is that internal coordinates require only $3N - 6$ (or $3N - 5$ for linear molecules) coordinates to describe a system fully since the 3 translational and 3 rotational degrees of freedom only have meaning in an external reference frame. N in this case is, of course, the number of atoms in the system. Given the exponential scaling of the number of points required to describe a quartic force field (QFF), or really any potential energy surface (PES), with the number atoms, this difference of 6 coordinates can have a large effect. As a result, using internal coordinates can lead to a substantial decrease in computational cost for such a PES.

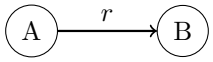
1.1 Simple-Internal Coordinates

Simple-internal coordinates are the basic building blocks of symmetry-internal coordinates, the major focus of the rest of the manual. The types of simple-internal coordinates supported by INTDER are described in the following subsections.

1.1.1 Stretch

The bond stretch, referred to as `STRE` in the input file, is the distance between two atoms, as shown in Fig. 1. When the direction matters, the distance is computed from atom `A` to atom `B`, using the equation given in Eqn. 1, where a_i and b_i represent the ith components of the Cartesian coordinates of atoms `A` and `B`, respectively.

Figure 1: A stretch

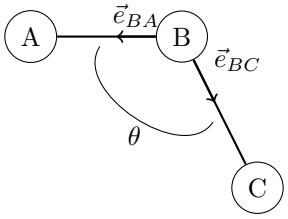


$$r = \sqrt{\sum_i (b_i - a_i)^2} \tag{1}$$

1.1.2 Bend

The bend, referred to as `BEND` in the input file, is the angle between three atoms, as shown in Fig. 2.

Figure 2: A bend



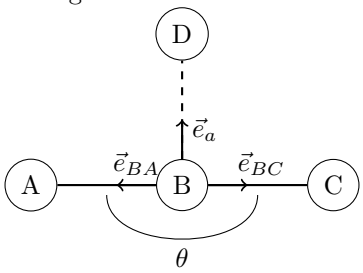
The value of the angle, θ is computed using Eqn. 2, where $\vec{e}_{ij}$ represents the unit vector pointing from atom i to atom j.

$$\theta = \text{acos}(\vec{e}_{BA} \cdot \vec{e}_{BC}) \tag{2}$$

1.1.3 Linear bend

The linear angle bend, referred to as `LIN1` in the input file, is similar to a regular bend, but atoms `A`, `B`, and `C` are in a line, so specifying the angle between them requires a fourth atom, `D`, which must be a dummy atom specified at the end of the geometry. Typically the coordinates of `D` are chosen to align with the central atom `B` along the axis of the molecule but to project along another axis, as shown in Fig. 3.

Figure 3: A linear bend



The value of the linear bend is computed using Eqn. 3 where $\vec{e}_a$ is the unit vector in the Cartesian direction of the dummy atom. Nothing in the code actually checks that it aligns with atom `B`, so the caller is responsible for making sure this is the case.

$$\theta = \text{asin}\left(\vec{e}_a \cdot (\vec{e}_{BC} \times \vec{e}_{BA})\right) \tag{3}$$

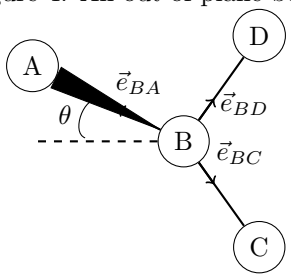
1.1.4 Out

The out-of-plane bend, referred to as `OUT` in the input file, is the angle between atom `A` and the plane formed by atoms `B`, `C`, and `D`, as shown in Fig. 4.

The value of θ is computed with the equations below, with the caveat that if $w_1 + w_2$ is greater than zero, the value of theta is set to π with the sign of θ, minus θ.

$$\vec{v}_5 = \vec{e}_{BC} \times \vec{e}_{BD} \tag{4}$$
$$w_1 = \vec{e}_{BA} \cdot \vec{e}_{BC} \tag{5}$$
$$w_2 = \vec{e}_{BA} \cdot \vec{e}_{BD} \tag{6}$$
$$\phi = \sin\left(\text{acos}(\vec{e}_{BC} \cdot \vec{e}_{BD})\right) \tag{7}$$
$$\theta = \text{asin}(\vec{e}_{BA} \cdot \vec{v}_5/\phi) \tag{8}$$

Figure 4: An out-of-plane bend

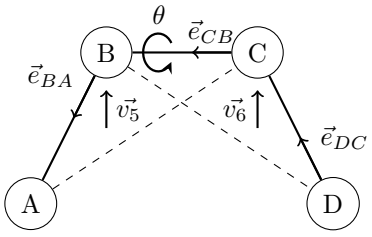


1.1.5 Torsion

The torsional angle, referred to as `TORS` in the input file, is the angle between the planes formed by the overlapping sets of atoms `A`, `B`, `C`, and `B`, `C`, `D`, as shown in Fig. 5. Vectors $\vec{v}_5$ and $\vec{v}_6$ are the vectors normal to the planes ABC and BCD, respectively, found by $\vec{e}_{BA} \times \vec{e}_{CB}$ and $\vec{e}_{DC} \times \vec{e}_{CB}$. To find the angle θ between these two normals, and thus between the planes, perform the following transformations:

$$w_2 = \vec{e}_{BA} \cdot \vec{e}_{CB} \tag{9}$$
$$w_3 = \vec{e}_{DC} \cdot \vec{e}_{CB} \tag{10}$$
$$s_2 = \sqrt{1 - w_2^2} \tag{11}$$
$$s_3 = \sqrt{1 - w_3^2} \tag{12}$$
$$w_4 = \vec{e}_{BA} \cdot \vec{v}_6 \tag{13}$$
$$w_5 = -\vec{v}_5 \cdot \vec{v}_6 \tag{14}$$
$$w = \frac{w_2}{s_2 s_3} \tag{15}$$
$$\theta_0 = \operatorname{asin} w \tag{16}$$
$$\theta = \begin{cases} \theta_0 & w_3 \geq 0 \\ \pi - \theta_0 & w_3 < 0 \end{cases} \tag{17}$$

Figure 5: A torsion



1.2 Symmetry-Internal Coordinates

With a solid foundation in simple-internal coordinates, symmetry-internal coordinates (SICs) require much less explanation. SICs are simply linear combinations of simple-internal coordinates that can be treated as a single unit. Common examples of SICs include the symmetric stretch in water, which is the positive combination (sum) of the O-H_1 stretch and the O-H_2 stretch; and the anti-symmetric stretch of water, which is the negative combination (difference) of the two O-H stretches. In the case of water, the third SIC only has a single component, the simple-internal bending coordinate. INTDER accepts linear combinations of

any number of simple-internal coordinates, but as you may imagine, the generation of highly-symmetrical SIC systems becomes very difficult. Describing how to approach such a problem is beyond the scope of this manual, if not beyond the scope of its author's abilities. Such is the major weakness of SICs in general.

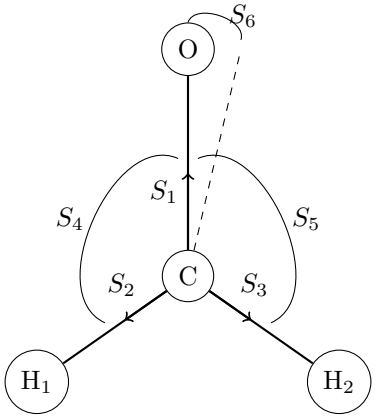
1.3 Input Format

The listing below shows an example of simple- and symmetry-internal coordinate input for the formaldehyde (H_2CO) molecule, followed by the geometry to help picture the meaning of these coordinates.

```
STRE      1     2
STRE      2     3
STRE      2     4
BEND      1     2     3
BEND      1     2     4
OUT       1     2     3     4
    1    1    1.000000000
    2    2    1.000000000    3    1.000000000
    3    4    1.000000000    5    1.000000000
    4    2    1.000000000    3   -1.000000000
    5    4    1.000000000    5   -1.000000000
    6    6    1.000000000
    0
      0.000000000          0.000000000         -1.137981497
      0.000000000          0.000000000          1.140677167
      0.000000000          1.771466895          2.235424580
      0.000000000         -1.771466895          2.235424580
```

The first section defines the internal coordinates themselves, given in more descriptive notation below and with atom labels corresponding to Fig. 6. This format is straightforward: the name of the coordinate occurs in the first column, followed by the number of the atom in the Cartesian geometry that follows in the order described in the previous sections.

Figure 6: Simple-internal coordinates of formaldehyde



$$S_1 = r(\mathrm{O} - \mathrm{C}) \tag{18}$$
$$S_2 = r(\mathrm{C} - \mathrm{H}_1) \tag{19}$$
$$S_3 = r(\mathrm{C} - \mathrm{H}_2) \tag{20}$$
$$S_4 = \angle(\mathrm{O} - \mathrm{C} - \mathrm{H}_1) \tag{21}$$
$$S_5 = \angle(\mathrm{O} - \mathrm{C} - \mathrm{H}_2) \tag{22}$$
$$S_6 = OUT(\mathrm{O} - \mathrm{CH}_2) \tag{23}$$

The second section defines the SICs as linear combinations of the simple-internals. The first column represents the SIC number, every pair of following columns represents a simple-internal coordinate and its coefficient. In practice all of these coefficients are 1.0, so this format is likely to change in the future. Positive coefficients, such as those for SIC 2 and SIC 3, are symmetric coordinates like the symmetric stretch for water discussed earlier. Similarly, negative coefficients, like the second components of SICs 4 and 5, correspond to anti-symmetric coordinates where one simple-internal coordinate increases while the other decreases. An SIC number of 0 at the start of the line signals the end of the SIC section.

SICs can also be written algebraically, as shown in the equations below. Note also the normalization constant that must be applied to give the whole SIC a magnitude of 1. This constant has the general form $\frac{\sqrt{N}}{N}$, where N is the number of simple-internal components, although the value for $N = 2$ is typically written as $\frac{1}{\sqrt{2}}$, which does not emphasize this form.

$$L_1 = r(\mathrm{O} - \mathrm{C}) \tag{24}$$

$$L_2 = \frac{1}{\sqrt{2}}[r(\mathrm{C} - \mathrm{H}_1) + r(\mathrm{C} - \mathrm{H}_1)] \tag{25}$$

$$L_2 = \frac{1}{\sqrt{2}}[\angle(\mathrm{O} - \mathrm{C} - \mathrm{H}_1) + \angle(\mathrm{O} - \mathrm{C} - \mathrm{H}_2)] \tag{26}$$

$$L_2 = \frac{1}{\sqrt{2}}[r(\mathrm{C} - \mathrm{H}_1) - r(\mathrm{C} - \mathrm{H}_1)] \tag{27}$$

$$L_2 = \frac{1}{\sqrt{2}}[\angle(\mathrm{O} - \mathrm{C} - \mathrm{H}_1) - \angle(\mathrm{O} - \mathrm{C} - \mathrm{H}_2)] \tag{28}$$

$$L_6 = OUT(\mathrm{O} - \mathrm{CH}_2) \tag{29}$$

As a final note, the example input contains exactly the right format for the Fortran version of INTDER, but the Rust version is much more forgiving with spacing and alignment. As long as whitespace separates the fields of each line, the Rust version should parse it correctly.

2 Displacements

The next section of the input file depends on the type of calculation to be run. The two options are SIC displacements to be converted to Cartesian geometries and SIC force constants to be converted to Cartesian force constants. This section is concerned with the former, and the next section covers the latter.

2.1 Input Format

The basic form of displacement input is as shown in the listing below.

```
DISP 413
    0
    1        -0.0200000000
    0
    1        -0.0150000000
    2        -0.0050000000
    0
    1        -0.0150000000
    3        -0.0050000000
    0
```

The Fortran version looks for the number following `DISP` to determine how many displacements to read, but the Rust version simply looks for `DISP` to signal to start reading displacements and will continue until the end of the file. Following this, each displacement is given by zero or more SIC indices followed by the size

of the displacement to take in that SIC. These displacements are expected to be in Ångstroms or radians, depending on the type of coordinate. The end of a single displacement is again signaled by a `0` at the start of the line. The reason for saying "zero or more SIC indices" before is that an immediate zero (not necessarily immediate in the `DISP` section, just without preceding SICs), as shown at the top of the previous listing, corresponds to a zero displacement or equilibrium geometry and is a valid input.

With that in mind, this listing shows 4 displacements: the zero displacement, one composed of only a -0.02 Å step in SIC 1, a combination of a -0.015 Å step in SIC 1 with a -0.005 Å step in SIC 2, and a similar combination of SICs 1 and 3.

Tip

If you need to verify that the number of displacements in your INTDER input file is correct, you can `grep` for these 0 lines:

```
$ grep -c '^    0$'
```

Just remember to subtract 1 for the instance in the SICs! This also assumes that the file is formatted as expected by the Fortran version. If you need more "flexible" formatting, you may need to adapt your regular expression to something like

```
$ grep -Ec '^\s+0\s*$'
```

Note also that now we need to pass the `-E` flag to take advantage of extended regular expressions that allow `\s`, for example.

2.2 Mathematical Details

For each of the displacements, $\vec{d_i}$ in the input, the initial set of SIC values, $\vec{L}_0$, is displaced to yield the desired SICs, $\vec{L}_d = \vec{L}_0 + \vec{d_i}$. Then, while the absolute value of the maximum deviation between the current SIC values $\vec{L}_i$ and $\vec{L}_d$ is greater than 1×10^{-14}, the Cartesian coordinates are adjusted by an iterative process. First, the SIC **B** matrix is constructed. The corresponding **D** matrix is just the product of **B** with its transpose: $\mathbf{D} = \mathbf{B}\mathbf{B}^\intercal$. Then the **A** matrix is simply $\mathbf{A} = \mathbf{B}^\intercal\mathbf{D}^{-1}$. This **A** matrix is multiplied by the difference between the desired and current SIC values to yield a step, $\vec{\delta}$, to be added to the Cartesian coordinates: $\vec{\delta} = \mathbf{A} * (\vec{L}_d - \vec{L}_i)$. Finally, the updated SIC values, $\vec{L}_{i+1}$, are computed from the resulting Cartesian geometry. As mentioned above, this process is repeated until convergence is reached or the maximum number of iterations (currently 20) is reached. While this process sounds straightforward at this level of abstraction, most of the effort goes into constructing the **B** matrix in the first place.

The SIC **B** matrix (denoted `BS` in the Fortran code and more aptly `sym_b_matrix` in my Rust code) requires only a fairly simple transformation (via multiplication by the **U** matrix) from the simple-internal **B** matrix, so most of the effort in this section is actually focused on the simple-internal coordinates and their **B** matrix. The discussion will only return to the construction of **U** at the very end.

The **B** matrix is an $N \times M$ matrix, where N corresponds to the number of simple-internal coordinates, and M is the number of Cartesian coordinates, or 3 times the number of atoms. Each row of **B** is the Wilson s-vector for the corresponding simple-internal coordinate. Rather, each row is conceptually a Wilson s-vector, but the actual form differs from that described by Wilson himself based on later developments in internal coordinate definitions. Like the coordinates themselves, each coordinate type has its own s-vector formulation and each of these is again described in the following subsections. Regardless of the specific formulation, the s-vectors simply describe the relationship between the simple-internal coordinates and the underlying Cartesian coordinates.

2.2.1 Stretch

3 Force Constants