CluE Oxide

CluE Oxide (Cluster Evolution Oxide) is a open source program for simulating electron spin decoherence ab initio using the Cluster Correlation Expansion [1, 2]. CluE Oxide takes as inputs a pbd file and an options file. The pdb file specifies the coordinates and elements types of all the atoms, and the options file specifies all the additional information needed, such as the pulse sequence the applied magnetic field, the central spin coordinates and so on.

0.1 Overview

The first steps CluE Oxide takes is to read the options and PDB files to construct the spin system. This includes applying periodic boundary conditions, and dropping non-magnetic nuclei as well as those beyond the system radius specified in the options. Isotope substitutions can also be applied at this step, if specified in the input options. In addition to isotope substitution, atoms in periodic boundary copies can be replaced with voids. This can be helpful in hydrogen rich media, where the proton may be diluted, but including all the deuterons in the simulation would be too expensive. The main time cost advantage of dropping the deuterons comes in when numerically integrating the Schödinger equation, but for large systems a Bernoulli trial at each atom is also expensive. Algorithmically, time is saved by avoiding N Bernoulli trials with a p success rate. In theses cases CluE Oxide samples a random number from the binomial distribution parameterized by N and p, and then selects that many random hydrons to initialize as protons, ignoring the rest.

Once the system is set up, CluE Oxide construct the tensors for the active spins, based on the user input. CluE Oxide will default to applying the point-dipole approximation for spin-spin coupling, but the hyperfine tensors can be explicitly set to custom values. Nuclear electric quadrupole coupling tensors can also be set, but default to zero. Tensor specification requires the eigenvalues, two eigenvectors, and which atoms to apply the tensor.

CluE Oxide has a filter algorithm that allows atoms to be specified by a combination of properties found in the PDB file, such as element, serial number, or residue, among others. The filters can either require or forbid qualifiers.

If unspecified CluE Oxide will assume that the applied magnetic field points along the z-direction of the PDB frame, different orientations can be specified. Additionally, CluE Oxide can compute the orientation averaged signal. CluE Oxide can average over custom orientation with custom weights, or it use random orientations (inefficient but simple) as well as orientations from a Lebedev grid, which integrates exactly the orientation dependence of the lowest order terms in a polynomial or spherical harmonic expansion[3]. Both the spin Hamiltonian and the Lebedev grid have inversion symmetry; this is leveraged to cut the number of calculation in half.

For each orientations, the set of clusters to be included in the n-CCE simulations must be calculated. The clusters are the subsets of the system that contain the central spin (only implicitly included since it is always there), and up to n bath spins that form a connected graph, where edges are defined between two bath spin when they are within some user specified cutoff. Several cutoff options are available, including an orientation independent distance cutoff, an orientation dependent dipole-dipole coupling, ΔA, and others based on the analytic solution of a simplified three spin system.

For large simulations, the primary time cost in CluE Oxide is often from diagonalizing and propagating the cluster Hamiltonians. In prototyping, a frequency-space method similar to that of [4] was tried, but likely due to the need to perform both a Fourier transform and inverse Fourier transform for each cluster, the time-domain integration of the Schrödinger equation is more efficient. The method CluE Oxide employ is to first diagonalize both electron spin-manifold Hamiltonians separately:

$$\hat{H}(m_{\mathrm{S}}) = \sum_n |\psi(m_{\mathrm{S}})\rangle\, E_n(m_{\mathrm{S}})\, \langle\psi(m_{\mathrm{S}})|\,. \tag{1}$$

And to construct the propagators in the eigenbasis for a small time increment, δt.

$$\hat{U}(\delta t, m_{\mathrm{S}}) = \sum_n |\psi(m_{\mathrm{S}})\rangle \exp\Big(-\mathrm{i}\frac{E_n(m_{\mathrm{S}})\delta t}{\hbar}\Big) \langle\psi(m_{\mathrm{S}})|\,. \tag{2}$$

This means that each cluster Hamiltonian only needs to be diagonalized once, as the later time propagators can be found via matrix multiplication.

$$\hat{U}(n_t\delta t, m_{\mathrm{S}}) = \hat{U}(\delta t, m_{\mathrm{S}})^{n_t}. \tag{3}$$

For some some of the longer time traces, it is helpful to have a small δt at early times and a longer δt at later times. The way CluE Oxide addresses this is allow multiple values of δt accompanied by the number of times each δt should be used. This requires that multiple propagators be calculated, but since the eigenvector have already been calculated, equation (2) can be used for multiple δts for a single diagonalization step.

Next at each time point in the simulated experiment, a total pulse sequence propagator is calculated as

$$\hat{U}_{n_t,m_{\mathrm{S}}} = \mathscr{T}\prod_p \hat{U}(\delta t, m_{\mathrm{S}p})^{n_t}, \tag{4}$$

where by using the $\hat{U}(\delta t, m_{\mathrm{S}p})^{n_t-1}$ from the previous steps, the cost scaling with the number of timepoints is linear. of each n_t is roughly the same.

The Heisenberg picture detection operator, $\hat{S}_+(t)$ in the high magnetic field limit, where the Zeeman basis is approximately the electron spin eigenbasis, can be written solely in terms of the propagators for the cluster in the as

$$\hat{S}_+(n_t) = \hat{U}^\dagger_{n_t,m'_{\mathrm{S}}}\hat{U}_{n_t,m_{\mathrm{S}}}. \tag{5}$$

For square d by d matrices A and B, the inner product between them is

$$\langle A, B\rangle := \mathrm{tr}(A^\dagger B). \tag{6}$$

To see why this definition makes sense, let n index the elements of the matrices. How exactly n indexes the matrices is unimportant as long as it is a consistent bijection between the d^2 row column pairs and d^2 integers. One possibility is that $n = r + dc$, where $r, c \in [0, d-1]$ are the row and column indices respectfully. In element-wise summation, the trace is

$$\mathrm{tr}(A^\dagger B) = \sum_{r,c} A^*_{r,c}B_{c,r}. \tag{7}$$

And the inner product between vectors is

$$\langle A, B\rangle = \sum_n A^*_n B_n. \tag{8}$$

Since there is a bijection between n and (r, c), these expression are equivalent; however, if only the trace is desired and not the matrix AB itself, the inner product method requires fewer steps, and is how CluE Oxide evaluates $\text{tr}\left(\hat{\rho}\hat{S}_+(t)\right)$.

After all the cluster signals are calculated the CCE auxiliary signals and products can be constructed. By the nature of the product, when calculating the n-CCE signal, all the n'-CCE signals, $n' < n$ are also calculated and saved.

If only a single orientation and isotopologue are requested, then CluE Oxide ends at this point. Different orientations require going back to the orientation generation step, and different isotopologues require rebuilding the system. After averaging over all the requested variations, CluE Oxide's primary returned data are the time domain signal n-CCE signal and the time axis. Depending on the input options other data may be saved, such as the clusters or individual signals from each orientation, the auxiliary signals, and system statistics including individual spin and methyl contributions.

0.2 Software

0.2.1 Installing

To start, please make sure that Rust is installed and up to date[5]. Before starting, it is a good idea to run the test suite.

```
cargo test
```

A few of the tests rely on random number generations and can occasionally fail. If this happens try running a second time: they should pass most of the time. To compile run the following.

```
cargo build --release
```

The compiled binary should be in *target/release/* and ready to use. An optional step is to make sure CluE Oxide is easy to access from the command line. One way to do this on Linux is make an alias by adding the following to the .bash_aliases file.

```
alias clue_oxide="path/to/clue/target/release/clue_oxide"
```

0.2.2 Running

To see the basic usage for CluE Oxide, run the following.

```
clue_oxide --help
```

The output is shown below.

```
Usage:
    clue_oxide input [-o "options"]

Options:
    -h, --help        Prints help information.
    -H, --hide-title Do not display title information.
    -l, --license     Prints license.
    -O, --option      input additional options.
    -V, --version     Prints version information.
    -W, --warrenty    Prints warrenty information.
```

To run CluE Oxide from the command line, an input file is needed. The input file format is .txt file, and is supplied to CluE Oxide as shown below.

```
clue_oxide input.txt
```

0.2.3 Examples

Before detailing the syntax of the input file, here is an example input for reference.

```
/*
Example CluE Oxide Input File
*/
input_structure_file = "MD_300K_25ns-30ns_center_0001.pdb";

detected_spin_position = centroid_over_serials([28,29]);
detected_spin_g_matrix = [2.0097, 2.0064, 2.0025];
detected_spin_g_y = diff(filter(tempo_n) , filter(tempo_o) );
detected_spin_g_x = diff(filter(tempo_c1) , filter(tempo_c19) );

radius = 25; // angstroms.

pulse_sequence = hahn;
magnetic_field = 1.2; // T.

number_timepoints = [101];
time_increments = [5e-8];

cluster_method = cce;
max_cluster_size = 4;

neighbor_cutoff_3_spin_hahn_mod_depth = 3.23e-5;
neighbor_cutoff_3_spin_hahn_taylor_4 = 1e17; // (rad/s)^4.

orientation_grid = lebedev(170);

#[filter(label = tempo_h)]
  residues in [TEM];
  elements in [H];

#[spin_properties(label = tempo_h, isotope = 1H)]
  tunnel_splitting = 80e3; // Hz.

#[filter(label = tempo_n)]
  residues in [TEM];
  elements in [N];

#[filter(label = tempo_o)]
  residues in [TEM];
  elements in [O];

#[filter(label = tempo_c1)]
  serials in [1];

#[filter(label = tempo_c19)]
  serials in [19];

#[spin_properties(label = tempo_n, isotope = 14N)]
  hyperfine_coupling = [20,20,100]*1e6;
  hyperfine_x = diff(bonded(tempo_c1),bonded(tempo_c19));
  hyperfine_y = diff(particle,bonded(tempo_o));

  electric_quadrupole_coupling = [-0.28, -1.47, 1.75]*1e6;
  electric_quadrupole_x = diff(bonded(tempo_c1),bonded(tempo_c19));
  electric_quadrupole_y = diff(particle,bonded(tempo_o));

#[config]
write_orientation_signals = "";
write_clusters = "";
```

The input script is not Turing complete, but is sufficient for starting CluE Oxide, and as will be discussed later, CluE Oxide can run from Turing complete languages like Bash, Python, and Rust. The syntax is inspired by C, C++, Rust, MATLAB and Python: all normal lines end in semicolons and double slashes indicate that the rest of line is a comment. Block commenting with “/*” and “*/” also works. Pound signs indicate that the following lines are to be interpreted differently. Line 5 specifies the input structure as a PDB file. Line 7 places the detected electron between the nitrogen and oxygen of TEMPO. Line 8–10 specifies the electron g-matrix. Line 12 says to only use a 25 Å radius sphere of the PDB, applying periodic boundary conditions if need be. Line 14 specifies the pulse sequence to be simulated. Line 15 declares the applied magnetic field strength in tesla: Line 18 defines the time increments used evaluating propagators; for Carr-Purcell sequences with n_π π-pulses, the time increment on the returned time-axis will be $(n_\pi + 1)\Delta t$. the applied magnetic field is always in along the z-axis. Line 20 determines which cluster approximation to use, and line 21 specifies the maximum cluster size to be used. Lines 23 and 24 specify neighbor cutoffs for determining clusters. Line 27 selects a 170 point Lebedev grid[3]. Lines 29–31 declare a filter selecting the TEMPO hydrogens.

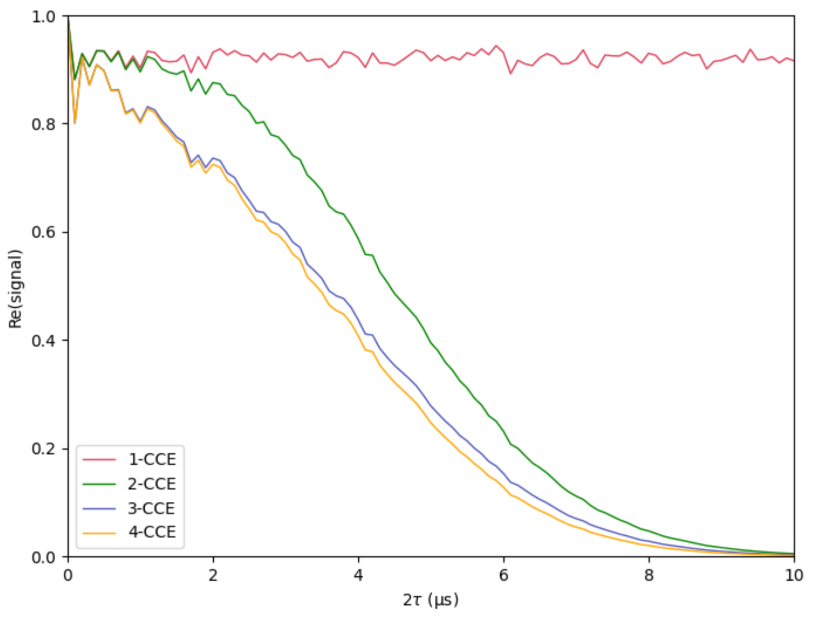

Figure 1: These Hahn echo simulation show the output of the previous example.

Each filter is required to starts with a *#[filter(label = my_label)]*, followed by various conditions; in this case the residue (as indicated in the PDB file), and the element. Line 33 and 34 set the methyl tunnel splitting for the TEMPO to 80 kHz, by referencing the previous filter. This is done by using *#[spin_properties(label = my_label, isotope = my_isotope)]*, where *my_label* matches the filter selecting the TEMPO hydrogens. Lines 36–48 declare more filters. Lines 50–57 sets the TEMPO nitrogen's hyperfine tensor and electric quadrupole coupling tensor. Note that one filter is used in *#[spin_properties(label = tempo_n, isotope = 14N)]* to select the TEMPO nitrogen and other filters are used to specify atoms for orientating the coupling tensors. Line 59 sets the mode back to *#[config]*; there is an implied *#[config]* at the start of every CluE Oxide file. Line 60 and 61 turn off saving orientation dependent signals an clusters, respectively.

0.3 Input File

0.3.1 #[config]

The main section of the input file is *#[config]*, and is the default when no mode is declared; however, the line

```
#[config]
```

will set CluE Oxide to read in *#[config]* mode and is needed to exit other modes back to *#[config]*. The available fields are explained below.

field	**values**	**description**
clash_distance_pbc	float	minimum distance in Å for which particles in periodic boundary condition copies are allowed to overlap particle in the primary cell
cluster_batch_size	int	number of clusters to evaluate between saves
cluster_method	r2cce, cce	method of evaluating clusters
detected_g_matrix	[float; 3]	eigenvalues of the detected spin g-matrix
detected_g_x	vector	x-direction of the detected spin g-matrix
detected_g_y	vector	y-direction of the detected spin g-matrix
detected_g_z	vector	z-direction of the detected spin g-matrix
detected_spin_position	spin_coor	coordinates in Å of the detected spin
input_structure_file	string	path to the input PDB
load_geometry	cube, sphere	shape of the spin system
magnetic_field	float	magnetic field strength in T of the applied magnetic field
max_cluster_size	int	maximum cluster size to be evaluated
neighbor_cutoff_3_spin_hahn_mod_depth	float	minimum modulation depth for the three spin hahn echo for an edge to be placed between two spins
neighbor_cutoff_3_spin_hahn_taylor_4	float	minimum magnitude of the $\mathscr{O}\big((2\tau)^4\big)$ coefficient, in the three spin hahn echo Taylor series, $\lvert k\omega^4\rvert$, in $(\mathrm{rad/s})^4$, for an edge to be placed between two spins
neighbor_cutoff_delta_hyperfine	float	minimum $\lvert\Delta A_{mn}\rvert := \lvert A_{zz,n} - A_{zz,m}\rvert$ in Hz for an edge to be placed between two spins
neighbor_cutoff_dipole_dipole	float	minimum $\lvert -b_{xx} - b_{yy}\rvert$ in Hz for an edge to be placed between two spins
neighbor_cutoff_dipole_perpendicular	float	minimum $\lvert b_\perp\rvert$ in Hz for an edge to be placed between two spins
number_timepoints	[int]	number of timepoints used for each time increment
number_system_instances	int	number of times the system will be simulated
orientation_grid	ori_grid	grid used for orientation averaging
pulse_sequence	cp-n_π, hahn	pulse sequence to simulate with n_π π-pulses
radius	float	radius in Å from the average position of the detected spin to use
save_dir	string	directory where data will be saved
temperature	float	temperature in K used to calculate each clusters density matrix
time_increments	[float]	list of Δts in s used for propagation
write_auxiliary_signals	string	file name to save cluster auxiliary signals under
write_bath	string	file name to save bath spins under
write_clusters	string	file name to save clusters under
write_info	string	directory name to save general information under
write_exchange_groups	string	file name to save exchange groups (such as methyls) under
write_structure_pdb	string	PDB file name to save the spin system under

ClusterMethod

There are two cluster evolutions schemes available in CluE Oxide, *cce*, and *r2cce*. The *cce* method implements ensemble CCE [1, 2] and has no explicit limit on the maximum cluster size. The *r2cce* method is a restricted version of CCE, using the analytic solution to the simplified three spin model from [6], and is therefore limited to a cluster size of two.

DensityMatrixMethod

There are options for setting the density matrix of a cluster. The first to assume the high nuclear temperature limit, in which case the density matrix is the identity matrix. The other option is to use $\hat{\rho} = \frac{1}{Z}\exp(-\frac{1}{k_{\mathrm{B}}T}\frac{\hat{H}_+ + \hat{H}_-}{2})$, where $\hat{H}_\pm$ are the cluster Hamiltonians for each of the detected spin manifolds. Z a normalization constant to ensures $\mathrm{tr}(\hat{\rho}) = 1$; k_{B} is the Boltzmann constant and T is the bath temperature. To set the temperature to 20 K for example, include the following in the CluE Oxide input file.

```
temperature = 20; // K.
```

This will switch CluE Oxide from using the identity matrix to the thermal density matrix.

DetectedSpinCoordinates

The detected spin coordinates have several ways to specified. They can be specified directly.

```
detected_spin_position = [1.2, -3, 0];
```

They can be specified as the centroid of atoms in the PDB file.

```
detected_spin_position = centroid_over_serials([28,29]);
```

Both of these methods assume that the detected spin is localized to a single point. If the spin is delocalized of multiple points, a csv file can be used to provide $|\Psi(x, y, z)|^2$.

```
detected_spin_position = read_csv("xyzw.csv");
```

Each row of the file "xyzw.csv" should be four comma separated floating point numbers, specifying the x, y, z and weight, $w = |\Psi(x, y, z)|^2 \mathrm{d}x\mathrm{d}y\mathrm{d}z$, for the coordinates.

LoadGeometry

The *load_geometry* can be either *sphere*, which is the default, or *cube*. The *sphere load_geometry* includes only bath particle that are within the specified *radius* of the detected spin. The *cube load_geometry* includes all bath particles from all periodic boundary duplicates. Periodic boundary duplicates are constructed as rectangular prism where there are $1 + 2\mathrm{ceil}(r/a)$ cell along each dimension, where r is the *radius* and a in the cell edge length along the dimension in question.

OrientationGrid

The orientation grid specifies the directions and weights for the applied static magnetic field. If no grid is given, the magnetic field is directed along the z-axis of the PDB file. A lebedev grid [3] can be specified.

```
orientation_grid = lebedev(302);
```

A random grid is also an option. Here is an example of using a single random direction.

```
orientation_grid = random(1);
```

And as with the detected spin position, a csv file x, y, z and weight entries can be used for a custom grid.

```
orientation_grid = read_csv("xyzw.csv");
```

PulseSequence

To simulate a Carr-Purcell pulse sequence with two π-pulses

```
pulse_sequence = cp-2;
```

A Hahn echo can be selected as shown.

```
pulse_sequence = hahn;
```

Or a Hahn echo is also a Carr-Purcell pulse sequence with one π-pulse.

```
pulse_sequence = cp-1;
```

SystemInstance

For simulations with random elements, such as a randomized isotope distribution, running the over multiple instances can be important. For example to run the simulation 1000 times each with a random direction, use the following.

```
number_system_instances = 1000;
orientation_grid = random(1);
```

Note that this is different from the following.

```
number_system_instances = 1;
orientation_grid = random(1000);
```

The former chooses a random isotopologue and a random direction 1000 times, while the latter uses a single random isotopologue but 10000 random directions. As an in between, the following chooses 900 random isotopologues, and for each of them chooses 100 random orientations.

```
number_system_instances = 900;
orientation_grid = random(100);
```

TimeAxis

The time-axis is specified in two part, the number of timepoints, and the time increments. For example, the CluE Oxide input file could include the following.

```
number_timepoints = [101,90];
time_increments = [5,50]*1e-9; // seconds.
```

CluE Oxide will read this and use $\Delta t = 5$ ns for the first 101 timepoints, and then $\Delta t = 50$ ns for the remaining 90 timepoints. There will be 191 timepoints total. Note that these are the Δts for the smallest propagator unit, so a Carr-Purcell pulse sequence with n_π π-pulses will have a $\Delta t_{\text{axis}} = (n_\pi + 1)\Delta t$. So if the above example is for a Hahn echo ($n_\pi = 1$), $\Delta t_{\text{axis}} = 10$ ns for the first 101 timepoints $\in [0, 1]$ μs and then $\Delta t_{\text{axis}} = 100$ ns for the next 90 timepoints $\in [1.1, 10]$ μs.

VectorSpecifiers

The fields that require values of "vector" have several options. First, a vector can be a list.

```
detected_g_x = [1,0,0];
detected_g_z = [0,0,1];
```

Note that exactly two axes should be specified: the third axis is found with the appropriate cross product. The specified axes do not need to be normalized or perfectly orthogonal, since CluE Oxide will orthonormalize the vectors itself. The reason for this is allow for directions to be specified by properties of the system. The following lines say to use the difference between the atoms in the *filter(tempo_c1)*, and the atoms in *filter(tempo_c19)* for the x-direction, and to use difference between the atoms in the *filter(tempo_n)*, and the atoms in *filter(tempo_o)* for the y-direction. Filters are explained in the next section, but the union of the atoms within the *diff()* must contain exactly two atoms for a unique vector to be specified.

```
detected_g_x = diff(filter(tempo_c1) , filter(tempo_c19) );;
detected_g_y = diff(filter(tempo_n) , filter(tempo_o) );;
```

0.3.2 #[filter(label = my_label)]

The options file can have zero, one, or multiple *#[filter(label = my_label)]* sections. Each *#[filter(label = my_label)]* section must have a unique label, by replacing *my_label* with the desired label. Under the *#[filter(label = my_label)]* header, the filter conditions that can have multiple matches are listed as

```
property in [allowed_value_1, allowed_value_n, ...];
```

or

```
property not in [forbidden_value_1, forbidden_value_n, ...];
```

The distance filter only takes maximum and minimum values and is specified as

```
distance <= r_max; // angstroms
distance >= r_min; // angstroms
```

where *r_max* and *r_min* are floating point numbers specifing the maximum and minimum distances in Å from the detected spin that the particles in the filter can be. CluE Oxide will find all the spins that fit all specified criteria. Note that depending upon how the criteria are defined it is possible that some spins will fall under multiple filters. To resolve this ambiguity, each spin is placed under the last applicable filter defined. The following table shows the possible filter criteria.

field	**values**	**description**
bonded_elements	[H,He,...]	elements of bonded atoms to match (in) or avoid (not in)
bonded_residues	[GLY,...]	residues of bonded atoms as specified in the PDB file to match (in) or avoid (not in)
bonded_serials	[int]	PDB serial numbers of bonded atoms to match (in) or avoid (not in)
bonded_residue_sequence_numbers	[int]	PDB residues sequence numbers of bonded atoms to match (in) or avoid (not in)
cell_ids	[int]	periodic boundary duplicate id where the particle resides, with the detected spin residing in cell 0
distance	float	maximum (<=) or minimum (>=) allowed distance in Å from the detected spin
elements	[H,He,...]	elements to match (in) or avoid (not in)
residues	[GLY,...]	residues as specified in the PDB file to match (in) or avoid (not in)
residue_sequence_numbers	[int]	residues sequence numbers as specified in the PDB file to match (in) or avoid (not in)
serials	[int]	serial numbers as specified in the PDB file to match (in) or avoid (not in)

0.3.3 #[sructure_properties(label = my_label)]

The options file can have zero, one, or multiple *#[sructure_properties(label = my_label)]* sections, where *my_label* should match to a *#[filter(label = my_label)]*. The properties specified under *#[sructure_properties(label = my_label)]* will be applied to all spins that fall under the corresponding *#[filter(label = my_label)]*. The following table shows the available properties.

field	values	description
extracell_isotope_abundances	{1H: 0.9, 2H: 0.1}	possible isotopes and probabilities for periodic copies of the primary cell
extracell_void_probability	float	chance that each particle will be removed if it is not in the primary cell
isotope_abundances	{1H: 0.9, 2H: 0.1}	possible isotopes and probabilities

0.3.4 #[spin_properties(label = my_label, isotope = my_isotope)]

The options file can have zero, one, or multiple *#[spin_properties(label = my_label, isotope = my_isotope)]* sections, where *my_label* should match to a *#[filter(label = my_label)]*. The properties specified under #[spin_properties(label = my_label, isotope = my_isotope)] will be applied to all spins of isotope *my_isotope* that fall under the corresponding *#[filter(label = my_label)]*. The reason to differentiate between *#[spin_properties(label = my_label, isotope = my_isotope)]* and *#[sructure_properties(label = my_label)]* is isotope specificity: *#[sructure_properties(label = my_label)]* applies to all particles in the corresponding filter, while *#[spin_properties(label = my_label, isotope = my_isotope)]* only applies to the specified isotope in the corresponding filter. The following table shows the available properties.

field	values	description
active	bool	specifies if the particle should be used for spin dynamics
electric_quadrupole_coupling	[float; 3]	eigenvalues in Hz of the electric quadrupole tensor
electric_quadrupole_x	vector	x-direction of the electric_quadrupole tensor
electric_quadrupole_y	vector	y-direction of the electric_quadrupole tensor
electric_quadrupole_z	vector	z-direction of the electric_quadrupole tensor
hyperfine_coupling	[float; 3]	eigenvalues in Hz of the hyperfine tensor
hyperfine_x	vector	x-direction of the hyperfine tensor
hyperfine_y	vector	y-direction of the hyperfine tensor
hyperfine_z	vector	z-direction of the hyperfine tensor
tunnel_splitting	float	tunnel splitting in Hz for particle in an exchange group

VectorSpecifiers

Two tensor axes must be specified in addition to the coupling. For example, the following sets the nitrogen hyperfine coupling to $A_{xx} = A_{yy} = 20$ MHz and $A_{zz} = 100$ MHz. It defines the x-direction of the tensor as the vector between the two carbons bonded to the nitrogen: since the nitrogen in TEMPO in bonded to only two carbon the vector is uniquely defined. The y-direction is defined as between the particle, the nitrogen in this case, and the oxygen that is bonded to the particle.

```
#[spin_properties(label = tempo_n, isotope = 14N)]
hyperfine_coupling = [20,20,100]*1e6; // Hz.
hyperfine_x = diff(bonded(tempo_c),bonded(tempo_c));
hyperfine_y = diff(particle,bonded(tempo_o));;
```

The arguments *diff()* can take are tabulated below. The table assumes that the user defined *#[filter(label = my_filter)]*.

argument	description
bonded(my_filter)	particles that are are in my_filter and bonded to reference particle
filter(my_filter)	particles that are are in my_filter
particle	the reference particle itself
same_molecule(my_filter)	particles that are are in my_filter and in the same molecule as the reference particle

0.4 Shell Scripting

The previous interface is useful for an isolated simulation, but to run multiple simulations that are nearly the same except for a few parameters being investigated, running CluE Oxide as a scrip is helpful. To this end there are several methods available. The first is via a shell script: CluE Oxide allows additional option to be appended to a base options file with the "-O" flag. Here is an example Bash script to run several different input structures with the same setting.

```bash
#!/usr/bin/bash

clue="path/to/clue"

options="path/to/options.txt"

files=(
    structure_1.pdb
    structure_2.pdb
    structure_n.pdb
)
for file in "${files[@]}"; do
    $clue $options -O "\"input_structure_file = $file;\""
done
```

Note that CluE Oxide interprets this the same as just appending the options to the end of the options file, the command line options must be be compatible with the options file. This means that the option should not already be set in the options file, and that the options file leaves off in the correct mode. For example if "#[config]" options are to be set, the options file should not end in another mode like "#[filter(label=my_label)]".

0.5 pyCluE

Additionally, CluE Oxide has a Python interface.

0.5.1 Installing

Within the CluE Oxide source directory, navigate to the pyclue directory.

```bash
cd pyclue
```

The Python interface uses maturin[7] to compile.

```bash
cd pyclue
python3 -m venv <path/to/virtual/environment>
source <path/to/virtual/environment/bin/activate>
pip install maturin
maturin build
```

One potential issue is that when maturin tries to compile CluE Oxide, it can fail to see the operating system unique flags, and will try to use them all. To account for this, open CluE Oxide's Cargo.toml file and comment out everything under the unneeded operating system section. For example, to compile on Linux add comments as shown below.

```
[target.'cfg(unix)'.dependencies]
ndarray-linalg = { version = "0.15", features = ["openblas-static"] }

#[target.'cfg(windows)'.dependencies.ndarray-linalg]
#version = '0.15.0'
#features = ['intel-mkl']
```

Once built, navigate to *target/wheels*, source the desired Python environment, and use pip to install the wheel.

```
cd target/wheels
source <path/to/installation/environment/bin/activate>
pip install <pyclue.whl>
```

0.5.2 Running

Running CluE Oxide via Python is done through the *CluEOptions* class.

```
class clue_oxide.CluEOptions(config = {},filter_list =[], structure_properties_list = [],
    spin_properties_list = [])
```

parameter	values	description
config	dictionary of strings	general config input
filter_list	[Filter]	filter input
structure_properties_list	[StructureProperties]	non-isotope specific properties input
spin_properties_list	[SpinProperties]	isotope specific properties input

The *Filter* class signature is shown below.

```
class clue_oxide.Filter(label, criteria = {})
```

The *label* parameter should be a string and serves the same purpose as the label in *#[filter(label = my_label)]* discussed earlier. The *criteria* parameter is a dictionary of strings. Similarly *StructureProperties* in analogous to a *#[structure_properties(label = my_label)]* section.

```
class clue_oxide.StructureProperties(label, properties = {})
```

And *SpinProperties* in analogous to a *#[spin_properties(label = my_label, isotope = my_isotope)]* section.

```
class clue_oxide.SpinProperties(label, isotope, properties = {})
```

Note that the additional string, *isotope*, is required.

Below in an example where an MD frame of TEMPO solvated in water is run with different tunnel splittings. Note that the pyCluE specific syntax is similar is similar to the standard CluE Oxide input files.

```
import clue_oxide as clue
import matplotlib
from matplotlib import pyplot as plt
import numpy as np

def main():

  # Initialize options.
  options = clue.CluEOptions(
      config = {
        "input_structure_file": "\"MD_300K_25ns-30ns_center_0001.pdb\"",
        "radius": "25", # angstroms
        "detected_spin_position": "centroid_over_serials([28,29])",
        "detected_spin_g_matrix": "[2.0097, 2.0064, 2.0025]",
        "detected_spin_g_y": "diff(filter(tempo_n) , filter(tempo_o) )",
        "detected_spin_g_x": "diff(filter(tempo_c1) , filter(tempo_c19) )",
        "number_timepoints": "[101]",
        "time_increments": "[5e-8]", # s
        "cluster_method": "cce",
        "max_cluster_size": "3",
        "magnetic_field": "1.2", # T
        "pulse_sequence": "hahn",
        "neighbor_cutoff_3_spin_hahn_mod_depth": "3.23e-5",
        "neighbor_cutoff_3_spin_hahn_taylor_4": "1e17", # (rad/s)^4
      },
      filter_list = [
        clue.Filter(label = "tempo_h", criteria = {
          "residues in": "[TEM]",
          "elements in": "[H]",
        }),
        clue.Filter(label = "tempo_n", criteria = {
          "residues in": "[TEM]",
          "elements in": "[N]",
        }),
        clue.Filter(label = "tempo_o", criteria = {
          "residues in": "[TEM]",
          "elements in": "[O]",
        }),
        clue.Filter(label = "tempo_c1", criteria = {
          "residues in": "[TEM]",
          "elements in": "[C]",
          "serials in": "[1]"
        }),
        clue.Filter(label = "tempo_c19", criteria = {
          "residues in": "[TEM]",
          "elements in": "[C]",
          "serials in": "[19]"
        }),
      ],
      spin_properties_list = [
        clue.SpinProperties(label = "tempo_n", isotope = "14N", properties = {
          "hyperfine_coupling": "[20,20,100]*1e6", # Hz
          "hyperfine_x": "diff(bonded(tempo_c1),bonded(tempo_c19))",
          "hyperfine_y": "diff(particle,bonded(tempo_o))",
          "electric_quadrupole_coupling": "[-0.28, -1.47, 1.75]*1e6", # Hz
          "electric_quadrupole_x": "diff(bonded(tempo_c1),bonded(tempo_c19))",
          "electric_quadrupole_y": "diff(particle,bonded(tempo_o))",
        }),
      ])


  # Initialize figure.
  plt.rc('font', size=12)
  plt.rc('axes', labelsize=12)

  fig, (ax) = plt.subplots(1,1,figsize=(8, 6) );
  cmap = matplotlib.colormaps['viridis']

  # Set desired methyl tunnel splittings.
  tunnel_splittings = np.array([0,20,40,80,100])*1e3;

  # Loop over tunnel splittings.
  for nut in tunnel_splittings:

    # Set tunnel splitting.
    options.set_spin_properties(label = "tempo_h", isotope = "1H",
        key = "tunnel_splitting", value = f"{nut}")

    # Print CluE input file.
    options.print()

    # Run CluE.
    time , signal = options.run()

    # Plot output.
    ax.plot(np.array(time)*1e6,np.real(np.array(signal)),
      color = cmap(nut/max(tunnel_splittings)), linewidth=1);


  # Polish the plot.
  norm = matplotlib.colors.Normalize(vmin=0, vmax=max(tunnel_splittings)*1e-3)

  sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
  sm.set_array([])

  plt.colorbar(sm, label=r"$\nu_\mathrm{t}$ (kHz)", orientation="vertical")
  ax.set_xlabel(r"$2\tau$  (s)");
  ax.set_ylabel("normalized echo amplitude");
  ax.axis([0,10,0,1.0] )

  # Save figure.
  plt.savefig('CluE-sims.png')

if __name__ == "__main__":
  main();
```

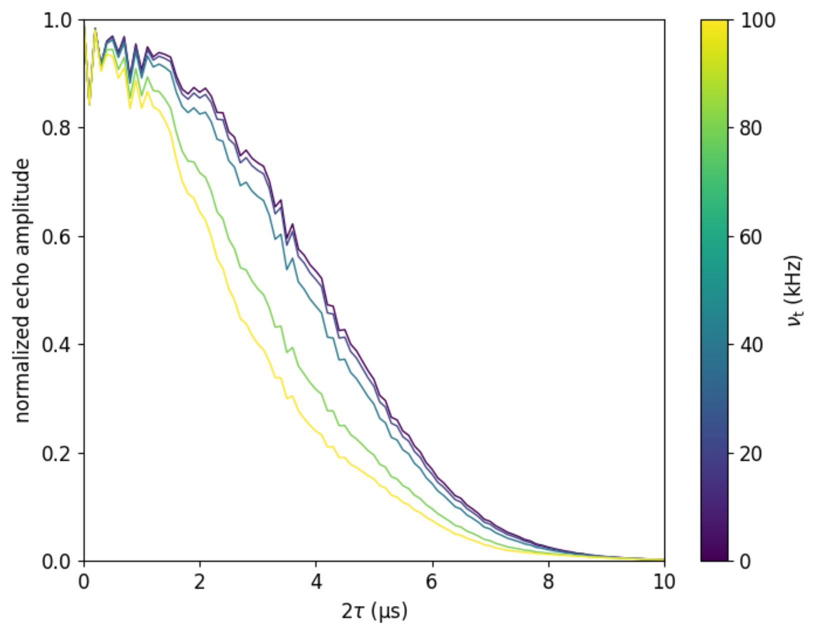

Figure 2: These Hahn echo simulation show the output of the previous example. The color-axis indicates the tunnel splitting.

Here the input is modified before sending it to CluE Oxide, so settings can be overwritten in the script. Figure 2 shows the output. Note that to save on computation time, these simulation are only at the 3-CCE level, and only use one frame at a single orientation.

0.6 Rust

Finally, since CluE Oxide is written in Rust, it is fairly straight forward to integrate CluE Oxide into a Rust program. Here is a script to run CluE Oxide over different proton concentrations. The output is shown in figure 3.

```rust
use clue_oxide as clue;
use clue_oxide::{
  physical_constants::Isotope,
  config::particle_config::{IsotopeAbundance,IsotopeDistribution}
};

fn main() {

    // Initialize options.
    let config = match clue::config::Config::from("
        input_structure_file = \"TEMPO_wat_gly_70A.pdb\";
        radius = 25; // angstroms.

        detected_spin_position = centroid_over_serials([28,29]);
        detected_spin_g_matrix = [2.0097, 2.0064, 2.0025];
        detected_spin_g_y = diff(filter(tempo_n) , filter(tempo_o) );
        detected_spin_g_x = diff(filter(tempo_c1) , filter(tempo_c19) );
        number_timepoints = [101,140];

        time_increments = [50,500]*1e-9;
        cluster_method = cce;
        max_cluster_size = 2;
        magnetic_field = 1.2; // T.
        pulse_sequence = hahn;

        number_system_instances = 10;
        orientation_grid = random(1);

        neighbor_cutoff_distance = 4; // angstroms.

      #[filter(label = tempo_n)]
        residues in [TEM];
        elements in [N];

      #[filter(label = tempo_o)]
        residues in [TEM];
        elements in [O];

      #[filter(label = tempo_c1)]
        serials in [1];

      #[filter(label = tempo_c19)]
        serials in [19];

      #[spin_properties(label = tempo_n, isotope = 14N)]
        hyperfine_coupling = [20,20,100]*1e6;
        hyperfine_x = diff(bonded(tempo_c1),bonded(tempo_c19));
        hyperfine_y = diff(particle,bonded(tempo_o));

        electric_quadrupole_coupling = [-0.28, -1.47, 1.75]*1e6;
        electric_quadrupole_x = diff(bonded(tempo_c1),bonded(tempo_c19));
        electric_quadrupole_y = diff(particle,bonded(tempo_o));

      #[filter(label = solvent_h)]
        residues not in [TEM];
        elements in [H];

      #[structure_properties(label = solvent_h)]
        isotope_abundances = {1H: 1, 2H: 0};

    "){
      Ok(cfg) => cfg,
      Err(err) => {
        eprintln!("CluE Error: {}.",err);
        return;
      },
    };


    // Find where the solvent properties are stored.
    let mut solvent_h_idx: Option<usize> = None;

    for (idx, p_cfg) in config.particles.iter().enumerate(){
      if p_cfg.label == "solvent_h"{
        solvent_h_idx = Some(idx);
        break;
      }
    }

    let solvent_h_idx = solvent_h_idx.expect("Could not find solvent_h_idx.");


    // Initialize desired mole fractions.
    let proton_fractions = vec![1.0, 0.75, 0.5, 0.25, 0.0];

    // Loop over mole fractions.
    for &frac in proton_fractions.iter(){

      let mut frac_config = config.clone();

      frac_config.save_name = Some(format!("CluE-proton_frac_{}",frac));

      let properties = frac_config.particles[solvent_h_idx]
        .properties.as_mut().expect("No properties for solvent_h");

      // Set isotope mole fractions.
      let isotope_abundances = vec![
        IsotopeAbundance{isotope: Isotope::Hydrogen1, abundance: frac},
        IsotopeAbundance{isotope: Isotope::Hydrogen2, abundance: 1.0 - frac},
      ];

      properties.isotopic_distribution = IsotopeDistribution{
          isotope_abundances,
          extracell_void_probability: None
      };

      // Run CluE.
      match clue::run(frac_config){
        Ok(_) => (),
        Err(err) => eprintln!("CluE Error: {}.",err),
      }
    }
}
```

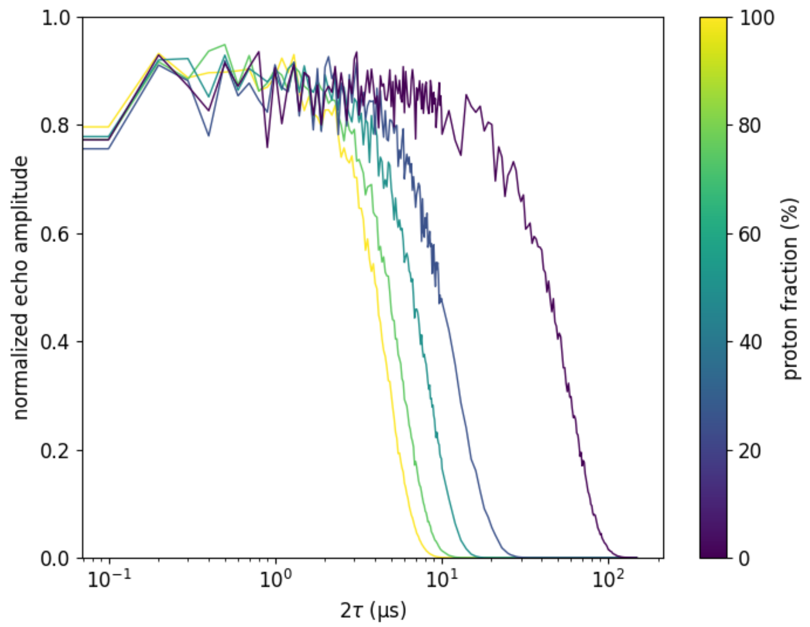

Figure 3: These Hahn echo simulation show the output of the previous example. The color-axis indicates the proton fraction. Note that the 2τ-axis is logarithmic.

The primary way to use CluE Oxide with Rust is to create an instance of *Config* and feed it to the run function. The easiest way to initialize an instance of *Config* is to use *Config::from(input: &str)*; this uses the same syntax as the CluE Oxide input files. Note that because the each new *frac_config* is modified after it has been constructed, the safety check to ensure values are not set multiple times is passed and properties can be modified directly. Every field of *Config* has a data type that is either an *Option<T>* or a *Vec::<T>*, for generic type *T*; values of *None* or an empty vector are considered unset. Which entries are required depends on the details of the simulation. CluE Oxide will return an error explaining what is missing if it needs an option that is unset. Several fields have default values, which can be set with the set_defaults method. Note that set_defaults will not overwrite user set values. This method is called automatically when using the command line or python interface. The *Config* struct is defined as shown below, with the defaults shown as comments.

```
pub struct Config{
  pub clash_distance: Option<f64>, // default: Some(1e-12)
  pub clash_distance_pbc: Option<f64>,
  pub cluster_batch_size: Option<usize>, // default: Some(1000)
  pub cluster_method: Option<ClusterMethod>,
  pub density_matrix: Option<DensityMatrixMethod>,
    // default(temperature = None):  DensityMatrixMethod::Identity
    // default(temperature = Some(T)):  DensityMatrixMethod::Thermal
  pub detected_spin_g_matrix: Option<TensorSpecifier>,
  pub detected_spin_identity: Option<Isotope>,
  pub detected_spin_multiplicity: Option<usize>,
  pub detected_spin_position: Option<DetectedSpinCoordinates>,
  pub detected_spin_transition: Option<[usize;2]>,
  pub input_structure_file: Option<String>,
  pub load_geometry: Option<LoadGeometry>,
    // default: Some(LoadGeometry::Sphere)
  pub magnetic_field: Option<Vector3D>,
  pub max_cluster_size: Option<usize>,
  pub neighbor_cutoff_delta_hyperfine: Option<f64>,
  pub neighbor_cutoff_dipole_dipole: Option<f64>,
  pub neighbor_cutoff_dipole_perpendicular: Option<f64>,
  pub neighbor_cutoff_distance: Option<f64>,
  pub neighbor_cutoff_3_spin_hahn_mod_depth: Option<f64>,
  pub neighbor_cutoff_3_spin_hahn_taylor_4: Option<f64>,
  pub number_system_instances: Option<usize>, // Some(1)
  pub number_timepoints: Vec::<usize>,
  pub orientation_grid: Option<OrientationAveraging>,
  pub particles: Vec::<ParticleConfig>,
  pub pdb_model_index: Option<usize>,
    // default: Some(0)
  pub pulse_sequence: Option<PulseSequence>,
  pub remove_partial_methyls: Option<bool>,
    // default(no tunnel splitting): Some(false)
    // default(with tunnel splitting): Some(true)
  pub root_dir: Option<String>, // default: Some("./".to_string())
  pub radius: Option<f64>,
  pub rng_seed: Option<u64>,
  pub save_name: Option<String>,
  pub temperature: Option<f64>,
  time_axis: Vec::<f64>,
  pub time_increments: Vec::<f64>,
  pub write_auxiliary_signals: Option<String>,
    // default: Some("auxiliary_signals".to_string())
  pub write_bath: Option<String>,
    // default: Some("bath".to_string())
  pub write_clusters: Option<String>,
    // default: Some("clusters".to_string())
  pub write_info: Option<String>,
    // default: Some("info".to_string())
  pub write_exchange_groups: Option<String>,
    // default: Some("exchange_groups".to_string())
  pub write_orientation_signals: Option<String>,
    // default: Some("orientations".to_string())
  pub write_structure_pdb: Option<String>,
    // default: Some("spin_system".to_string())
  pub write_tensors: Option<String>,
}
```

Bibliography

(1) Yang, W.; Liu, R.-B. Phys. Rev. B **2008**, *78*, 085315.

(2) Yang, W.; Liu, R.-B. Phys. Rev. B **2009**, *79*, 115320.

(3) Lebedev, V.; Laikov, D. Dokl. Math. **1999**, *59*, 477–481.

(4) Stoll, S.; Britt, R. D. Phys. Chem. Chem. Phys. **2009**, *11*, 6614.

(5) Foundation, R. Rust A language empowering everyone to build reliable and efficient software. `https://www.rust-lang.org/` (accessed 07/08/2023).

(6) Witzel, W.; Das Sarma, S. Phys. Rev. B **2006**, *74*, 035322.

(7) PyO3 PyO3/maturin `https://github.com/PyO3/maturin` (accessed 07/08/2023).