paramagpy Documentation

Release 0.1

Henry Orton

October 24, 2018

Table of Contents

class `paramagpy.metal.`**`Metal`** (*position=(0, 0, 0)*, *eulers=(0, 0, 0)*, *axrh=(0, 0)*, *mueff=0.0*, *shift=0.0*, *temperature=298.15*, *t1e=0.0*, *B0=18.79*, *taur=0.0*)

An object for paramagnetic chi tensors and delta-chi tensors. This must be created by specifying position, euler angles and eigenvalues only.

classmethod `anisotropy_to_eigenvalues` (*axial*, *rhombic*)

Calculate [dx,dy,dz] eigenvalues from axial and rhombic tensor anisotropies (axial and rhombic parameters). Calculations assume traceless tensor.

axial : *float*
the axial anisotropy of the tensor

rhombic : *float*
the rhombic anisotropy of the tensor

euler_angles : *array of floats*
the euler angles [alpha,beta,gamma] in radians by ZYZ convention

`dipole_shift_tensor` (*position*)

Calculate the chemical shift tensor at the given postition due to the paramagnetic dipole tensor field

position : *array floats*
the position (x, y, z) in meters

dipole_shift_tensor : *3x3 array*
the tensor describing chemical shift at the nuclear position

`dsa_r1` (*position*, *gamma*, *csa=0.0*, *ignorePara=False*)

Calculate R1 relaxation due to Curie Spin
atom

value : *float*
The R1 relaxation rate in /s

`dsa_r2` (*position*, *gamma*, *csa=0.0*, *ignorePara=False*)

Calculate R2 relaxation due to Curie Spin
atom :

value : *float*
The R2 relaxation rate in /s

classmethod `eigenvalues_to_anisotropy` (*dx*, *dy*, *dz*)

Calculate axial and rhombic tensor anisotropies from eigenvalues dx,dy,dz

dx, dy, dz : *floats*
the eigenvalues of the tensor. These are the principle axis magnitudes

axial, rhombic : *tuple of floats*
the tensor anisotropies

fast_pcs (*posarray*)
Rapidly calculatew the psuedo-contact shift at `n' positions. This efficient algorithm calculates the PCSs for an array of positions and is best used where speed is required for fitting.

posarray : *array of positions with shape (n,3)*
the position (x, y, z) in meters

pcs : *array of floats with shape (n,1)*
the peudo-contact shift in parts-per-million (ppm)

static fast_second_invariant_squared (*tensorarray*)
Calculate the second invariant at some position due to the magnetic susceptibility

position : *array floats*
the position (x, y, z) in meters

secondInvariant : *float*
the second invariant of the shift tensor

classmethod make_tensor (*x, y, z, axial, rhombic, alpha, beta, gamma, lanthanide=None, temperature=298.15*)
Make a ChiTensor isntance from given parameters. This is designed to use pdb coordinates (x, y, z) and euler angles from an output like Numbat.

x, y, z : *floats*
tensor position in pdb coordiante in Angstroms

axial, rhombic : *floats*
the tensor anisotropies in units 10^-32

alpha, beta, gamma : *floats*
the euler angles in degrees that maps the tensor to the pdb (I think?)

ChiTensor : *object*
a tensor object for calulating paramagnetic effects on nuclear spins in the pdb coordinate

pcs (*position*)
Calculate the psuedo-contact shift at the given postition

position : *array floats*
the position (x, y, z) in meters

pcs : *float*
the pseudo-contact shift in parts-per-million (ppm)

racs (*csa*)
Calculate the residual anisotropic chemical shift at the given postition. The partial alignment induced by an anisotropic magnetic susecptiblity causes the chemical shift tensor at a nuclear position to average to a value different to the isotropic value.

csa : *3 x 3 array*
the chemical shift anisotropy tensor

racs : *float*
the residual anisotropic chemical shift in parts-per-million (ppm)

rads (*position*)
Calculate the residual anisotropic dipolar shift at the given postition. The partial alignment induced by an anisotropic magnetic susecptiblity causes the dipole shift tensor at a nuclear position to average to a value different to the PCS.

position : *array floats*
the position (x, y, z) in meters

rads : *float*
the residual anisotropic dipole shift in parts-per-million (ppm)

rdc (*vector*, *gam1*, *gam2*)
Calculate Residual Dipolar Coupling (RDC)

vector : *[x,y,z] array of float*
internuclear vector in meters

gam1 : *float*
gyromagnetic ratio of spin 1 in rad/s/T

gam2 :
gyromagnetic ratio of spin 2 in rad/s/T

tensor : *ChiTensor object*
a paramagnetic tensor object from which <delta_tensor> 3x3 traceless matrix attribute must be present

rdc : *float*
the RDC in Hz

static second_invariant_squared (*tensor*)
Calculate the second invariant at some position due to the magnetic susceptibility

position : *array floats*
the position (x, y, z) in meters

secondInvariant : *float*
the second invariant of the shift tensor

static spec_dens (*tau*, *omega*)
Calculate spectral density at omega

omega : *float*
the spectral density argument

value : *float*
the value of the spectral denstiy at <omega>

paramagpy.metal.**euler_to_matrix** (*eulers*)
Calculate a rotation matrix from euler angles using ZYZ convention

eulers : *array of floats*
the euler angles [alpha,beta,gamma] in radians by ZYZ convention.

matrix : *numpy 3x3 matrix object*
the rotation matrix

paramagpy.metal.**matrix_to_euler** (*M*)
Calculate Euler angles from a rotation matrix using ZYZ convention

M : *3x3 array*
a rotation matrix

eulers : *array of floats*
the euler angles [alpha,beta,gamma] in radians by ZYZ convention

paramagpy.fit.**clean_indices** (*indices*)
Uniquely map a list of integers to their smallest size. For example: [7,4,7,9,9,10,1] -> [4 2 4 0 0 1 3]

indices : *array-like integers*
a list of integers

new_indices : *array-like integers*
the mapped integers with smallest size

paramagpy.fit.**extract_csa** (*data*)
Extract CSA tensors from atoms

data : *list of lists*
A list with elements [Atom, value, error], where Atom is an Atom object, value is the PCS/RDC/PRE value, and error is the uncertainty

csas : *array of 3x3 arrays*
an array of each CSA tensor

paramagpy.fit.**extract_pcs** (*data*)
Extract values required for PCS calculations

data : *list of lists*
A list with elements [Atom, value, error], where Atom is an Atom object, value is the PCS value, and error is the uncertainty

tuple : *(atom coordinates, PCS values, PCS errors, atom indices)*
all information required for PCS calculations

paramagpy.fit.**extract_pre** (*data*)
Extract values required for PRE calculations

data : *list of lists*
A list with elements [Atom, value, error], where Atom is an Atom object, value is the PRE value, and error is the uncertainty

tuple : *(atom coordinates, PRE values, PRE errors, atom indices)*
all information required for PRE calculations

paramagpy.fit.**extract_rdc** (*data*)
Extract values required for RDC calculations

data : *list of lists*
A list with elements [Atom, value, error], where Atom is an Atom object, value is the RDC value, and error is the uncertainty

tuple : *(inter-atomic vector, gamma values, RDC values,*
RDC errors, atom indices)
all information required for RDC calculations

paramagpy.fit.**nlr_fit_metal_from_pcs** (*initMetals*, *pcss*, *params*, *sumIndices=None*, *userads=False*, *useracs=False*, *progress=None*)
Fit deltaChi tensor to PCS values using non-linear regression.

initMetals : *list of Metal objects*
a list of metals used as starting points for fitting. a list must always be provided, but may also contain only one element. If multiple metals are provided, each metal is fitted to their respective PCS dataset by index, but all are fitted to a common position.

pcss : *list of PCS datasets*
each PCS dataset must correspond to an associated metal for fitting. each PCS dataset has structure [Atom, value, error], where Atom is an Atom object, value is the PCS/RDC/PRE value and error is the uncertainty

params : *list of str*
the parameters to be fit. For example ['x','y','z','ax','rh','a','b','g','shift']

sumIndices : *list of arrays of ints, optional*
each index list must correspond to an associated pcs dataset. each index list contains an index assigned to each atom. Common indices determine summation between models for

ensemble averaging. If None, defaults to atom serial number to determine summation between models.

userads : *bool, optional*
include residual anisotropic dipolar shielding (RADS) during fitting

useracs : *bool, optional*
include residual anisotropic chemical shielding (RACS) during fitting. CSA tensors are taken using the <csa> method of atoms.

progress : *object, optional*
to keep track of the calculation, progress.set(x) is called each iteration and varies from 0.0 -> 1.0 when the calculation is complete.

metals : *list of metals*
the metals fitted by NLR to the PCS data provided

`paramagpy.fit.`**`nlr_fit_metal_from_pre`** (*initMetals*, *pres*, *params*, *sumIndices=None*, *rtypes=None*, *usesbm=True*, *usedsa=True*, *usecsa=False*, *progress=None*)
Fit deltaChi tensor to PCS values using non-linear regression.

initMetals : *list of Metal objects*
a list of metals used as starting points for fitting. a list must always be provided, but may also contain only one element. If multiple metals are provided, each metal is fitted to their respective PCS dataset by index, but all are fitted to a common position.

pcss : *list of PCS datasets*
each PCS dataset must correspond to an associated metal for fitting. each PCS dataset has structure [Atom, value, error], where Atom is an Atom object, value is the PCS/RDC/PRE value and error is the uncertainty

params : *list of str*
the parameters to be fit. For example ['x','y','z','ax','rh','a','b','g','shift']

sumIndices : *list of arrays of ints, optional*
each index list must correspond to an associated pcs dataset. each index list contains an index assigned to each atom. Common indices determine summation between models for ensemble averaging. If None, defaults to atom serial number to determine summation between models.

userads : *bool, optional*
include residual anisotropic dipolar shielding (RADS) during fitting

useracs : *bool, optional*
include residual anisotropic chemical shielding (RACS) during fitting. CSA tensors are taken using the <csa> method of atoms.

progress : *object, optional*
to keep track of the calculation, progress.set(x) is called each iteration and varies from 0.0 -> 1.0 when the calculation is complete.

metals : *list of metals*
the metals fitted by NLR to the PCS data provided

`paramagpy.fit.`**`sphere_grid`** (*origin*, *radius*, *points*)
Make a grid of cartesian points within a sphere

origin : *float*
the centre of the sphere

radius : *float*
the radius of the sphere

points : *int*
the number of points per radius

array : *array of [x,y,z] coordinates*
the points within the sphere

paramagpy.fit.**svd_calc_metal_from_pcs** (*pos*, *pcs*, *idx*)
Solve PCS equation by single value decomposition. This function is generally called by higher methods like <svd_gridsearch_fit_metal_from_pcs>

pos : *array of [x,y,z] floats*
the atomic positions in meters

pcs : *array of floats*
the PCS values in ppm

idx : *array of ints*
an index assigned to each atom. Common indices determine summation between models for ensemble averaging.

tuple : *(calc, sol)*
calc are the calculated PCS values from the fitted tensor sol is the solution to the linearised PCS equation and consists of the tensor matrix elements

paramagpy.fit.**svd_calc_metal_from_pcs_offset** (*pos*, *pcs*, *idx*)
Solve PCS equation by single value decomposition with offset. An offset arising from referencing errors between diamagnetic and paramagnetic datasets can be accounted for using this method. This function is generally called by higher methods like <svd_gridsearch_fit_metal_from_pcs>
NOTE: the factor of 1E26 is required for floating point error mitigation

pos : *array of [x,y,z] floats*
the atomic positions in meters

pcs : *array of floats*
the PCS values in ppm

idx : *array of ints*
an index assigned to each atom. Common indices determine summation between models for ensemble averaging.

tuple : *(calc, sol)*
calc are the calculated PCS values from the fitted tensor sol is the solution to the linearised PCS equation and consists of the tensor matrix elements and offset

paramagpy.fit.**svd_gridsearch_fit_metal_from_pcs** (*metals*, *pcss*, *sumIndices=None*, *origin=None*, *radius=20.0*, *points=16*, *offsetShift=False*, *progress=None*)
Fit deltaChi tensor to PCS values using Single Value Decomposition over a grid of points in a sphere.

metals : *list of Metal objects*
a list of metals used as starting points for fitting. a list must always be provided, but may also contain only one element. If multiple metals are provided, each metal is fitted to their respective PCS dataset by index, but all are fitted to a common position.

pcss : *list of PCS datasets*
each PCS dataset must correspond to an associated metal for fitting. each PCS dataset has structure [Atom, value, error], where Atom is an Atom object, value is the PCS/RDC/PRE value and error is the uncertainty

sumIndices : *list of arrays of ints, optional*
each index list must correspond to an associated pcs dataset. each index list contains an index assigned to each atom. Common indices determine summation between models for ensemble averaging. If None, defaults to atom serial number to determine summation between models.

origin : *float, optional*
the centre of the gridsearch of positions in Angstroms. If None, the position of the first metal is used

radius : *float, optional*
the radius of the gridsearch in Angstroms.

points : *int, optional*
the number of points per radius in the gridsearch

offsetShift : *bool, optional*
if True, an offset value added to all PCS values is included in the SVD fitting. This may arise due to a referencing error between diamagnetic and paramagnetic PCS datasets and may be used when many data points are available. Default False, no offset is included in the fitting.

progress : *object, optional*
to keep track of the calculation, progress.set(x) is called each iteration and varies from 0.0 -> 1.0 when the calculation is complete.

minmetals : *list of metals*
the metals fitted by SVD to the PCS data provided

paramagpy.fit.**unique_pairing** (*a*, *b*)
Uniquely map two integers to a single integer. The mapped space is minimum size. The input is symmetric.
a : int b : int

c : *int*
unique symmetric mapping (a, b) -> c

class paramagpy.protein.**CustomStructure** (**arg*, ***kwargs*)
docstring for CustomStructure

class paramagpy.protein.**CustomStructureBuilder** (**arg*, ***kwargs*)
docstring for CustomStructureBuilder

init_atom (*name*, *coord*, *b_factor*, *occupancy*, *altloc*, *fullname*, *serial_number=None*, *element=None*)
Create a new Atom object. Arguments:

- name - string, atom name, e.g. CA, spaces should be stripped
- coord - Numeric array (Float0, size 3), atomic coordinates
- b_factor - float, B factor
- occupancy - float
- altloc - string, alternative location specifier
- fullname - string, atom name including spaces, e.g. " CA "
- element - string, upper case, e.g. "HG" for mercury

init_structure (*structure_id*)
Initialize a new Structure object with given id.

Arguments:

- id - string

paramagpy.protein.**rotation_matrix** (*axis*, *theta*)
Return the rotation matrix associated with counterclockwise rotation about the given axis by theta radians.

axis : *array of floats*
the [x,y,z] axis for rotation.

matrix : *numpy 3x3 matrix object*
the rotation matrix

class paramagpy.dataparse.**DataContainer** (**args*, ***kwargs*)
docstring for DataContainer

- *Index*
- *Module Index*
- *Search Page*

p

U