Masci-tools Documentation

Release 0.15.0

The JuDFT team

May 03, 2023

CONTENTS

This package was developed in the process of developing the AiiDA-FLEUR and AiiDA_KKR plugins to AiiDA. It contains helper functions that can help with common pre- and postprocessing steps of the FLEUR and KKR codes developed at the Forschungszentrum Jülich (see also the juDFT website for more information).

If you use this package please cite: …

CHAPTER

ONE

REQUIREMENTS TO USE THIS CODE:

- `lxml`
- `h5py`
- `numpy`
- `matplotlib`
- `bokeh` (optional)
- `seaborn`
- `ase`
- `mendeleev`
- `click`
- `click-completion`
- `PyYAML`
- `tabulate`
- `deepdiff`
- `humanfriendly`
- `more_itertools`

CHAPTER

TWO

INSTALLATION INSTRUCTIONS:

Install from pypi the latest release

```
pip install masci-tools
```

or from the masci-tools source folder any branch

```
pip install .
# or which is very useful to keep track of the changes (developers)
pip install -e .
```

CHAPTER

THREE

ACKNOWLEDGMENTS:

We acknowledge partial support from the EU Centre of Excellence "MaX – Materials Design at the Exascale" (http://www.max-centre.eu). (Horizon 2020 EINFRA-5, Grant No. 676598) We thank the AiiDA team for their help and work. Also the vial exchange with developers of AiiDA packages for other codes was inspiring.

CHAPTER

FOUR

USER'S GUIDE

4.1 User guide

This is the masci-tools user's guide.

4.1.1 Using the Fleur input/output parsers

Contents

4.1.1.1 Parser for the Fleur inp.xml file

The fleur `inp.xml` contains all the information about the setup of a fleur calculation. To use this information in external scripts or aiida-fleur, the information needs to be parsed from the `.xml` format somehow.

For this purpose the *`inpxml_parser()`* is implemented. The usage is shown below. The input file is parsed recursively and all information is put into the dictionary.

```
from masci_tools.io.parsers.fleur import inpxml_parser

input_dict = inpxml_parser('/path/to/random/inp.xml')

#The call below will output warnings about failed conversions in the warnings dictionary
warnings = {'parser_warnings': []}
input_dict = inpxml_parser('/path/to/random/inp.xml', parser_info_out=warnings)
```

The conversion of each attribute or text is done according to the FleurInputSchema for the same version, which is stored in this repository for versions from `0.27` to `0.35`. The following table shows the version compatibility of the input parser.

File Version	Compatible
0.27 - 0.36	Yes
0.37	Fallback to 0.36

4.1.1.2 Parser for the Fleur out.xml file

For the `out.xml` file a similar parser is implemented. However, since the output file contains a lot more information, which is not always useful the *outxml_parser()* is defined a lot more selectively. But the usage is almost completely identical to the input file.

```
from masci_tools.io.parsers.fleur import outxml_parser

#The default is that only the last stable iteration is parsed
output_dict = outxml_parser('/path/to/random/out.xml')

#Here all iterations are parsed
output_dict = outxml_parser('/path/to/random/out.xml', iteration_to_parse='all')

#Or the 5.
output_dict = outxml_parser('/path/to/random/out.xml', iteration_to_parse=5)

#The call below will output warnings about failed conversions in the warnings dictionary
warnings = {'parser_warnings': []}
output_dict = outxml_parser('/path/to/random/out.xml', parser_info_out=warnings)
```

For each iteration the parser decides based on the type of fleur calculation, what things should be parsed. For a more detailed explanation refer to the *Updating or adapting the Fleur Parsers*.

The following table shows the version compatibility of the output parser. For versions before `0.34` the file version corresponds to the input version, since the output version is `0.27` for all versions before this point.

File Version	Compatible
0.27 - 0.29	0.29 version is assumed (no XML validation)
0.30 - 0.31	Yes (no XML validation)
0.32	No (Does not exist for any release version of fleur)
0.33	Yes (no XML validation)
0.34 - 0.36	`Yes
0.37 -	Fallback to 0.36

Using File handles

Both the *inpxml_parser()* and *outxml_parser()* can also be used with file handles like shown below.

```
from masci_tools.io.parsers.fleur import inpxml_parser

with open('/path/to/random/inp.xml', 'rb') as file:
    input_dict = inpxml_parser(file)
```

Notice that the file has to be opened in binary mode.

4.1.1.3 XML getter functions

There are a number of functions for extracting specific parts of the XML files in the *xml_getters* module. The following are available:

- *get_fleur_modes()*: Get information about the mode of the fleur calculation
- *get_nkpts()*: Get the (for older versions approximate if not `kPointList` is used) number of kpoints to be used in the calculation
- *get_cell()*: Get the Bravais matrix of the system
- *get_parameterdata()*: Get the information about the calculation parameters needed to reproduce a calculation starting from the inpgen
- *get_structuredata()*: Get the structure from the xml file (atom positions + unit cell)
- *get_kpointsdata()*: Get the defined kpoint sets (single/multiple) from the xml file (kpoints + weights + unit cell)
- *get_relaxation_information()*: Get the relaxation history and current displacements
- *get_symmetry_information()*: Get the symmetry operations used in the calculation

All of these are used in the same way

```
from masci_tools.io.fleur_xml import load_inpxml
from masci_tools.util.xml.xml_getters import get_fleur_modes

xmltree, schema_dict = load_inpxml('/path/to/inp.xml')

fleur_modes = get_fleur_modes(xmltree, schema_dict)
print(fleur_modes)
```

4.1.1.4 Using the `schema_dict_util` functions

If only a small amount of information is required from the input or output files of fleur the full parsers might be overkill. But there are a number of utility functions allowing easy access to information from the `.xml` files without knowing the exact XPath expressions for each version of the input/output. A code example extracting information from a input file is given below.

```
from masci_tools.io.fleur_xml import load_inpxml
from masci_tools.util.schema_dict_util import evaluate_attribute, eval_simple_xpath

#First we create a xml-tree from the input file and load the desired input schema␣
↪dictionary
xmltree, schema_dict = load_inpxml('/path/to/inp.xml')
root = xmltree.getroot()

#Here an example of extracting some attributes. The interface to all functions in
#schema_dict_util is the same

#Number of spins
spins = evaluate_attribute(root, schema_dict, 'jspins')

#Planewave cutoff (notice the names are case-insensitive, 'KMAX' would work as well)
kmax = evaluate_attribute(root, schema_dict, 'kmax')
```

(continues on next page)

(continued from previous page)

```
#Some attributes need to be specified further for a distinct path
#`radius` exists both for atom species and atom groups so we give a phrase to distinguish␣
↪them
mt_radii = evaluate_attribute(root, schema_dict, 'radius', contains='species')

#But we can also make implicit constraints
# 1. Get some element in the xml tree, where the path is more specified. In the example␣
↪lets
#    get the element containing all species
# 2. If we evaluate the `radius` attribute now on the species elements, we do not need
#    the contains parameter, since from the point of the species element there is only␣
↪one possibility
#    for the `radius` attribute

species = eval_simple_xpath(root, schema_dict, 'atomSpecies')
mt_radii = evaluate_attribute(species, schema_dict, 'radius')
```

To manage the context of these functions the *FleurXMLContext()* is available to write the same code as above more concisely.

```
from masci_tools.io.fleur_xml import load_inpxml, FleurXMLContext
xmltree, schema_dict = load_inpxml('/path/to/inp.xml')

with FleurXMLContext(xmltree, schema_dict) as root:
   spins = root.attribute('jspins')
   noco = root.attribute('l_noco', default=False)

   #Not nesting the context we need to specify which elements are meant
   mt_radii = root.attribute('radius', contains='species')

   #Nesting using find (the first element is return)
   with root.find('atomspecies') as all_species:
         mt_radii = all_species.attribute('radius')

   #Nesting using iter (each iteration returns a new context for the next element)
   mt_radii = []
   for species in root.iter('species'):
         mt_radii.append(species.attribute('radius'))
```

4.1.2 FleurXMLModifier

4.1.2.1 Description

The *FleurXMLModifier* class can be used if you want to change anything in a `inp.xml` file in an easy and robust way. It will validate all the changes you wish to do and apply all these changes to a given `inp.xml` and produce a new XML tree.

4.1.2.2 Usage

To modify an existing `inp.xml`, a *FleurXMLModifier* instance has to be initialised. After that, a user should register certain modifications which will be cached. They will be applied on a given `inp.xml`. However, the provided `inp.xml` will not be changed but only a modified XML tree is returned, which you can store in a new `.xml` file.

```
from masci_tools.io.fleurxmlmodifier import  FleurXMLModifier

fm = FleurXMLModifier()                                    # Initialise␣
↪FleurXMLModifier class
fm.set_inpchanges({'dos' : True, 'Kmax': 3.9 })            # Add changes
new_xmltree, _ = fm.modify_xmlfile('/path/to/original/inp.xml') #Apply
```

4.1.2.3 User Methods

General methods

- *FleurXMLModifier.modify_xmlfile()*: Applies the registered changes to a given `inp.xml` (and optional `n_mmp_mat` file)
- *FleurXMLModifier.changes()*: Displays the current list of changes.
- *FleurXMLModifier.undo()*: Removes the last task or all tasks from the list of changes.

Modification registration methods

The registration methods can be separated into two groups. First of all, there are XML methods that require deeper knowledge about the structure of an `inp.xml` file. All of them require an xpath input and start their method names start with `xml_`:

- *FleurXMLModifier.xml_set_attrib_value_no_create()*: Set attributes on the result(s) of the given xpath
- *FleurXMLModifier.xml_set_text_no_create()*: Set text on the result(s) of the given xpath
- *FleurXMLModifier.xml_create_tag()*: Insert an xml element in the xml tree on the result(s) of the given xpath.
- *FleurXMLModifier.xml_delete_tag()*: Delete an xml element in the xml tree on the result(s) of the given xpath.
- *FleurXMLModifier.xml_delete_att()*: Delete an attribute in the xml tree on the result(s) of the given xpath.
- *FleurXMLModifier.xml_replace_tag()*: Replace an xml element on the result(s) of the given xpath.

On the other hand, there are shortcut methods that already know some paths:

- *FleurXMLModifier.set_species()*: Specific user-friendly method to change species parameters.
- *FleurXMLModifier.clone_species()*: Method to create a clone of a given species with optional modifications
- *FleurXMLModifier.set_atomgroup()*: Specific method to change atom group parameters.
- *FleurXMLModifier.set_species_label()*: Specific user-friendly method to change a species of an atom with a certain label.
- *FleurXMLModifier.set_atomgroup_label()*: Specific method to change atom group parameters of an atom with a certain label.

- *FleurXMLModifier.switch_species()*: user-friendly method for switching the atom species of a atom group
- *FleurXMLModifier.switch_species_label()*: user-friendly method for switching the atom species of a atom group with an atom with a certain label.
- *FleurXMLModifier.set_nkpts()*: user-friendly method for setting the `kPointCount` (**Only for MaX4 and older**)
- *FleurXMLModifier.set_kpath()*: user-friendly method for setting the path for a bandstructure calculations (**Only for MaX4 and older**)
- *FleurXMLModifier.set_kpointlist()*: user-friendly method for setting/creating a `kPointlist` from lists
- *FleurXMLModifier.switch_kpointset()*: user-friendly method for switching the used kpoint set in a calculation (**Only for MaX5 and newer**)
- *FleurXMLModifier.set_inpchanges()*: Specific user-friendly method for easy changes of attribute key value type.
- *FleurXMLModifier.shift_value()*: Specific user-friendly method to shift value of an attribute.
- *FleurXMLModifier.shift_value_species_label()*: Specific user-friendly method to shift value of an attribute of an atom with a certain label.
- *FleurXMLModifier.set_attrib_value()*: user-friendly method for setting attributes in the xml file by specifying their name
- *FleurXMLModifier.set_first_attrib_value()*: user-friendly method for setting the first occurrence of an attribute in the xml file by specifying its name
- *FleurXMLModifier.add_number_to_attrib()*: user-friendly method for adding to or multiplying values of attributes in the xml file by specifying their name
- *FleurXMLModifier.add_number_to_first_attrib()*: user-friendly method for adding to or multiplying values of the first occurrence of the attribute in the xml file by specifying their name
- *FleurXMLModifier.set_text()*: user-friendly method for setting text on xml elements in the xml file by specifying their name
- *FleurXMLModifier.set_first_text()*: user-friendly method for setting the text on the first occurrence of an xml element in the xml file by specifying its name
- *FleurXMLModifier.set_simple_tag()*: user-friendly method for creating and setting attributes on simple xml elements (only attributes) in the xml file by specifying its name
- *FleurXMLModifier.set_complex_tag()*: user-friendly method for creating complex tags in the xml file by specifying its name
- *FleurXMLModifier.create_tag()*: User-friendly method for inserting a tag in the right place by specifying it's name
- *FleurXMLModifier.delete_tag()*: User-friendly method for delete a tag by specifying it's name
- *FleurXMLModifier.delete_att()*: User-friendly method for deleting an attribute from a tag by specifying it's name
- *FleurXMLModifier.replace_tag()*: User-friendly method for replacing a tag by another by specifying its name
- *FleurXMLModifier.set_nmmpmat()*: Specific method for initializing or modifying the density matrix file for a LDA+U calculation (details see below)
- *FleurXMLModifier.rotate_nmmpmat()*: Specific method for rotating a block/blocks of the density matrix file for a LDA+U calculation (details see below) in real space

- *FleurXMLModifier.align_nmmpmat_to_sqa()*: Specific method for aligning a block/blocks of the density matrix file for a LDA+U calculation (details see below) in real space with the SQA already specified in the `inp.xml`

4.1.2.4 Modifying the density matrix for LDA+U calculations

The above mentioned *FleurXMLModifier.set_nmmpmat()*, *FleurXMLModifier.rotate_nmmpmat()* and *FleurXMLModifier.align_nmmpmat_to_sqa()* take a special role in the modification registration methods, as the modifications are not done on the `inp.xml` file but the density matrix file `n_mmp_mat` used by Fleur for LDA+U calculations. The resulting new `n_mmp_mat` file is returned next to the new `inp.xml` by the *FleurXMLModifier.modify_xmlfile()*.

The code example below shows how to use this method to add a LDA+U procedure to an atom species and provide an initial guess for the density matrix.

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
# Add LDA+U procedure
fm.set_species('Nd-1', {'ldaU':{'l': 3, 'U': 6.76, 'J': 0.76, 'l_amf': 'F'}})
# Initialize n_mmp_mat file with the states m = -3 to m = 0 occupied for spin up
# spin down is initialized with 0 by default, since no n_mmp_mat file is provided
fm.set_nmmpmat('Nd-1', orbital=3, spin=1, state_occupations=[1,1,1,1,0,0,0])
new_xmltree, add_files = fm.modify_xmlfile('/path/to/original/inp.xml')
print(add_files['n_mmp_mat'])
```

Note: The `n_mmp_mat` file is a simple text file with no knowledge of which density matrix block corresponds to which LDA+U procedure. They are read in the same order as they appear in the `inp.xml`. For this reason the `n_mmp_mat` file can become invalid if one adds/removes a LDA+U procedure to the `inp.xml` after the `n_mmp_mat` file was initialized. Therefore any modifications to the `n_mmp_mat` file should be done after adding/removing or modifying the LDA+U configuration.

4.1.3 General `HDF5` file reader

Fleur uses the HDF5 library for output files containing large datasets. The masci-tools library provides the *reader.HDF5Reader* class to extract and transform information from these files. The `h5py` library is used to get information from `.hdf` files.

4.1.3.1 Basic Usage

The specifications of what to extract and how to transform the data are given in the form of a python dictionary. Let us look at a usage example; extracting data for a bandstructure calculation from the `banddos.hdf` file produced by Fleur.

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands

#The HDF5Reader is used with a contextmanager to safely handle
#opening/closing the h5py.File object that is produced to extract information
with HDF5Reader('/path/to/banddos.hdf') as h5reader:
   datasets, attributes = h5reader.read(recipe=FleurBands)
```

The method *reader.HDF5Reader.read()* produces two python dictionaries. In the case of the `FleurBands` recipe these contain the following information.

- **`datasets`**
  - Eigenvalues converted to eV shited to `E_F=0` (if available in the `banddos.hdf`) and split up into spin-up/down and flattened to one dimension
  - The kpath projected to 1D and reshaped to same length as weights/eigenvalues
  - The weights (flattened) of the interstitial region, each atom, each orbital on each atom for all eigenvalues
- **`attributes`**
  - The coordinates of the used kpoints
  - Positions, atomic symbols and indices of symmetry equivalent atoms
  - Dimensions of eigenvalues (`nkpts` and `nbands`)
  - Bravais matrix/Reciprocal cell of the system
  - Indices and labels of special k-points
  - Fermi energy
  - Number of spins in the calculation

The following pre-defined recipes are stored in *masci_tools.io.parsers.hdf5.recipes*:

- Recipe for `banddos.hdf` for bandstructure calculations
- Recipe for `banddos.hdf` for standard density of states calculations
- Different DOS modes are also supported (`jDOS`, `orbcomp`, `mcd`)

If no recipe is provided to the *reader.HDF5Reader*, it will create the `datasets` and `attributes` as two nested dictionaries, exactly mirroring the structure of the `.hdf` file and converting datasets into numpy arrays.

For big datasets it might be useful to keep the dataset as a reference to the file and not load the dataset into memory. To achieve this you can pass `move_to_memory=False`, when initializing the reader. Notice that most of the transformations will still implicitly create numpy arrays and after the hdf file is closed the datasets will no longer be available.

4.1.3.2 Structure of recipes for the `reader.HDF5Reader`

The recipe for extracting bandstructure information form the `banddos.hdf` looks like this:

```
def bands_recipe_format(group: Literal['Local', 'jDOS', 'Orbcomp', 'MCD'], simple: bool␣
↪= False) -> HDF5Recipe:
    """
    Format for bandstructure calculations retrieving weights from the given group

    :param group: str of the group the weights should be taken from
    :param simple: bool, if True no additional weights are retrieved with the produced␣
↪recipe

    :returns: dict of the recipe to retrieve a bandstructure calculation
    """

    if group == 'Local':
```

(continues on next page)

(continued from previous page)

```
        atom_prefix = 'MT:'
        weight_atom_terminator = '[spdf]'
    elif group == 'jDOS':
        atom_prefix = 'jDOS:'
        weight_atom_terminator = '[spdf]'
    elif group == 'Orbcomp':
        atom_prefix = 'ORB:'
        weight_atom_terminator = ','
    elif group == 'MCD':
        atom_prefix = 'At'
        weight_atom_terminator = 'NC'
    else:
        raise ValueError(f'Unknown group: {group}')

    recipe = HDF5Recipe({
        'datasets': {
            'eigenvalues': {
                'h5path':
                f'/{group}/BS/eigenvalues',
                'transforms': [
                    AttribTransformation(name='shift_by_attribute',
                                         attrib_name='fermi_energy',
                                         kwargs={
                                             'negative': True,
                                         }),
                    Transformation(name='multiply_scalar', args=(HTR_TO_EV,)),
                    Transformation(name='split_array', kwargs={
                        'suffixes': ['up', 'down'],
                        'name': 'eigenvalues'
                    }),
                    Transformation(name='flatten_array')
                ],
                'unpack_dict':
                True
            },
            'kpath': {
                'h5path':
                '/kpts/coordinates',
                'transforms': [
                    AttribTransformation(name='multiply_by_attribute',
                                         attrib_name='reciprocal_cell',
                                         kwargs={'transpose': True}),
                    Transformation(name='calculate_norm', kwargs={'between_neighbours': True}),
                    Transformation(name='cumulative_sum'),
                    AttribTransformation(name='repeat_array_by_attribute', attrib_name='nbands'),
                ]
            },
        },
        'attributes': {
            'group_name': {
```

(continues on next page)

(continued from previous page)

```
                'h5path': f'/{group}',
                'transforms': [
                    Transformation(name='get_name'),
                ],
            },
            'kpoints': {
                'h5path': '/kpts/coordinates',
            },
            'nkpts': {
                'h5path': '/Local/BS/eigenvalues',
                'transforms': [Transformation(name='get_shape'),
                               Transformation(name='index_dataset', args=(1,))]
            },
            'nbands': {
                'h5path': '/Local/BS/eigenvalues',
                'transforms': [Transformation(name='get_shape'),
                               Transformation(name='index_dataset', args=(2,))]
            },
            'atoms_elements': {
                'h5path': '/atoms/atomicNumbers',
                'description': 'Atomic numbers',
                'transforms': [Transformation(name='periodic_elements')]
            },
            'n_types': {
                'h5path':
                '/atoms',
                'description':
                'Number of atom types',
                'transforms':
                [Transformation(name='get_attribute', args=('nTypes',)),
                 Transformation(name='get_first_element')]
            },
            'atoms_position': {
                'h5path': '/atoms/positions',
                'description': 'Atom coordinates per atom',
            },
            'atoms_groups': {
                'h5path': '/atoms/equivAtomsGroup'
            },
            'reciprocal_cell': {
                'h5path': '/cell/reciprocalCell'
            },
            'bravais_matrix': {
                'h5path': '/cell/bravaisMatrix',
                'description': 'Coordinate transformation internal to physical for atoms
↪',
                'transforms': [Transformation(name='multiply_scalar', args=(BOHR_A,))]
            },
            'special_kpoint_indices': {
                'h5path': '/kpts/specialPointIndices',
                'transforms': [Transformation(name='shift_dataset', args=(-1,))]
            },
```

(continues on next page)

(continued from previous page)

```
            'special_kpoint_labels': {
                'h5path': '/kpts/specialPointLabels',
                'transforms': [Transformation(name='convert_to_str')]
            },
            'fermi_energy': {
                'h5path':
                '/general',
                'description':
                'fermi_energy of the system',
                'transforms': [
                    Transformation(name='get_attribute', args=('lastFermiEnergy',)),
                    Transformation(name='get_first_element')
                ]
            },
            'spins': {
                'h5path':
                '/general',
                'description':
                'number of distinct spin directions in the system',
                'transforms':
                [Transformation(name='get_attribute', args=('spins',)),
                 Transformation(name='get_first_element')]
            }
        }
    })

    if simple:
        return recipe

    recipe['datasets']['weights'] = {
        'h5path':
        f'/{group}/BS',
        'transforms': [
            Transformation(name='get_all_child_datasets', kwargs={'ignore': ['eigenvalues', 'kpts']}),
            AttribTransformation(name='add_partial_sums',
                                 attrib_name='atoms_groups',
                                 args=(f'{atom_prefix}{{}}{weight_atom_terminator}'.format,),
                                 kwargs={
                                     'make_set': True,
                                     'replace_format': f'{atom_prefix}{{}}'.format
                                 }),
            Transformation(name='split_array', kwargs={'suffixes': ['up', 'down']}),
            Transformation(name='flatten_array')
        ],
        'unpack_dict':
        True
    }

    return recipe
```

Each recipe can define the `datasets` and `attributes` entry (if one is not defined, a empty dict is returned in its place).

Each entry in these sections has the same structure.

```
#Example entry from the FleurBands recipe

'fermi_energy': {
        'h5path':
        '/general',
        'description':
        'fermi_energy of the system',
        'transforms': [
            Transformation(name='get_attribute', args=('lastFermiEnergy',), kwargs={}),
            Transformation(name='get_first_element', args=(), kwargs={})
        ]
    }
```

All entries must define the key `h5path`. This gives the initial dataset for this key, which will be extracted from the given `.hdf` file. The key of the entry corresponds to the key under which the result will be saved to the output dictionary.

If the dataset should be transformed in some way after reading it, there are a number of defined transformations in *masci_tools.io.parsers.hdf5.transforms*. These are added to an entry by adding a list of namedtuples (*reader.Transformation* for general transformations; *reader.AttribTransformation* for attribute transformations) under the key `transforms`. General Transformations can be used in all entries, while transformations using an attribute value can only be used in the `datasets` entries. Each namedtuple takes the `name` of the transformation function and the positional (`args`), and keyword arguments (`kwargs`) for the transformation. Attribute transformations also take the name of the attribute, whose value should be passed to the transformation in `attrib_name`.

At the moment the following transformation functions are pre-defined:

General Transformations:

- *get_first_element()*: Get the index `0` of the dataset
- *index_dataset()*: Get the index `index` of the dataset
- *slice_dataset()*: Slice the given dataset with the given argument
- *get_shape()*: Get the shape of the dataset
- *tile_array()*: Use np.tile to repeat dataset a given amount of times
- *repeat_array()*: Use np.repeat to repeat each element in the dataset a given amount of times
- *get_all_child_datasets()*: extract all datasets contained in the current hdf group and enter them into a dict
- *merge_subgroup_datasets()*: extract all datasets contained in the subgroups of the current hdf group and enter them into a dict in a list (or one numpy array)
- *stack_datasets()*: Stack the given datasets in the dictionary along a given axis
- *shift_dataset()*: Shift the given dataset with a scalar value
- *multiply_scalar()*: Multiply the given dataset with a scalar value
- *multiply_array()*: Multiply the given dataset with a given array
- *convert_to_complex_array()*: Convert real dataset to complex array
- *calculate_norm()*: Calculate norm of list of vectors (either absolute or difference between subsequent entries)
- *cumulative_sum()*: Calculative cumulative sum of dataset
- *get_attribute()*: Get the value of one given attribute on the dataset
- *attributes()*: Get all defined attributes on the dataset as a dict

- `move_to_memory()`: Convert dataset to numpy array (if not already done implicitly)
- `flatten_array()`: Create copy of dataset flattened into one dimension
- `split_array()`: Split the given dataset along its first index and store result in a dictionary with keys with suffixes
- `convert_to_str()`: Convert datatype of dataset to string
- `periodic_elements()`: Convert atomic numbers to their atomic symbols

Transformations using an attribute:

- `multiply_by_attribute()`: Multiply dataset by value of attribute (both scalar and matrix)
- `shift_by_attribute()`: Shift the given dataset with the value of an attribute
- `repeat_array_by_attribute()`: Call `repeat_array()` with the value of an attribute as argument
- `tile_array_by_attribute()`: Call `tile_array()` with the value of an attribute as argument
- `add_partial_sums()`: Sum over entries in dictionary datasets with given patterns in the key (Pattern is formatted with given attribute value)

Custom transformation functions can also be defined using the `hdf5_transformation()` decorator. For some transformation, e.g. `get_all_child_datasets()`, the result will be a subdictionary in the `datasets` or `attributes` dictionary. If this is not desired the entry can include `'unpack_dict': True`. With this all keys from the resulting dict will be extracted after all transformations and put into the root dictionary.

4.1.4 Plotting Fleur DOS/bandstructures

This section discusses how to obtain plots of data in the `banddos.hdf` for density of states and bandstructure calculations.

In the following bandstructure and DOS plots are explained. Each section leads with the names of the recipes from the `recipes` module that can be used with the explained visualization function.

All Fleur specific plotting routines are found in `fleur` have implementations for both the matplotlib and bokeh plotting libraries and can be customized heavily. For an explanation on customizing plots refer to *General Plotting routines*.

Contents

- *Selecting specific DOS components*
- *Plotting DOS without spinpolarization*
- *Selecting a specific spin channel*

4.1.4.1 Reading `banddos.hdf` files

The process here is divided in two parts. First we extract and transform the data in a way to make it easy to plot via the *`reader.HDF5Reader`*. For a detailed explanation of the capabilities of this tool refer to *General HDF5 file reader*. Here we show the basic usage:

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands

with HDF5Reader('files/banddos_bands.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurBands)

print(f"The following datasets were read: {list(data.keys())}")
print(f"The following attributes were read: {list(attributes.keys())}")
```

```
The following datasets were read: ['eigenvalues_up', 'kpath', 'INT_up', 'MT:1d_up',
↪'MT:1f_up', 'MT:1p_up', 'MT:1s_up', 'Sym_up', 'Total_up', 'MT:1_up']
The following attributes were read: ['group_name', 'kpoints', 'nkpts', 'nbands', 'atoms_
↪elements', 'n_types', 'atoms_position', 'atoms_groups', 'reciprocal_cell', 'bravais_
↪matrix', 'special_kpoint_indices', 'special_kpoint_labels', 'fermi_energy', 'spins']
```

4.1.4.2 Bandstructures

Compatible Recipes for the *`reader.HDF5Reader`*:

- `FleurBands`: Default recipe reading in the kpoints, eignevalues and weights for atom and orbital contributions
- `FleurSimpleBands`: Reads in only the kpoints and eigenvalues and now weights
- `FleurOrbcompBands`: In addition to the eigenvalues the weights from an orbital decomposition calculation are read in
- `FleurjDOSBands`: In addition to the eigenvalues the weights from a jDOS calculation are read in
- `FleurMCDBands`: In addition to the eigenvalues the weights from a MCD calculation are read in
- *`recipes.get_fleur_bands_specific_weights()`*: Function to generate a recipe for reading in the eigenvalues+a provided list of weights

The bandstructure visualization *`plot_fleur_bands()`* can be used to plot

1. Non-spinpolarized/spinpolarized bandstructures
2. Bandstructures with emphasized weights on all eigenvalues (Also non-spinpolarized and spinpolarized)

Standard bandstructure

To plot a simple bandstructure without any weighting we just have to pass the data, that the *HDF5Reader* provided to the *plot_fleur_bands()*.

The two examples below show the resulting plots for a non-psinpolarized system (bulk Si) and a spin-polarized system (Fe fcc). For both systems the necessary code is exactly the same and is shown above the plots. The shown plots are the ones for the matplotlib plotting backend:

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands
from masci_tools.vis.fleur import plot_fleur_bands

#Read in data
with HDF5Reader('files/banddos_bands.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurBands)

#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_bands(data,
                      attributes,
                      limits={'y': (-13, 5)},
                      markersize=10)
```

Bandstructure with weights

To plot a simple bandstructure with weighting we do the same procedure as above, but we pass in the entry we want to use for weights. These correspond to the entries in the `banddos.hdf` file (for example the weight for the s-orbital on the first atom type is called `MT:1s`). The weights will be used to change the size and color (according to a colormap) to indicate regions of high weight.

The two examples below show the resulting plots for a non-psinpolarized system (bulk Si) weighted for the s-orbital on the first atom and a spin-polarized system (Fe fcc) with weights for the d-orbital on the first atom type. For both systems the necessary code is exactly the same and is shown above the plots. The shown plots are the ones for the matplotlib plotting backend:

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands
from masci_tools.vis.fleur import plot_fleur_bands

#Read in data
with HDF5Reader('files/banddos_bands.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurBands)

#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_bands(data,
                      attributes,
                      weight='MT:1s',
                      limits={'y': (-13,5)})
```

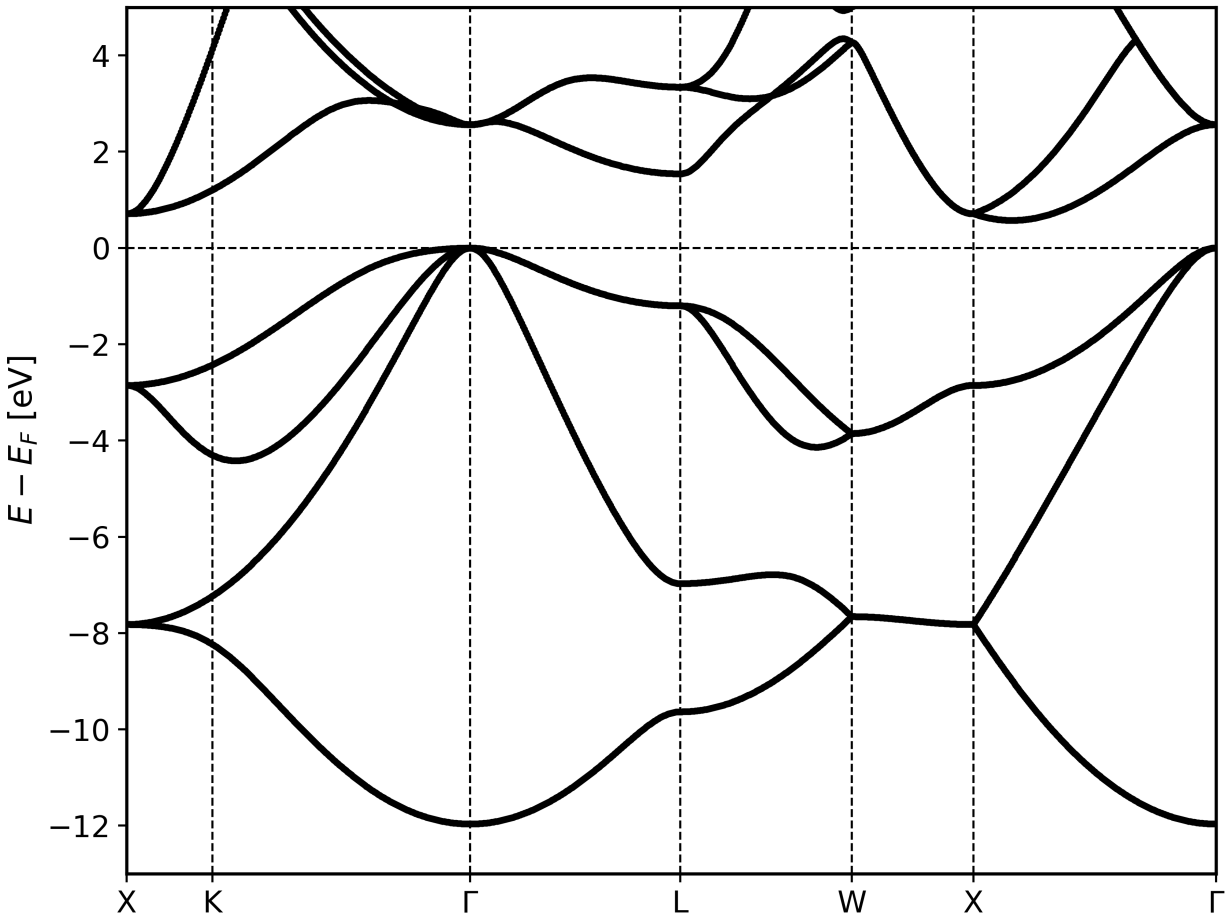


Fig. 1: Non spinpolarized bandstructure for a bulk Si structure

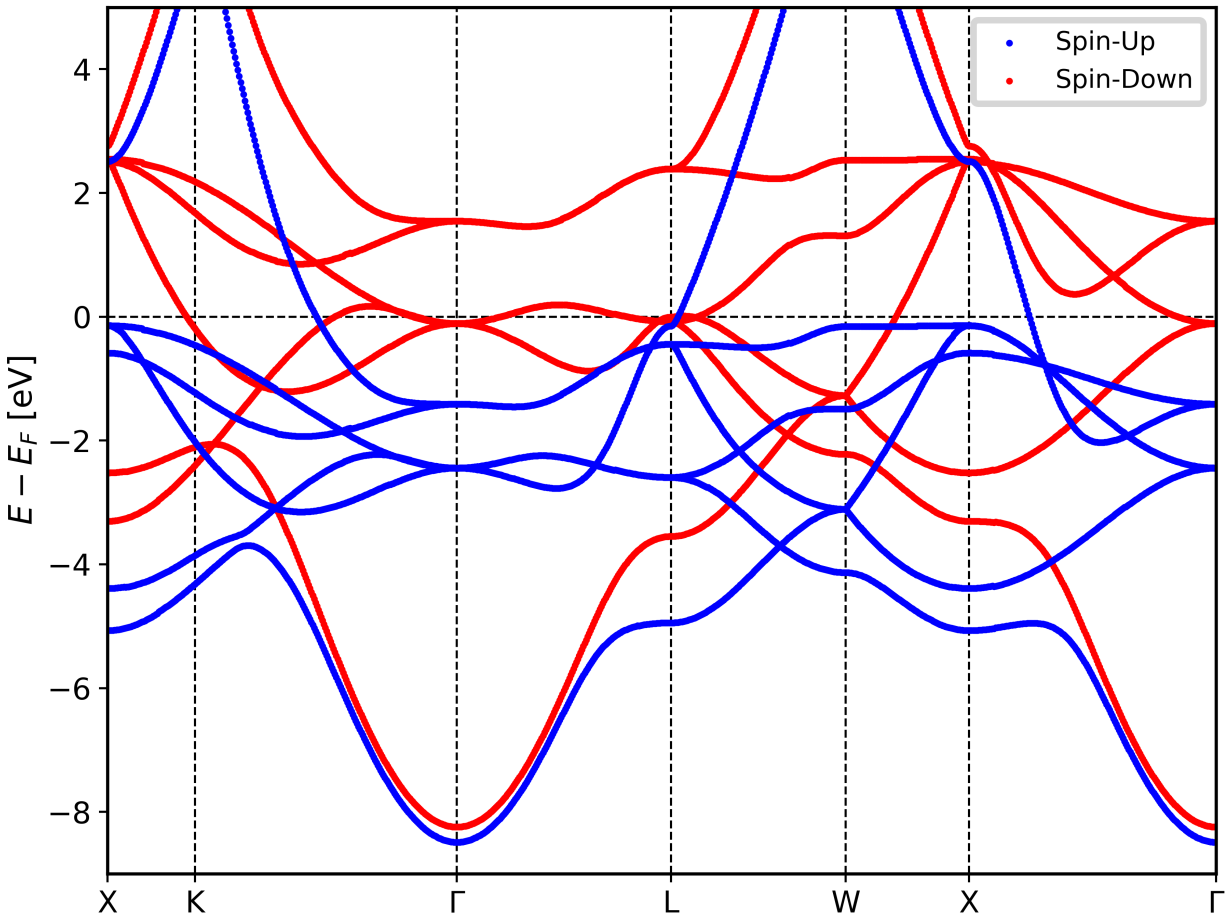


Fig. 2: Spinpolarized bandstructure for a bulk Fe fcc structure

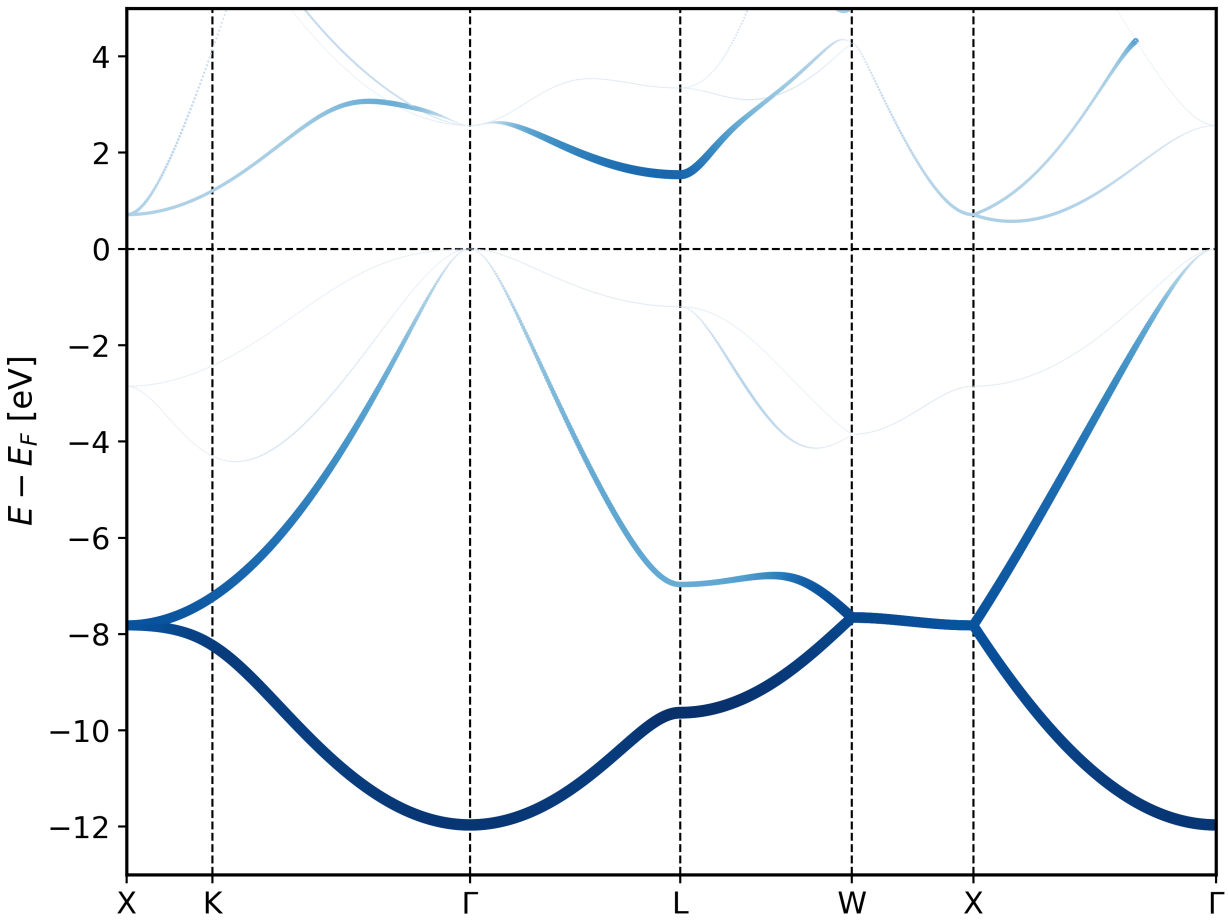


Fig. 3: Non spinpolarized bandstructure for a bulk Si structure. The s-like character inside the Muffin-tin sphere is highlighted

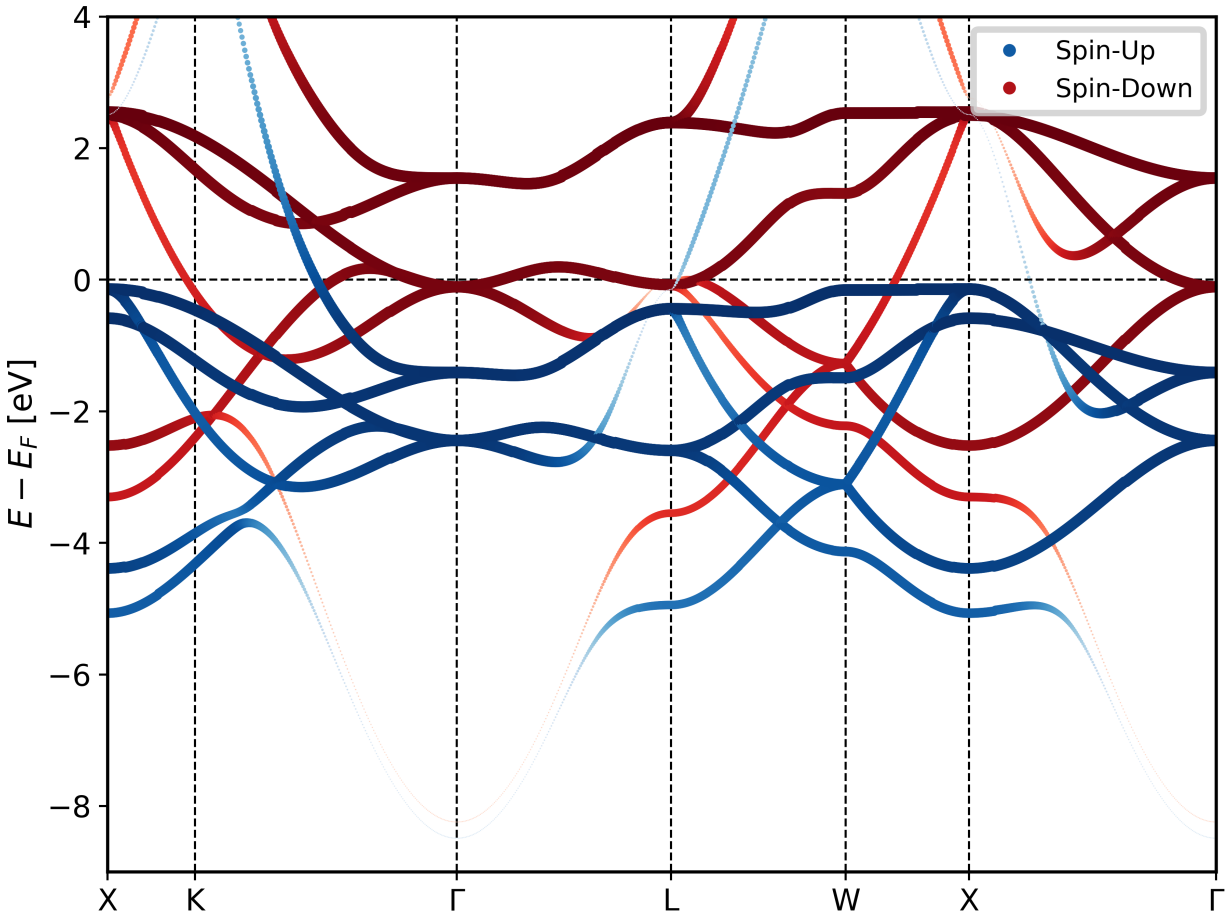


Fig. 4: Spinpolarized bandstructure for a bulk Fe fcc structure. The d-like character inside the Muffin-tin sphere is highlighted

Plotting options for bandstructure plots

The *plot_fleur_bands()* function has a couple of options to modify, what is being displayed from the `banddos.hdf` file. Below we show a few examples of ways to use these options, together with examples of resulting plots.

Plotting bandstructure without spinpolarization

Providing `spinpol=False` will display the bandstructure as non spinpolarized, even if there are two spins in the data. Works for both non-weighted and weighted bandstructures.

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands
from masci_tools.vis.fleur import plot_fleur_bands

#Read in data
with HDF5Reader('files/banddos_spinpol_bands.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurBands)

#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_bands(data,
                      attributes,
                      limits={'y': (-9,4)},
                      markersize=10,
                      spinpol=False)
```

Selecting a specific spin channel

Providing `only_spin='up'` or `'down'` will plot only the given spin channel

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands
from masci_tools.vis.fleur import plot_fleur_bands

#Read in data
with HDF5Reader('files/banddos_spinpol_bands.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurBands)

#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_bands(data,
                      attributes,
                      limits={'y': (-9,4)},
                      only_spin='up',
                      markersize=10,
                      color='darkblue')
```

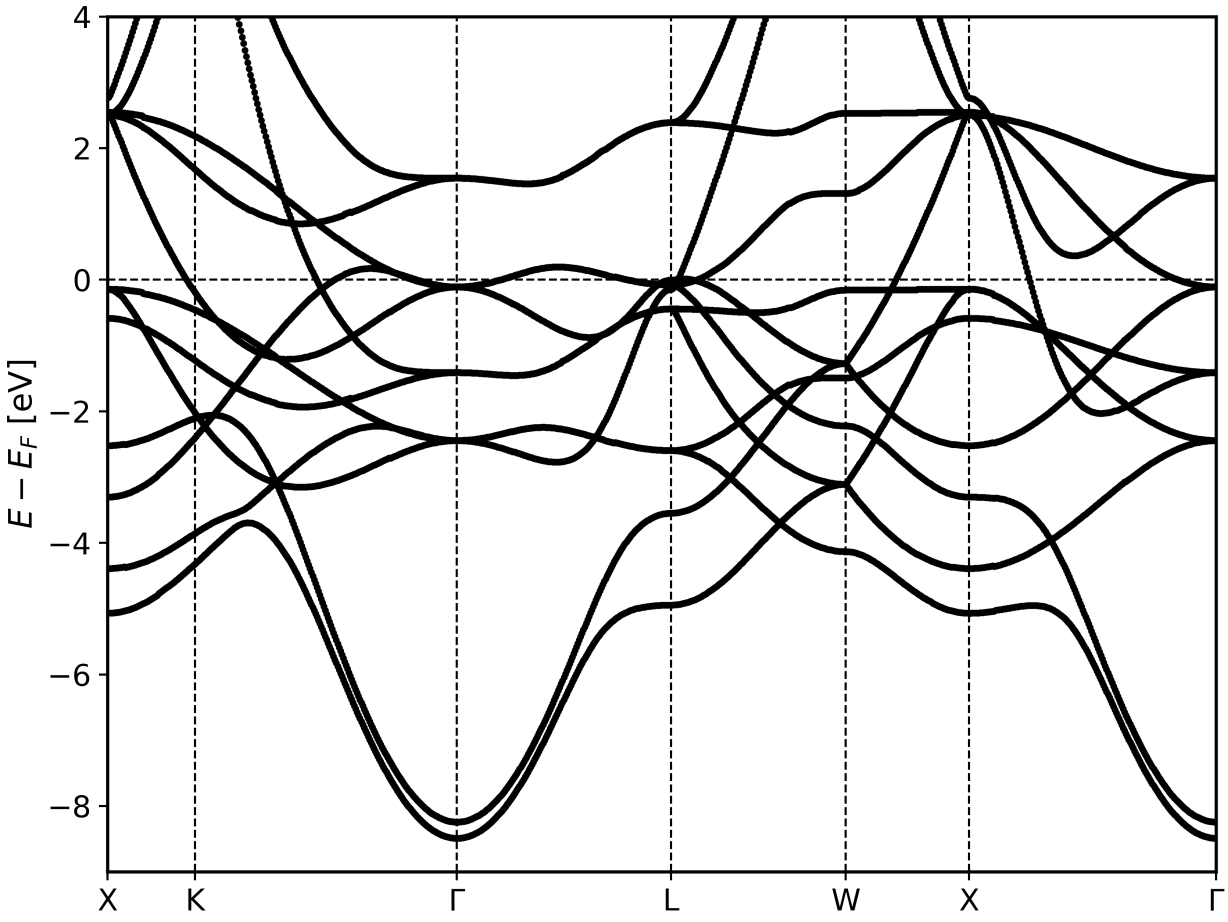


Fig. 5: Non spinpolarized bandstructure for a bulk Fe fcc structure.

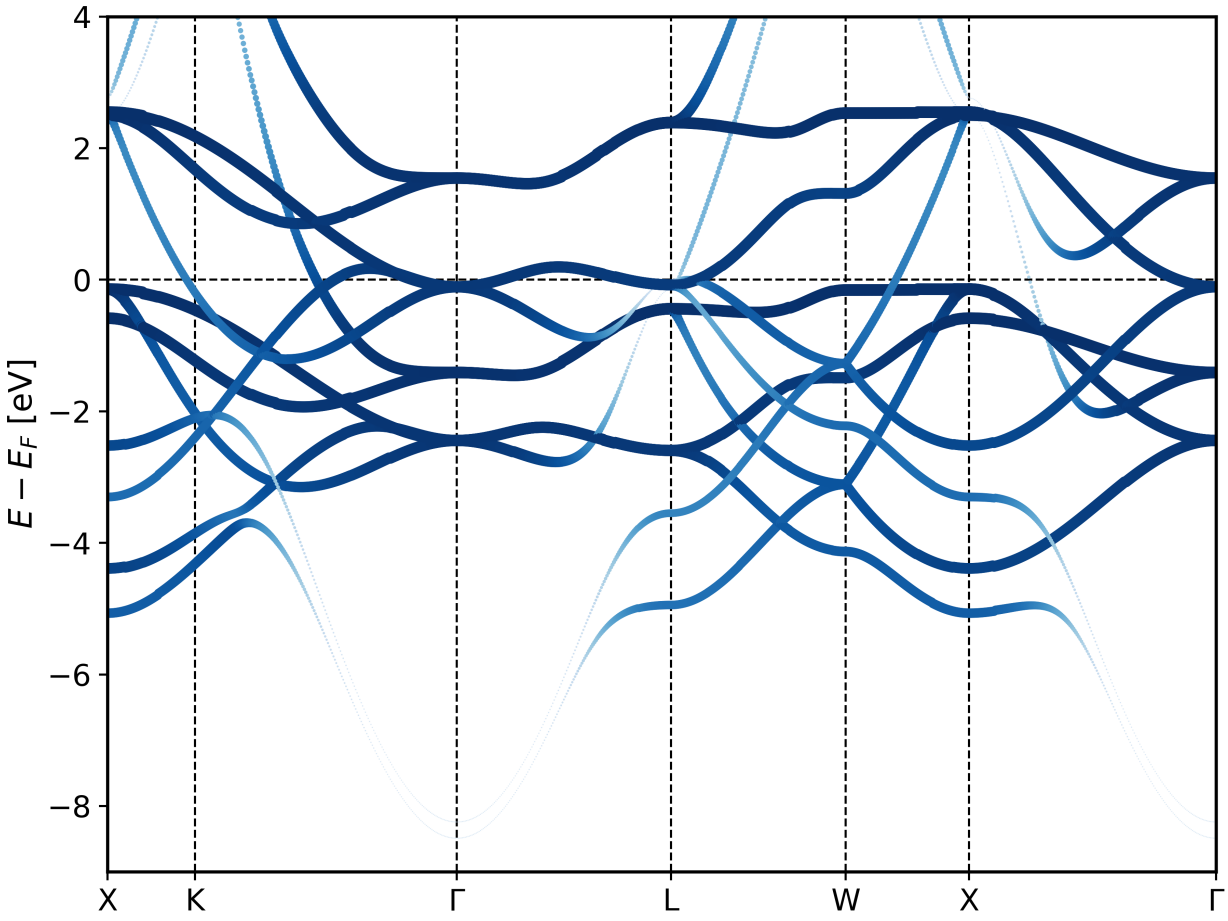


Fig. 6: Non spinpolarized bandstructure for a bulk Fe fcc structure. The d-like character inside the Muffin-tin sphere is highlighted

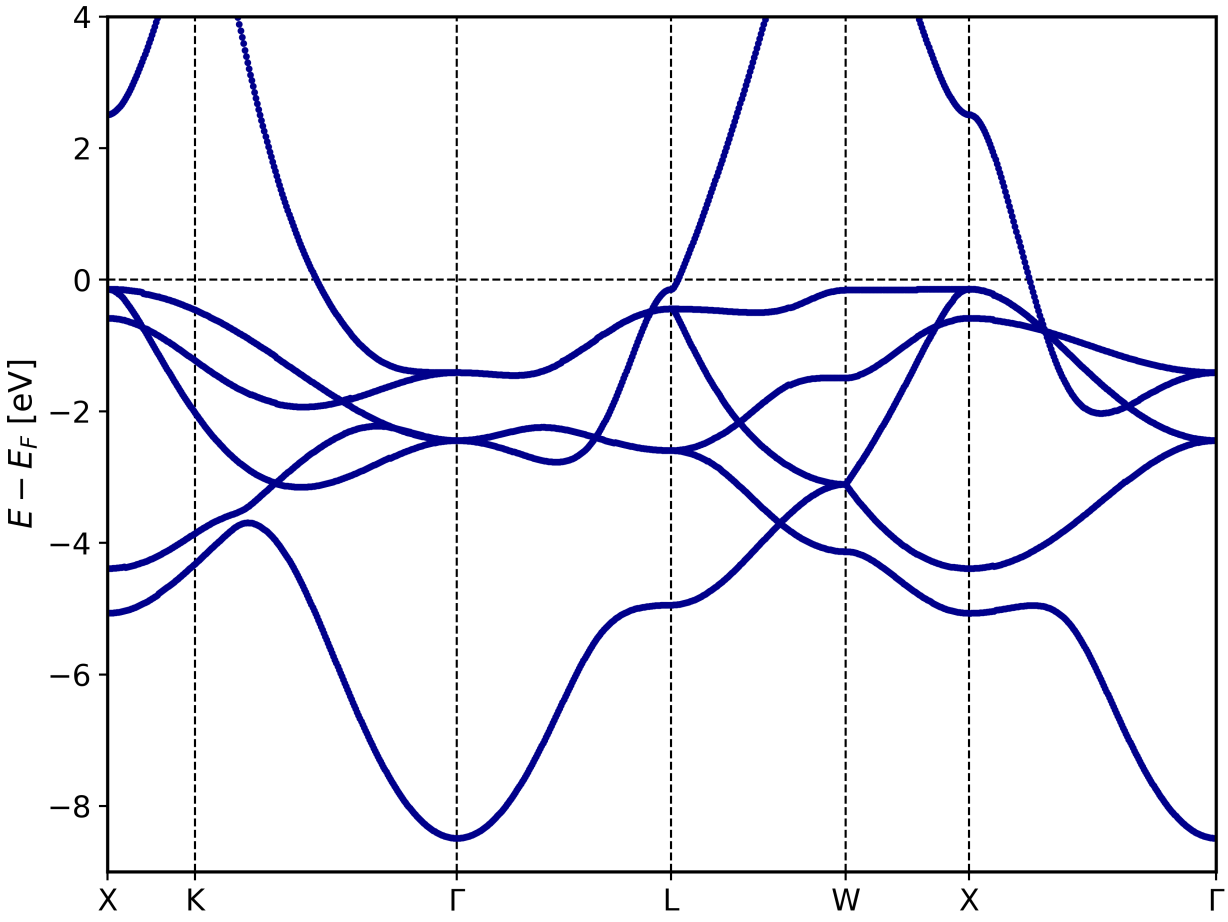


Fig. 7: Bandstructure for a bulk Fe fcc structure (only spin up).

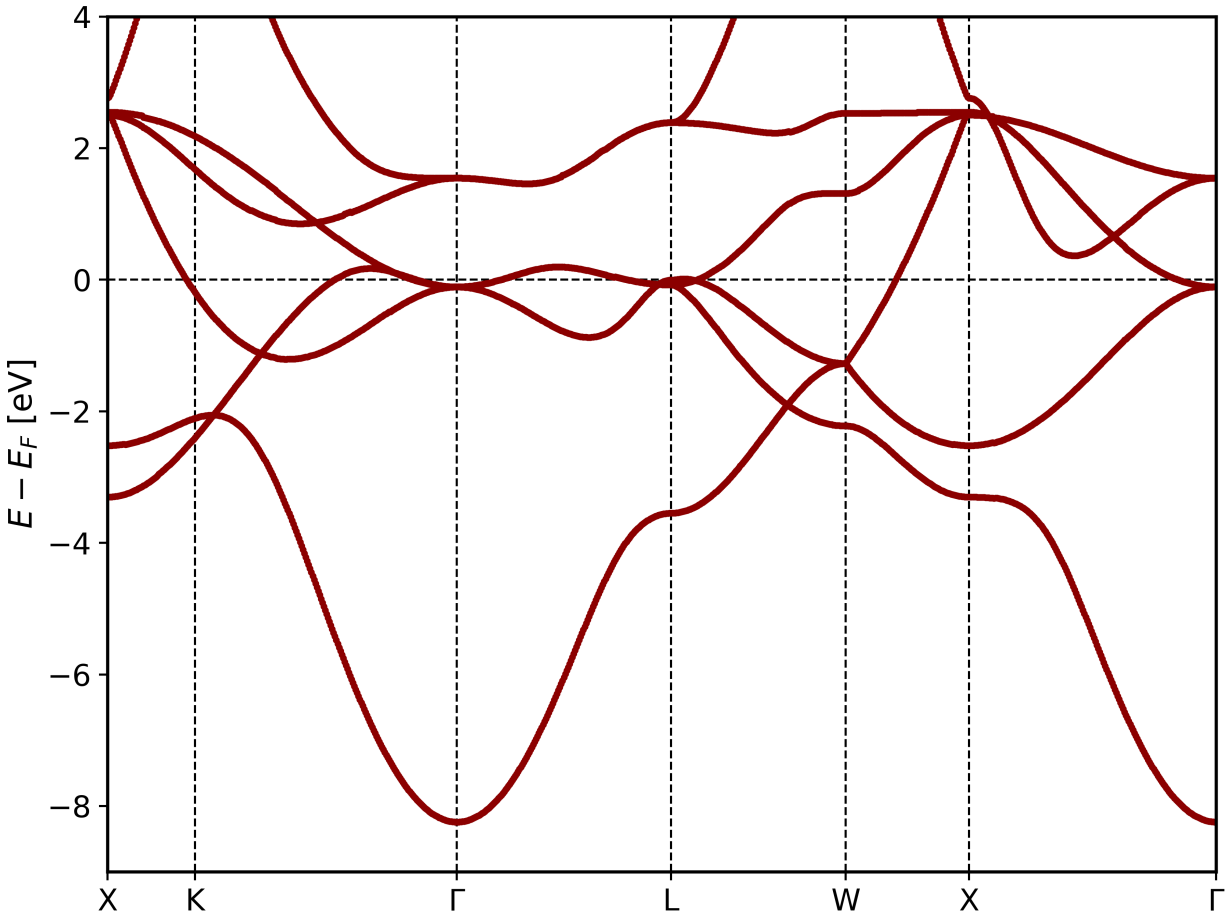


Fig. 8: Bandstructure for a bulk Fe fcc structure (only spin down). The color is changed manually

4.1.4.3 Density of States

Compatible Recipes for the *HDF5Reader*:

- `FleurDOS`: Default recipe reading in the total, interstitial, vacuum, atom and l-channel resolved DOS
- `FleurORBCOMP`: Read in the DOS from an orbital decomposition calculation
- `FleurJDOS`: Read in the DOS from a jDOS calculation
- `FleurMCD`: Read in the DOS from a MCD calculation

The dos visualization *plot_fleur_dos()* can be used to plot non spinpolarized and spinpolarized DOS, with selection of which components to plot.

Standard density of states plot

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurDOS
from masci_tools.vis.fleur import plot_fleur_dos

#Read in data
with HDF5Reader('files/banddos_dos.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurDOS)

#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_dos(data,
                    attributes,
                    limits={'energy': (-13,5)})
```

Plotting options for DOS plots

The *plot_fleur_dos()* function has a couple of options to modify, what is being displayed from the `banddos.hdf` file. Below we show a few examples of ways to use these options, together with examples of resulting plots.

Selecting specific DOS components

The DOS is made up of a lot of contributions that can be displayed separately.

Here we list the options that are available and show example plots for only selecting the atom projected compinents of the density of states

- `plot_keys`: Can be used to provide a explicit list of keys you want to display (Same format as in the `banddos.hdf`)
- `show_total`: Control, whether to show the total density of states (default `True`)
- `show_interstitial`: Control, whether to show the interstitial contribution of the density of states (default `True`)
- `show_atoms`: Control, which total atom projected DOS to show. Can be either the string `all` (All components are shown), the value `None` (no components are shown) or a list of the integer indices of the atom types that should be displayed (default `all`)

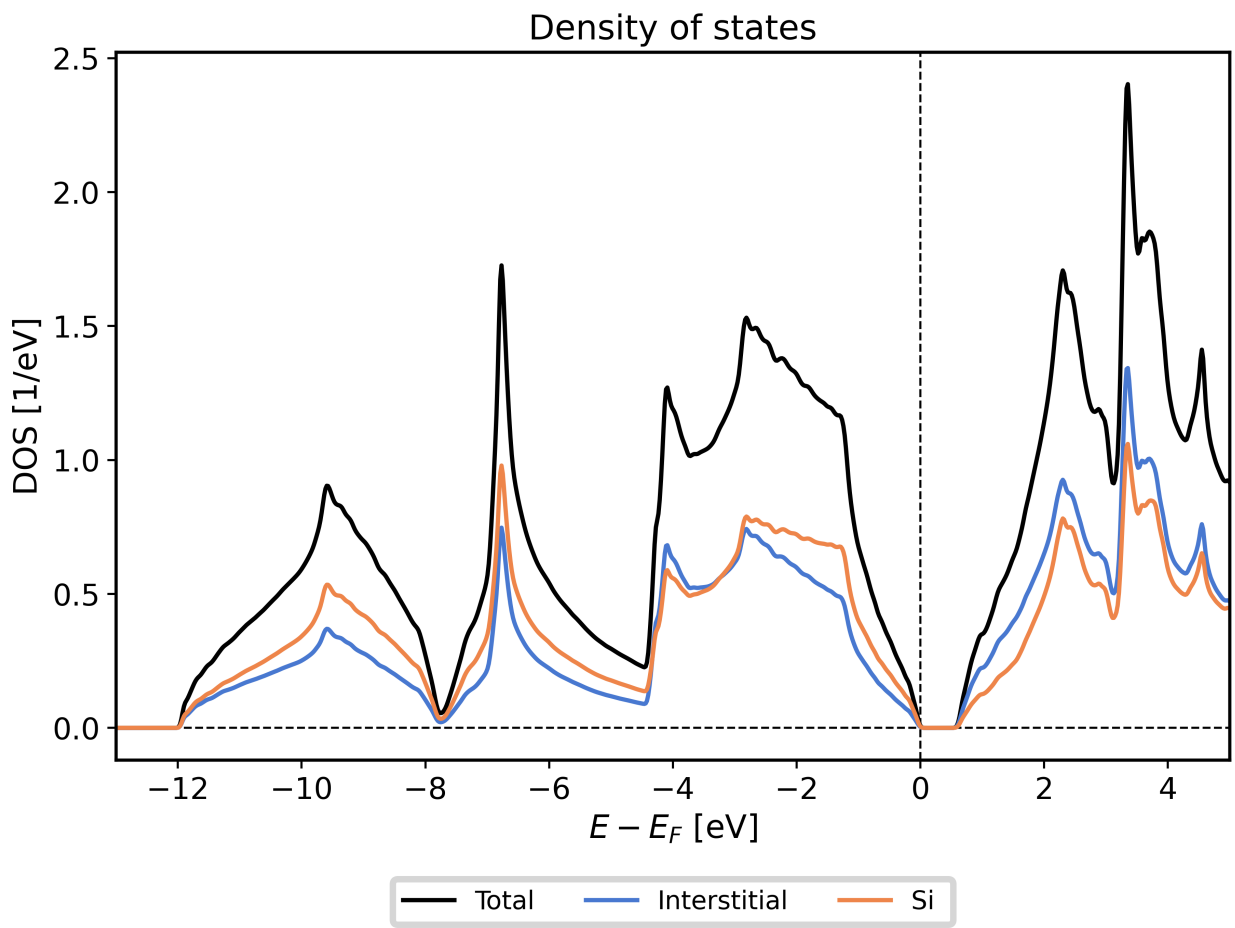


Fig. 9: Non spinpolarized DOS for a bulk Si structure

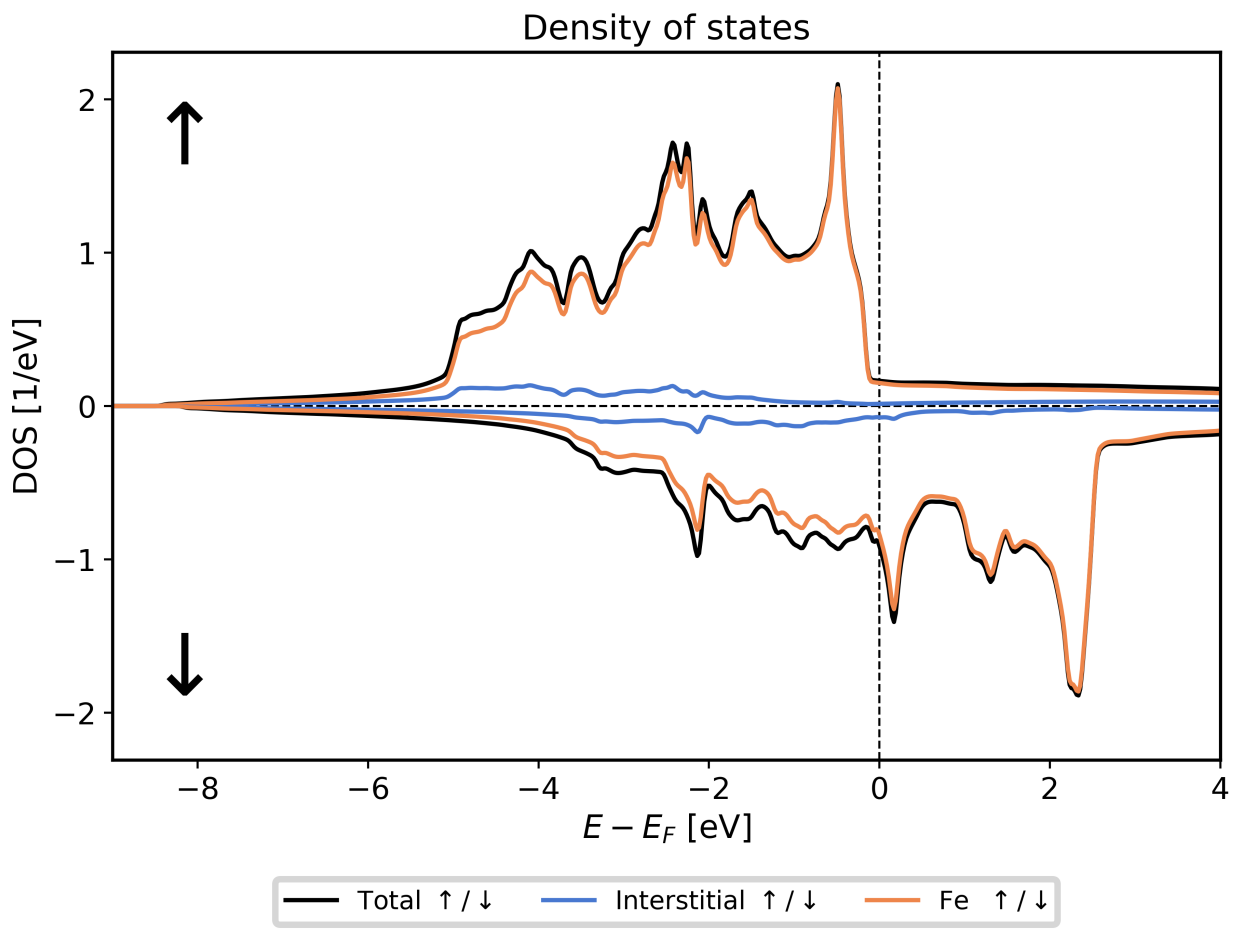


Fig. 10: Spinpolarized DOS for a bulk Fe fcc structure

- `show_lresolved`: Control, on which atoms to show the orbital projected DOS. Can be either the string `all` (All components are shown), the value `None` (no components are shown) or a list of the integer indices of the atom types for which to display the orbital components (default `None`)

Below an example of only displaying the atom projected DOS together with their orbital contributions is shown.

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurDOS
from masci_tools.vis.fleur import plot_fleur_dos

#Read in data
with HDF5Reader('files/banddos_dos.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurDOS)

ax = plot_fleur_dos(data, attributes,
                    show_total=False,
                    show_interstitial=False,
                    show_lresolved='all',
                    limits={'energy': (-13,5)})
```

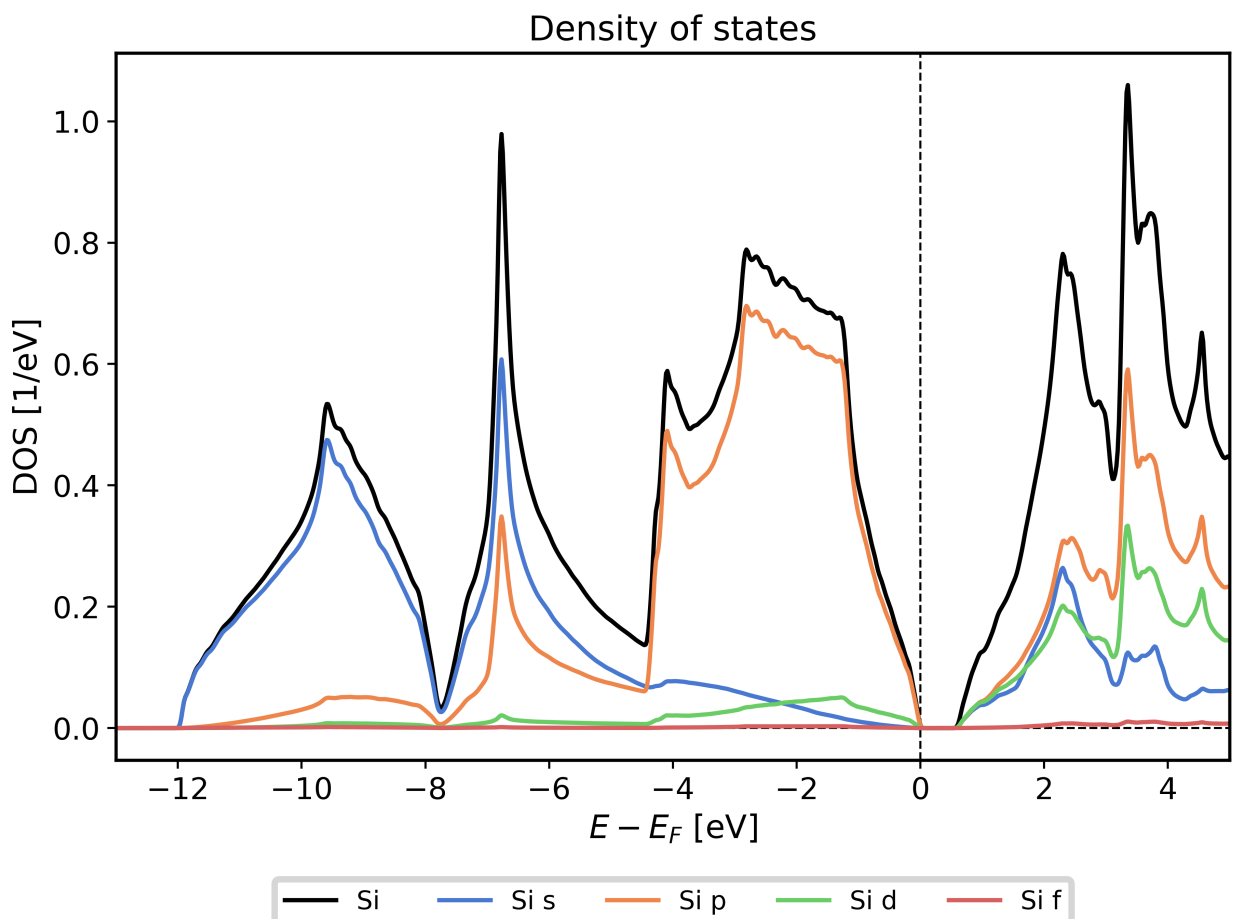


Fig. 11: Non spinpolarized DOS for a bulk Si structure. Only the atom and l-channel projected DOS is shown

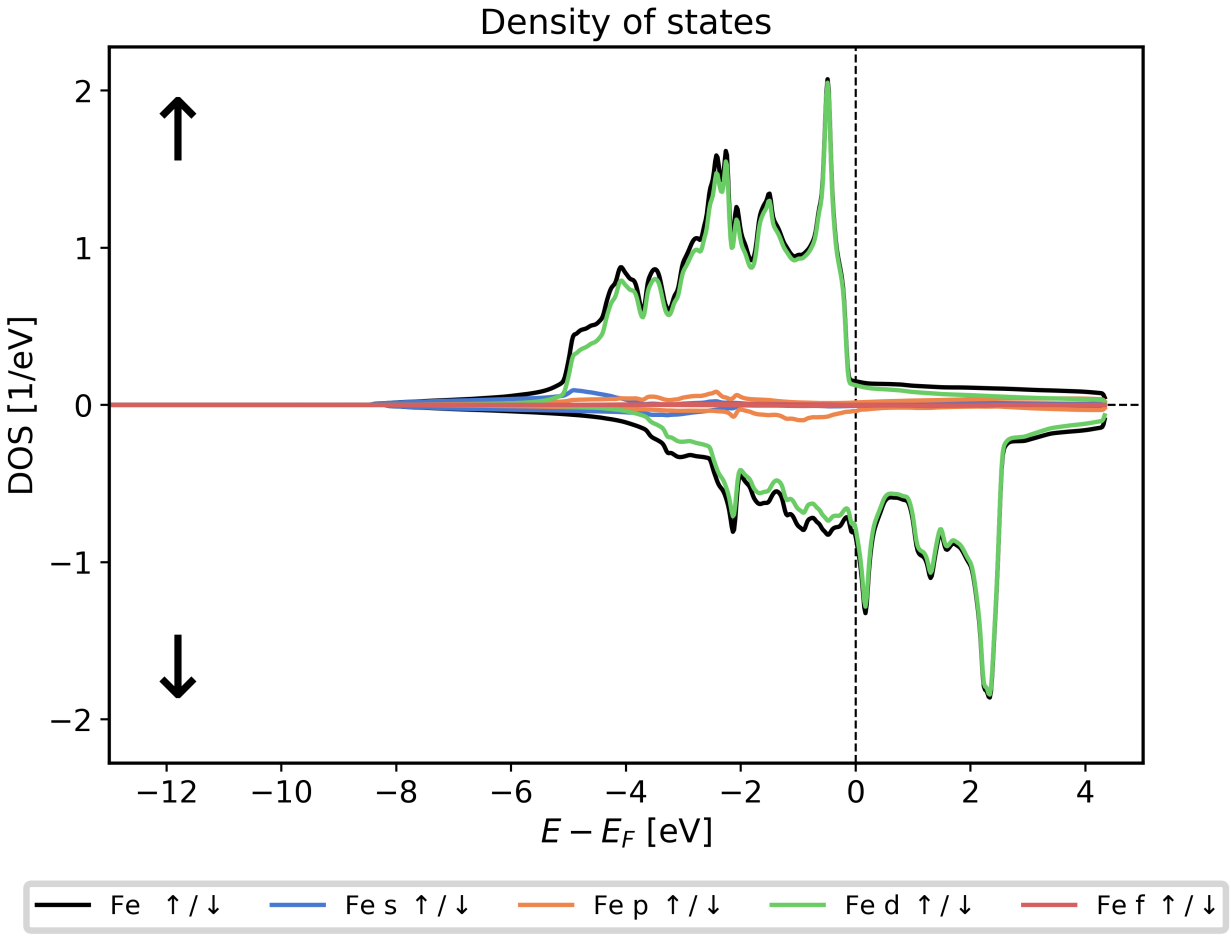


Fig. 12: Non spinpolarized DOS for a bulk Fe fcc structure. Only the atom and l-channel projected DOS is shown

Plotting DOS without spinpolarization

Providing `spinpol=False` will display the DOS as non spinpolarized, even if there are two spins in the data.

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurDOS
from masci_tools.vis.fleur import plot_fleur_dos

#Read in data
with HDF5Reader('files/banddos_spinpol_dos.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurDOS)

#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_dos(data,
                    attributes,
                    limits={'energy': (-9,4)},
                    spinpol=False)
```

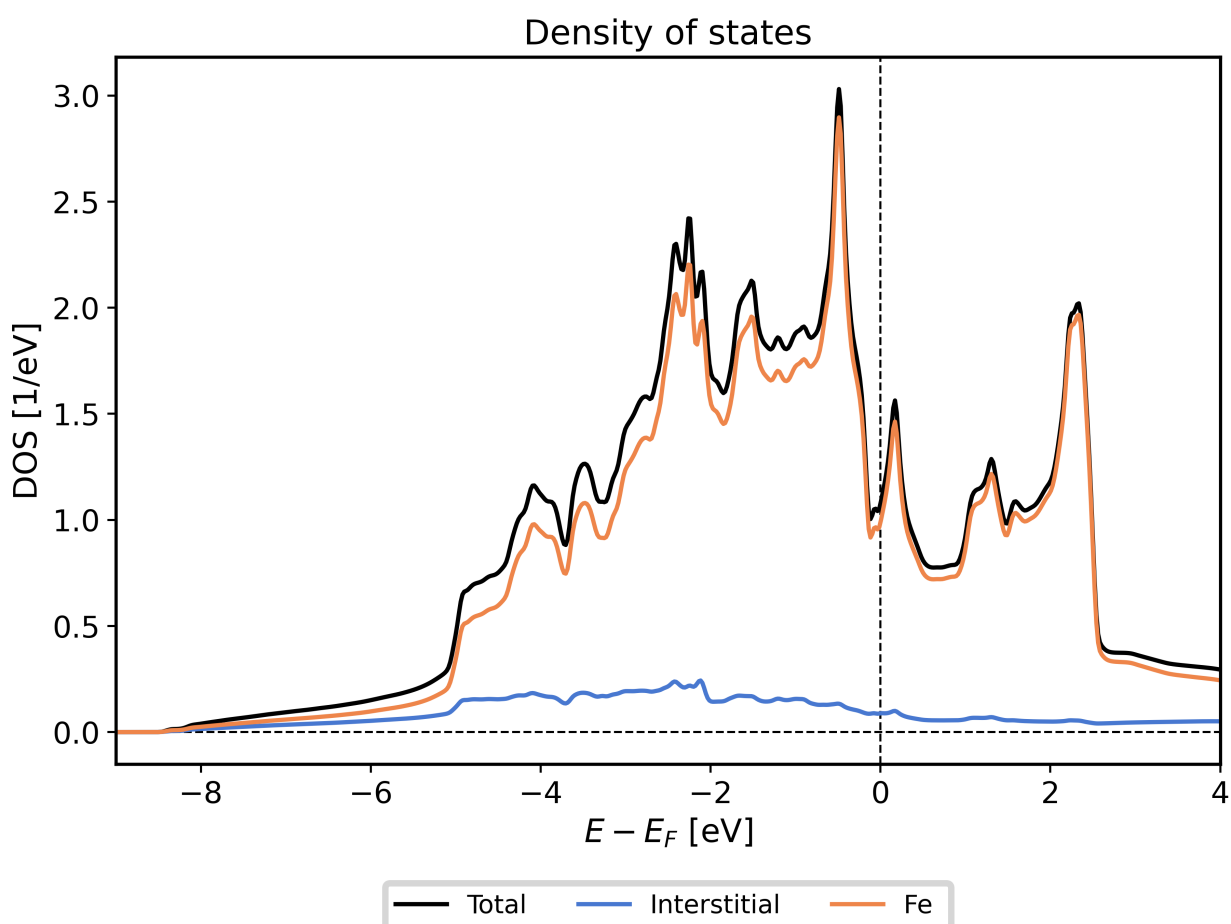


Fig. 13: Non spinpolarized DOS for a bulk Fe fcc structure.

Selecting a specific spin channel

Providing `only_spin='up'` or `'down'` will plot only the given spin channel

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurDOS
from masci_tools.vis.fleur import plot_fleur_dos

#Read in data
with HDF5Reader('files/banddos_spinpol_dos.hdf') as h5reader:
   data, attributes = h5reader.read(recipe=FleurDOS)

#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_dos(data,
                    attributes,
                    limits={'energy': (-9,4)},
                    only_spin='up')
```

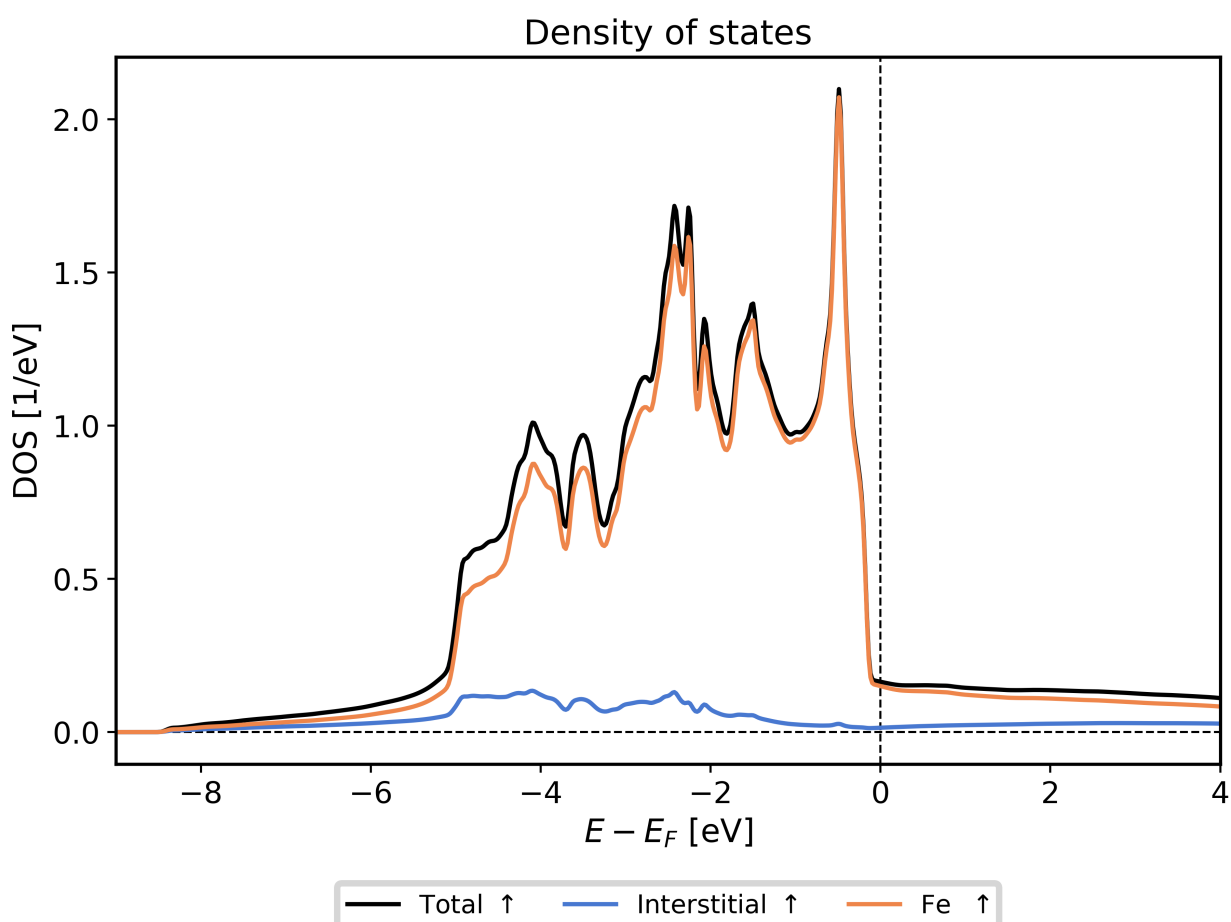


Fig. 14: DOS for a bulk Fe fcc structure (only spin up).

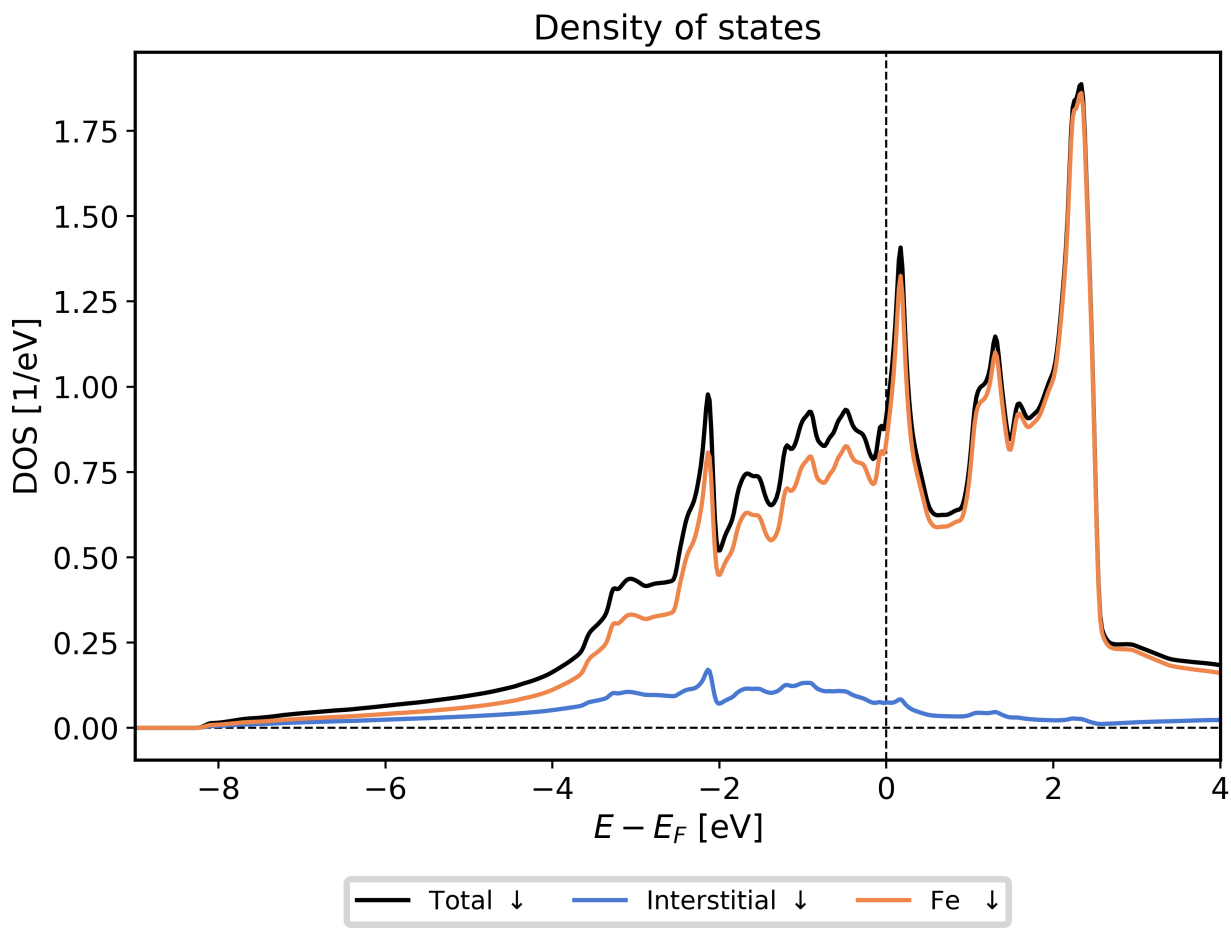


Fig. 15: DOS for a bulk Fe fcc structure (only spin down).

4.1.5 Plotting KKR DOS/bandstructures

This section discusses how the standard output files for density of states and bandstructure calculations of a KKR calculation can be visualized.

Contents

4.1.5.1 Density of states

```
#Example: KKR DOS

from masci_tools.vis.kkr_plot_dos import dosplot

# the path can be a relative or absolute path to the directory
# where the dos.atom files reside (i.e. the dir where the DOS calculation ran)
dosplot('files/kkr_dos/', color='k', lw=4, marker='v', ls=':', ms=8)
```

```
files/kkr_dos/dos.atom1
```

Where the `color`, `lw`, etc inputs are optional settings which customize the plot:

We can also use this to show the l-decomposed DOS (red line are d-orbitals):

```
#Example: KKR DOS, l-resolved

dosplot('files/kkr_dos/', totonly=False)
```

```
files/kkr_dos/dos.atom1
```

4.1.5.2 Bandstructure

```
#Example: KKR bandstructure

from masci_tools.vis.kkr_plot_bandstruc_qdos import dispersionplot

# the path can be a relative or absolute path to the directory
# where the qdos files reside (i.e. the dir where the qdos calculation ran)
dispersionplot('files/kkr_bandstruc/', ptitle='bulk Cu')
```

Which can also be customized with keyword arguments to the `dispersionplot` function:

```
#Example: KKR bandstructure with custom color map
```

(continues on next page)

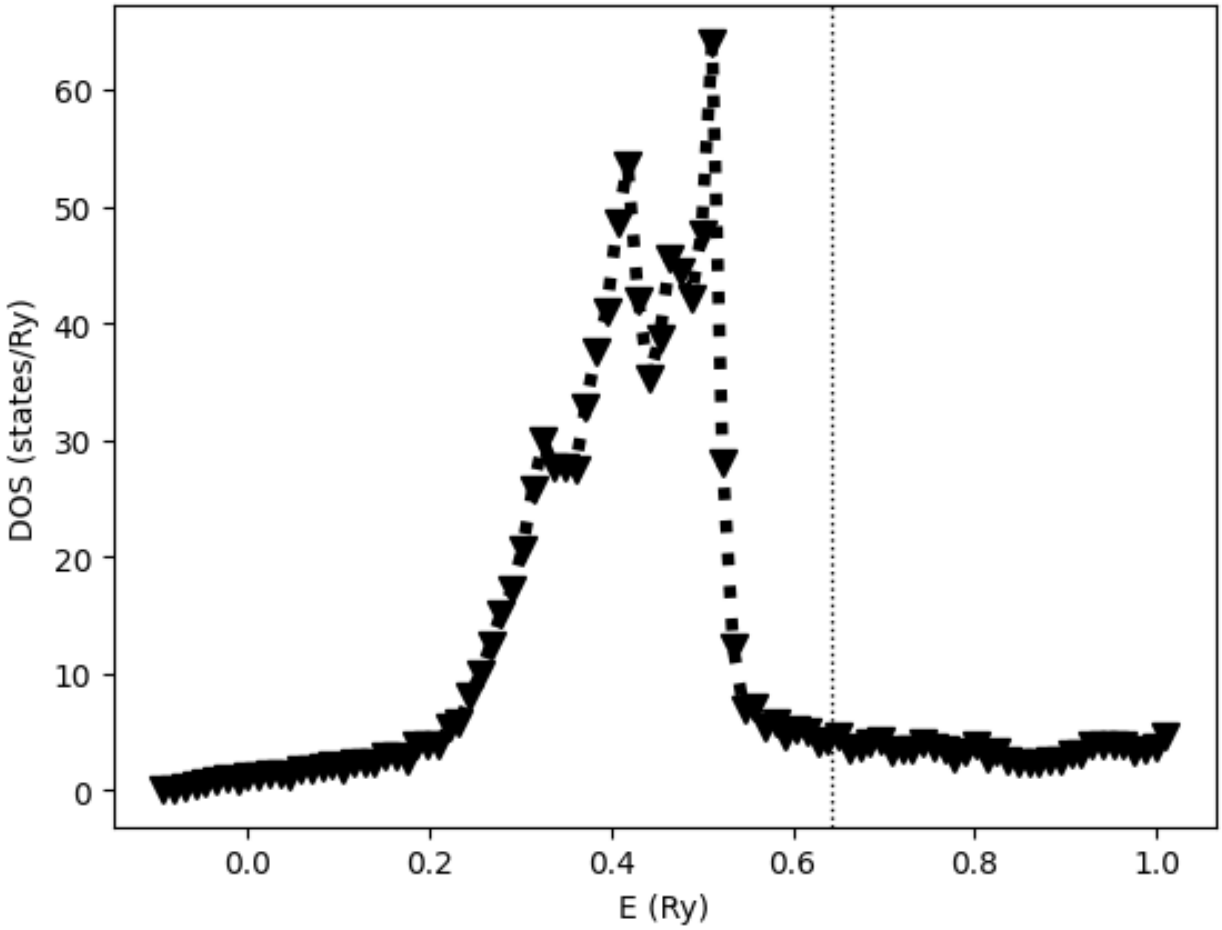


Fig. 16: DOS of bulk fcc Cu.

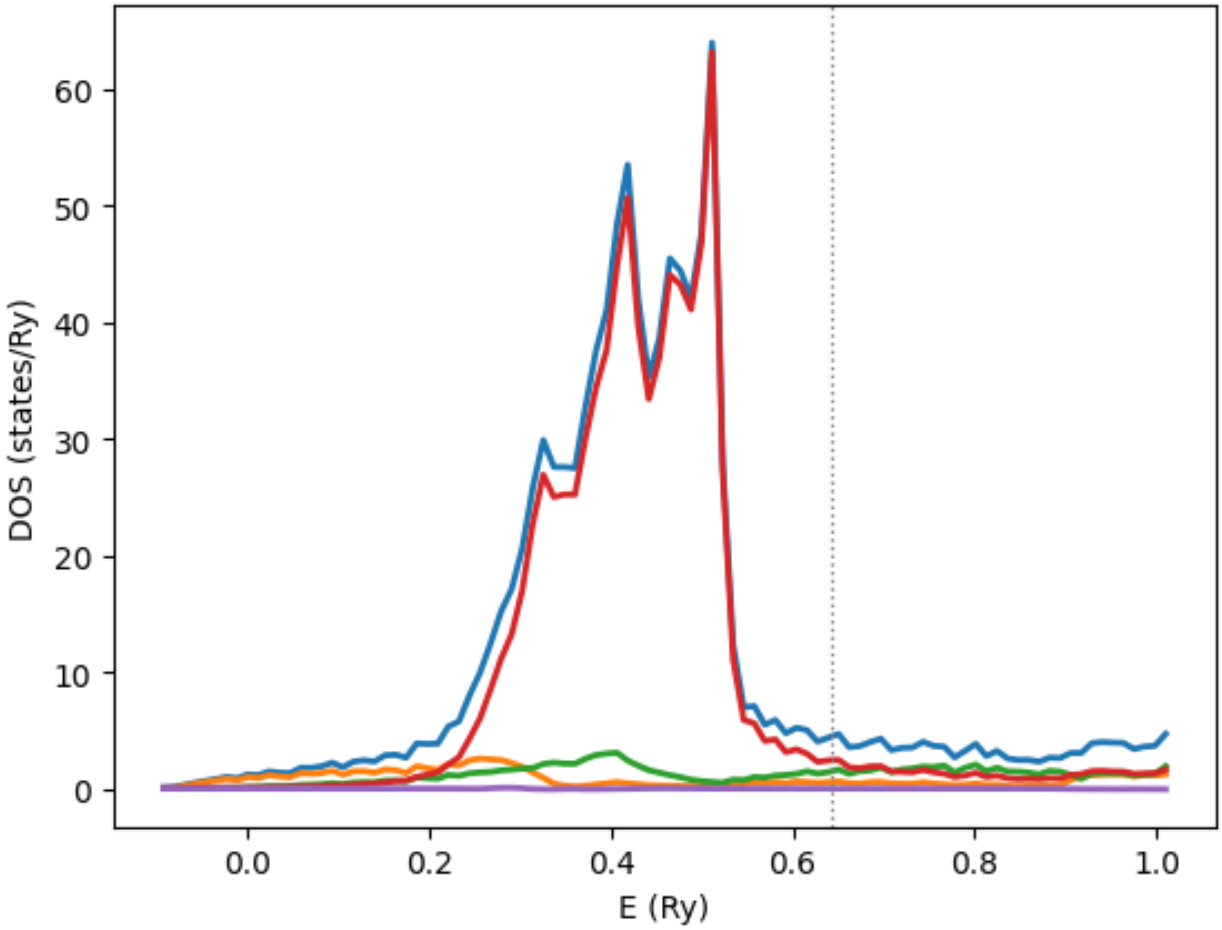


Fig. 17: DOS of bulk fcc Cu, l-resolved.

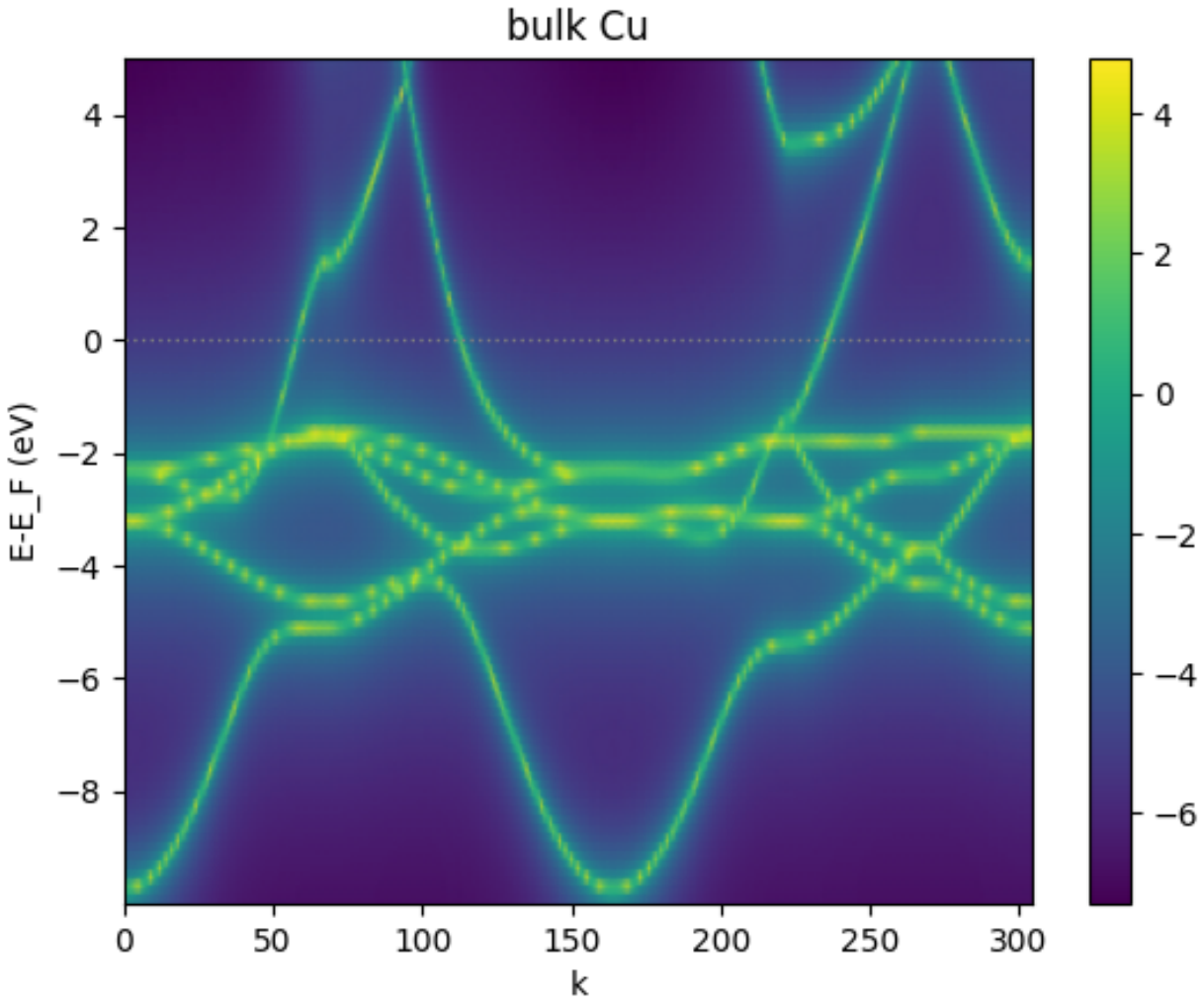


Fig. 18: Bandstructure of bulk fcc Cu.

(continued from previous page)

```
dispersionplot('files/kkr_bandstruc/', ptitle='bulk Cu', cmap='binary', clims=[-2,2],␣
↪clrbar=False)
```

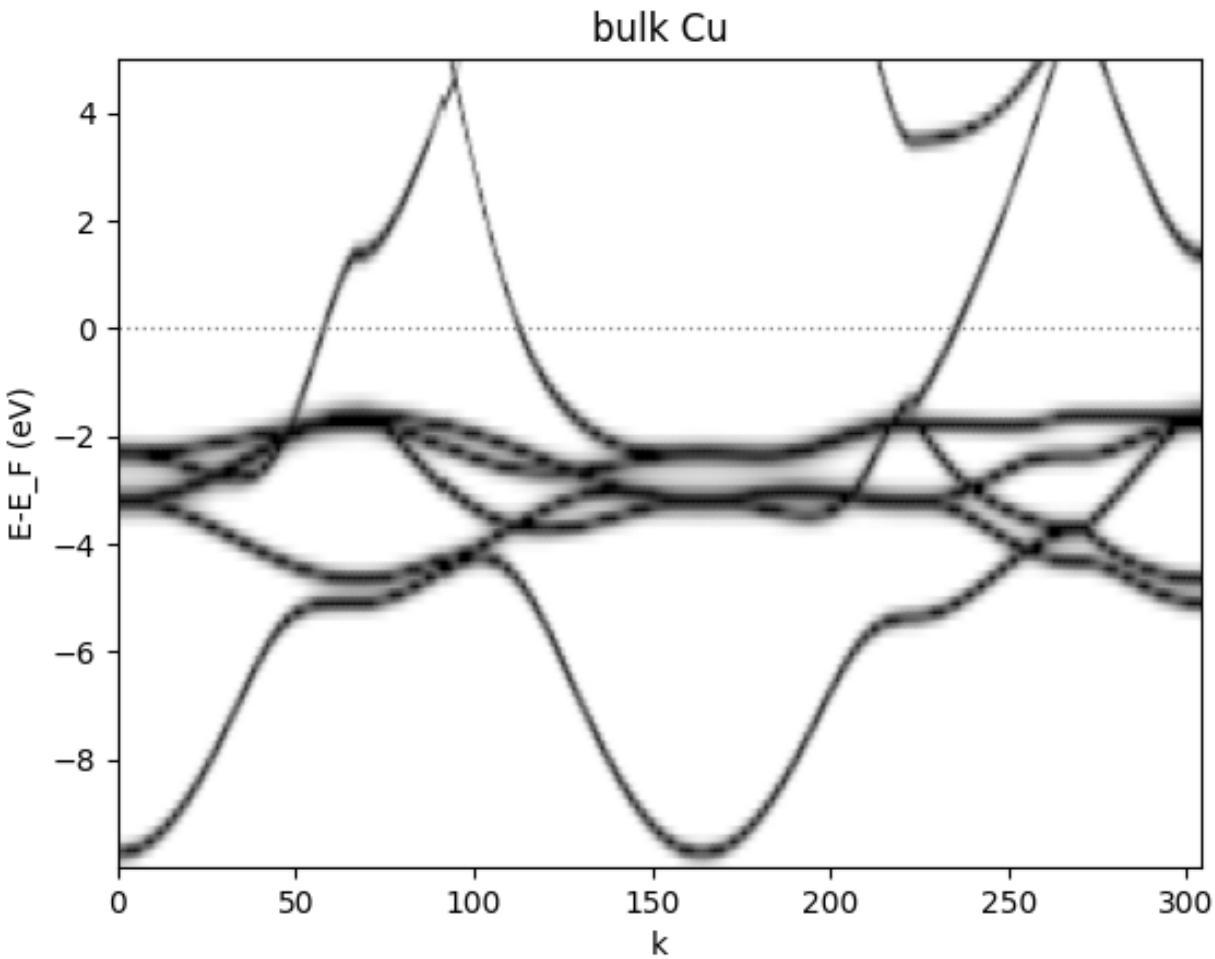


Fig. 19: Bandstructure of bulk fcc Cu.

Fermi surface

Constant energy contours can be calculated by using a single energy point in a qdos calculation with a 2D k-point grid defined in the `qvec.dat` input file to KKRhost. For example, this can then be used to visualize a cut through the Fermi surface of a material.

```
#Example: KKR bandstructure

from masci_tools.vis.kkr_plot_FS_qdos import FSqdos2D

# the path can be a relative or absolute path to the directory
# where the qdos files reside (i.e. the dir where the qdos calculation ran)
FSqdos2D('PATH/TO/OUTPUT-FILES/')
```

4.1.6 General Plotting routines

The plotting of data is always a common task that needs to be performed. However, there is a lot of variation in how someone might want plots to look or be arranged. Some plots might also need to be interactive to be of a real use.

For these reasons the `masci_tools` library provides utility for general plotting and template functions for common plots made when working with DFT methods. There are two plotting backends available:

- matplotlib: Mainly used for non-interactive plots
- bokeh: Mainly used for interactive plots

4.1.6.1 Available Routines

For both of these there are a lot of plotting routines available (both general or specific to a problem). All of these routines will return the used `Axes` object in the case of matplotlib or the `figure` produced by bokeh for custom modifications.

`common` (Can be used for both backends):

- *`scatter()`*: Make a scatterplot with varying size and color of the points for multiple sets of data
- *`line()`*: Make a lineplot with multiple sets of data
- *`dos()`*: Plot a general density of states (non-spinpolarized)
- *`spinpol_dos()`*: Plot a general density of states (spinpolarized)
- *`bands()`*: Plot a general bandstructure (non-spinpolarized)
- *`spinpol_bands()`*: Plot a general bandstructure (spinpolarized)

`matplotlib`:

- *`single_scatterplot()`*: Make a scatterplot with lines for a single set of data
- *`multiple_scatterplots()`*: Make a scatterplot with lines for multiple sets of data
- *`multi_scatter_plot()`*: Make a scatterplot with varying size and color of the points for multiple sets of data
- *`colormesh_plot()`*: Make 2D plot with the data represented as color
- *`waterfall_plot()`*: Make 3D plot with the `scatter3D` function of matplotlib
- *`surface_plot()`*: Make 3D plot with the `plot_surface` function of matplotlib
- *`multiplot_moved()`*: Plot multiple sets of data above each other with a configurable shift
- *`histogram()`*: Make a histogram plot
- *`barchart()`*: Make a barchart plot
- *`multiaxis_scatterplot()`*: Make a plot containing multiple sets of data distributed over multiple subplots in a grid
- *`plot_convex_hull2d()`*: Make a 2D plot of a convex hull
- *`plot_residuen()`*: Make a residual plot for given real and fit data. Can also produce a histogram of the deviations
- *`plot_convergence()`*: Plot the convergence behaviour of charge density distances and energies
- *`plot_lattice_constant()`*: Plot the energy curve with changing unit cell volume
- *`plot_dos()`*: Plot a general density of states (non-spinpolarized)
- *`plot_spinpol_dos()`*: Plot a general density of states (spinpolarized)

- *plot_bands()*: Plot a general bandstructure (non-spinpolarized)
- *plot_spinpol_bands()*: Plot a general bandstructure (spinpolarized)
- *plot_spectral_function()*: Plot a spectral function (colormesh along kpath)

bokeh:

- *bokeh_scatter()*: Make a scatterplot for a single set of data
- *bokeh_multi_scatter()*: Make a scatterplot for a multiple sets of data
- *bokeh_line()*: Make a line plot for multiple sets of data
- *bokeh_dos()*: Plot a general density of states (non-spinpolarized)
- *bokeh_spinpol_dos()*: Plot a general density of states (spinpolarized)
- *bokeh_bands()*: Plot a general bandstructure (non-spinpolarized)
- *bokeh_spinpol_bands()*: Plot a general bandstructure (spinpolarized)
- *bokeh_spectral_function()*: Plot a spectral function (colormesh along kpath)
- *periodic_table_plot()*: Make a interactive plot of data for the periodic table
- *plot_lattice_constant()*: Plot the energy curve with changing unit cell volume
- *plot_convergence()*: Plot the convergence behaviour of charge density distances and energies
- *matrix_plot()*: Plot a grid of rectangles with color scaling and added text

If you have ideas for new useful and beautiful plotting routines you are welcome to contribute. Refer to the sections *Using the Plotter class* and *Using the PlotData class* for a guide on how to get started.

4.1.6.2 Providing Data

Data can be provided to plotting functions in two main ways:

1. The first arguments and data arguments are given the keys in a mapping, which should be used. The corresponding mapping is provided via the `data` keyword argument
2. The first arguments and data arguments are given the data that should be plotted against each other.

The following two code blocks are equivalent in terms of the provided data.

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-10,10,100)
y1 = x**2
y2 = 20*np.sin(x)

#The data is split up according to fixed rules that the plot function defines.
#The default behaviour is that a list of lists is interpreted as multiple separate plots
ax = multiple_scatterplots(x, [y1, y2])
```

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-10,10,100)
y1 = x**2
```

(continues on next page)

(continued from previous page)

```
y2 = 20*np.sin(x)
data = {'x': x, 'y1': y1, 'y2': y2}

ax = multiple_scatterplots('x', ['y1', 'y2'], data=data)
```

4.1.6.3 Customizing Plots

You might want to change the parameters of your plot. From changing the color, linestyle or shape of the markers there are a million options to tweak. These can be set by simply passing the keyword arguments with the desired parameters to the plotting function. The names of these parameters mostly correspond to the same names as in the plotting library that is used in the plotting function. However, there are some deviations and also some special keywords that you can use. We will go over the most important ones in this section accompanied with concrete code examples. For a reference of the defaults defined in the `masci_tools` library you can refer to *matplotlib_plotter.MatplotlibPlotter* and *bokeh_plotter.BokehPlotter* for a complete reference.

The most important special keywords are listed below. If there are deviating names for these in `matplotlib` and `bokeh` plotting functions both names are written in the order `matplotlib` or `bokeh`:

- `limits`: This is used to set the bounds of the axis specifically. Provided in form of a dictionary. For example passing `limits={'x': (-5,5)}` will set the x-axis limits between `-5` and `5` and `limits={'x': (-5,5), 'y':(0,10)}` will set the y-axis limits in addition
- `scale`: Used to set the scaling of the axis in the plots. Also provided in form of a dictionary. For example passing `scale={'x': 'log', 'y': 'log'}` will set both axis to logarithmic scaling `scale={'y': 'log'}` will only to it for the y-axis
- `lines` or `straight_lines`: Easy way to draw help lines into the plot. Provided in form of a dictionary. For example passing `lines={'vertical': 0}` will draw a vertical line at `x=0` `lines={'horizontal': [1, 5,10]}` will draw three horizontal lines at `y=1, 5 or 10` respectively
- `plot_labels` or `legend_labels`: Defines labels for the legend of a plot
- `labels for axis`: Normally called `xlabel` or `ylabel`, but specialized plot routines might have different names
- `title`: Title for the produced plot
- `saving options`: `show=True` call the plotting library specific show routines (default). For matplotlib you can also specify `saveas='filename'` and `save_plots=True` to save the plot to file

In the following we will look at examples using the matplotlib plotting functions in *plot_methods*. The options are the same for the bokeh plotting routines in *bokeh_plots*.

Single plots

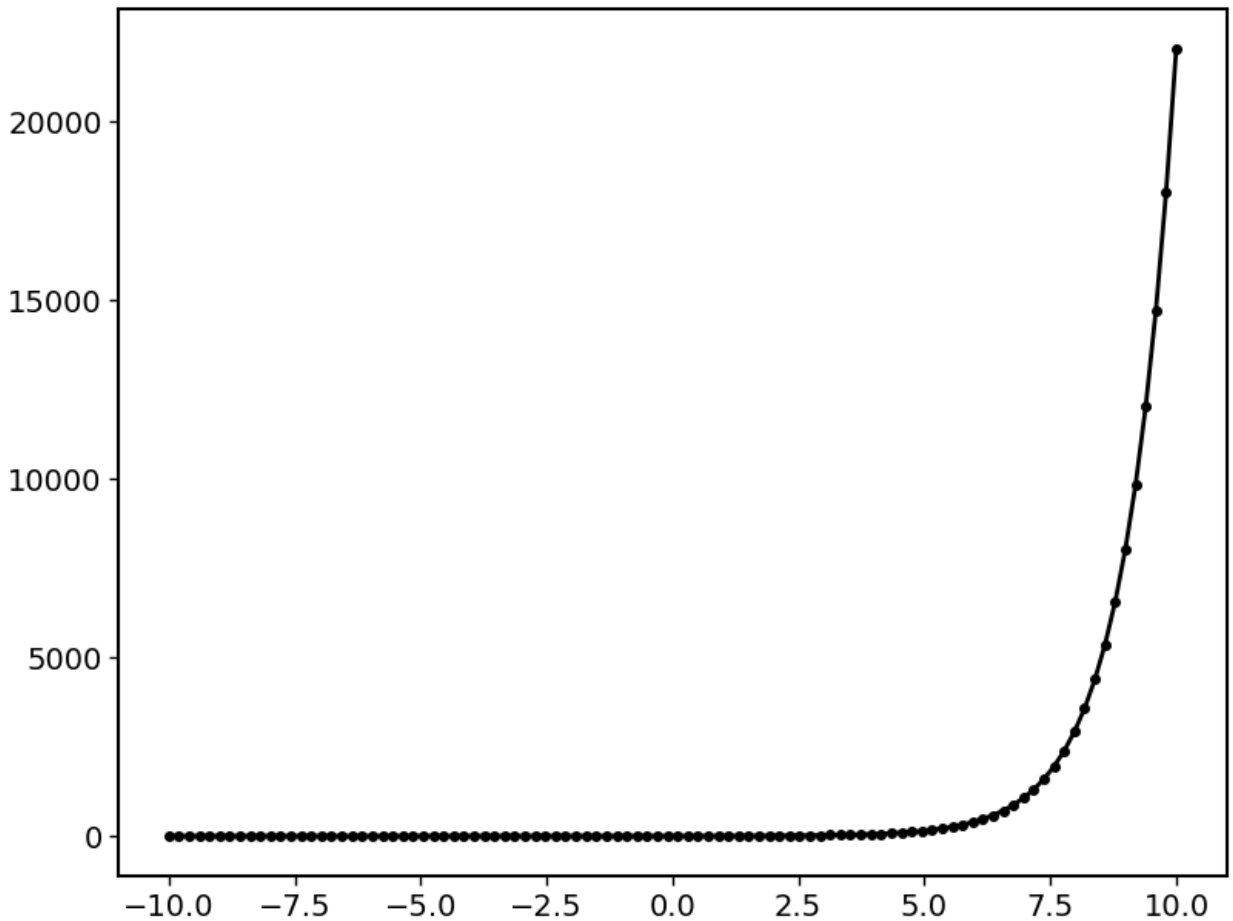
We start from the default result of calling the *plot_methods.single_scatterplot()* function with an exponential function. Afterwards we go through examples of modifying this call in one particular way. All of these can be combined to customize the plot to your desire

```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)
```

(continues on next page)

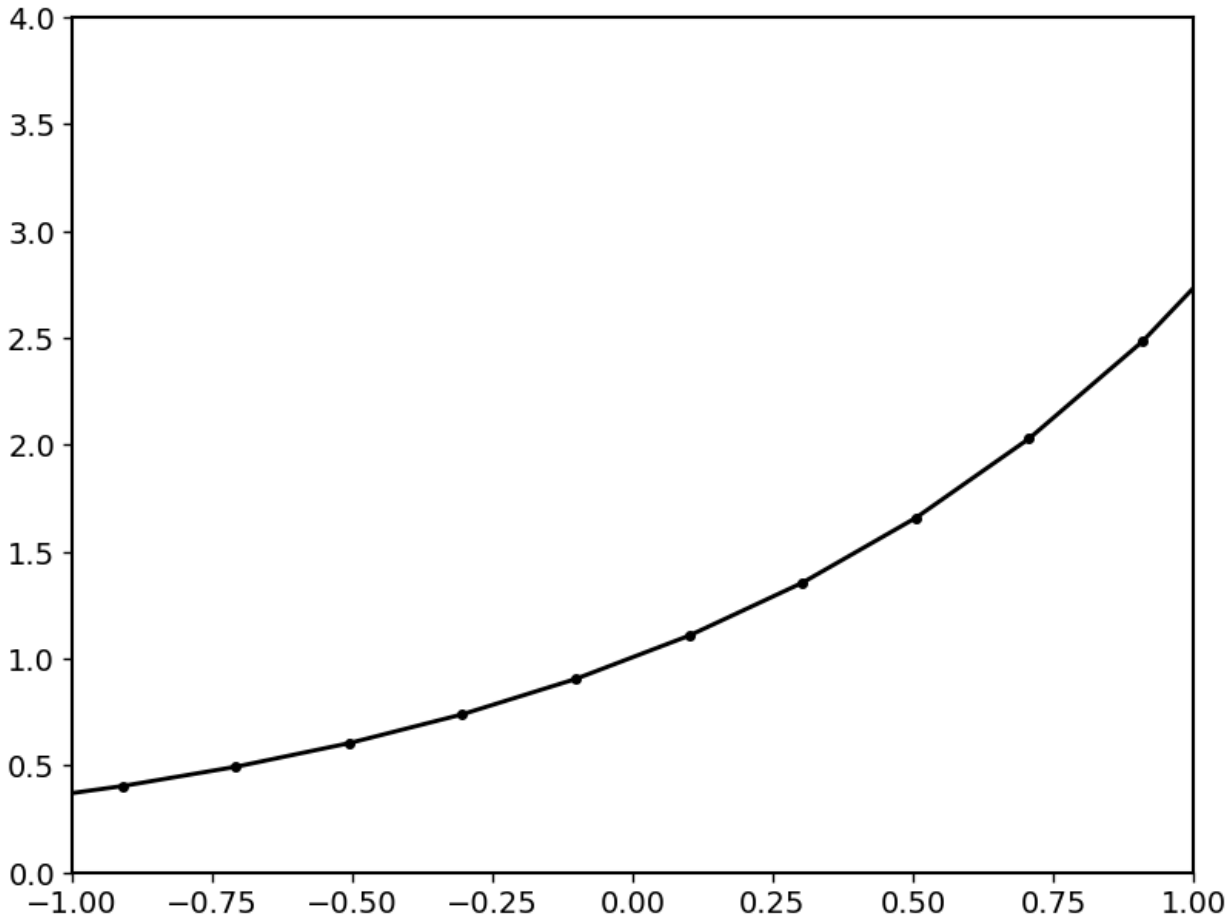
(continued from previous page)

```
ax = single_scatterplot(x,y)
```



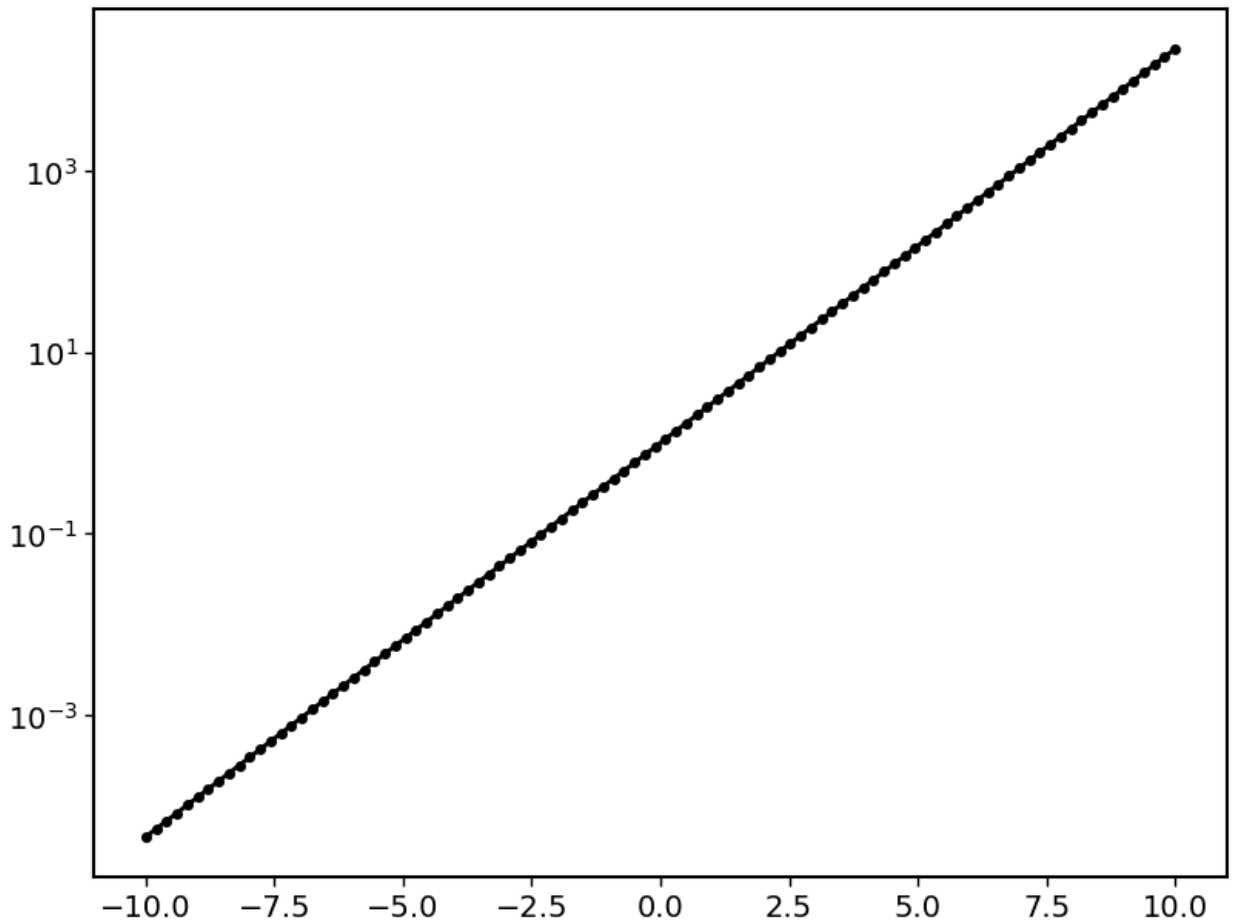
Setting limits

```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, limits={'x': (-1,1), 'y': (0,4)})
```



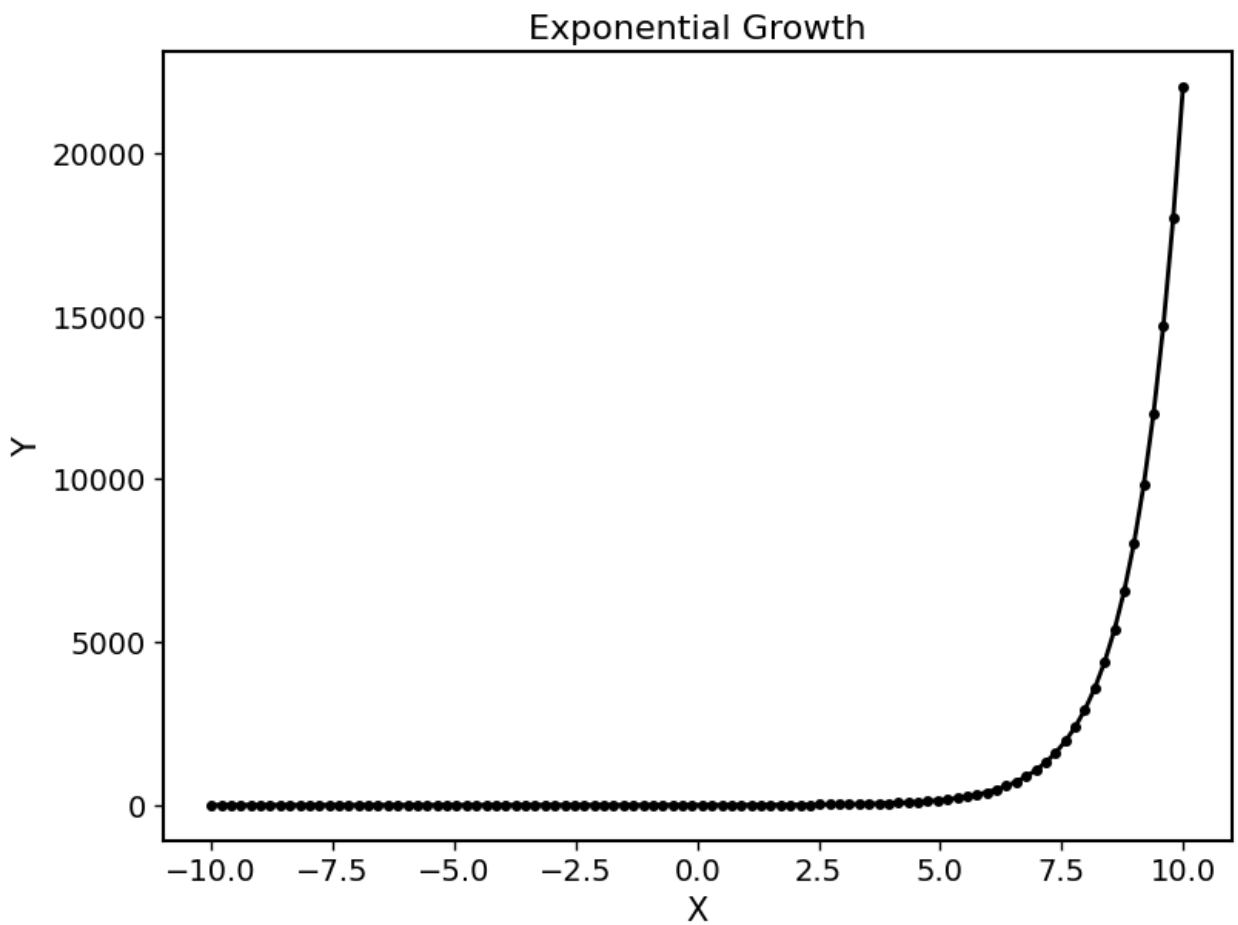
Modifying the scale of the axis

```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, scale={'y': 'log'})
```



Setting labels on the axis and a title

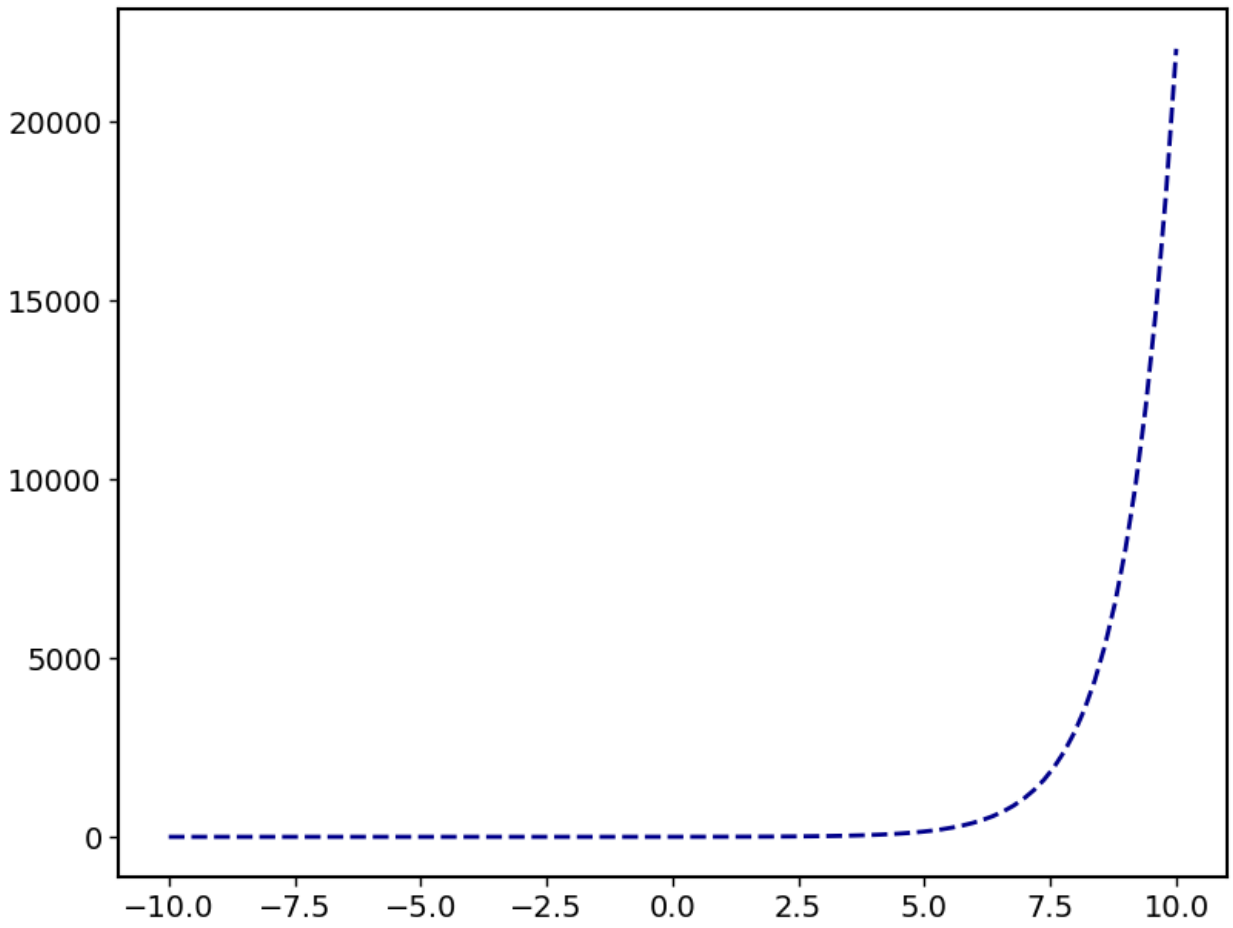
```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, xlabel='X', ylabel='Y', title='Exponential Growth')
```



Modifying plot parameters

See the matplotlib documentation for complete references of possible options

```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, color='darkblue', linestyle='--', marker=None)
```



Setting user defaults

If you wish to change some parameters for all the plots you want to do, you can use the functions *plot_methods.set_mpl_plot_defaults()* or *bokeh_plots.set_bokeh_plot_defaults()* for the matplotlib and bokeh plotting library respectively. These functions accept the same keyword arguments as above and they will be applied to all the next plots that you do.

You can reset the changes to the defaults with *plot_methods.reset_mpl_plot_defaults()* or *bokeh_plots.reset_bokeh_plot_defaults()*

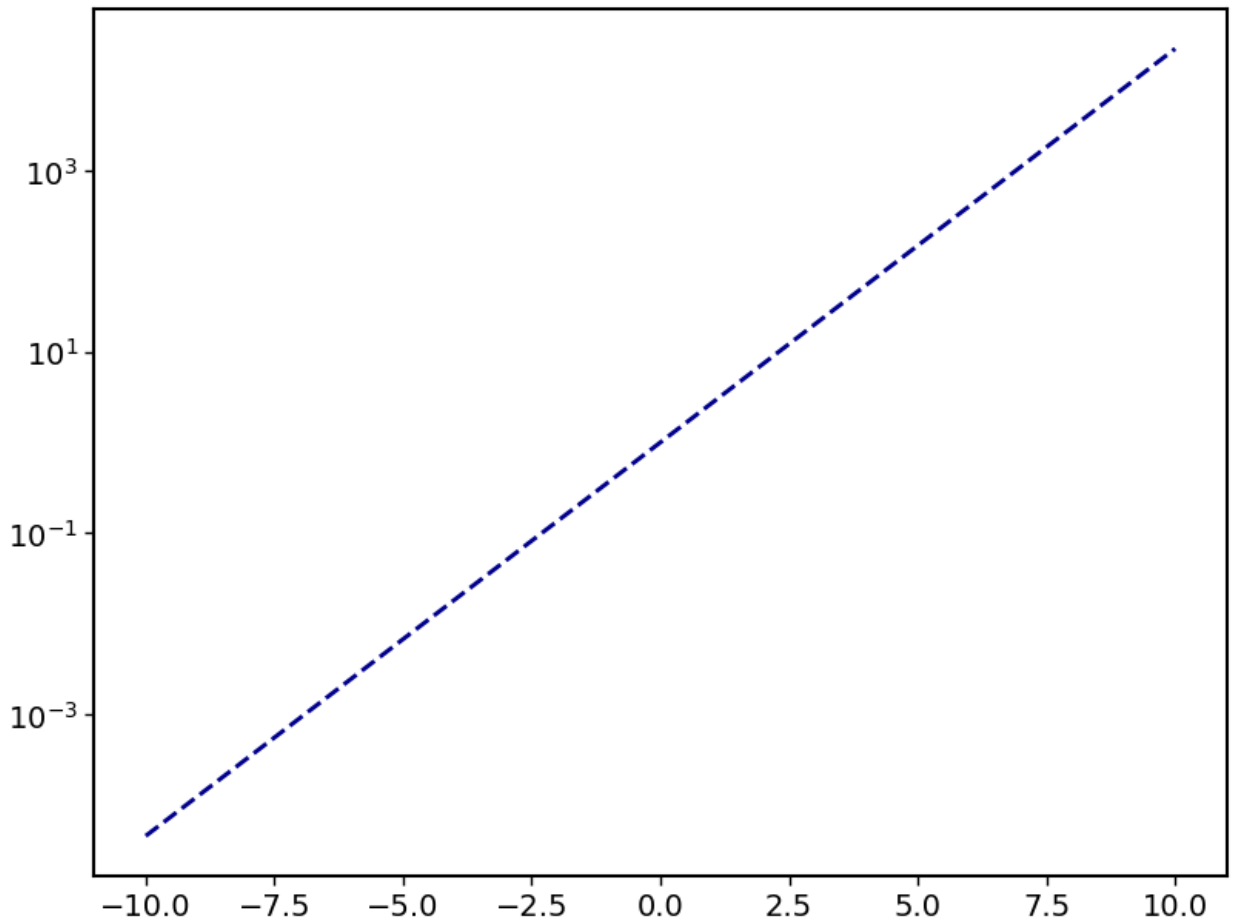
Note: You can still override these defaults by simply passing in another value for the parameter you wish to overwrite in the call to a plotting function

```
from masci_tools.vis.plot_methods import single_scatterplot, set_mpl_plot_defaults
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

set_mpl_plot_defaults(color='darkblue', linestyle='--', marker=None)
```

(continues on next page)

(continued from previous page)

```
ax = single_scatterplot(x,y, scale={'y': 'log'})
```



Resetting defaults:

```
from masci_tools.vis.plot_methods import reset_mpl_plot_defaults
reset_mpl_plot_defaults()
```

Multiple plots

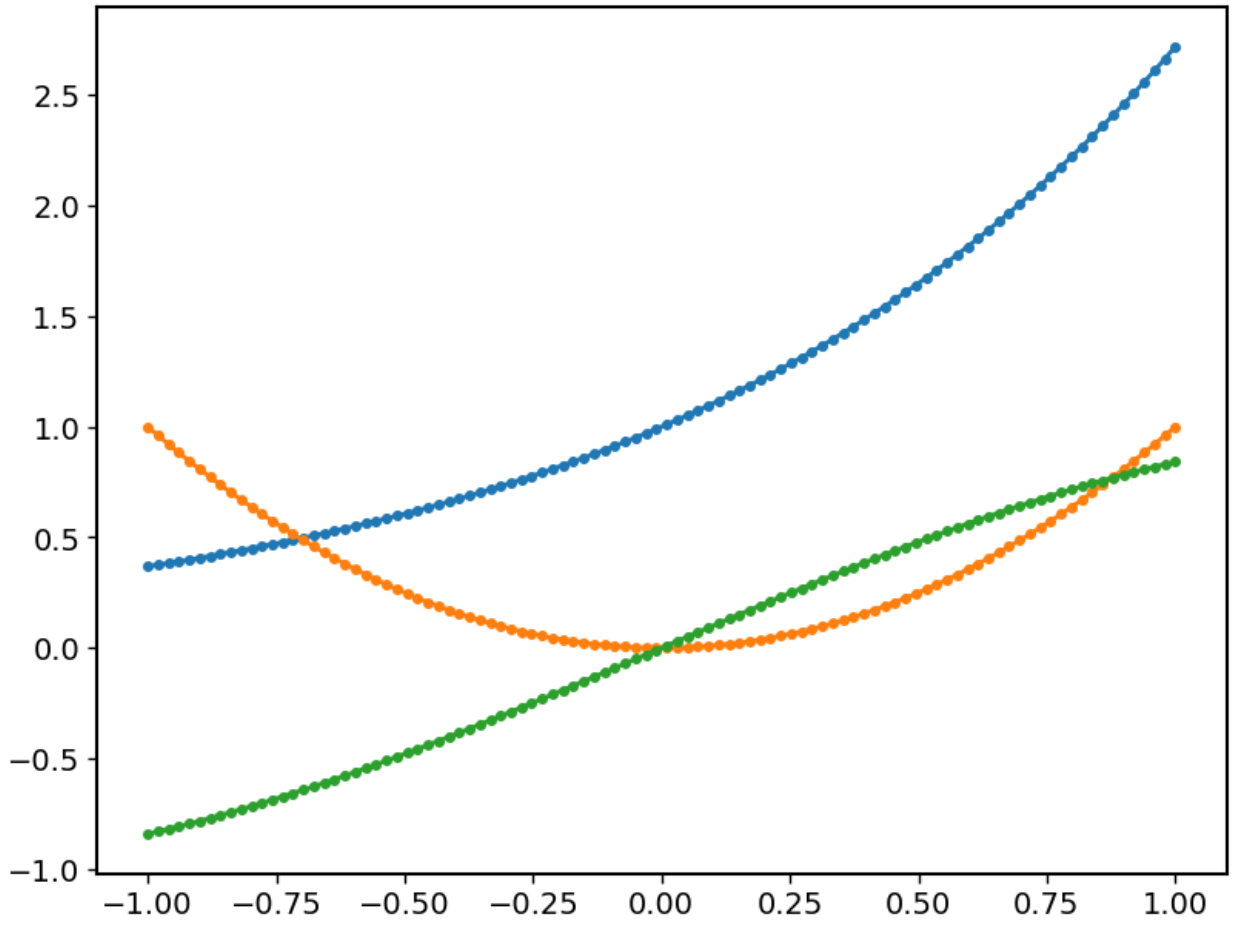
Many plotting routines accept multiple sets of data to plot. An example of this is the *plot_methods.multiple_scatterplots()* function. The usage of these is essentially the same. However, some parameters can be changed for each data set to plot. These include but are not limited to `linestyle`, `linewidth`, `marker`, `markersize` and `color`. These parameters can either be set to a single value applying it to all data sets, or can be specified for some/all data sets with unspecified values being replaced with the current defaults. This second way can be done in two ways (Both of the below examples have the same effect):

1. List of values (`None` for unspecified values) Example: `linestyle=['-', None, '--']`
2. Dictionary with integer indices Example: `linestyle={0:'-', 2:'--'}`

> **Warning:** Specifying parameters for multiple data sets is only valid for the parameters passed into the function. Setting defaults with values for multiple data sets is not supported

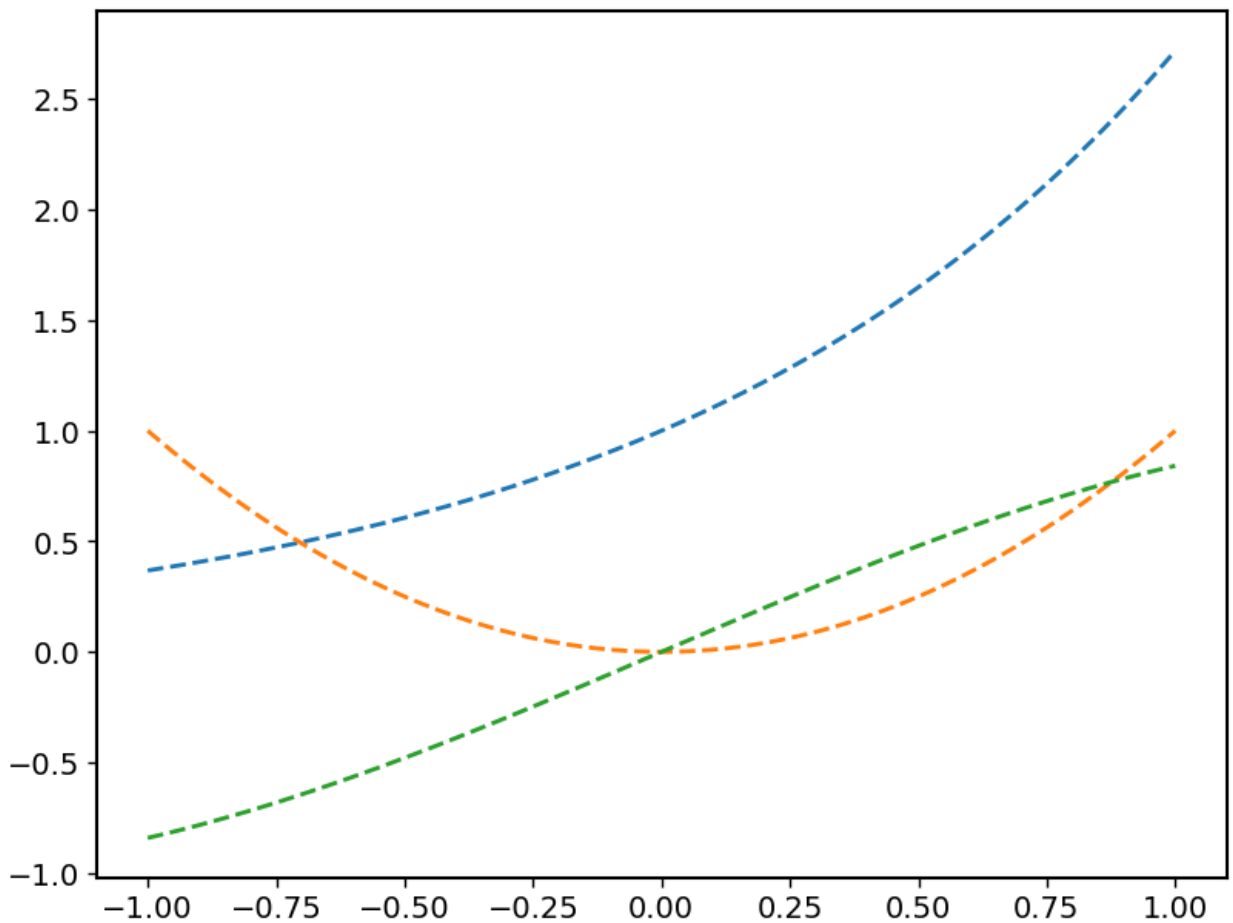
Default plot

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-1,1,100)
y = np.exp(x)
y2 = x**2
y3 = np.sin(x)

ax = multiple_scatterplots([x, x, x], [y, y2, y3])
```



Changing parameters on all plots

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-1,1,100)
y = np.exp(x)
y2 = x**2
y3 = np.sin(x)

ax = multiple_scatterplots([x, x, x], [y, y2, y3], linestyle='--', marker=None)
```



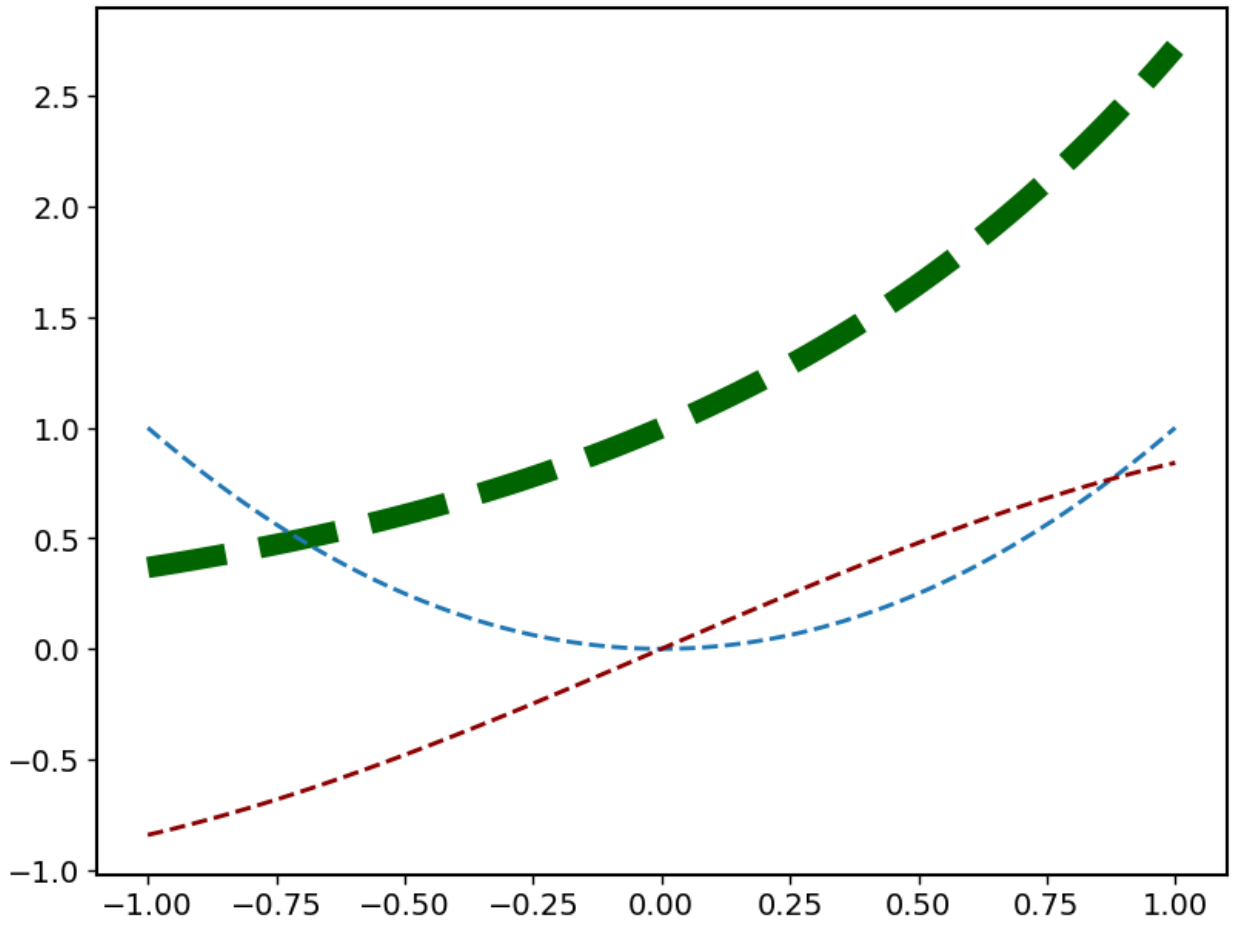
Changing parameters on specific plots

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-1,1,100)
y = np.exp(x)
y2 = x**2
y3 = np.sin(x)
```

(continues on next page)

(continued from previous page)

```
ax = multiple_scatterplots([x, x, x], [y, y2, y3],
                           linestyle='--',
                           marker=None,
                           color=['darkgreen', None, 'darkred'],
                           linewidth={0: 10})
```



CHAPTER

FIVE

DEVELOPER'S GUIDE

5.1 Developers Guide

This is the developers guide for masci-tools

5.1.1 Test suite of masci-tools

5.1.1.1 Installation

Install the package with the `testing` extra:

```
pip install -e .[testing]
```

5.1.1.2 Running the tests

Run `pytest` in the `tests` path of masci-tools:

```
cd tests
pytest .
```

5.1.1.3 Regenerating test files

With some code changes an update of the reference files for the tests is required. This can be done with the `--force-regen` option:

```
pytest --force-regen .
```

5.1.2 Updating or adapting the Fleur Parsers

Note: A more detailed walkthrough of the important design decisions concerning the implementation Fleur XML parsers and IO capabilities can be found in this document

Each input and output file for Fleur has a correspong XML-Schema, where the structure of these files are defined.

To be able to parse such files efficiently and without hardcoding their structure we extract all necessary information about the schemas in the modules under *`fleur_schema`*. The resulting python dictionaries can be accessed via the

classes *InputSchemaDict* and *OutputSchemaDict*. The easiest way to instantiate one of these objects is to use the *InputSchemaDict.fromVersion()* or *OutputSchemaDict.fromVersion()* methods by providing the desired version string.

5.1.2.1 Adding/modifying a Fleur Schema:

The command `masci-tools fleur-schema add <path-to-schema-file>` can be used to add the schema to the repository. If the schema for the specified version already exists an error is raised. To ignore this error the option `--overwrite` can be used.

5.1.2.2 Adapting the outxml_parser:

In contrast to the input file parser *inpxml_parser()*, which parses all information available, the *outxml_parser()* has to be more flexible. The out file has much more information which might not be always useful for users. Therefore the selection of what is parsed has to be much more specific.

This selection is expressed in the context of tasks. In general this corresponds to things like: - Total energy - Charge density distances - Magnetic moments - and so on ...

These are expressed in a definition in form of a dictionary. Below a simple example (Total energy) is shown, which parses the `value` and `units` attribute of the `totalEnergy` tag. The hardcoded known parsing tasks can be found in *default_parse_tasks*.

```
total_energy_definition = {
    'energy_hartree': {
        'parse_type': 'singleValue',
        'path_spec': {
            'name': 'totalEnergy'
        }
    },
}
```

The definition of a task can consist of multiple keys (in this case only `energy_hartree`), which by default correspond to the keys in the resulting output dictionary. Each key has to contain the `parse_type` and `path_spec` key. The `parse_type` defines the method used to extract the information.

The following are possible:

attrib
Will parse the value of the given attribute

text
Will parse the text of the given tag

numberNodes
Will return the number of nodes for the given tag

exists
Will return, whether the given tag exists

attrib_exists
Will return, whether the given attribute exists

allAttribs
Will parse all known attributes at the given tag into a dictionary

parentAttribs
Will parse all known attributes at the given tag into a dictionary, but for the parent of the tag

singleValue
Special case of allAttribs to parse value and units attribute for the given tag

xmlGetter
Will execute a function in the module *xml_getters* given in the `name` entry

The `path_spec` key specifies how the key can be uniquely identified. It can contain the following specifications:

name
Name of the wanted tag/attribute

contains
A phrase, which has to occur in the path

not_contains
A phrase, which has to not occur in the path

exclude
list of str. Only valid for attributes (these are sorted into different categories `unique`, `unique_path` and `other`). This attribute can exclude one or more of these categories

All except the `name` key are optional and should be constructed so that there is only one possible choice. Otherwise an exception is raised. There are other keywords, which can be entered here. These control how the parsed data is entered into the output dictionary. For a definition of these keywords, please refer to *default_parse_tasks*.

Each task can also contain a number of control keys, determining when to perform the tasks. Each of these keys begins with an underscore. All of these are optional. The following are valid:

_general
bool, if True (default False) the task is not performed for each iteration but once on the root of the file

_minimal
bool, if True the task is performed even when `minimal_mode = True` is given

_modes
list of tuples specifying requirements on the `fleur_modes` for the task. For example `[('jspins', 2), ('soc', True)]` will only perform the task for a magnetic SOC calculation

_conversions
list of str, giving the names of functions to call after this task. Functions given here have to be decorated with the *conversion_function()* decorator

_special
bool, if True (default False) this task is *NEVER* added automatically and has to be added by hand

5.1.2.3 Migrating the parsing tasks

These task definitions might have to be adapted for new fleur versions. Some changes might be possible to make in *default_parse_tasks* directly without breaking backwards compatibility. If this is not possible there is a decorator *register_migration()* to define a function that is recognized by the *outxml_parser()* to convert between versions. A usage example is shown below.

```
from masci_tools.io.parsers.fleur import register_migration
import copy

@register_migration(base_version='0.33', target_version='0.34')
def migrate_033_to034(definition_dict):
  """
```

(continues on next page)

(continued from previous page)

```
    Fictitious migration from 0.33 to 0.34
    Moves the `number_of_atom_types` attribute from reading a simple
    attribute to counting the number of atomGroups in the input section
    And removes orbital_magnetic_moments task
    """

    #IMPORTANT: First copy the original dict
    new_dict = copy.deepcopy(definition_dict)

    #If a task is incompatible remove it from the defintion_dict
    new_dict.pop('orbital_magnetic_moments')

    new_dict['general_out_info'].pop('number_of_atom_types')
    new_dict['general_inp_info']['number_of_atom_types'] = {
        'parse_type': 'numberNodes',
        'path_spec': {
            'name': 'atomGroup'
        }
    }

    return new_dict
```

5.1.3 Using the `Plotter` class

5.1.3.1 Description

The *`Plotter`* class aims to provide a framework, which can be used to handle default values and collect common codeblocks needed for different plotting frameworks. The *`Plotter`* class is a base class that should be subclassed for different Plotting backends. See *`MatplotlibPlotter`* or *`BokehPlotter`* for examples. The subclass provides a dictionary of all the keys that should be handled by the plotter class. The Plotter class provides a hierarchy of overwriting these parameters (Higher numbers take precedence).

1. Function defaults set with *`Plotter.set_defaults()`* with `default_type='function'`
2. Global defaults set with *`Plotter.set_defaults()`*
3. Parameters set with *`Plotter.set_parameters()`*

The subclasses should then also provide the plotting backend specific useful code snippets. For example showing colorbars, legends, and so on …

For a list of these functions you can look at the respective documentation (*`MatplotlibPlotter`* or *`BokehPlotter`*)

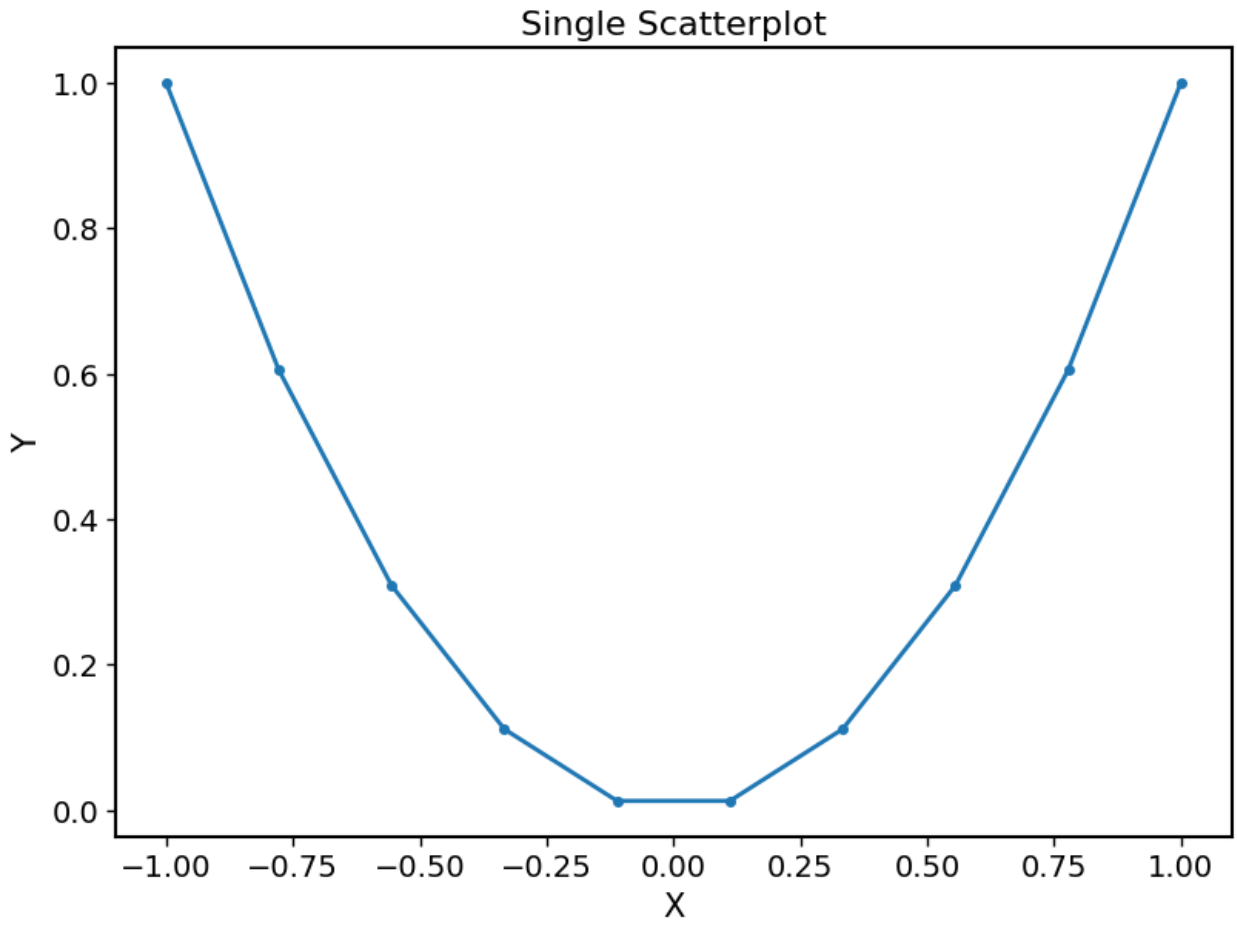
5.1.3.2 Writing a plotting function

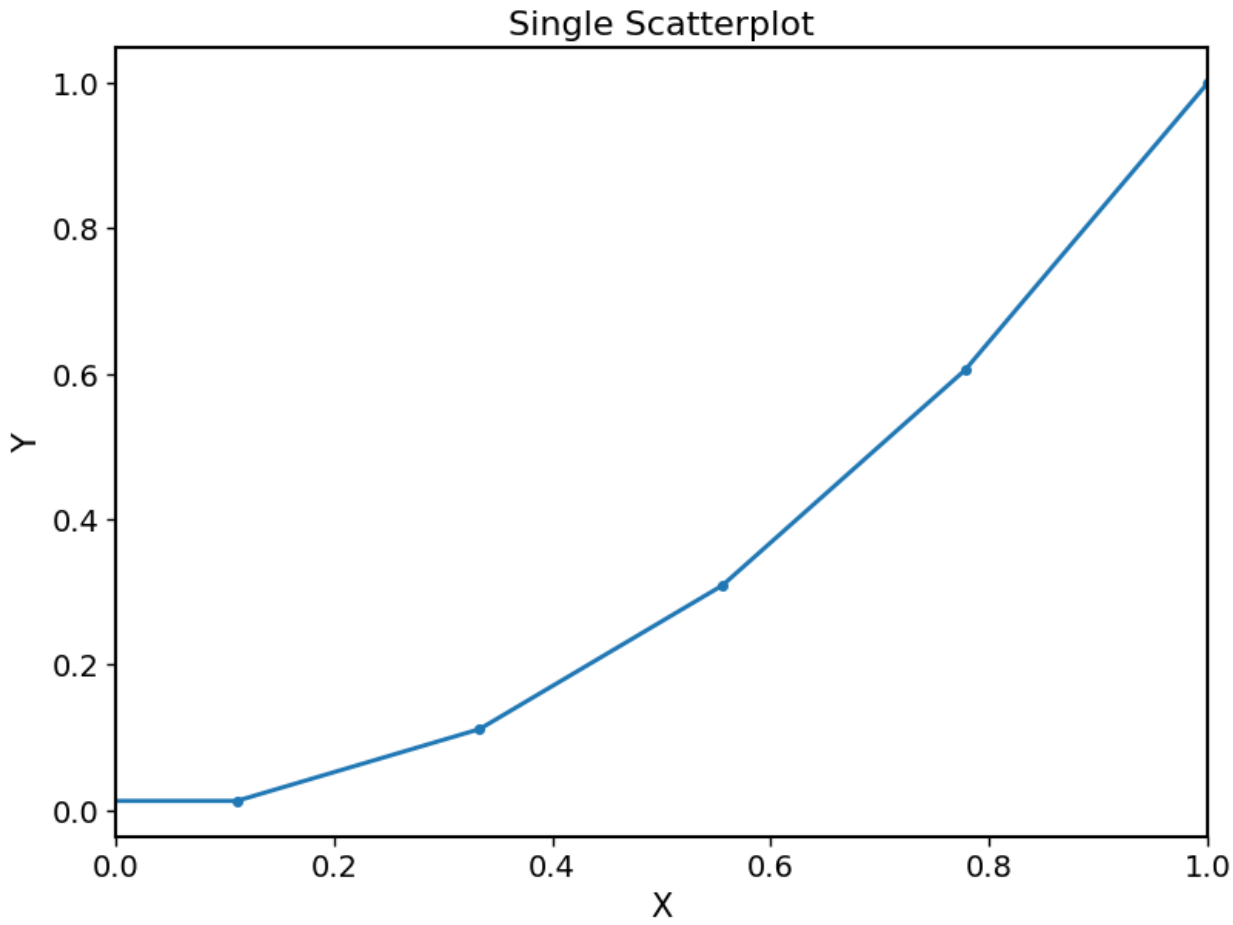
In the following we will go through a few examples of how to write a simple plotting function using the *`Plotter`* class. We will be focusing on the *`MatplotlibPlotter`*, but all of this is very similar for other plotting backends.

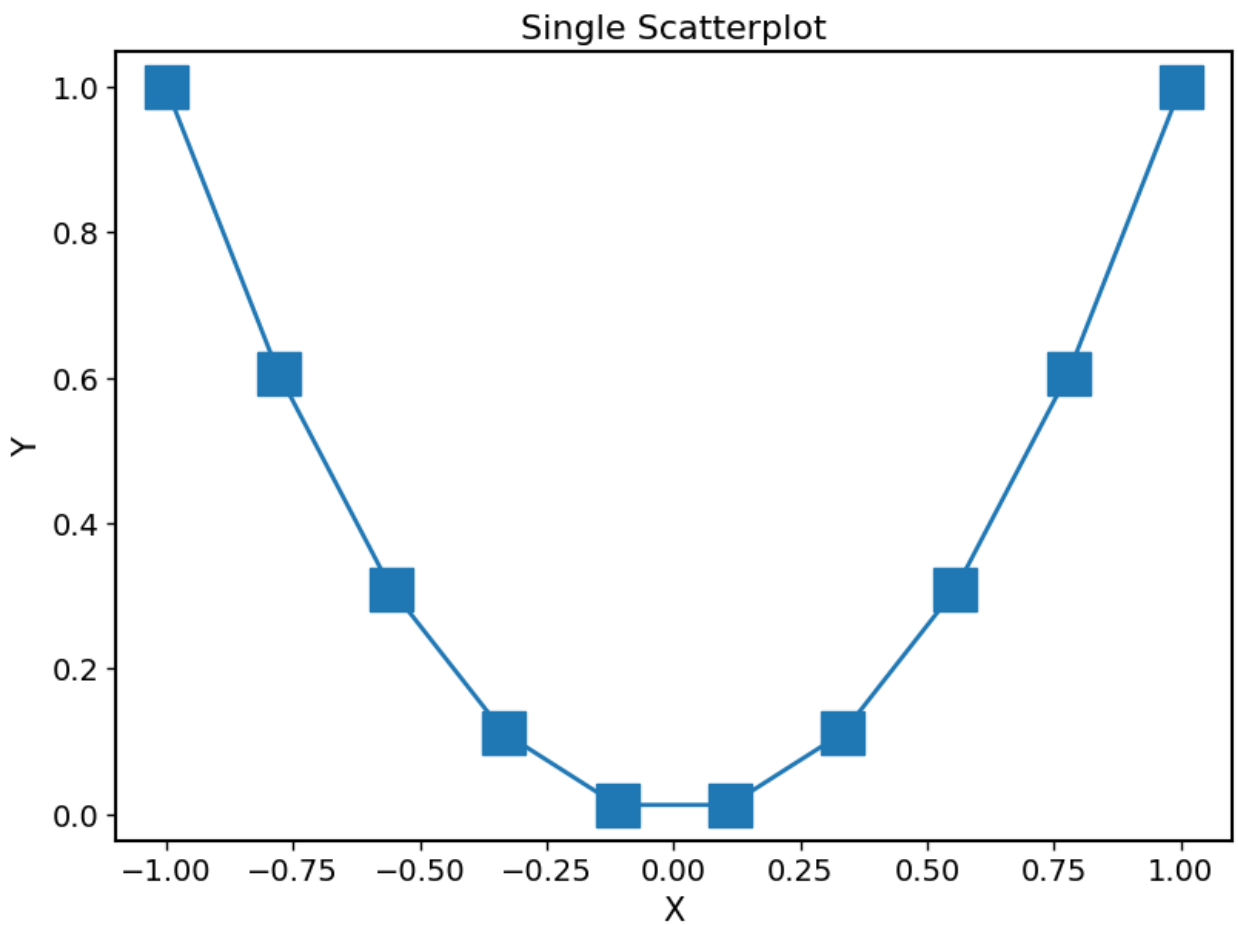
Local instance

Even though the *Plotter* class is meant to be used globally or on the module level, it can also be useful locally for simplifying simple plotting scripts. Here we have a example of a function producing a single plot with the given data for the x and y coordinates.

```
def plot_with_defaults(x,y,**kwargs):
   from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter

   #First we instantiate the MatplotlibPlotter class
   plot_params = MatplotlibPlotter()

   #Now we process the given arguments
   plot_params.set_parameters(**kwargs)

   #Set up the axis, on which to plot the data
   ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

   #The plot_kwargs provides a way to get the keyword arguments for the
   #actual plotting call to `plot` in this case.
   plot_kwargs = plot_params.plot_kwargs()

   ax.plot(x, y, **plot_kwargs)

   #The MatplotlibPlotter has a lot of small helper functions
   #In this case we just want to set the limits and scale of the
   #axis if they were given
   plot_params.set_scale(ax)
   plot_params.set_limits(ax)
   plot_params.save_plot('figure')

   return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#Some examples
plot_with_defaults(x, y)
plot_with_defaults(x, y, limits={'x': (0,1)})
plot_with_defaults(x, y, marker='s', markersize=20)
```


Single Scatterplot
1.0
0.8
0.6
0.4
0.2
0.0
Y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X


Single Scatterplot
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X




```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

Global/Module level instance

The local instance already gives us reusable code snippets to avoid common pitfalls when doing matplotlib/bokeh plots. But when instantiating the *Plotter* class locally we have no way of letting the user modify the global defaults.

However, when handling global state we need to be careful to not leave the instance of the *Plotter* class in an inconsistent state. If an error is thrown inside the plotting routine the parameters would stay set and may lead to very unexpected results. For this reason every plotting function using a global or module level instance of these plotters should be decorated with the *ensure_plotter_consistency()* decorator. This does two things:

1. If an error occurs in the decorated function the parameters will be reset before the error is raised
2. It makes sure that nothing inside the plotting routine changed the user defined defaults

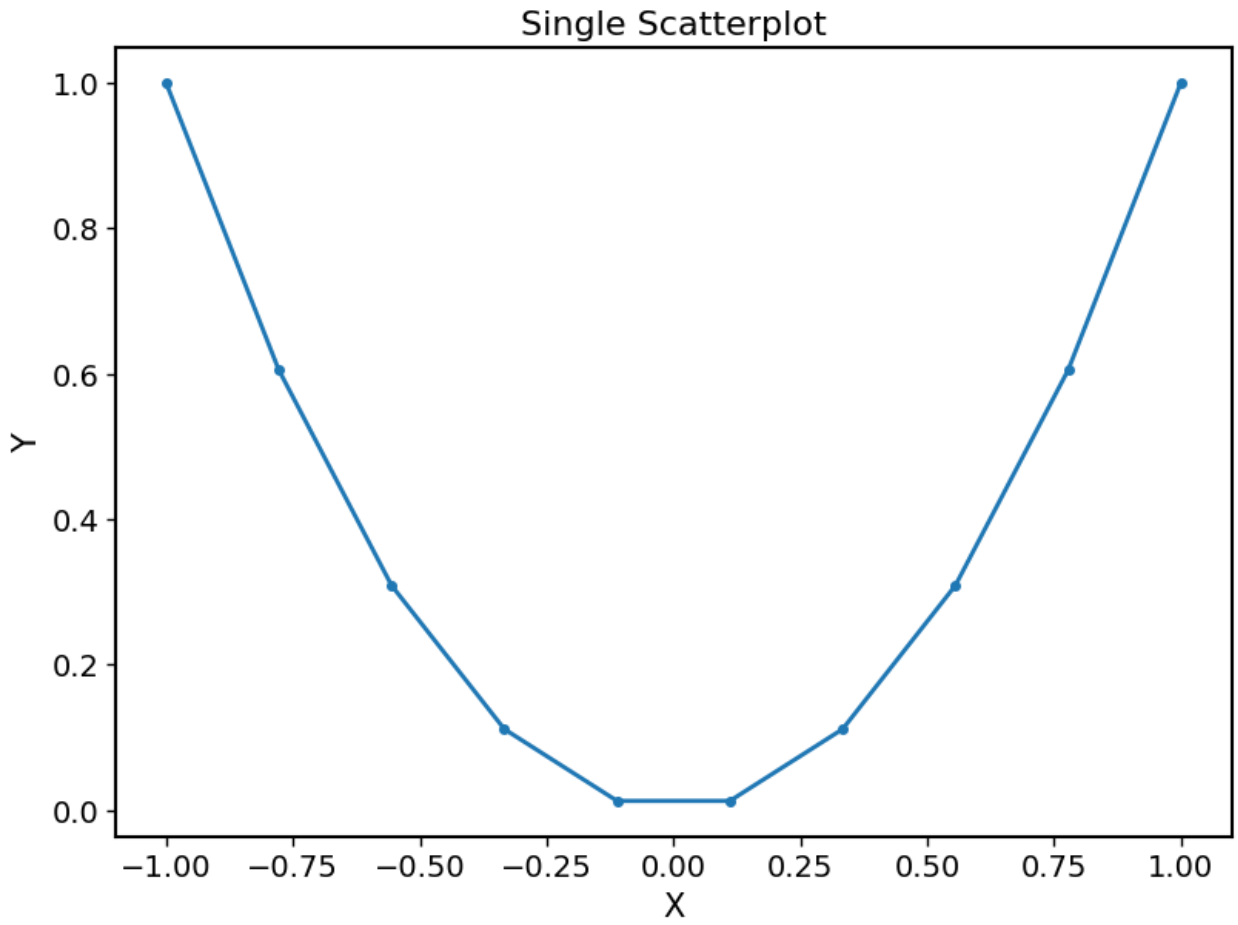
Let us take the previous example and convert it to use a global instance

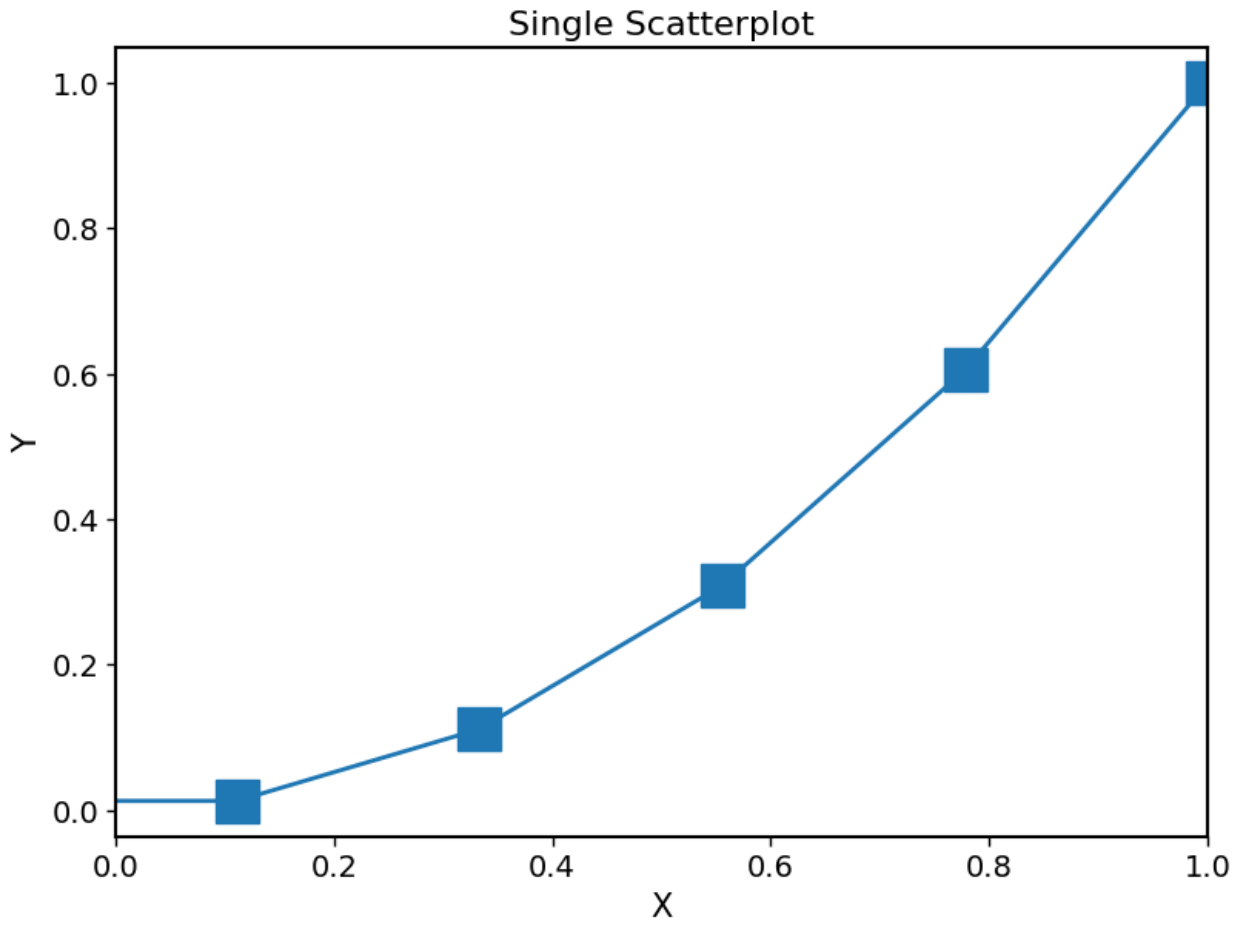
```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis.parameters import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()
```

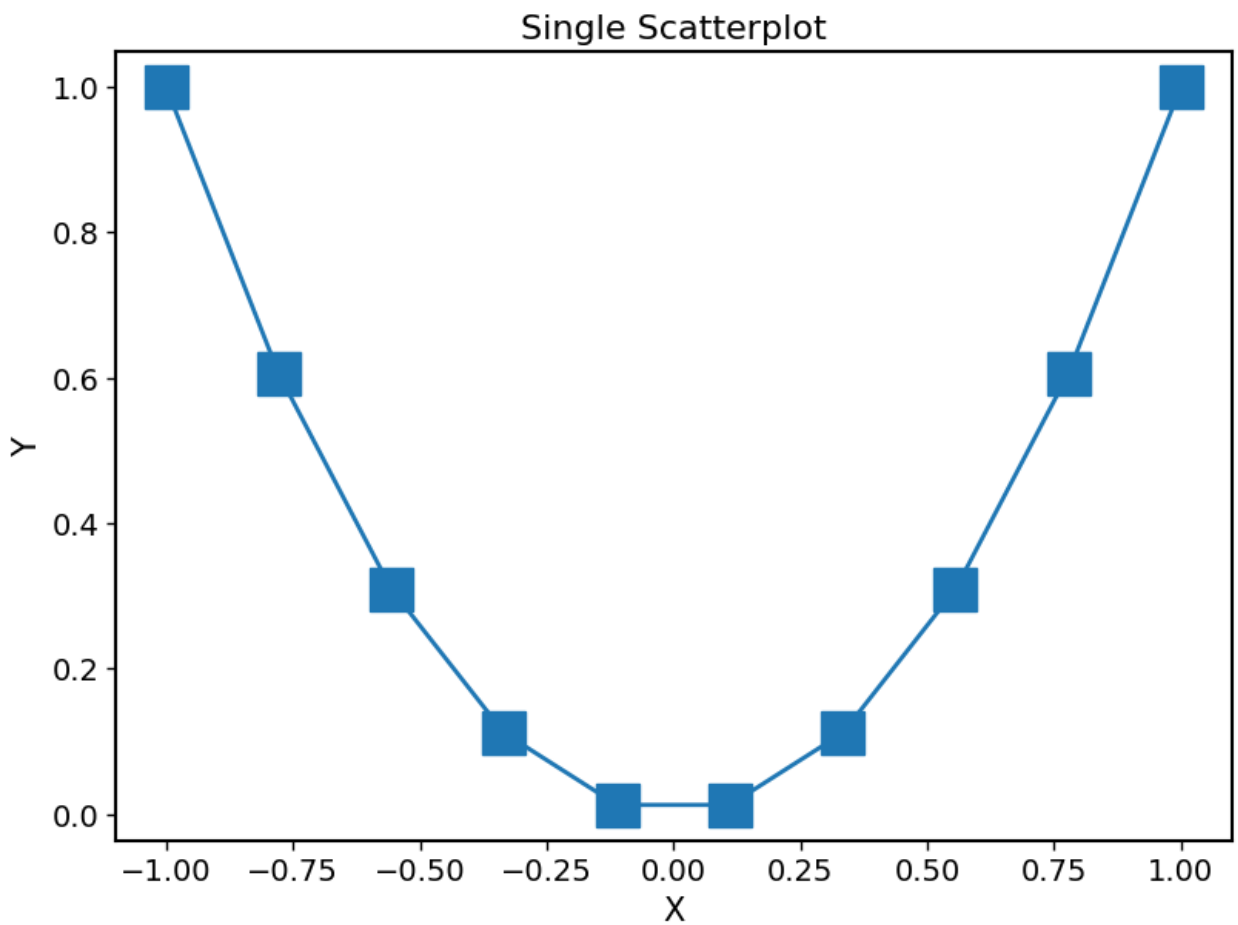
(continues on next page)

(continued from previous page)

```
#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

   #Now we process the given arguments
   plot_params.set_parameters(**kwargs)

   #Set up the axis, on which to plot the data
   ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

   #The plot_kwargs provides a way to get the keyword arguments for the
   #actual plotting call to `plot` in this case.
   plot_kwargs = plot_params.plot_kwargs()

   ax.plot(x, y, **plot_kwargs)

   #The MatplotlibPlotter has a lot of small helper functions
   #In this case we just want to set the limits and scale of the
   #axis if they were given
   plot_params.set_scale(ax)
   plot_params.set_limits(ax)
   plot_params.save_plot('figure')

   return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#Some examples
plot_with_defaults(x, y)
plot_params.set_defaults(marker='s', markersize=20)
plot_with_defaults(x, y, limits={'x': (0,1)})
plot_with_defaults(x, y)
```


Single Scatterplot
1.0
0.8
0.6
0.4
0.2
0.0
Y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X


Single Scatterplot
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X




```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

The *Plotter.set_defaults()* method is exposed in the two main modules for plotting *masci_tools.vis.plot_methods* *masci_tools.vis.bokeh_plots* as the functions *masci_tools.vis.plot_methods.set_mpl_plot_defaults()* and *masci_tools.vis.bokeh_plots.set_bokeh_plot_defaults()* specific to the plotter instance that is used in these modules.

Function defaults

Some functions may want to set function specific defaults, that make sense inside the function, but may not be useful globally. The following example sets the default `linewidth` for our function to 6.

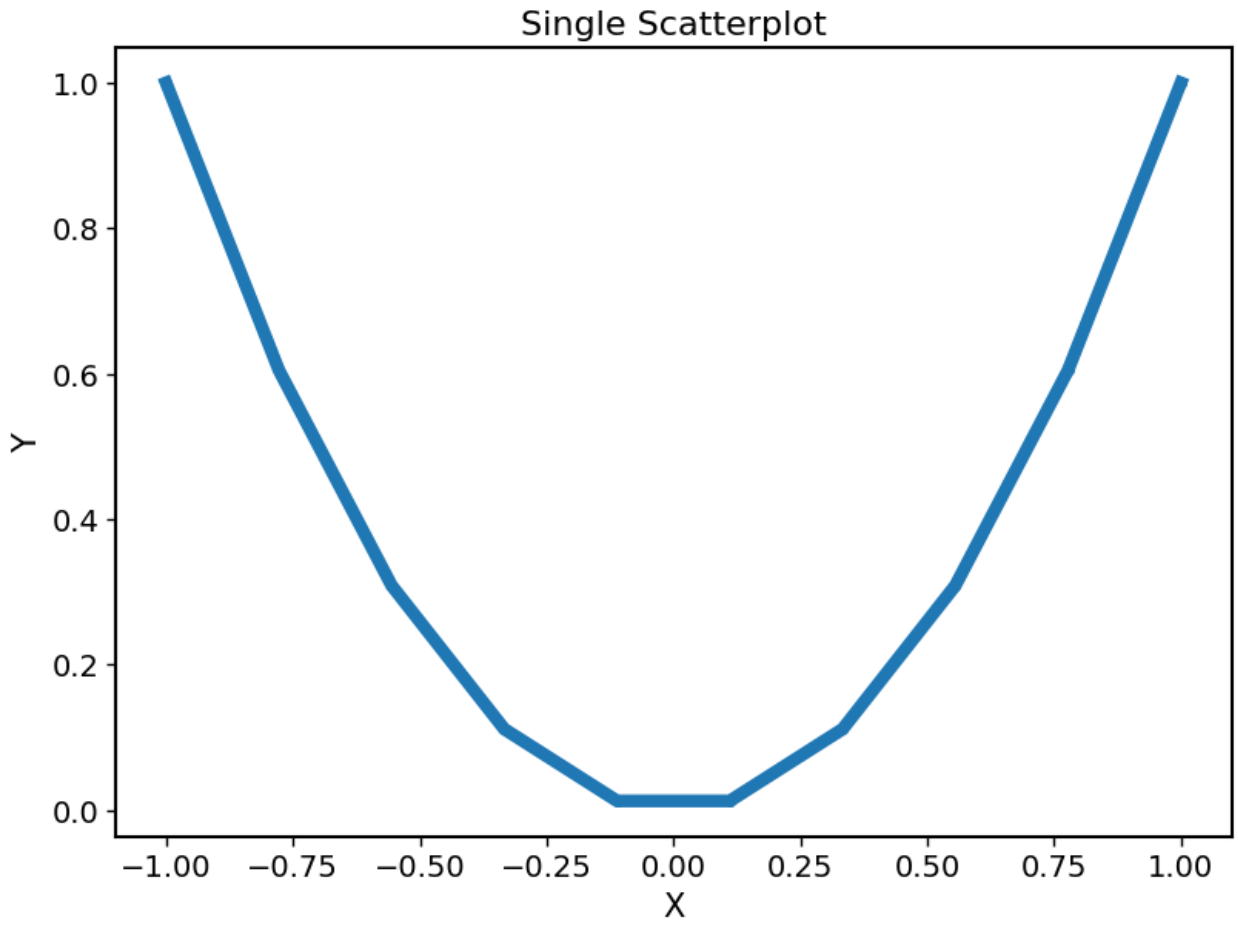
Note: Function defaults are also reset by the *ensure_plotter_consistency()* decorator, when the plotting function terminates successfully or in an error

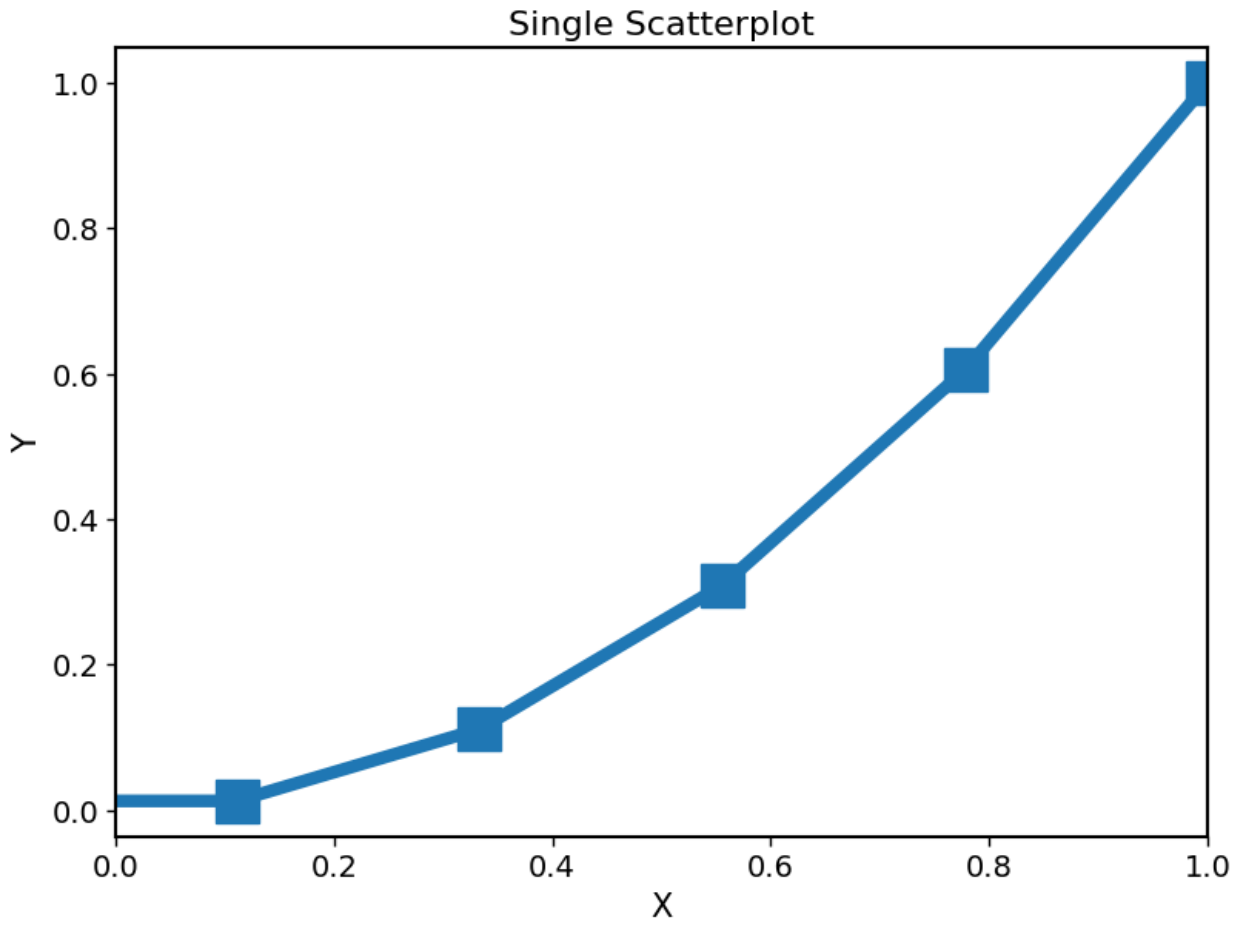
```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis.parameters import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()
```

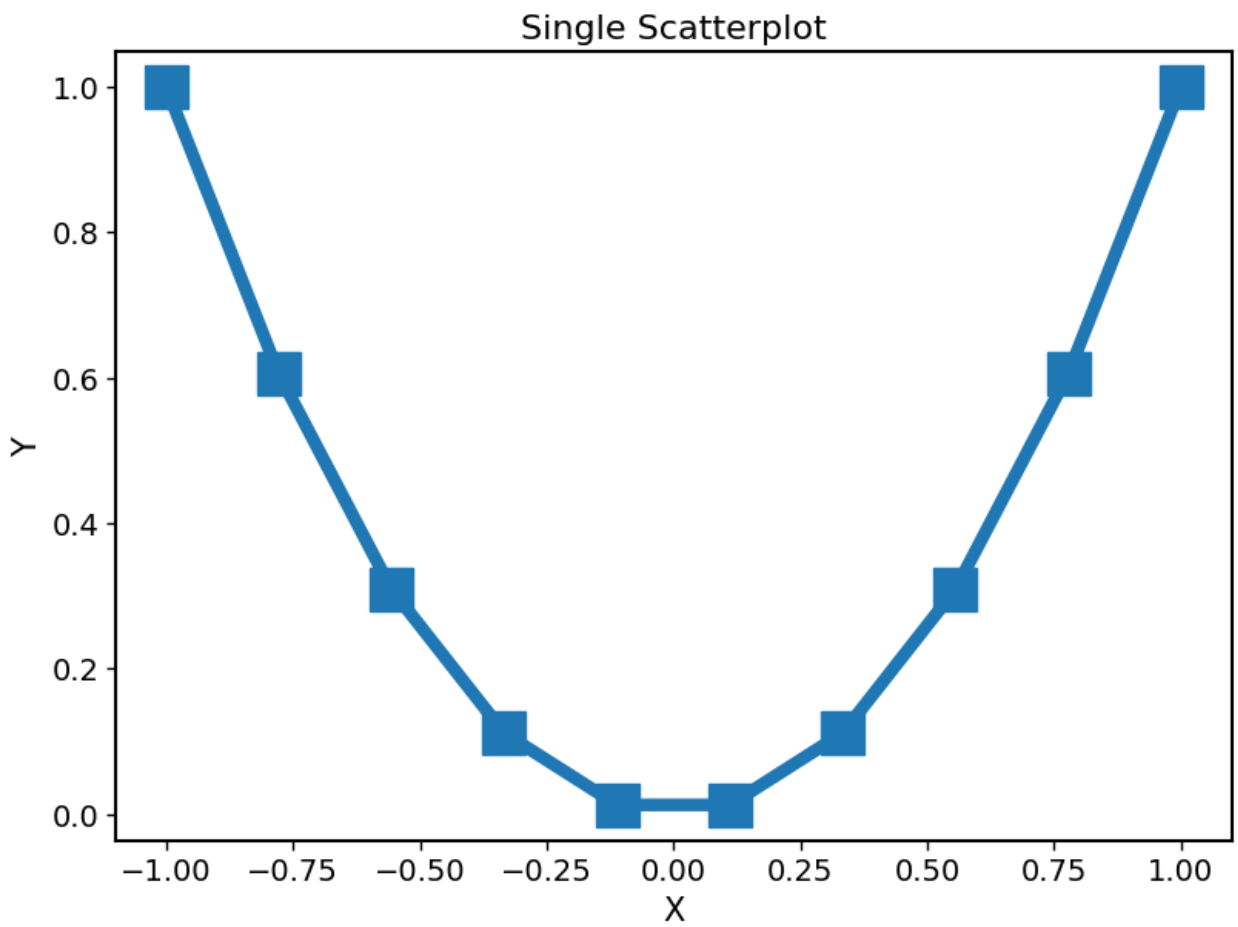
(continues on next page)

(continued from previous page)

```
#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)

    #Now we process the given arguments
    plot_params.set_parameters(**kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()

    ax.plot(x, y, **plot_kwargs)

    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)
    plot_params.save_plot('figure')

    return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#Some examples
plot_with_defaults(x, y)
plot_params.set_defaults(marker='s', markersize=20)
plot_with_defaults(x, y, limits={'x': (0,1)})
plot_with_defaults(x, y)
```


Single Scatterplot
1.0
0.8
0.6
0.4
0.2
0.0
Y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X


Single Scatterplot
1.0
0.8
0.6
0.4
0.2
0.0
Y
0.0
0.2
0.4
0.6
0.8
1.0
X




```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

Passing keyword arguments directly to plot calls

The plotter classes have a restricted set of keys that they recognize as valid parameters. This set is of course not complete, since there is a vast number of parameters you can set for all plotting backends. In our previous examples unknown keys will immediately lead to an error in the call to *Plotter.set_parameters()*. To enable this functionality we can provide the `continue_on_error=True` as an argument to this method.

Then the unknown keys are ignored and are returned in a dictionary. Additionally you can explicitly bypass the plotter object if you provide arguments in a dictionary with the name `extra_kwargs` it will be ignored, unpacked and returned along with the unknown keys

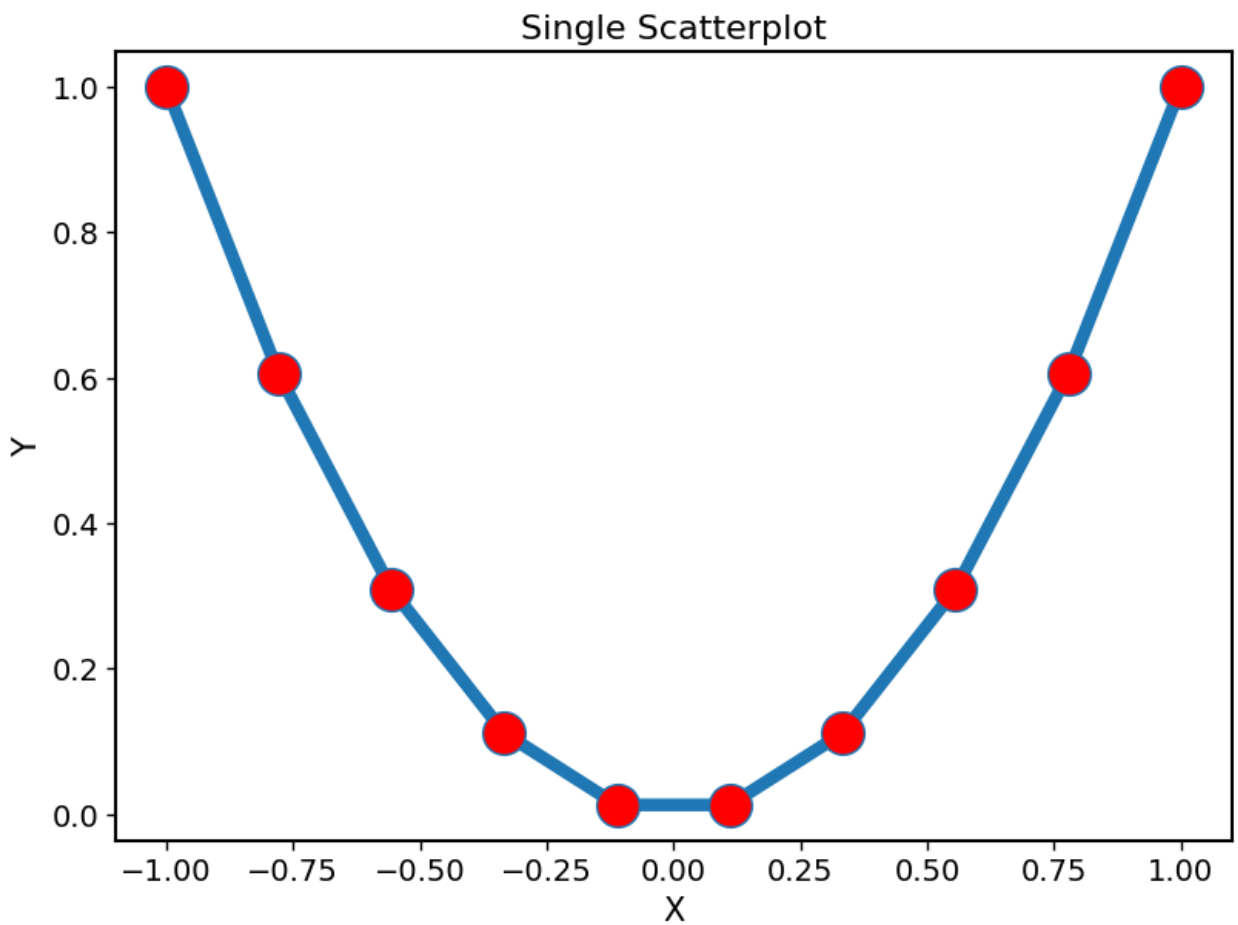
> **Warning:** Be careful with the this feature and especially the `extra_kwargs`, since there is no check for name clashes with this argument. You might also run into situations, where arguments of different names collide with arguments provided by the *Plotter*

```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis.parameters import ensure_plotter_consistency
```

(continues on next page)

(continued from previous page)

```
#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)

    #Now we process the given arguments (unknown ones are returned)
    kwargs = plot_params.set_parameters(continue_on_error=True, **kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()

    ax.plot(x, y, **plot_kwargs, **kwargs)

    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)
    plot_params.save_plot('figure')

    return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#The key markerfacecolor is not known to the MatplotlibPlotter
plot_with_defaults(x, y, markerfacecolor='red', markersize=20)
```




```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

Multiple plotting calls

The plotter classes also provide support for multiple plotting calls with different data sets in a single plotting function. To enable this feature we need to set two properties on the *Plotter*; `single_plot` to `False` and `num_plots` to the number of plot calls made in this function. The plot specific parameters can then be specified in two ways. Shown behind the two ways is the way to set the color of the second data set to `red`.

1. List of values (`None` for unspecified values) `[None,'red']`
2. Dict with integer indices for the specified values `{1: 'red'}`

Unspecified values are replaced with the previously set defaults.

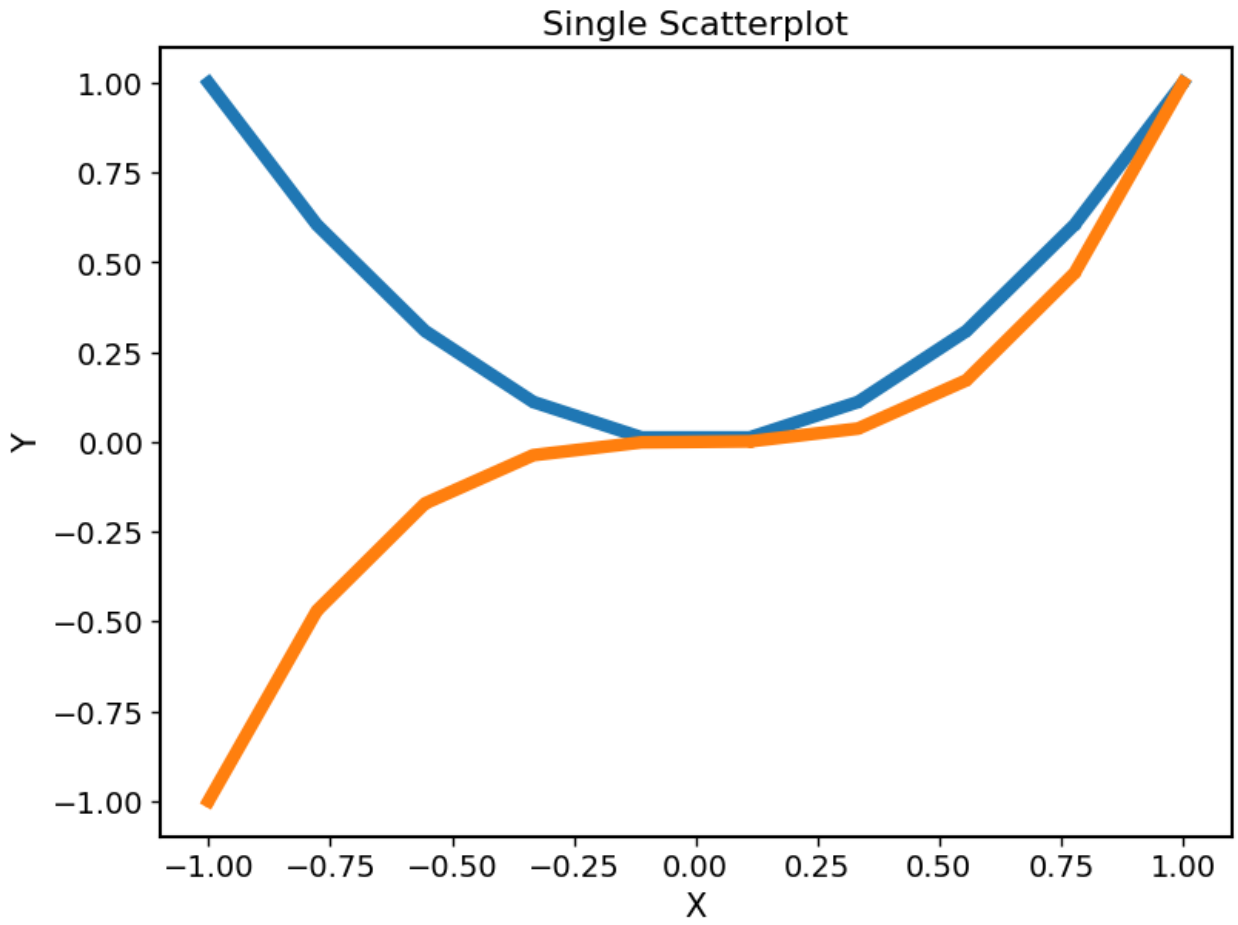
Note: The `num_plots` and `single_plot` properties are also reset by the *ensure_plotter_consistency()*

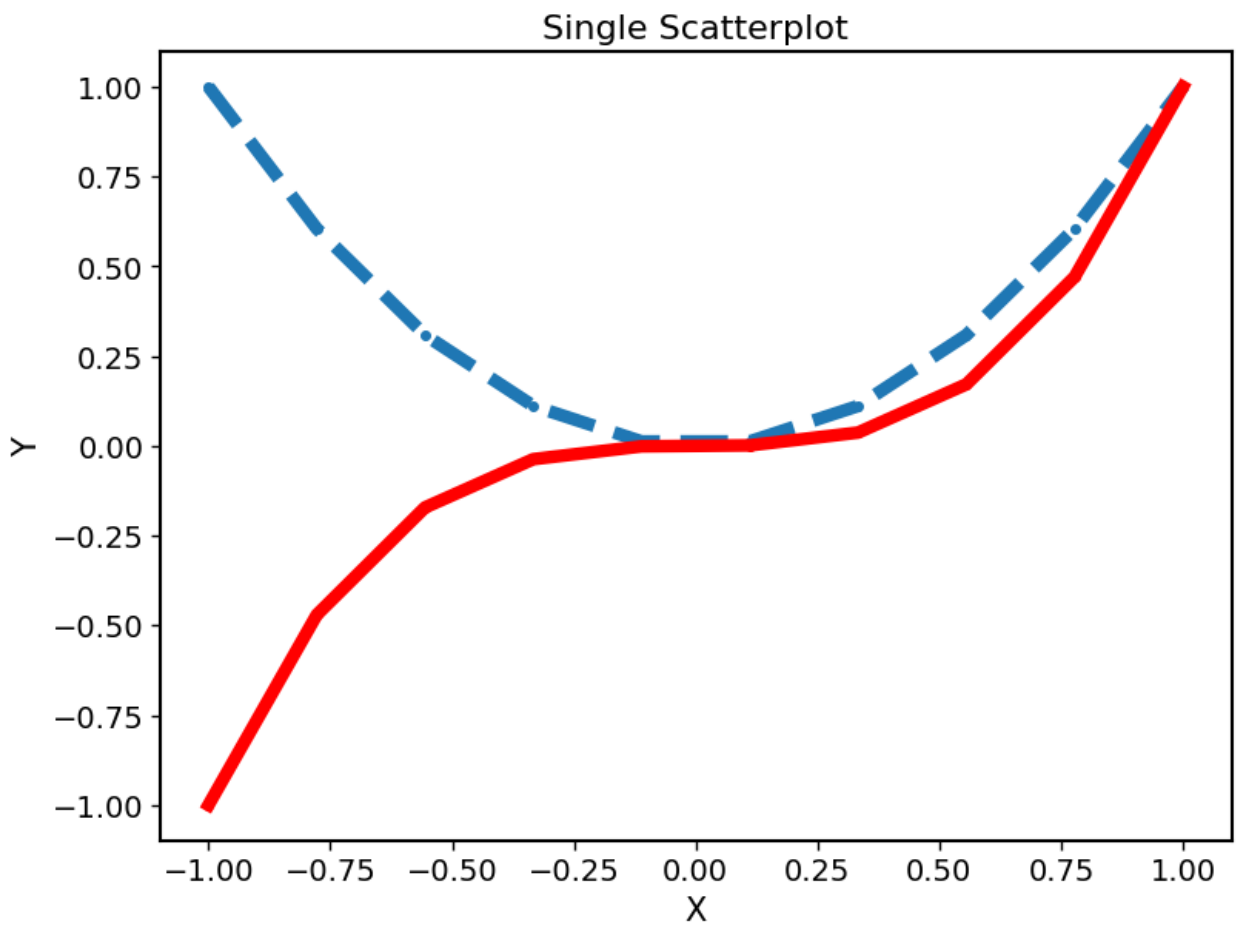
```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis.parameters import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
```

(continues on next page)

(continued from previous page)

```
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_2lines_with_defaults(x,y,**kwargs):

   plot_params.single_plot = False
   plot_params.num_plots = 2

   #Set the function defaults
   plot_params.set_defaults(default_type='function', linewidth=6)

   #Now we process the given arguments (unknown ones are returned)
   kwargs = plot_params.set_parameters(continue_on_error=True, **kwargs)

   #Set up the axis, on which to plot the data
   ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

   #The plot_kwargs provides a way to get the keyword arguments for the
   #actual plotting call to `plot` in this case.
   #For multiple plots this will be a list of dicts
   #of length `num_plots`
   plot_kwargs = plot_params.plot_kwargs()

   ax.plot(x[0], y[0], **plot_kwargs[0], **kwargs)
   ax.plot(x[1], y[1], **plot_kwargs[1], **kwargs)

   #The MatplotlibPlotter has a lot of small helper functions
   #In this case we just want to set the limits and scale of the
   #axis if they were given
   plot_params.set_scale(ax)
   plot_params.set_limits(ax)
   plot_params.save_plot('figure')

   return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2
y2 = x**3

#The key markerfacecolor is not known to the MatplotlibPlotter
plot_2lines_with_defaults([x,x], [y,y2])
plot_2lines_with_defaults([x,x], [y,y2],
                          color={1:'red'}, linestyle=['--',None])
```


Single Scatterplot
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X




```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

Custom function specific parameters

You might have situations, where you want to have some function specific parameters, that should pull from the previously set defaults or even a custom default value.

The *Plotter.add_parameter()* method is implemented exactly for this purpose. It creates a new key to be handled by the plotter class and with the arguments `default_from` or `default_value` we can specify what the defaults should be. `default_value` sets a specific value, `default_from` specifies a key from the plotter class from which to take the default value.

The *plot_kwargs()* method then can take keyword arguments to replace the arguments to take with your custom parameters

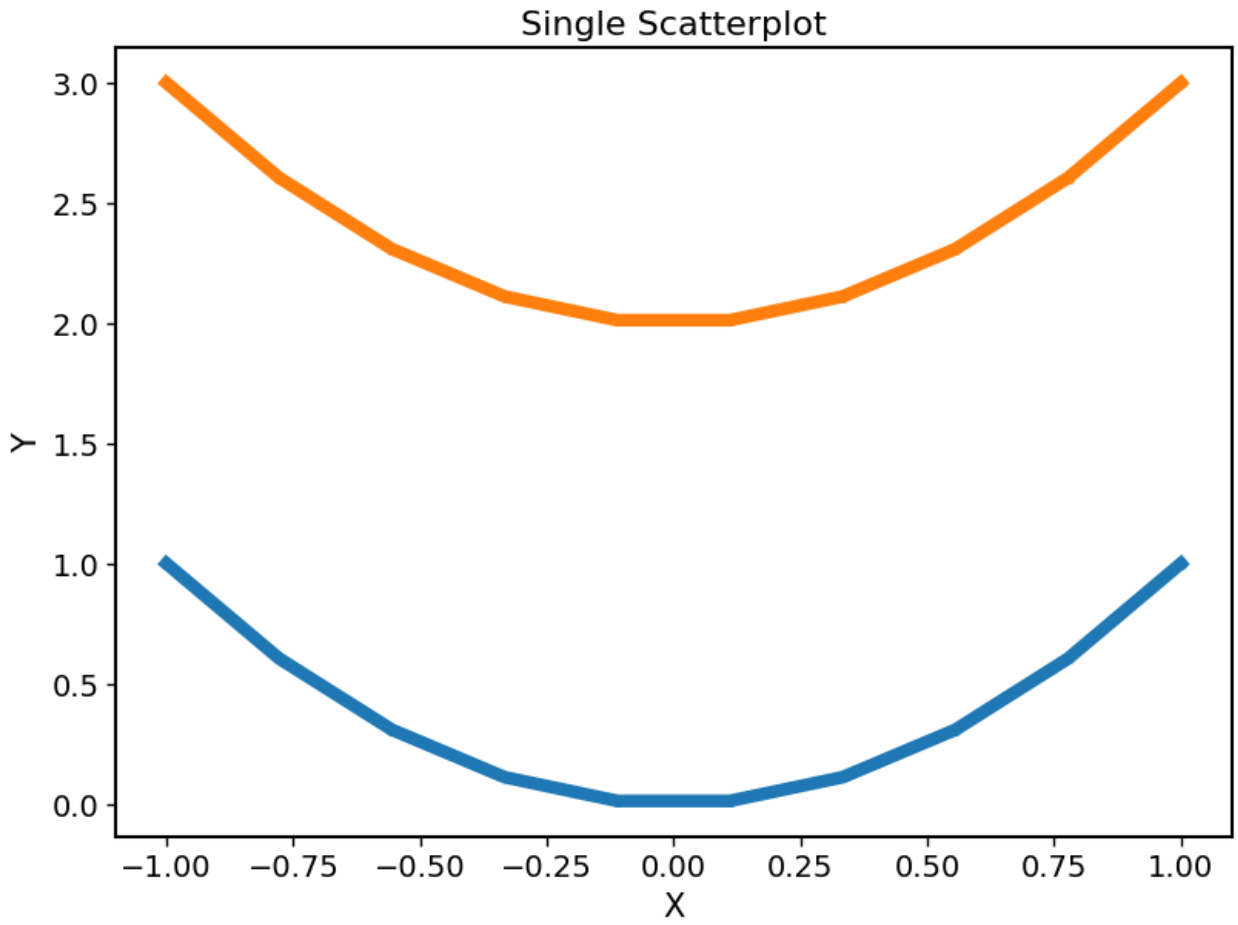
Note: These added parameters live on the function defaults and parameters level, meaning they will be removed by the *ensure_plotter_consistency()* decorator after the function finishes

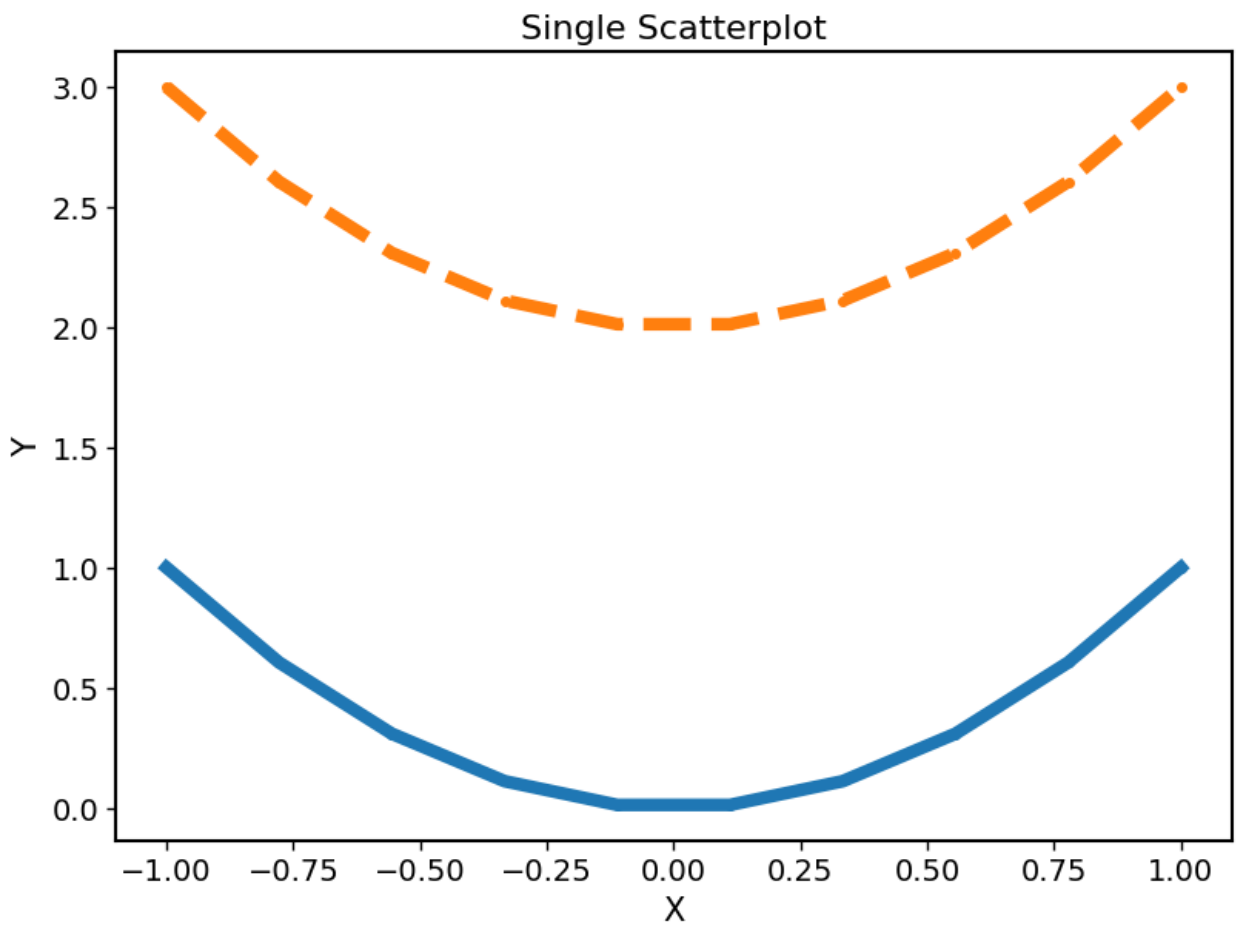
```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis.parameters import ensure_plotter_consistency
```

(continues on next page)

(continued from previous page)

```
#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_shifted_with_defaults(x,y,**kwargs):

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)

    plot_params.add_parameter('linestyle_shifted',
                              default_from='linestyle')

    #Now we process the given arguments (unknown ones are returned)
    kwargs = plot_params.set_parameters(continue_on_error=True, **kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()
    ax.plot(x, y, **plot_kwargs, **kwargs)

    #This call replaces the parameter linestyle with our custom
    #parameter linestyle_shifted
    plot_kwargs = plot_params.plot_kwargs(linestyle='linestyle_shifted')
    ax.plot(x, y+2, **plot_kwargs, **kwargs)

    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)
    plot_params.save_plot('figure')

    return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

plot_shifted_with_defaults(x, y)
plot_shifted_with_defaults(x, y, linestyle_shifted='--')
```


Single Scatterplot
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X




```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

Nested plotting functions

More complex plotting routines might want to call other plotting routines to simplify their structure. However, this has a side-effect when working with the *Plotter* class and the *ensure_plotter_consistency()* decorator. Since the decorator resets the parameters and function defaults after a plotting function has been called you lose everything that you might have modified in the enclosing plotting function.

If you do need access to these parameters after calling a nested plotting function the *NestedPlotParameters()* contextmanager is implemented. It defines a local scope, in which a plotting function can change the parameters and function defaults. After exiting the local scope the parameters and function defaults are always in the same state as when the `with` block was entered (Even if an error is raised). The nested plotting function will also start with the state that was set before.

Usage is shown here:

```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis.parameters import ensure_plotter_consistency
from masci_tools.vis.parameters import NestedPlotParameters
```

(continues on next page)

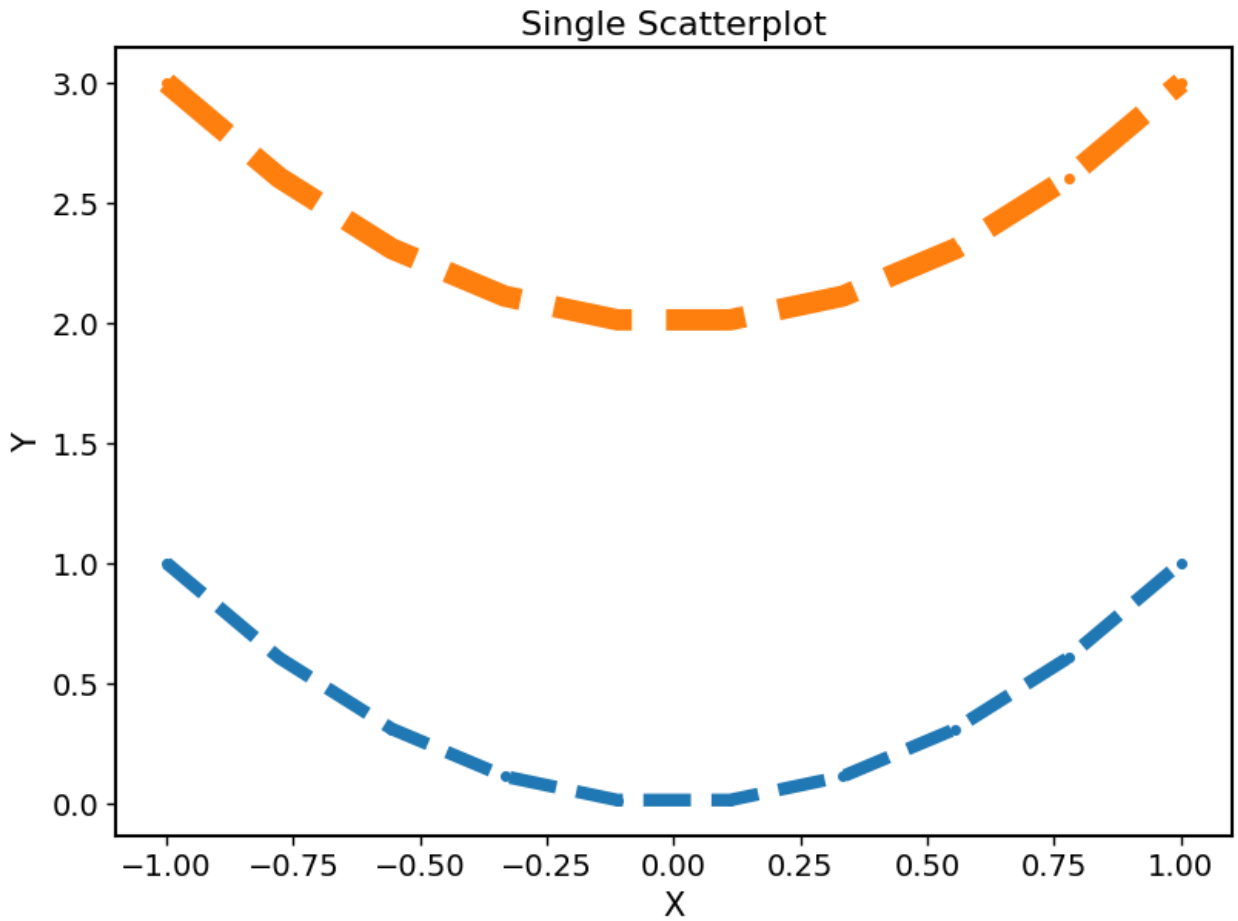
(continued from previous page)

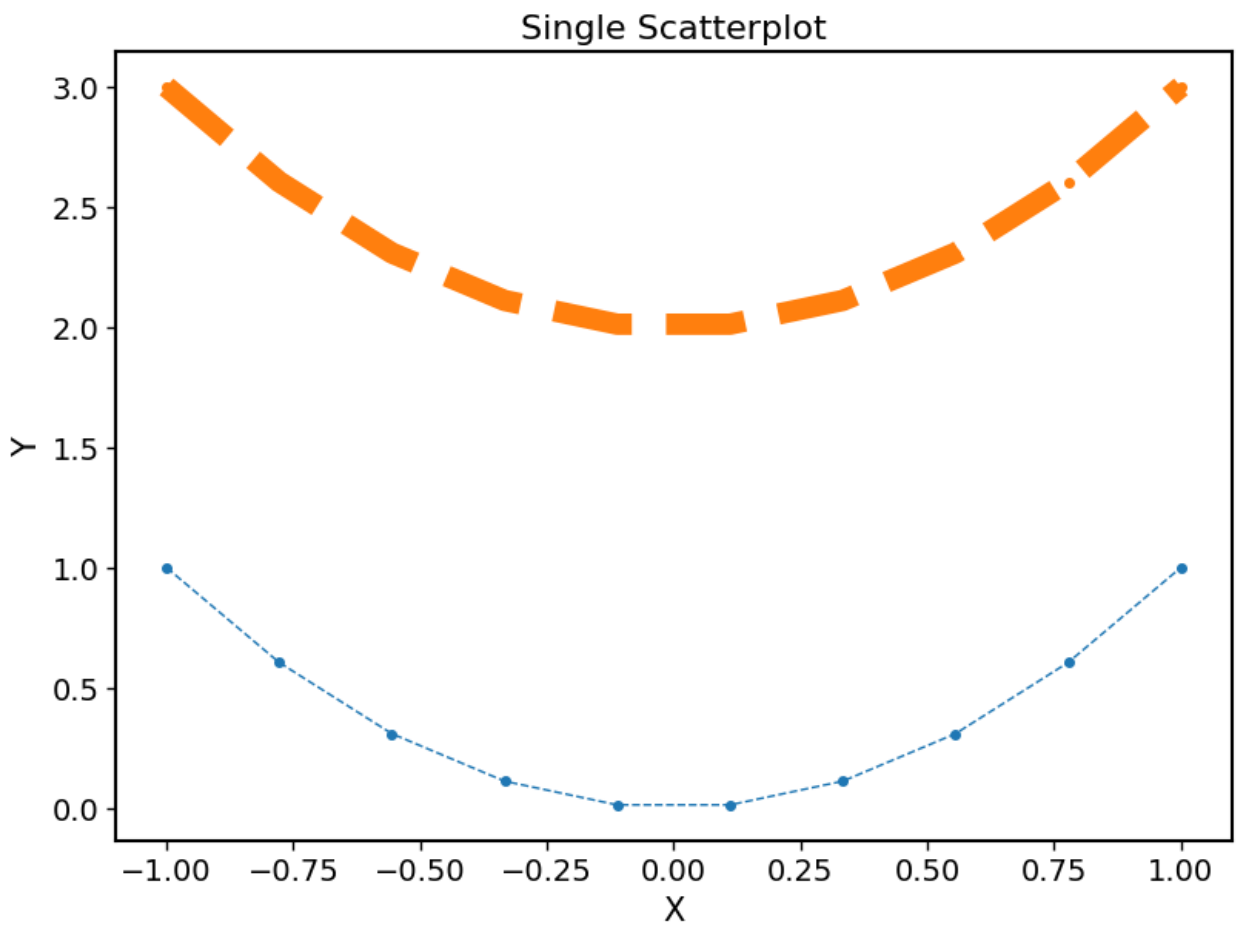
```
#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

@ensure_plotter_consistency(plot_params)
def nested_plot_function(x, y, **kwargs):

   plot_params.set_defaults(default_type='function',
                            linewidth=10, linestyle='--')

   #The contextmanager also needs a reference to the plotter object
   #to manage
   with NestedPlotParameters(plot_params):
      ax = plot_with_defaults(x,y,**kwargs)

   #Will plot with the above set defaults
   plot_kwargs = plot_params.plot_kwargs()
   ax.plot(x, y+2, **plot_kwargs)
   plot_params.save_plot('figure')

   return ax

@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

   #Set the function defaults
   plot_params.set_defaults(default_type='function', linewidth=6)

   #Now we process the given arguments
   plot_params.set_parameters(**kwargs)

   #Set up the axis, on which to plot the data
   ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

   #The plot_kwargs provides a way to get the keyword arguments for the
   #actual plotting call to `plot` in this case.
   plot_kwargs = plot_params.plot_kwargs()

   ax.plot(x, y, **plot_kwargs)

   #The MatplotlibPlotter has a lot of small helper functions
   #In this case we just want to set the limits and scale of the
   #axis if they were given
   plot_params.set_scale(ax)
   plot_params.set_limits(ax)

   return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2
```

(continues on next page)

(continued from previous page)

```
nested_plot_function(x, y)
nested_plot_function(x, y, linewidth=1)
```






```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

5.1.4 Using the `PlotData` class

5.1.4.1 Description

The *`PlotData`* class simplifies supporting data to plotting functions in multiple ways, while keeping the plotting functions themselves simple and easy to understand.

The basic idea of *`PlotData`* is to mimic the behaviour of the `data` argument in matplotlib or the `source` argument in bokeh. Suppose we have our data for a plot in a dictionary `d`, which has the keys `x`, `y1` and `y2`. If we now want to plot both `y` keys against `x` we can do this in the following way.

```
from masci_tools.vis.data import PlotData

plot_data = PlotData(d, x='x', y=['y1', 'y2'])

for entry, source in plot_data.items():
   #entry has the keys needed to get the data from the source
   #and source is the mapping to use

   print(entry.x, entry.y) #Yields x, y1 in the first loop and x, y2 in the second
```

(continues on next page)

(continued from previous page)

```
#Now we can plot the data
#for example plt.plot(entry.x, entry.y, data=source)
```

The keys are automatically expanded to be of the same length, if this is possible. There are three iteration modes, with the same names as for dicts:

keys
: Yields `namedtuple` with the keys for each plot

values
: Yields `namedtuple` with the values corresponding to the keys for each plot

items
: Yields the `keys` and their corresponding mapping for each plot

All of these functions have an argument `first`, which will only return the first element if it is given as `True`.

Note: The names `x` and `y` in the example above are completely arbitrary. The names for the columns and the fields on the `namedtuple` are determined by the keyword arguments given to *`PlotData`* at initialization

Note: At the moment the types of mappings accepted in the *`PlotData`* class are limited to `dict`, `pd.DataFrame` and `ColumnDataSource` (bokeh) objects

5.1.4.2 Initializing `PlotData` without a mapping

Users might want to provide data directly as arrays. If this should be allowed, there is a function *`process_data_arguments()`* to allow for this option. This function can either take a `data` argument with a mapping and the same keyword arguments as the *`PlotData`*.

```
from masci_tools.vis.data import process_data_arguments

plot_data = process_data_arguments(data=d, x='x', y=['y1','y2'])
```

Or you can provide the arrays directly without a `data` argument

```
from masci_tools.vis.data import process_data_arguments

#x,y1,y2 are the actual arrays
plot_data = process_data_arguments(x=x, y=[y1,y2])
```

If no `data` argument is given the keyword arguments are assumed to contain the data and they will be processed according to three rules:

1. If the data is a multidimensional array (list of lists, etc.) and it is not forbidden by the given argument the first dimension of the array is iterated over and interpreted as separate entries (if the data was previously split up into multiple sets a length check is performed)
2. If the data is a one-dimensional array and of a different length than the number of defined data sets it is added to all previously existing entries
3. If the data is a one-dimensional array and of the same length as the number of defined data sets each entry is added to the corresponding data set

Note: List or array in this context refers to `list`, `np.array` and `pd.Series`

5.1.4.3 Available routines on `PlotData`

There are a couple of routines for mutating/copyying or getting information about the data in a *`PlotData`* instance. These are not meant to be used heavily and should be used for typical simple work done for plot data processing, i.e. scaling, shifting, getting limits, ...

Note: The term data key in the following section refers to the keys of the keyword arguments given to *`PlotData`* at initialization or the fields on the namedtuples returned by iterating over an instance

- *`PlotData.get_keys()`*: Get all the keys for a given data key in a list
- *`PlotData.get_values()`*: Get all the values for a given data key in a list
- *`PlotData.min()`*: Get the minimum value for a given data key. A mask can be passed to further select the data. If `separate=True` is passed a list of minimum values for each plot is returned
- *`PlotData.max()`*: Get the maximum value for a given data key. A mask can be passed to further select the data. If `separate=True` is passed a list of maximum values for each plot is returned
- *`PlotData.apply()`*: Apply a lambda function to transform the data of a given data key (in-place!!)
- *`PlotData.get_function_result()`*: Apply a function to a given data key and return the results (Does not change the data)
- *`PlotData.sort_data()`*: Sort the data by the given data keys
- *`PlotData.group_data()`*: Group the data by the given data keys
- *`PlotData.shift_data()`*: Shift the data of a given data key either globally or with different shifts for each plot
- *`PlotData.copy_data()`*: Copy data to a of one data key to a new data key
- *`PlotData.distinct_datasets()`*: Return how many different datasets exist for a given data key

Warning: The methods *`PlotData.sort_data()`* and *`PlotData.group_data()`* will always convert the data sources to `pd.DataFrame` objects if they are not already.

CHAPTER

SIX

REFERENCE

6.1 Reference

6.1.1 Changelog

6.1.1.1 latest

full changelog

Nothing here yet

6.1.1.2 v.0.15.0

full changelog

Added

- Added support for Python `3.11` [#227]
- Added `FleurInputSchema.xsd` and `FleurOutputSchema.xsd` for the MaX6.2 release of fleur (file version `0.37`) [#240]
- KKR parser: Added keywords for Kkrimp BdG [#235]

Bugfixes

- Fixed order of added comments in `readd_inpgen_comments`. Previously they would end up in reverse order
- Fixed extraction of global magnetic moments from `out.xml` [#230]/[#231]

6.1.1.3 v.0.14.0

full changelog

Added

- Command `masci-tools convert-inpgen` to transform common structure formats, e.g. `cif` into inpgen files. Uses `ase` or `pymatgen` with the corresponding fleur plugins `ase-fleur` and `pymatgen-io-fleur` (install additional dependencies `cmdline-extras`)[#215]
- Functions `get_inpgen_comments` and `readd_inpgen_comments` to keep the comments of the inp.xml file containing the inpgen commandline and file content through functions using `clear_xml`

Improvements

- `write/read_fleur_inpgen` now supports the magnetic moment definitions in the inpgen input file introduced in MaX-6.1 (also the `scf` namelist) [#213]
- `get_structuredata` also reads out the magnetic moments [#213]
- `FleurXMLModifier.modify_xmlfile` now keeps the inpgen comments by default (controlled by option `keep_inpgen_comments`)

Bugfixes

- Serveral fixes for Fleur DOS plots [#212]:
  - Fixed wrong summation of atom weights for files containing 10 or more atomtypes
  - Fixed crashes using a custom weight not conforming to the naming scheme of weights in the `banddos.hdf`
  - Fixed specification of parameters by DOS label not working if the spin suffix is omitted

6.1.1.4 v.0.13.0

full changelog

Improvements

- `set_kpointmesh` now also writes out the `nx/ny/nz` attributes fro the dimensions of the kpoint mesh
- `get_structure_data`, `get_parameter_data` and `get_kpoints_data` are renamed to `get_structuredata`, `get_parameterdata` and `get_kpointsdata` to match the names of the corresponding functions in aiida-fleur. Old names are available with deprecations [#208]
- `FleurXMLModifier` now supports changes to input files with not yet available Fleur schemas, if the changes are compatible with the last available file schema [#209]

Bugfixes

- Bugfix in XML setters `set_inpchanges` and `set_attrib_value`, setting the `xcFunctional` key was previously not case-insensitive in constrast with all other keys
- Fixed crash in `get_parameter_data`. This function would previously crash if a kpoint mesh without `nx/ny/nz` attributes was used and the first point in the list was the gamma point

6.1.1.5 v.0.12.0

full changelog

Added

- Added XPath evaluation functions with runtime tpye checking of the results of the expressions [#181]
- Command `masci-tools fleur-schema pull <branch>` to update/add Fleur Schema files from the iffgit more easily [#184]
- New XML setters [#183]:
  - Setting XC functional explicitly + LibXC support (`set_xcfunctional`),
  - Creating a kpoint path using `ase` (`set_kpointpath`)
  - Creating a kpoint mesh with symmetry reduction using `spglib`. Should be equivalent to the `gamma@grid=nx,ny,nz` kpoint generator in `inpgen` (`set_kpointmesh`)
- Added `FleurInputSchema.xsd` and `FleurOutputSchema.xsd` for the MaX6.1 release of fleur (file version `0.36`) [#196]

Bugfixes

- Add clearer error message if `None` is passed to the `convert_to_xml` functions. This would happen for example using the `set_inpchanges` function with `{'minDistance': None}` [#182]
- Fixed `masci-tools fleur-schema add` with `--from-git` flag. Previously it would still check for the existence of the Schema file locally [#184]
- `get_fleur_modes`: `gw` mode renamed to `spex` and now stores the actual integer value of the attribute [#185]
- Bugfix in `clear_xml`, when multiple XML comments are present outside the root element [#193]
- Bugfix in `reverse_xinclude`. This would previously break when reexcluding trees already containing a `relaxation` tag and would end up with two `xi:include` tags for the `relax.xml` [#194]
- Bugfix for `FleurXMLModifier`. The `task_list` property would incorrectly enter a `kwargs` key if the modifying function in question has an explicit `**kwargs` argument
- Bugfix in matplotlib plots with placement of multiple colorbars (e.g. weighted spin-polarized bandstructure) [#198]

6.1.1.6 v.0.11.3

full changelog

Improvements

- Changes to KKR plotting routine `dispersionplot` for compatibility with AiiDA v2.0
- Connecting vectors for intersite `GreensFunction` are now saved in Angstroem. For better interoperability with ase, pymatgen, AiiDA

For Developers

- Relaxed CI requirements for docs build. Nitpicky mode is no longer required to pass but is treated as a hint to look into the warnings

6.1.1.7 v.0.11.2

full changelog

Improvements

- Several changes in KKR IO functions to make them compatible with `aiida-core>=2.0` [#175]
- Add function to calculate fourier transform of e.g. `J_ij` constants calculated from Green's functions (`masci_tools.tools.greensf_calculations.heisenberg_reciprocal`)

Bugfixes

- Fixed nondeterministic order in bokeh regression tests if multiple dictionaries with the same values but differing keys in the same list (e.g. Providing the same data for different columns)
- Fixed wrong names for columns entered in `decompose_jij_tensor`, i.e. `J_ji -> J_ij`

Deprecations

- Deprecated the unused modules `util/kkr_rms_tracker.py` and `util/modify_potential.py`

6.1.1.8 v.0.11.1

full changelog

Bugfixes

- Bugfix in `reverse_xinclude`. Version `0.11.0` broke this function for versions, where the `relaxation` tag was not allowed.

6.1.1.9 v.0.11.0

full changelog

This release adds some improvements to the XML and HDF5 handling mainly for better AiiDA-Fleur support. Also major updates to documentation configurations and Green's function calculations.

Added

- Added `FleurElementMaker` class. This can be used to create XML elements compatible with a given version from scratch. Has case-insensitivity and converts values to strings for XML [#159] Example

```
from masci_tools.util.xml.builder import FleurElementMaker
E = FleurElementMaker.fromVersion('0.35')
new_kpointset = E.kpointlist(
    *(
        E.kpoint(kpoint, weight=weight, label=special_labels[indx]) if indx in␣
↪special_labels else
        E.kpoint(kpoint, weight=weight) for indx, (kpoint, weight) in␣
↪enumerate(zip(kpoints, weights))
    ),
    name=name,
    count=nkpts,
    type=kpoint_type)
```

- Function `serialize_xml_arguments` to `masci_tools.util.xml.common_functions` to remove XML elements/trees from positional/keyword arguments and replace them with string representations of the XML. Can be used in AiiDA-Fleur
- Module `masci_tools.util.ipython` and ipython extension (`%load_ext masci_tools`). Adds syntax highlighted XML tree output and creating HTML syntax highlighted diffs of XML trees [#158]
- Added calculation of Jij Tensor from intersite Green's functions [#170]

Improvements

- Added `name` entry to `SchemaDict.tag_info` which contains the tag name in the original case [#159]
- `convert_to_xml` is made more strict. Conversion `int` to `str` uses the `{:d}` format specifier and string conversion is no longer always attempted [#159]
- Improvements to Colorbar creation in matplotlib plotting methods. Limits are now set consistently with `limits={'color': (low, high)}` in the plot and colorbar. Spinpolarized bandplots now show two colorbars for the two colormaps if requested
- `get_parameter_data` now also extracts the `gamma` switch for kpoint generation for more consistent roundtrips. This is only set if the first kpoint in the mesh is the gamma point and there are multiple
- `load_inpxml` and `load_outxml` now consistently accept the XML file given as a string of the content. The content no longer has to be manually encoded as bytes

- The method `FleurXMLModifier.modify_xmlfile` now always returns two things. The modified XML tree and a dictionary with all additional file contents, e.g. `n_mmp_mat`.
- Support for aligning spin/real-space frames of Green's functions. Several further imporvements/bugfixes for Green's function modules [#170]

Bugfixes

- Bugfix for `outxml_parser` returning a nested list for Hubbard 1 distances, where a flat list was expected. Removed `force_list` argument from the parsing task definition
- Fixes in `plot_fleur_bands`, when providing custom weights without spin suffixes, i.e. `_up/_down`
- Fix in `HDF5Reader`. IO like objects without an attached filename would lead to an early error. This is the case for example for some readers in the file repository implementation used in AiiDA v2.0
- Fix in `HDF5Reader`. The file handles for compressed files in the AiiDA v2 repository have to be copied into a temporary file first before they can be used

For Developers

- Docs: Updated `sphinx` and `sphinx-autodoc-typehints` versions and build docs on python 3.10 [#156]
- Docs: Converted to `MyST` markdown and where appropriate introduced `myst-nb` for executing code cells in the documentation, e.g. generate plotting examples [#157]
- Bokeh regression tests now strip out the bokeh version from the test files
- Added pre-commit hook, which generates the docstrings for the `FleurXMLModifier` registration methods from their XML setter function counterparts [#166]

6.1.1.10 v0.10.1

full changelog

Bugfixes

- Remove accidentally left in debug print in `outxml_parser`

6.1.1.11 v.0.10.0

full changelog

This release provides several new features in the XML modification/evaluation for Fleur XML files and bugfixes. Multiple problems when working with DFT+U density matrix files are also fixed.

Added

- New XML setter `align_nmmpmat_to_sqa` to rotate the density matrix file according to SQAs specified either for noco or second variation SOC [#140]
- Added `task_list` property to `FleurXMLModifier` to construct a list which can be used to replicate the same `FleurXMLModifier` with the `fromList()` classmethod [#149]
- Added `FleurXMLContext`, which acts as a holder of th XML elements, schema dictionary, constants and logger to reduce the amount of information/clutter in functions evaluating things from the XML file [#152]

  Note: The class `ParseTasks` used in the `outxml_parser` was simplified and placed into the `outxml_parser` module and the decorator `register_parsing_function` was removed. This was done without deprecation since they were exclusively used in the `outxml_parser` and were the main cause of cyclic import problems previously
- Added several predefined conversions to/from input version `0.35` to `inpxml_converter` [#153]

Improvements

- Added `inverse` argument to nmmpmat XML setters. These will correctly produce the inverse rotation operation for the given angles. Also allow setting `orbital='all'` in `rotate_nmmpmat` to rotate all blocks by the given angles [#140]
- The XML setters `create_tag`, `replace_tag` and their low-level equivalents now also accept XML strings, i.e. `<example attribute="1"/>`, as arguments for the elements to create/replace [#145]

Bugfixes

- Fix for XML setters operating on the DFT+U density matrix file. Previously these functions would not map the density matrix blocks correctly if multiple atomgroups shared the same species containing `ldaU` tags [#140]
- Added missing prefactor `(-1)^(m-mp)` to `get_wigner_matrix()`
- Added basic tests of `masci_tools.tools.greensfunction` module and fixed several bugs found due to this [#150]
- Fixed bug in XML setters operating on the DFT+U density matrix file not correctly extracting the number of spin blocks when only setting `l_mperp`
- Fixed bug, when using the `FleurXMLModifier` directly (not in `aiida-fleur`), included XML files were not handled
- Fixed bug in `outxml_parser`, when the XML file had to be repaired and more than one iteration was present the wrong iteration was chosen as the last stable iteration [#152]

Deprecated

- The module `masci_tools.io.io_fleurxml` is renamed to `masci_tools.io.fleur_xml` [#152]
- The module `masci_tools.util.parse_task_decorator` is removed. All decorators are now availaibe under `masci_tools.io.parsers.fleur` [#152]

For Developers

- Added `py.typed` marker to masci-tools, since a large part of the outside facing code (especially the XML APIs are typed). With this marker other packages can use the typehints in this package

6.1.1.12 v.0.9.1

full changelog

Added

- Standalone function `masci_tools.tools.fleur_inpxml_converter.convert_inpxml` to allow conversions of `inp.xml` files within a python runtime without needing to go via the commandline

Bugfixes

- Fixed bug in bokeh testing fixtures using the wrong folder for fallback versions
- Fixed bug not correctly converting complex numbers from the Fleur xml files if they have whitespace at beginning/end

6.1.1.13 v.0.9.0

full changelog

Added

- New `bokeh` plot routine for matrix plot of rectangle patches with added texts [#124]
- Added TS contribution to free energy to output of `outxml_parser`

Improvements

- Several arguments in XML setter functions were renamed for more consistent signatures. The main changes are
  - `attributedict/change_dict` -> `changes`
  - `attributename/attribv` -> `name/value`
  - `add_number` -> `number_to_add`

  The old signatures are still supported with deprecations if called via the `FleurXMLModifier` [#118]
- Remove constraint on `bokeh` version (previously `<=1.4.0`) [#122]
- Add `only_spin` option and calculate complete non-spinpolarized DOS for `spinpol=False` in `plot_fleur_dos` [#125]
- Refactored `CFCalculation`, i.e. naming of attributes handling of cutoffs. Added classmethod to construct instance directly from numpy arrays [#127]
- Refactored plotting methods for `CFCalculation` to allow the same parameter freedom as for the other matplolib routines [#127]

- Improvement to labels and legends in DOS/bandstructure plots. Matplotlib plots now put the legend centered below the plot and added latex labels to axis and ticks in bokeh (version `2.4.0` needed) [#133]
- `io_nmmpmat`: Allow negative indices in `read_nmmpmat_block` and raise error for invalid index

Bugfixes

- Fix for signatures of `set_text/set_first_text`. These contained names of attribute setting functions [#118]
- Fix for validating arguments in `FleurXMLModifier` not accepting an argument named `name` when passed by keyword. [#118]
- Fixed problems in `masci_tools.testing.bokeh` when adding files for new bokeh versions [#122]
- Several fixes for `plot_fleur_dos`. Using the `area_plot` or specifying `color` explicitly could mess up the color order [#132]
- Fixed bug in `validate_nmmpmat` and consequently `FleurXMLModifier` not correctly validating denisty matrix files with certain off-diagonal elements being negative [#135]
- Fix for `HDF5Reader` for compatibility for file handles in `aiida-core` 2.0. The file handles coming from the file repository have no directly attached extension so the check if the file is a hdf file cannot be performed

For developers

- Fixed upload of pytest-mpl results artifacts to include the whole directory with images and not just the HTML file [#117]
- Updated typing to newer version of `lxml-stubs` (`0.4.0`) [#123]

6.1.1.14 v.0.8.0

full changelog

Added

- Added `FleurInputSchema.xsd` and `FleurOutputSchema.xsd` for the MaX6 release of fleur (file version `0.35`) [#112]
- New XML getter function: `get_special_kpoints` extracts the labelled kpoints from a kpoint list (for now only implemented for Max5 or later)
- Added extraction of Hubbard 1 input and distances in the `outxml_parser` for fleur (distances only available starting from version `0.35`) [#108]
- Added extraction of global vector of magnetic moments in non-collinear calculations in `outxml_parser` under the key `magnetic_vec_moments` starting from version `0.35`

Improvements

- Fleur schema parsing functions now recognize a new alias from the fleur schemas `FortranComplex` which is a number of the form `(float,float)`. Converters for complex values are added. (Note: Complex numbers should not yet be used in the `outxml_parser`, since AiiDA (<2.0) does not support complex numbers yet) [#106]
- Added `IncompatibleSchemaVersions` error when a combination of output and input version for `OutputSchemaDict` is given, for which it is known that no XML schema can be compiled
- `xml_getters` functions can now be used in the task definitions of the `outxml_parser` to keep information consistent. This example definition will insert the structure data, i.e. a tuple of atoms, bravais matrix and periodic boundary conditions into the output dictionary. `{'parse_type':'xmlGetter', 'name': 'get_structure_data'}` [#107]
- The `_conversions` key in the `outxml_parser` now accepts namedtuples `Conversion` to enable passing additional arguments to these functions. [#109]
- Adjusted `get_cell` to understand the `bravaisMatrixFilm` input introduced with the MaX6 release of fleur [#110]
- Improved detection, whether a given xpath contains a tag including stripping predicates. Added function `contains_tag` in `masci_tools.util.xml.common_functions` [#113]
- Refactored bokeh plot routine `periodic_table_plot` to make use of the plot parameters utilities [#114]
- `get_parameter_data` now extracts LOs with higher energy derivatives or `HELO` type, as they are supported by the newest versions of the inpgen. The old behaviour of dropping all non `SCLO` and `eDeriv="0"` LOs is available via the option `allow_special_los=False`

Bugfixes

- Fix in `load_inpxml` and `load_outxml` (this also effects the `inpxml/outxml_parser`). Previously file handle like objects not directly subclassing `io.IOBase` would lead to an exception
- Added patch for `OutputSchemaDict` objects with `FleurOutputSchema.xsd` files before version `0.35`. The attribute `qPoints` in the DMI output was actually called `qpoints` in these schemas, making it impossible to retrieve this attribute
- Fixed behaviour of relative XPath methods of `SchemaDict` which did not correctly handle root tags, whose names are contained in other tag names, for example `bravaisMatrix` and `bravaisMatrixFilm` from the new file version `0.35`

Deprecated

- Passing strings in the `_conversions` key in task definitions for the `outxml_parser`. Use `masci_tools.util.parse_utils.Conversion` instead. [#109]

For developers

- Reorganized visualization tests, making the regeneration of baseline images with `pytest-mpl` easier [#101]
- Switched build system from `setuptools` to `flit`, since this way all the configuration can be specified in the `pyproject.toml` and a lot of duplication of information is avoided (e.g. version numbers) [#102]

6.1.1.15 v.0.7.2

full changelog

Bugfixes

- Fixed regression in `set_atomgroup_label` and `set_species_label`. These functions could be used in previous versions with atom labels, that do not exist. This is not possible in `v.0.7.1`. Since some parts of the `aiida-fleur` plugin relied on this the behaviour has to be kept.

6.1.1.16 v.0.7.1

full changelog

Added

- `XPathBuilder` class for specifying complex conditions on xpaths with a dictionary. Added `filters` argument to `schema_dict_util` and `xml_setters` functions for this dictionary [#96]

Bugfixes

- Fixed issue with `MANIFEST.in`, where non-python files from the `tools` subpackage were not included in the built packages
- Fixed bug not correctly processing the plot limits in `plot_fleur_bands` in excluding points outside the plot area for better performance
- Fix for HDF5 transformation `add_partial_sums` if not all formatted patterns are present in the dataset, e.g. if a bandstructure/DOS is calculated for only selected atoms

For developers

- More strict `mypy` configuration and moved a lot of the annotations to modern syntax with `from __future__ import annotations`
- Added `pyupgrade` hook to automatically do some easy refactoring, i.e. removing compatibility workarounds move to modern syntax. Set to apply changes compatible with `3.7` and later

6.1.1.17 v.0.7.0

full changelog

Commandline interface, refactoring of SchemaDict/XML functions and major improvements for package configuration/tooling for developers. Added support for python `3.10`. Dropped testing for python `3.6`.

Added

- Added click command line interface (available as `masci-tools`). Can add Fleur XML Schema files with validation (Also directly pulled from the Fleur git), use the XML parsing functions and interface to the fleur visualization routines [#49]
- Added `Tabulator` for use of creating `pandas.Dataframes` from attributes of python objects. Used in `aiida-jutools` to tabulate attributes of aiida nodes
- Added `optional_tasks` argument to `outxml_parser`. Adds tasks marked with `'_optional': True` to the performed tasks [#81]
- Added visualization routine for spectral functions (colormesh plot with path though Brillouin zone)
- Added tool for converting `inp.xml` files between different file versions (Available through the click CLI `masci-tools inpxml`) [#88]
- Added three new XML setters: `clone_species` (Create and modify a species starting from an existing one), `switch_species/switch_species_label` for switching the species attribute of atom groups with additional checks
- `outxml_parser`: Total Energy is now taken from the output `freeEnergy` in the `out.xml`

Improvements

- Refactored SchemaDict code. Moved routines `get_tag_xpath` and similar to methods on the SchemaDict. If the path cannot be determined custom exceptions `NoPathFound` and `NoUniquePathFound` are now raised [#84]
- Added utility to `OutputSchemaDict` to create absolute paths into `iteration` elements in `out.xml`. Added support for this option in `schema_dict_util` functions with `iteration_path=True`
- All basic XML modification functions now accept either a `ElementTree` or `Element`. and warn they find no nodes to operate on
- Improved capabilities of green's function tool, can now be used with radially resolved/k-resolved Green's functions
- Improved performance of Fleur XML Schema parsing by switching from the `xpath` method on the `ElementTree` to constructing a `XPathEvaluator` object [#89]
- All `xml_getters` functions can now also be used with `out.xml` files
- `set_atomgroup/set_atomgroup_label` now use `switch_species` if the species attribute is changed
- `set_atomgroup` now supports the `all-<search string>` syntax for species argument, equivalent to `set_species`
- Improved behaviour of spin-polarized DOS plots for duplicating all plot parameters for spin-down components (previously only color was repeated)

Bugfixes

- Fixed several issues in version handling of Schema dictionaries. It is now possible to add a new schema and have it work (with warnings) without needing to change any code (`masci-tools fleur-schema add <path/to/schema>`)
- Bugfix in `evaluate_tag` not handling the combination of options `subtags=True` and `text=True` correctly. Previously some results were overwritten.
- Fixed accidental change in `write_inpgen_file` in comparison to old `aiida-fleur` implementation. Now the species name is always appended to the position in the inpgen file if it is not equal to the atom symbol
- Fixed behaviour of `get_parameter_data` for inputs with local orbitals with higher energy derivatives. These cannot be created by the inpgen and so are dropped
- Fixes in `xml_setters` to allow consistent creation of multiple tags for setting text or attributes

Deprecated

- The `fleur_schema` subpackage was moved from `masci_tools.io.parsers.fleur` to `masci_tools.io.parsers` to avoid circular import issues [#87]

Removed

- `get_structure_data` now returns `AtomSiteProperties` for the atom information by default. The default value of `site_namedtuple` is now `True`

For developers

- Moved configuration of `yapf`, `pylint` and `pytest` into `pyproject.toml`
- Made test suite executable from the root-folder (Some file paths were not transferrable when changing the execution directory)
- Added `test_file` fixture, which constructs the absolute filepath to files in the `tests/files` folder to reduce the difficulty of moving test files around and reorganizing the pytest suite
- Updated pylint (`2.11`), pytest (`6.0`) in `setup.py`
- Added `mypy` pre-commit hook. Checked files are specified explicitly [#86].
- Added typing to majority of XML functions (with stubs package `lxml-stubs`) and large parts of the `io` and `util` subpackages
- Dropped testing for python `3.6` in CI

6.1.1.18 v.0.6.2

full changelog

Small bugfixes and refactoring for plotting routines

Added

- Common plot routines for equation of states `eos_plot` and plotting scf convergence `convergence_plot`
- Replaced old convergence plot routines with single routines for bokeh/matplotlib named `plot_convergence`

Improvements

- Moved eos and convergence plots to use `PlotData` class
- Fixed some edge cases of bokeh testing fixtures. Previously some plots would crash the code for normalizing the json

Bugfixes

- Fixed bugs in convergence routines for using them in aiida-fleur

6.1.1.19 v.0.6.1

Release fixing a small issue with publishing version `0.6.0` to zenodo

6.1.1.20 v.0.6.0

full changelog

This release contains major improvements to plotting methods and new tools. Also the fleur parsing functions were improved

Added

- `PlotData` class for handling data passed to plotting methods very flexibly [#54] (For more information see the relevant users guide or developers guide sections in the documentation)
- `masci_tools.vis.common` module for plotting methods with common interfaces for bokeh/matplotlib [#71]
- `get_parameter_data` also extracts kpt mesh specifications for the input generator
- Exposed and improved bokeh testing fixtures in `masci_tools.testing.bokeh` for use in higher level packages
- `greensf_calculations` module in `tools` with sample functions for calculating properties with green's functions from fleur
- Added two options `line_plot` and `separate_bands` to bandstructure plots. While `line_plot` is obvious (no weighted bandstructures possible), `separate_bands` allows to set parameters for single selected bands. These options can also be combined

Improvements

- Added option `only_used` to `get_kpoints_data` to get only the `kPointList` referenced in the `kPointListSelection` tag
- Made `constants` argument to `schema_dict_util` functions completely optional. Will raise an exception if a undefined constant is encountered
- Bandstructure plots now exclude points outside the plotting area to speed up these plots significantly for systems with a large number of bands
- Refactored attribute/text type definitions in `SchemaDict` objects. Now unified under one structure. Both attributes and texts can now be recognized to contain multiple values [#64]
- Added `spin_arrows` option to toggle spin arrows in `plot_spinpol_dos` for matplotlib. Previously this was only possible for bokeh
- Added options to create different types of bar plots to `barchart`: Available are `'stacked'` (default), `'grouped'`, `'independent'` (positions can be defined for each data set)
- Exceptions occurring in `transforms` for `HDF5Reader` are now bundled into `HDF5TransformationError` to allow easier error handling
- Added MT keys to `kkrparams`

Bugfixes

- Fix for `write_inpgen_file`, which was incorrectly inserting the `'X'` (empty sphere) element into inpgen files
- Fix for `read_inpgen_file`, which could not handle inpgen files with comments on certain lines in the inpgen file
- Several fixes for `plot_fleur_dos` not using the standard DOS calculation but orbital decompositions and so on
- Adjusted default `dpi` for matplotlib to `100` to avoid problems with size when using `plt.show()` instead of saving

6.1.1.21 v.0.5.0

full changelog

This release contains fleur inpgen IO capabilities and many major improvements

Added

- IO routines for reading/writing inpgen input files
- `get_structure_data` now returns the relaxed structure if a `relaxation` section is present
- `get_parameter_data` can read in the electron configuration if requested
- `get_symmetry_information` xml getter to get all defined symmetry operations
- `get_structure_data` and `get_parameter_data` now norm the species names to get consistent ids for usage in the inpgen files [#70]

Improvements

- Added another possible value to environment variable `MASCI_TOOLS_USE_OLD_CONSTANTS` to get the values of constants in between commits c171563 and 66953f8 [#66]
- `spex` attribute is now used in addition to old `gw` attribute to determine whether a fleur input file is from a SPEX calculation
- `evaluate_tag` can now also get the text and recursively parse all subtags
- Changed default of `multiply_by_eqiv_atoms` in `plot_fleur_dos` to `True`
- `barchart` can now make horizontal bar charts
- `get_cell` now also takes the `scale` attribute into account
- Made conversion to angstroem optional (done by default) in the xml getters. Can be turned off with the options `convert_to/from_angstroem`
- Deprecated: Old atom position output in `get_structure_data`. Use option `site_namedtuple=True` to get new output (includes species information)

Bugfixes

- Fixes to make `plot_convex_hull_2d` work
- Fix in `evaluate_text` make `constants` argument optional
- Fix for `get_structure_data`; Added missing unit conversion for z-coordinate of film positions

6.1.1.22 v.0.4.10

full changelog

This release contains bugifxes for the visualization routines

Bugfixes

- Fixed issue for histogram plot, which prevented usage for e.g. stacked histograms with multiple datasets
- Fixed usage of dictionary arguments in plot_fleur_dos, that should not be used to specify parameters for multiple datasets (e.g. limits, lines)
- Updated documentation table of plot parameters to put all dictionaries into literal blocks. Otherwise single quotes appear differently on read the docs

6.1.1.23 v.0.4.9

full changelog

Improvements

- Added saving/loading of plot defaults in json files to Plotter class (No exposed functions yet in plot_methods or bokeh_plots)
- DOS plot parameters in plot_fleur_dos can now be specifified by label, e.g. color={'Total_up': 'blue'}

Bugfixes

- Various Bugfixes and improvements to ChemicalElements class, greensfunction tools and plotting methods
- Fixed wrong atom label generation in DOS plots (#55)

6.1.1.24 v.0.4.8

full changelog

Added

- Tool to analyze/work with greensfunctions calculated by Fleur `masci_tools.tools.greensfunction`

Improvements

- Several improvements of KKR parsers/parameters [#13]
- Introduced patching functions for the schema dictionaries to manually correct ambiguities [#48]
- Improvements of the `HDF5Reader` and recipes for Fleur DOS/bandstructure calculation
- For devs:
  - pylint warnings are no longer fatal for the CI jobs
  - tests folder moved outside package directory

Bugfixes

- Bugfixes and improvements in the plotting functions for DOS/bandstructures
- Bugfixes in `xml_setters` and `fleurxmlmodifier` modules

6.1.1.25 v.0.4.7

full changelog

Added

- Introduced higher level XML modification functions for `xml_delete_tag`, `xml_delete_att`, `xml_replace_tag`

Improvements

- moving conversion factors for energy and bohr to angstroem conversion to NIST values in KKR parts of the code. This can also be disabled for backwards compatibility by setting the environment flag `MASCI_TOOLS_USE_OLD_CONSTANTS`
- Added missing energy unit alias for Fleur input files
- `xml_delete_tag`, `xml_delete_att`, `xml_replace_tag` now also support the `occurrences` argument
- FleurXMLModifier improvements. Modification registration methods will now test the given arguments against the modifying functions immediately to fail early for errors. `fromList` classmethod allows the instantiation with a known list of tasks to perform. These are passed through the same procedure as the normal registration of changes

6.1.1.26 v.0.4.6

full changelog

Bugfixes

- Fix for the `clear_xml` function, where comments could end up in the set of included tags. This lead to failures in aiida-fleur

6.1.1.27 v.0.4.5

full changelog

Added

- Introduced function to split a xmltree back up into the included subtrees and the main tree with `xi:include` tags
- clear_xml now returns a set of the tags, that were included

Bugfixes

- various bugfixes for xml modifying functions
- A special case for `theta`, `phi` and `ef_shift` attributes of forceTheorem tags, since they are not correctly typed in the Inputschema

6.1.1.28 v.0.4.4

full changelog

Added

- XML getter methods for number of kpoints and relaxation information
- XML setter methods for manipulating kpoints

Improvements

- top level `create_tag` now also accepts a etree.Element
- XML getters now also accept etree.Element
- Added `etree.indent` calls to keep modified `inp.xml` clean (raises lxml dependency constraint to 4.5)
- `io_fleurxml` functions now pass keyword arguments to XMLParser
- Re-add `fleur_modes` to output_dict

Bugfixes

- Bugfix for relative xpaths

6.1.1.29 v.0.4.3

full changelog

Added

- Added utility for creating schema dictionary version specific functions
- Added XML getter functions adapted from aiida-fleur fleurinpdata methods [#40]

Improvements

- Improved logging and failure handling of XML parser functions [#39]
- Improved handling of `schema_dict_util` functions with nodes away from the XML root of the file [#41]

6.1.1.30 v.0.4.2

full changelog

Added

- Added key descriptions to Plotter objects and `get_mpl_help` and `get_bokeh_help` functions in plotting modules for getting descriptions of parameters
- Two new recipes for `HDF5Reader` for bandstructures (for reading in no or specific weights besides eigenvalues)

Improvements

- MatplotlibPlotter and BokehPlotter now have a autogenerated table of descriptions and default values in the docstring (not as nicely formatted) and in the sphinx build (really nicely formatted)
- `save_format` in matplotlib plots can now be a list of formats
- Various visual improvements to band/DOS plots:
  - Bandstructure size scaling adjusted to not produce massive bands
  - Bandstrcuture spin up components are now potted on top by default
  - size/color scaling now done with respect to the maximum in the plotting region
  - DOS added spin arrows in spin-polarized case
  - DOS inverted x-axis in vertical plot (spin down now on the right side)
  - DOS added symmetric limits in DOS direction for spin-polarized plots
  - DOS default figsize flipped for vertical plots

6.1.1.31 v.0.4.1

full changelog

Bugfixes

- Fix for `plot_lattice_constant` to make it work in aiida-fleur

6.1.1.32 v.0.4.0

full changelog

Added

- Parsers for Fleur `inp.xml`/`out.xml` files. These are robust with respect to changing file versions [#3]
- General parser for `.hdf` files [#30]
- Functionality for modifying `inp.xml` files [#23]
- Fleur visualization routines `plot_fleur_dos` and `plot_fleur_bands`
- IO Module for creating/reading and manipulating `n_mmp_mat` files for LDA+U calculations in fleur [#31]
- Tool for calculating crystal field coefficients [#22]

Improvements

- Refactored parameter handling in plotting methods. Introduced `Plotter` class for consistent behaviour and easy extendability of plotting capabilities [#6]

6.1.2 Source code documentation

6.1.2.1 Visualisation and Plotting

Fleur specific Plotting

Plotting routine for fleur density of states and bandstructures

`masci_tools.vis.fleur.`**`plot_fleur_bands`**(*bandsdata*, *bandsattributes*, *spinpol=True*, *only_spin=None*, *backend=None*, *weight=None*, ***kwargs*)

Plot the bandstructure previously extracted from a *banddos.hdf* via the `HDF5Reader`

This routine expects datasets and attributes read in with a *FleurBands* recipe from `recipes` or something producing equivalent data

Parameters

- **`bandsdata`** – dataset dict produced by the *FleurBands* recipe
- **`attributes`** – attributes dict produced by the *FleurBands* recipe
- **`spinpol`** – bool, if True (default) use the plot for spin-polarized bands if the data is spin-polarized
- **`only_spin`** – optional str, if given only the specified spin components are plotted
- **`backend`** – specify which plotting library to use ('matplotlib' or 'bokeh')
- **`weight`** – str, name of the weight (without spin suffix *_up* or *_dn*) you want to emphasize

All other Kwargs are passed on to the underlying plot routines

- Matplotlib: `plot_bands()`, `plot_spinpol_bands()`
- Bokeh: `bokeh_bands()`, `bokeh_spinpol_bands()`

masci_tools.vis.fleur.**plot_fleur_bands_characterize**(*bandsdata*, *bandsattributes*, *weight_names*, *weight_colors*, *spinpol=True*, *only_spin=None*, *backend=None*, ***kwargs*)

Plot the bandstructure previously extracted from a *banddos.hdf* via the `HDF5Reader` with points colored according to the maximum weight from a selection of weights. Can be used to show what character dominates each band

This routine expects datasets and attributes read in with a *FleurBands* recipe from `recipes` or something producing equivalent data

Parameters

- **bandsdata** – dataset dict produced by the *FleurBands* recipe
- **attributes** – attributes dict produced by the *FleurBands* recipe
- **weight_names** – list of str with the names of the weights that should be considered in the characterization
- **weight_color** – list of colors associated with each weight. If spin-polarized bandstructures should be shown with different colors the list should be twice as long as the weights
- **spinpol** – bool, if True (default) use the plot for spin-polarized bands if the data is spin-polarized
- **only_spin** – optional str, if given only the specified spin components are plotted
- **backend** – specify which plotting library to use ('matplotlib' or 'bokeh')

All other Kwargs are passed on to `plot_fleur_bands()`

masci_tools.vis.fleur.**plot_fleur_dos**(*dosdata*, *attributes*, *spinpol=True*, *only_spin=None*, *multiply_by_equiv_atoms=True*, *plot_keys=None*, *show_total=True*, *show_interstitial=True*, *show_sym=False*, *show_atoms='all'*, *show_lresolved=None*, *key_mask=None*, *backend=None*, ***kwargs*)

Plot the density of states previously extracted from a *banddos.hdf* via the `HDF5Reader`

This routine expects datasets and attributes read in with the *FleurDOS* (Or related DOS modes) recipe from `recipes` or something producing equivalent data

The limits for the axes can be specified either with `x` and `y` or `energy` and `dos`. Mixing the two options is not possible

Parameters

- **dosdata** – dataset dict produced by the *FleurDOS* recipe
- **attributes** – attributes dict produced by the *FleurDOS* recipe
- **spinpol** – bool, if True (default) use the plot for spin-polarized dos if the data is spin-polarized
- **only_spin** – optional str, if given only the specified spin components are plotted
- **backend** – specify which plotting library to use ('matplotlib' or 'bokeh')

Arguments for selecting the DOS components to plot: :type plot_keys:

param plot_keys
optional str list of str, defines the labels you want to plot

All other Kwargs are passed on to the underlying plot routines

- Matplotlib: *plot_dos()*, *plot_spinpol_dos()*
- Bokeh: *bokeh_dos()*, *bokeh_spinpol_dos()*

masci_tools.vis.fleur.**sum_weights_over_atoms**(*data*, *attributes*, *atoms_to_sum*, *entry_name*)

Create sums of atom components over specified atoms. They are entered with the same suffixes as in the original data, but with the given entry_name as prefix

Parameters

- **data** – datasets dict produced by the HDF5Reader with a recipe for DOS or bandstructure
- **attributes** – attributes dict produced by the HDF5Reader with a recipe for DOS or bandstructure
- **atoms_to_sum** – list of ints for the atoms, which should be summed
- **entry_name** – str prefix to be entered for the summed entries

Returns

dict with the summed entries

KKR specific Plotting

Tool to plot a Fermi surface calculated with the qdos mode of KKR

masci_tools.vis.kkr_plot_FS_qdos.**FSqdos2D**(*p0='./'*, *s=20*, *reload_data=False*, *clrbar=True*, *atoms=None*, *ax=None*, *nosave=False*, *noalat=False*, *cmap=<matplotlib.colors.LinearSegmentedColormap object>*, *noplot=False*, *return_data=False*, *pclrmesh=False*, *logscale=True*, *ef=None*)

plotting routine for dos files

dispersionplot function for plotting KKR bandstructures (i.e. qdos) files

masci_tools.vis.kkr_plot_bandstruc_qdos.**dispersionplot**(*p0='./'*, *data_all=None*, *totonly=True*, *s=20*, *ls_ef=':'*, *lw_ef=1*, *units='eV_rel'*, *noefline=False*, *color=''*, *reload_data=False*, *clrbar=True*, *logscale=True*, *nosave=False*, *atoms=None*, *ratios=False*, *atoms2=None*, *noscale=False*, *newfig=False*, *cmap=None*, *alpha=1.0*, *qcomponent=-2*, *clims=None*, *xscale=1.0*, *raster=True*, *atoms3=None*, *alpha_reverse=False*, *return_data=False*, *xshift=0.0*, *yshift=0.0*, *plotmode='pcolor'*, *ptitle=None*, *ef=None*, *as_e_dimension=None*, *scale_alpha_data=False*, *shading='gouraud'*, *verbose=False*)

plotting routine for qdos files - dispersion (E vs. q)

masci_tools.vis.kkr_plot_bandstruc_qdos.**load_data**(*p0='./'*, *reload_data=False*, *nosave=False*, *atoms=None*, *ratios=False*, *atoms2=None*, *atoms3=None*, *ef=None*, *verbose=False*)

load the qdos data

Plotting function for KKR dos from files

masci_tools.vis.kkr_plot_dos.**dosplot**(*p0='./', totonly=True, color='', label='', marker='', lw=2, ms=5, ls='-', ls_ef=':', lw_ef=1, units='Ry', noefline=False, interpol=False, allatoms=False, onespin=False, atoms=None, lmdos=False, lm=None, nofig=False, scale=1.0, shift=0, normalized=False, xyswitch=False, efcolor='', return_data=False, xscale=1.0, xshift=0.0, yshift=0.0, filled=False, spins=2*)

plotting routine for dos files

Tool to visualize the KKR shapefunctions

masci_tools.vis.kkr_plot_shapefun.**change_zoom**(*ax, zoom_range, center=None*)

Change the zoom of a 3d plot

Author
Philipp Ruessmann

Parameters

- **ax** – axis which is zoomed
- **zoom_range** (float) – range to which the image is zoomed, total range from center-zoom_range to center+zoom_range
- **center** (Optional[list]) – center of the zoomed region (optional, defaults to origin)

Return type
None

masci_tools.vis.kkr_plot_shapefun.**plot_shapefun**(*pos, out, mode*)

Creates a simple matplotlib image to show the shapefunctions given it's positions in the unit cell, the atoms's vertices in *ut* and the plotting mode

Author
Philipp Ruessmann

Parameters

- **pos** – positions of the centers of the cells
- **verts** – array of vertices of the shapefunction (outlines of shapes)
- **mode** – 'all' or 'single' determines whether or not all shapes are combined in a single figure or plotted as individual figures

Returns ax
return the axis in which the plot was done (useful to pass to 'change_zoom' and 'zoom_in' functions of this module

masci_tools.vis.kkr_plot_shapefun.**zoom_in**(*ax, atm, pos, zoom_range=10*)

Zoom into shapefun of a single atom

Author
Philipp Ruessmann

Parameters

- **ax** – axis in which shapefun plot is found
- **atm** – atom index whose shapefunction is zoomed
- **pos** – array of positions of centers of the shapes (needed to shift center of zommed region to correct atom
- **zoom_range** – range of the zoomed region (optional, defaults to 10)

General Plotting

Here basic functionality is provided for setting default parameters for plotting and ensuring consistent values for these

masci_tools.vis.parameters.**F**

Generic Callable type

alias of TypeVar('F', bound=Callable[[...], Any])

masci_tools.vis.parameters.**NestedPlotParameters**(*plotter_object*)

Contextmanager for nested plot function calls Will reset function defaults and parameters to previous values after exiting

Parameters
plotter_object (*Plotter*) – Plotter instance

Return type
Generator[None, None, None]

class masci_tools.vis.parameters.**Plotter**(*default_parameters*, *general_keys=None*, *key_descriptions=None*, *type_kwargs_mapping=None*, *kwargs_postprocess_rename=None*, ***kwargs*)

Base class for handling parameters for plotting methods. For different plotting backends a subclass can be created to represent the specific parameters of the backend.

Parameters

- **default_parameters** (dict[str, Any]) – dict with hardcoded default parameters
- **general_keys** (Optional[set[str]]) – set of str optional, defines parameters which are not allowed to change for each entry in the plot data

Kwargs in the __init__ method are forwarded to *Plotter.set_defaults()* to change the current defaults away from the hardcoded parameters.

The Plotter class creates a hierarchy of dictionaries for lookups on this object utilizing the *ChainMap* from the *collections* module.

The hierarchy is as follows (First entries take precedence over later entries):

- *parameters*: set by *set_parameters()* (usually arguments passed into function)
- *user defaults*: set by *set_defaults()*
- *function defaults*: set by *set_defaults()* with *default_type='function'*
- *global defaults*: Hardcoded as fallback

Only the *parameters* can represent parameters for multiple sets of plot calls. All others are used as fallback for specifying non-specified values for single plots

The current parameters can be accessed by bracket indexing the class. A example of this is shown below.

```
parameter_dict = {'fontsize': 16, 'linestyle': '-'}

params = Plotter(parameter_dict)

#Accessing a parameter
print(params['fontsize']) # 16

#Modifying a parameter
```

(continues on next page)

(continued from previous page)

```
params['fontsize'] = 20
print(params['fontsize']) # 20

#Creating a parameter set for multiple plots

#1. Set the properties to the correct values
params.single_plot = False
params.num_plots = 3

#2. Now we can set a property either by providing a list or a integer indexed dict
#   Both of the following examples set the linestyle of the second and third plot␣
↪to '--'
params['linestyle'] = [None, '--', '--']
params['linestyle'] = {1: '--', 2: '--'}

# Not specified values are replaced with the default value for a single plot
print(params['linestyle']) # ['-', '--', '--']

#In lists properties can also be indexed via tuples
print(params[('linestyle', 0)]) # '-'
print(params[('linestyle', 1)]) # '--'

#Changes to the parameters and properties are reset
params.reset_parameters()

print(params['linestyle']) # '-'
```

add_parameter(*name*, *default_from=None*, *default_val=None*)

Add a new parameter to the parameters dictionary.

Parameters

- **name** (str) – str name of the parameter
- **default_from** (Optional[str]) – str (optional), if given a entry is created in the current defaults with the name and the default value of the key *default_from*

Return type
None

static convert_to_complete_list(*given_value*, *single_plot*, *num_plots*, *default=None*, *key=''*)

Converts given value to list with length num_plots with None for the non-specified values

Parameters

- **given_value** (Any) – value passed in, for multiple plots either list or dict with integer keys
- **single_plot** (bool) – bool, if True only a single parameter is allowed
- **num_plots** (int) – int, if single_plot is False this defines the number of plots
- **default** (Optional[Any]) – default value for unspecified entries
- **key** (str) – str of the key to process

Return type
Any

static dict_of_lists_to_list_of_dicts(*dict_of_lists*, *single_plot*, *num_plots*, *repeat_after=None*, *ignore_repeat=None*)

Converts dict of lists and single values to list of length num_plots or single dict for single_plot=True

Parameters

- **dict_of_lists** (dict[str, list[Any]]) – dict to be converted
- **single_plot** (bool) – boolean, if True only a single parameter set is allowed
- **num_plots** (int) – int of the number of allowed plots

Return type
list[dict[str, Any]]

Returns
list of dicts

expand_parameters(*original_length*, ***kwargs*)

Expand parameters to a bigger number of plots. New length has to be a multiple of original length. Only lists of length <= orginal_length are expanded. Also expands function defaults

Parameters

- **orginal_length** – int of the old length
- **kwargs** (Any) – arguments to expand

Return type
dict[str, Any]

Returns
expanded kwargs

get_description(*key*)

Get the description of the given key

Parameters
key (str) – str of the key, for which the description should be printed

Return type
None

get_dict()

Return the dictionary of the current defaults. For use of printing

Return type
dict[str, Any]

get_multiple_kwargs(*keys*, *ignore=None*)

Get multiple parameters and return them in a dictionary

Parameters

- **keys** (set[str]) – set of keys to process
- **ignore** (Union[str, list[str], None]) – str or list of str (optional), defines keys to ignore in the creation of the dict

Return type
dict[str, Any]

is_general(*key*)

Return, whether the key is general (meaning only related to the whole plots)

Parameters
key (str) – str of the key to check

Return type
bool

Returns
bool, whether the key is general

load_defaults(*filename='plot_defaults.json'*)

Load defaults from a json file.

Parameters
filename (Union[str, bytes, Path, PathLike, IO[Any]]) – filename,from where the defaults should be taken

Return type
None

property num_plots: int

Integer property for number of plots produced

plot_kwargs(*plot_type='default', ignore=None, extra_keys=None, post_process=True, list_of_dicts=True, **kwargs*)

Creates a dict or list of dicts (for multiple plots) with the defined parameters for the plotting calls of different types

Parameters

- **plot_type** (str) – type of plot
- **ignore** (Union[str, list[str], None]) – str or list of str (optional), defines keys to ignore in the creation of the dict
- **extra_keys** (Optional[set[str]]) – optional set for additional keys to retrieve
- **post_process** (bool) – bool, if True the parameters are cleaned up for inserting them directly into bokeh plotting functions

Return type
Any

Kwargs are used to replace values by custom parameters:

Example for using a custom markersize:

```
p = Plotter(type_kwargs_mapping={'default': {'marker'}})
p.add_parameter('marker_custom', default_from='marker')
p.plot_kwargs(marker='marker_custom')
```

This code snippet will return the standard parameters for a plot, but the value for the marker will be taken from the key *marker_custom*

remove_added_parameters()

Remove the parameters added via *Plotter.add_parameter()*

Return type
None

reset_defaults()

Resets the defaults to the hardcoded defaults in _PLOT_DEFAULTS.

Return type
None

reset_parameters()

Reset the parameters to the current defaults. The properties single_plot and num_plots are also set to default values

Return type
None

save_defaults(*filename='plot_defaults.json'*, *save_complete=False*)

Save the current defaults to a json file.

Parameters

- **filename** (Union[str, bytes, Path, PathLike, IO[Any]]) – filename, where the defaults should be stored
- **save_complete** (bool) – bool if True not only the overwritten user defaults but also the unmodified hardcoded defaults are stored

Return type
None

set_defaults(*continue_on_error=False*, *default_type='global'*, ***kwargs*)

Set the current defaults. This method will only work if the parameters are not changed from the defaults. Otherwise a error is raised. This is because after changing the defaults the changes will be propagated to the parameters to ensure consistency.

Parameters
continue_on_error (bool) – bool, if True unknown key are simply skipped

Default_type
either 'global' or 'function'. Specifies, whether to set the global defaults (not reset after function) or the function defaults

Return type
dict[str, Any]

Kwargs are used to set the defaults.

set_parameters(*continue_on_error=False*, ***kwargs*)

Set the current parameters.

Parameters
continue_on_error (bool) – bool, if True unknown key are simply skipped and returned

Return type
dict[str, Any]

Kwargs are used to set the defaults.

set_single_default(*key*, *value*, *default_type='global'*)

Set default value for a single key/value pair

Parameters

- **key** (str) – str of the key to set
- **value** (Any) – value to set the key to

Default_type
either 'global' or 'function'. Specifies, whether to set the global defaults (not reset after function) or the function defaults

Return type
None

property single_plot: bool

Boolean property if True only a single Plot parameter set is allowed

masci_tools.vis.parameters.**ensure_plotter_consistency**(*plotter_object*)

Decorator for plot functions to ensure that the Parameters are reset even if an error occurs in the function Additionally checks are performed that the parameters are reset after execution and the defaults are never changed in a plot function

Parameters
plotter_object (*Plotter*) – Plotter instance to be checked for consistency

Return type
Callable[[TypeVar(F, bound= Callable[..., Any])], TypeVar(F, bound= Callable[..., Any])]

This module contains classes and functions to make plotting functions more flexible with respect to the used data. This way plotting functions can both allow the flexible usage of lists, arrays directly or dataframes together with the keys that should be used

class masci_tools.vis.data.**ColumnDataSourceWrapper**(*wrapped*)

Wrapper around bokeh.models.ColumnDataSource to give it a __getitem__ and __setitem__ method

Used in the *PlotDataIterator* for easier handling of these types

class masci_tools.vis.data.**PlotData**(*data*, *use_column_source=False*, *same_length=False*, *strict_data_keys=False*, *copy_data=False*, ***kwargs*)

Class for iterating over the data in a dict or dataframe with automatic filling in of single defined keys to get a list of keys to extract.

The iteration allows for implicit definition of data for multiple plot sets, without excessive copying of the given data

Usage Example

```
from masci_tools.vis.data import PlotData
import numpy as np

#Let's say we have one energy grid and a couple of functions
#defined on this energy grid.
#We collect these in a dict

x = np.linspace(-10,10,100)
data = {'x': x, 'y1': np.sin(x), 'y2':np.cos(x), 'y3', x**2}

p = PlotData(data, x='x', y=['y1', 'y2', 'y3'])

#If we now iterate over this object it will result in the data
#for y being returned together with the x data (The same would work the other way␣
↪around)
for entry in p:
   print(entry.x) #'x' entry
```

(continues on next page)

(continued from previous page)

```
    print(entry.y) #'y1' then 'y2' and finally 'y3'

#Additionally data for z, color and size can be defined
```

Parameters

data – object or list of objects which can be bracket indexed with the given keys e.g. dicts, pandas dataframes, …

Same_length

bool if True and any sources are dicts it will be checked for same dimensions in (ALL) entries (not only for keys plotted against each other)

Strict_data_keys

bool if True no new data keys are allowed to be entered via `copy_data()`

Kwargs are used to specify the columns in a namedtuple If a list is given for any of the keys the data will be expanded to a list of namedtuple with the same length

add_data_key(*data_key*, *keys=None*)

Add a new column of data keys

Parameters

- **data_key** – string of the new data key to add
- **keys** – None, Index into data or list of index into the data to initialize the values to

apply(*data_key*, *lambda_func*, *apply_to_whole_array=True*, ***kwargs*)

Apply a function to a given data column for all entries

> **Warning:** This operation is done in-place. Meaning if there are multiple data entries pointing to the same data set and only one should be modified by this method, the data needs to be copied beforehand using `copy_data()`

Parameters

- **data_key** – name of the data key to apply the function
- **lambda_func** – function to apply to the data

copy_data(*data_key_from*, *data_key_to*, *prefix=None*, *rename_original=False*, *force=False*)

Copy the data for a given data key to another one

Parameters

- **data_key_from** – data key to copy from
- **data_key_to** – data key to copy to
- **prefix** – optional prefix to use for the renamed data entries. Can be used to avoid name clashes. If not given the data keys are used
- **rename_original** – optional bool (default False). If True the original entries are renamed instead of the ones under `data_key_to`

property data_keys

Return the registered data keys for this instance

distinct_datasets(*data_key*)

Return how many different data sets are present for the given data key

Parameters
data_key – The data key to analyse

Returns
int of the number of different datasets

get_function_result(*data_key*, *func*, *list_return=False*, *as_numpy_array=False*, ***kwargs*)

Apply a function to a given data column for all entries and return the results

Parameters

- **data_key** – name of the data key to apply the function to
- **func** – function to apply to the data to get the results if func is a string then it will be used to get the attribute with the corresponding name from the source and call it

get_keys(*data_key*)

Get the keys for a given data column for all entries

Parameters
data_key – name of the data key to return the keys

Returns
list of keys, corresponding to the entries for the given data in the sources

get_mask(*mask*, *data_key=None*)

Get mask list for use with the Data in this instance

Parameters

- **mask** – either list of callable, if it is callable it is used in *get_function_result()* together with the `data_key` argument
- **data_key** – str to be used for the data key if mask is a callable

:param

get_values(*data_key*)

Get the values for a given data column for all entries

Parameters
data_key – name of the data key to return the values

Returns
list of values, corresponding to the entries for the given data in the sources

group_data(*by*, ***kwargs*)

Group the data by the given data_key(s) or other arguments for groupby

Note: This function will convert the data arguments to `pd.Dataframe` objects

Parameters
by – str or list of str of the data_keys to sort by or other valid arguments for by in `pd.Dataframe.groupby()`

Kwargs are passed on to `pd.Dataframe.groupby()`

items(*first=False*, *mappable=False*)

Iterate over PlotData items. Returns the key and corresponding source for the data

Parameters

- **first** – bool, if True only the first entry is returned
- **mappable** – bool, if True only the data ColumnDataSources are wrapped to be mappable

keys(*first=False*)

Iterate over PlotData keys. Returns the keys for the corresponding sources

Parameters

first – bool, if True only the first entry is returned

mask_data(*mask*, *data_key=None*, *replace_value=None*)

Apply a given mask to the data inplace.

Note: This function will convert the data arguments to `pd.Dataframe` objects

---

Parameters

- **mask** – mask rgument passed to *get_mask()*
- **data_key** – data_key argument used by *get_mask()*

max(*data_key*, *separate=False*, *mask=None*, *mask_data_key=None*)

Get the maximum value for a given data column for all entries

Parameters

- **data_key** – name of the data key to determine the maximum
- **separate** – bool if True the maximum will be determined and returned for all entries separately
- **mask** – optional mask to select specific rows from the data entries
- **mask_data_key** – optional data key to be used when `mask` is a function

Returns

maximum value for all entries either combined or as a list

min(*data_key*, *separate=False*, *mask=None*, *mask_data_key=None*)

Get the minimum value for a given data column for all entries

Parameters

- **data_key** – name of the data key to determine the minimum
- **separate** – bool if True the minimum will be determined and returned for all entries separately
- **mask** – optional mask to select specific rows from the data entries
- **mask_data_key** – optional data key to be used when `mask` is a function

Returns

minimum value for all entries either combined or as a list

shift_data(*data_key*, *shifts*, *shifted_data_key=None*, *separate_data=True*, *negative=False*)

Apply shifts to a given data column for all entries

Parameters

- **data_key** – name of the data key to shift
- **shifts** – float or array of floats with the shifts to apply
- **shifted_data_key** – optional string, if given the data will be copied to this data key
- **separate_data** – bool, if True and shifted_data_key is not given the data will be copied to itself (This separates the data for all columns)
- **negative** – bool if True the shifts are applied with a minus sign

sort_data(*by_data_keys*, ***kwargs*)

Sort the data by the given data_key(s)

Note: This function will convert the data arguments to `pd.Dataframe` objects

Note: If there are multiple plot sets and only one data source. This function will expand the data to be one data source sorted according to the data_keys for each plot

Parameters

by_data_keys – str or list of str of the data_keys to sort by

Kwargs are passed on to `pd.Dataframe.sort_values()`

values(*first=False*)

Iterate over PlotData values. Returns the values for the data

Parameters

first – bool, if True only the first entry is returned

class masci_tools.vis.data.**PlotDataIterator**(*plot_data*, *mode='values'*, *mappable=False*)

Class containing the iteration behaviour over the `PlotData` class. Can be used in three modes:

- *keys*: Returns the keys to be entered in the corresponding data sources for each entry
- *values*: Returns the data for each entry
- *items*: Returns the keys and the data sources in a tuple

The keys and values are always returned in a `namedtuple` with fields corresponding to the set data keys

masci_tools.vis.data.**normalize_list_or_array**(*data*, *key*, *out_data*, *flatten_np=False*, *forbid_split_up=False*)

Split up a given list/numpy array or pd.Series to be used in the plotting methods

Parameters

- **data** – The (array-like) data to be normalized
- **key** – key under which to enter the new data
- **out_data** – dict containing previously normalized data
- **flatten_np** – bool, if True multidimensional numpy arrays are flattened

- **forbid_split_up** – bool, if True multidimensional arrays are not split up

The rules are the following:

- if `data` is a multidimensional array (list of lists, etc.) and it is not forbidden by the given argument the first dimension of the array is iterated over and interpreted as separate entries (if the data was previously split up into multiple sets a length check is performed)
- if `data` is a one-dimensional array and of a different length than the number of defined data sets it is added to all previously existing entries
- if `data` is a one-dimensional array and of the same length as the number of defined data sets each entry is added to the corresponding data set

Returns
list of dicts or dict containing the nomralized data

`masci_tools.vis.data.`**`process_data_arguments`**(*data=None*, *single_plot=False*, *use_column_source=False*, *flatten_np=False*, *forbid_split_up=None*, *same_length=False*, *copy_data=False*, ***kwargs*)

Initialize PlotData from np.arrays or lists of np.arrays or lists or a already given data argument, i.e. mapping

Parameters

- **data** – either None or Mapping to be used as the data in the PlotData class
- **single_plot** – bool, if True only a single dataset is allowed
- **use_column_source** – bool, if True all data arguments are converted to ColumnDataSource of bokeh
- **flatten_np** – bool, if True multidimensional numpy arrays are flattened (Only if data not given)
- **forbid_split_up** – set of keys for which not to split up multidimensional arrays
- **same_length** – bool if True and any sources are dicts it will be checked for same dimensions in (ALL) entries (not only for keys plotted against each other)
- **copy_data** – bool, if True the data argument will be copied

Kwargs define which keys belong to which data entries if data is given or they contain the data to be normalized

The following two example calls will both create a PlotData object with the same two plot data sets with the entries `x` and `y`:

```
import numpy as np

x = np.linspace(-10,10,100)
y1 = y**2
y2 = np.sin(x)

#Use a predefined data argument (a dict in this case) and the keys in the kwargs
p = process_data_arguments({'x': x, 'y1': y1, 'y2': y2}, x='x', y=['y1','y2'])

#Let the function normalize the given arrays
p = process_data_arguments=(x=x,y=[y1, y2])
```

Returns
A *`PlotData`* object corresponding to the given data

Collection of routines to be reused in plotting routines

masci_tools.vis.helpers.**exclude_points**(*plot_data*, **data_keys*, *limits*, *padding=0.1*)

Exclude points outside the given limits

Parameters

- **plot_data** (*PlotData*) – PlotData instance containing all the data for plots
- **data_keys** (str) – str of the keys to consider for excluding points
- **limits** (Optional[dict[str, tuple[float, float]]]) – dict of the set plot limits
- **padding** (float) – float, determines how far beyond the plot limits a point has to lie to be excluded (default 10%)

Return type
None

masci_tools.vis.helpers.**get_special_kpoint_ticks**(*kpoints*, *math_mode='$'*)

Process the high symmetry kpoints and ggf. replace with appropiate latex symbol

- Gamma/G is replaced with $Gamma$

Parameters

- **kpoints** (list[tuple[str, float]]) – list of tuples with label and position of the points
- **math_mode** (str) – Determines the symbol to enter math mode in latex

Return type
tuple[list[float], list[str]]

Returns
Ticks and their respective labels

masci_tools.vis.helpers.**mpl_single_line_or_area**(*axis*, *entry*, *source*, *area=False*, *area_vertical=False*, *area_enclosing_line=True*, *advance_color_cycle=False*, *area_line_alpha=1.0*, ***kwargs*)

Create a scatterplot, lineplot or area plot for the given entry on the matplotlib axis

Parameters

- **axis** – Axes to plot on
- **entry** – namedtuple of the entries to plot
- **source** – mapping containing the data to plot
- **area** – bool, if True fill_betweenx/y will be used to create an area plot
- **area_vertical** – bool, if True fill_betweeny will be used
- **area_enclosing_line** – bool if True the area plot will have another line plot around the edge
- **advance_color_cycle** – bool, if True the matplotlib color cycle will be advanced no matter what else is specified
- **area_line_alpha** – if an area plot is done this is the alpha parameter (transparency)

Kwargs are passed to the respective plotting routines

This modules provides common plotting functions dispatching to different plotting backends. At the moment the following backends are used:

- `matplotlib` ('mpl', 'matplotlib')
- `bokeh` ('bokeh')

The underlying plotting routines collected here should have the same signature for the data arguments; keyword arguments can be different.

class `masci_tools.vis.common.`**`PlotBackend`**(*value*)

Enumeration containing the possible names for each plotting backend Initialize using the *from_str()* method

At the moment the following are supported (case-insensitive)

- `matplotlib`: either *'mpl'* or *'matplotlib'*
- `bokeh`: *'bokeh'*

static default()

Return a *PlotBackend* instance corresponding to the current default backend

static from_str(*label*)

Initialize the *PlotBackend* from a given string

Parameters

label – str to use to initialize the backend if it is *None* the default is returned

Returns

PlotBackend instance corresponding to the label

`masci_tools.vis.common.`**`bands`**(*kpath*, *eigenvalues*, *backend=None*, *data=None*, ***kwargs*)

Plot the provided data for a bandstructure (non spin-polarized) Non-weighted, weighted, as a line plot or scatter plot, color-mapped or fixed colors are all possible options

Parameters

- **kpath** – data for the kpoints path (flattened to 1D)
- **eigenvalues** – data for the eigenvalues
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- `matplotlib`: *plot_bands()*
- `bokeh`: *bokeh_bands()*

Returns

Figure object for the used plotting backend

`masci_tools.vis.common.`**`dos`**(*energy_grid*, *dos_data*, *backend=None*, *data=None*, ***kwargs*)

Plot the provided data as a density of states (not spin-polarized). Can be done horizontally or vertical via the switch *xyswitch*

Parameters

- **energy_grid** – data for the energy grid of the DOS
- **dos_data** – data for all the DOS components to plot
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)

- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: *plot_dos()*
- bokeh: *bokeh_dos()*

Returns
Figure object for the used plotting backend

masci_tools.vis.common.**get_help**(*key*, *backend=None*)

Get a help string for a given parameter.

Parameters

- **key** – name of the parameter to get the parameter for
- **backend** – For which backend to get the description of the parameter

masci_tools.vis.common.**get_plotter**(*backend=None*)

Get the instance of the *Plotter* subclass used for the given plotting backend

Parameters
backend – For which backend to get the Plotter instance

masci_tools.vis.common.**line**(*xdata*, *ydata*, *backend=None*, *data=None*, ***kwargs*)

Plot the provided data as a line plot. Multiple data sets are possible

Parameters

- **xdata** – data for the x-axis
- **xdata** – data for the y-axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: *multiple_scatterplots()*
- bokeh: *bokeh_line()*

Returns
Figure object for the used plotting backend

masci_tools.vis.common.**load_defaults**(*backend=None*, *filename='plot_defaults.json'*)

Load defaults for the plot parameters from a file and set the contained defaults.

Parameters

- **backend** – For which backend to save the parameters
- **filename** – str of the filename to load the defaults from

masci_tools.vis.common.**reset_defaults**(*backend=None*)

Reset the defaults for theplot parameters to the original state.

Parameters
backend – For which backend to reset the parameters

masci_tools.vis.common.**save_defaults**(*backend=None*, *filename='plot_defaults.json'*, *save_complete=False*)

Save the defaults for the plot parameters.

Parameters

- **backend** – For which backend to save the parameters
- **filename** – str of the filename to save the defaults to
- **save_complete** – bool, if True also the hardcoded defaults are included

masci_tools.vis.common.**scatter**(*xdata*, *ydata*, *backend=None*, *data=None*, ***kwargs*)

Plot the provided data as a scatter plot. Varying size and color are possible. Multiple data sets are possible

Parameters

- **xdata** – data for the x-axis
- **xdata** – data for the y-axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: *multi_scatter_plot()*
- bokeh: *bokeh_multi_scatter()*

Returns
Figure object for the used plotting backend

masci_tools.vis.common.**set_default_backend**(*backend*)

Sets the default backend used when no explicit backend is specified.

Parameters
backend – Name of the backend to use

masci_tools.vis.common.**set_defaults**(*backend=None*, ***kwargs*)

Sets defaults for the plot parameters.

Parameters
backend – For which backend to set the parameters

The Kwargs are used to set the parameters of the specified backend

masci_tools.vis.common.**show_defaults**(*backend=None*)

Show the current set defaults for the plot parameters.

Parameters
backend – For which backend to show the parameters

masci_tools.vis.common.**spinpol_bands**(*kpath*, *eigenvalues_up*, *eigenvalues_dn*, *backend=None*, *data=None*, ***kwargs*)

Plot the provided data for a bandstructure (spin-polarized) Non-weighted, weighted, as a line plot or scatter plot, color-mapped or fixed colors are all possible options

Parameters

- **kpath** – data for the kpoints path (flattened to 1D)
- **eigenvalues_up** – data for the eigenvalues for spin-up

- **eigenvalues_dn** – data for the eigenvalues for spin-down
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: *plot_spinpol_bands()*
- bokeh: *bokeh_spinpol_bands()*

Returns
Figure object for the used plotting backend

masci_tools.vis.common.**spinpol_dos**(*energy_grid*, *dos_data_up*, *dos_data_dn*, *backend=None*, *data=None*, ***kwargs*)

Plot the provided data as a density of states (spin-polarized). Can be done horizontally or vertical via the switch *xyswitch*

Parameters

- **energy_grid** – data for the energy grid of the DOS
- **dos_data_up** – data for all the DOS components to plot for spin-up
- **dos_data_dn** – data for all the DOS components to plot for spin-down
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: *plot_spinpol_dos()*
- bokeh: *bokeh_spinpol_dos()*

Returns
Figure object for the used plotting backend

Matplotlib

This module contains a subclass of *Plotter* for the matplotlib library

class masci_tools.vis.matplotlib_plotter.**MatplotlibPlotter**(***kwargs*)

Class for plotting parameters and standard code snippets for plotting with the matplotlib backend.

Kwargs in the __init__ method are forwarded to setting default values for the instance

For specific documentation about the parameter/defaults handling refer to *Plotter*.

Below the current defined default values are shown:

Table 1: Plot Parameters

Name	Description	Default value
title_fontsize	Fontsize for the title of the figure	16

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
figure_kwargs	Arguments passed to *plt.figure* when creating the figure. Includes things like figsize, dpi, background color, ...	{'figsize': (8, 6), 'dpi': 100, 'facecolor': 'w', 'edgecolor': 'k', 'constrained_layout': True}
alpha	Float specifying the transparency of the title	1
axis_linewidth	Linewidth of the lines for the axis	1.5
use_axis_formatter	If True the labels will always not be formatted with an additive constant at the top	False
set_powerlimits	If True the threshold for switching to scientific notation is adjusted to 0,3	True
xticks	Positions of the ticks on the x axis	No Default
xticklabels	Labels for the ticks on the x-axis	No Default
yticks	Positions for the ticks on the y-axis	No Default
yticklabels	Labels for the ticks on the y-axis	No Default
invert_xaxis	If True the direction of the x-axis is inverted	False
invert_yaxis	If True the direction of the y-axis is inverted	False
color_cycle	If set this will override the default color cycle of matplotlib. Can be given as name of a colormap cycle or list of colors	No Default
color_cycle_always_advance	Always advance the color cycle even if the color was specified	False
sub_colormap	If a colormap is used this can be used to cut out a part of the colormap. For example (0.5,1.0) will only use the upper half of the colormap	No Default
linewidth	Linewidth for the plot(s)	2.0
linestyle	Linestyle for the plot(s)	-
marker	Shape of the marker to use for the plot(s)	o
markersize	Size of the markers to use in the plot(s)	4.0
color	Color to use in the plot(s)	No Default
zorder	z-position to use for the plot(s) (Is used to define fore- and background)	No Default
repeat_parameters	If set as integer the parameters for single plots (except labels) will be repeated after the given number of plots. Only implemented for multiple_scatterplots	No Default
edgecolor	Edgecolor to use in the plot(s)	No Default
facecolor	Facecolor to use in the plot(s)	No Default
plot_label	Label to use in the plot(s) for the legend	No Default
area_plot	If True fill_between(x) will be used to produce the plot(s)	False
area_vertical	Determines, whether to use fill_between or fill_betweenx for area plots	False
area_enclosing_line	If True a enclosing line will be drawn around the area	True
area_alpha	Transparency to use for the area in the area plot(s)	1.0
area_linecolor	Color for the enclosing line in the area plot(s)	No Default
plot_alpha	Transparency to use for the plot(s)	1.0
cmap	Colormap to use for scatter/pcolormesh or 3D plots	viridis
norm	If set this norm will be used to normalize data for the colormapping	No Default

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
shading	Shading to use for pcolormesh plots	gouraud
rasterized	Rasterize the pcolormesh when drawing vector graphics.	True
scale	Dict specifying the scales of the axis, e.g {'y': 'log'} will create a logarithmic scale on the y-axis	No Default
limits	Dict specifying the limits of the axis, e.g {'x': (-5,5)}	No Default
labelfontsize	Fontsize for the labels on the axis	15
lines	Dict specifying straight help-lines to draw. For example {'vertical': 0, 'horizontal': [-1,1]} will draw a vertical line at 0 and two horizontal at -1 and 1	No Default
line_options	Color, width, and more options for the help-lines	{'linestyle': '--', 'color': 'k', 'linewidth': 1.0}
font_options	Default font options that can be used for text annotations	{'family': 'serif', 'color': 'black', 'weight': 'normal', 'size': 16}
tick_paramsx	Parameters for major ticks on the x-axis (Size, fontsize, ...)	{'size': 4.0, 'width': 1.0, 'labelsize': 14, 'length': 5, 'labelrotation': 0}
tick_paramsy	Parameters for major ticks on the y-axis (Size, fontsize, ...)	{'size': 4.0, 'width': 1.0, 'labelsize': 14, 'length': 5, 'labelrotation': 0}
tick_paramsx_minor	Parameters for minor ticks on the x-axis (Size, fontsize, ...)	{'size': 2.0, 'width': 1.0, 'labelsize': 0, 'length': 2.5}
tick_paramsy_minor	Parameters for minor ticks on the y-axis (Size, fontsize, ...)	{'size': 2.0, 'width': 1.0, 'labelsize': 0, 'length': 2.5}

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
colorbar	If True and the function implements color mapping, a colorbar is shown	True
colorbar_options	Parameters for displaying the colorbar (Fontsize, ...)	{'pad': 0.05}
legend	If True a legend for the plot is shown	False
legend_show_data_labels	If True the column names from the data argument are shown if not overwritten	False
legend_remove_duplicates	If True duplicate legend labels are removed	False
legend_options	Parameters for displaying the legend (Fontsize, location, ...)	{'fontsize': ↪'large', 'linewidth': 3.0, 'loc': 'best', 'fancybox': True}
save_plots	if True the plots will be saved to file	False
save_format	Formats to save the plots to, can be single or list of formats	png
save_options	Additional options for saving the plots to file	{'transparent': ↪True}
tightlayout	If True the tight layout will be used (NOT IMPLEMENTED)	False
show	If True plt.show will be called at the end of the routine	True
save_raw_plot_data	If True the data for the plot is saved to file (NOT IMPLEMENTED)	False
raw_plot_data_format	Format in which to save the data for the plot (NOT IMPLEMENTED)	txt

draw_lines(*ax*)

Draw horizontal and vertical lines specified in the lines argument

Parameters

ax – Axes object on which to perform the operation

plot_kwargs(*plot_type='default'*, *ignore=None*, *extra_keys=None*, *post_process=True*, *list_of_dicts=True*, ***kwargs*)

Creates a dict or list of dicts (for multiple plots) with the defined parameters for the plotting calls of matplotlib

Parameters

- **ignore** – str or list of str (optional), defines keys to ignore in the creation of the dict
- **extra_keys** – optional set for additional keys to retrieve
- **post_process** – bool, if True the parameters are cleaned up for inserting them directly into matplotlib plitting functions

Kwargs are used to replace values by custom parameters:

Example for using a custom markersize:

```
p = MatplotlibPlotter()
p.add_parameter('marker_custom', default_from='marker')
p.plot_kwargs(marker='marker_custom')
```

This code snippet will return the standard parameters for a plot, but the value for the marker will be taken from the key *marker_custom*

prepare_plot(*title=None*, *xlabel=None*, *ylabel=None*, *zlabel=None*, *axis=None*, *minor=False*, *projection=None*)

Prepares the figure of a matplotlib plot, setting the labels/titles, ticks, …

Parameters

- **title** – str for the title of the figure
- **xlabel** – str for the label on the x-axis
- **ylabel** – str for the label on the y-axis
- **zlabel** – str for the label on the z-axis
- **axis** – matplotlib axes object, optional, if given the operations are performed on the object otherwise a new figure and subplot are created
- **minor** – bool, if True minor tick parameters are set
- **projection** – str, passed on to the add_subplot call

Returns
the created or modified axis object

save_plot(*saveas*)

Save the current figure or show the current figure

Parameters
saveas – str, filename for the resulting file

set_limits(*ax*)

Set limits of the axis

Parameters
ax – Axes object on which to perform the operation

set_scale(*ax*)

Set scale of the axis (for example 'log')

Parameters
ax – Axes object on which to perform the operation

show_colorbar(*ax*)

Print a colorbar for the plot

Parameters
ax – Axes object on which to perform the operation

show_legend(*ax*, *leg_elems=None*)

Print a legend for the plot

Parameters
ax – Axes object on which to perform the operation

static truncate_colormap(*cmap*, *minval=0.0*, *maxval=1.0*, *n=256*)

Cut off parts of colormap

Parameters

- **cmap** – cmap to truncate
- **minval** – minimum value of new colormap
- **maxval** – maximum value of new colormap
- **n** – number of colors in new colormap

Returns

colormap truncated to only hold colors between minval and maxval from old colormap

In this module are plot routines collected to create default plots out of certain output nodes from certain workflows with matplot lib.

Comment: Do not use any aiida methods, otherwise the methods in here can become tricky to use inside a virtual environment. Make the user extract thing out of aiida objects before hand or write something on top. Since usually parameter nodes, or files are plotted, parse a dict or filepath.

Each of the plot_methods can take keyword arguments to modify parameters of the plots There are keywords that are handled by a special class for defaults. All other arguments will be passed on to the matplotlib plotting calls

For the definition of the defaults refer to `MatplotlibPlotter`

masci_tools.vis.plot_methods.**CDF_voigt_profile**(*x*, *fwhm_g*, *fwhm_l*, *mu*)

Cumulative distribution function of a voigt profile implementation of formula found here: https://en.wikipedia.org/wiki/Voigt_profile # TODO: is there an other way then to calc 2F2? # or is there an other way to calc the integral of wofz directly, or use different error functions.

class masci_tools.vis.plot_methods.**PDF**(*pdf*, *size=(200, 200)*)

Display a PDF file inside a Jupyter notebook.

masci_tools.vis.plot_methods.**asymmetric_lorentz**(*x*, *fwhm*, *mu*, *alpha=1.0*, *beta=1.5*)

asymmetric lorentz function

L^alpha for x<=mu L^beta for x>mu See casexps LA

masci_tools.vis.plot_methods.**asymmetric_lorentz_gauss_conv**(*x*, *mu*, *fwhm_l*, *fwhm_g*, *alpha=1.0*, *beta=1.5*)

asymmetric Lorentzian with Gauss convoluted

masci_tools.vis.plot_methods.**asymmetric_lorentz_gauss_sum**(*x*, *mu*, *fwhm_l*, *fwhm_g*, *alpha=1.0*, *beta=1.5*)

asymmetric Lorentzian with Gauss convoluted

masci_tools.vis.plot_methods.**barchart**(*positions*, *heights*, ***, *width=0.35*, *xlabel='x'*, *ylabel='y'*, *title=''*, *bottom=None*, *alignment='vertical'*, *saveas='barchart'*, *bar_type='stacked'*, *axis=None*, *xerr=None*, *yerr=None*, *data=None*, *copy_data=False*, ***kwargs*)

Create a standard bar chart plot (this should be flexible enough) to do all the basic bar chart plots.

Parameters

- **positions** – arraylike data for the positions of the bars
- **heights** – arraylike data for the heights of the bars
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)

- **width** – float determines the width of the bars
- **axis** – Axes object where to add the plot
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction
- **bottom** – bottom values for the lowest end of the bars
- **bar_type** – type of the barchart plot. Either `stacked`, `grouped` or `independent`
- **alignment** – which direction the bars should be plotted (`horizontal` or `vertical`)
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *bar*

TODO: grouped barchart (meaning not stacked)

`masci_tools.vis.plot_methods.`**`colormesh_plot`**(*xdata*, *ydata*, *cdata*, *, *xlabel=''*, *ylabel=''*, *title=''*, *data=None*, *saveas='colormesh'*, *axis=None*, *copy_data=False*, ***kwargs*)

Create plot with pcolormesh

Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **cdata** – arraylike, data for the color values with a colormap
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *pcolormesh*

`masci_tools.vis.plot_methods.`**`construct_corelevel_spectrum`**(*coreleveldict*, *natom_typesdict*, *exp_references=None*, *scale_to=-1*, *fwhm_g=0.6*, *fwhm_l=0.1*, *energy_range=None*, *xspec=None*, *energy_grid=0.2*, *peakfunction='voigt'*, *alpha_l=1.0*, *beta_l=1.5*)

Constructrs a corelevel spectrum from a given corelevel dict

Params

Returns

list: [xdata_spec, ydata_spec, ydata_single_all, xdata_all, ydata_all, xdatalabel]

`masci_tools.vis.plot_methods.default_histogram(*args, **kwargs)`

Create a standard looking histogram (DEPRECATED)

`masci_tools.vis.plot_methods.doniach_sunjic(x, scale=1.0, E_0=0, gamma=1.0, alpha=0.0)`

Doniach Sunjic asymmetric peak function. tail to higher binding energies.

param x: list values to evaluate this function param scale: multiply the function with this factor param E_0: position of the peak param gamma, 'lifetime' broadening param alpha: 'asymmetry' parametera

See Doniach S. and Sunjic M., J. Phys. 4C31, 285 (1970) or http://www.casaxps.com/help_manual/line_shapes.htm

`masci_tools.vis.plot_methods.gauss_one(x, fwhm, mu)`

Returns a Lorentzian line shape at x with FWHM fwhm and mean mu

`masci_tools.vis.plot_methods.gaussian(x, fwhm, mu)`

Returns Gaussian line shape at x with FWHM fwhm and mean mu

`masci_tools.vis.plot_methods.get_mpl_help(key)`

Print the description of the given key in the matplotlib backend

Available defaults can be seen in `MatplotlibPlotter`

`masci_tools.vis.plot_methods.histogram(xdata, density=False, histtype='bar', align='mid', orientation='vertical', log=False, axis=None, title='hist', xlabel='bins', ylabel='counts', saveas='histogram', return_hist_output=False, data=None, copy_data=False, **kwargs)`

Create a standard looking histogram

Parameters

- **xdata** – arraylike, Data for the histogram
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **density** – bool, if True the histogram is normed and a normal distribution is plotted with the same mu and sigma as the data
- **histtype** – str, type of the histogram
- **align** – str, defines where the bars for the bins are aligned
- **orientation** – str, is the histogram vertical or horizontal
- **log** – bool, if True a logarithmic scale is used for the counts
- **axis** – Axes object where to add the plot
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **return_hist_output** – bool, if True the data output from hist will be returned
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *hist*

`masci_tools.vis.plot_methods.hyp2f2(a, b, z)`

Calculation of the 2F2() hypergeometric function, since it is not part of scipy with the identity 2. from here: https://en.wikipedia.org/wiki/Generalized_hypergeometric_function a, b,z array like inputs TODO: not clear to me how to do this… the identity is only useful if we mange the adjust the arguments in a way that we can use them… also maybe go for the special case we need first: 1,1,3/2;2;-z2

`masci_tools.vis.plot_methods.load_mpl_defaults(filename='plot_mpl_defaults.json')`

Load defaults for the matplotlib backend from a json file.

Parameters

filename – filename,from where the defaults should be taken

`masci_tools.vis.plot_methods.lorentzian(x, fwhm, mu)`

Returns a Lorentzian line shape at x with FWHM fwhm and mean mu

`masci_tools.vis.plot_methods.lorentzian_one(x, fwhm, mu)`

Returns a Lorentzian line shape at x with FWHM fwhm and mean mu

`masci_tools.vis.plot_methods.multi_scatter_plot(xdata, ydata, *, size_data=None, color_data=None, xlabel='', ylabel='', title='', data=None, saveas='mscatterplot', axis=None, copy_data=False, exclude_points_outside_plot_area=False, **kwargs)`

Create a scatter plot with varying marker size Info: x, y, size and color data must have the same dimensions.

Parameters

- **xdata** – str or arraylike, data for the x coordinate
- **ydata** – str or arraylike, data for the y coordinate
- **size_data** – str or arraylike, data for the markersizes (optional)
- **color_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **data** – Mapping giving the data for the plots (required if data arguments are str)
- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *scatter*

`masci_tools.vis.plot_methods.multiaxis_scatterplot(xdata, ydata, *, axes_loc, xlabel='', ylabel='', title='', num_cols=1, num_rows=1, saveas='mscatterplot', **kwargs)`

Create a scatter plot with multiple axes.

Parameters

- **xdata** – list of arraylikes, passed on to the plotting functions for each axis (x-axis)
- **ydata** – list of arraylikes, passed on to the plotting functions for each axis (y-axis)
- **axes_loc** – list of tuples of two integers, location of each axis
- **xlabel** – str or list of str, labels for the x axis
- **ylabel** – str or list of str, labels for the y-axis
- **title** – str or list of str, titles for the subplots
- **num_rows** – int, how many rows of axis are created
- **num_cols** – int, how many columns of axis are created
- **saveas** – str filename of the saved file

Special Kwargs:

param subplot_params
dict with integer keys, can contain all valid kwargs for `multiple_scatterplots()` with the integer key denoting to which subplot the changes are applied

param axes_kwargs
dict with integer keys, additional arguments to pass on to *subplot2grid* for the creation of each axis (e.g colspan, rowspan)

Other Kwargs will be passed on to all `multiple_scatterplots()` calls (If they are not overwritten by parameters in *subplot_params*).

masci_tools.vis.plot_methods.**multiple_scatterplots**(*xdata*, *ydata*, **, xlabel='', ylabel='', title='', data=None, saveas='mscatterplot', axis=None, xerr=None, yerr=None, area_curve=0, copy_data=False, exclude_points_outside_plot_area=False, **kwargs*)

Create a standard scatter plot with multiple sets of data (this should be flexible enough) to do all the basic plots.

Parameters

- **xdata** – str or arraylike, data for the x coordinate
- **ydata** – str or arraylike, data for the y coordinate
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, title of the figure
- **data** – Mapping giving the data for the plots (required if xdata and ydata are str)
- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction

- **area_curve** – if an area plot is made this arguments defines the other enclosing line defaults to 0
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib functions (*errorbar* or *fill_between*)

masci_tools.vis.plot_methods.**multiplot_moved**(*xdata*, *ydata*, *, *xlabel=''*, *ylabel=''*, *title=''*, *data=None*, *scale_move=1.0*, *min_add=0*, *saveas='mscatterplot'*, *copy_data=False*, ***kwargs*)

Plots all the scatter plots above each other. It adds an arbitrary offset to the ydata to do this and calls *multiple_scatterplots*. Therefore you might not want to show the yaxis ticks

Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **scale_move** – float, max*scale_move determines size of the shift
- **min_add** – float, minimum shift
- **saveas** – str specifying the filename (without file format)
- **copy_data** – bool, if True the data argument will be copied

Kwargs are passed on to the `multiple_scatterplots()` call

masci_tools.vis.plot_methods.**plot_bands**(*kpath*, *bands*, *, *data=None*, *size_data=None*, *color_data=None*, *special_kpoints=None*, *e_fermi=0*, *xlabel=''*, *ylabel='$E-E_F$ [eV]'*, *title=''*, *saveas='bandstructure'*, *markersize_min=0.5*, *markersize_scaling=5.0*, *scale_color=True*, *separate_bands=False*, *line_plot=False*, *band_index=None*, *copy_data=False*, ***kwargs*)

Plot the provided data for a bandstrucuture (non spin-polarized). Can be used to illustrate weights on bands via *size_data*

Parameters

- **kpath** – arraylike data for the kpoint data
- **bands** – arraylike data for the eigenvalues
- **size_data** – arraylike data the weights to emphasize (optional)
- **color_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot

- **e_fermi** – float (default 0), place the line for the fermi energy at this value
- **special_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **markersize_min** – minimum value used in scaling points for weight
- **markersize_scaling** – factor used in scaling points for weight
- **scale_color** – bool, if True (default) the weight will be additionally shown via a colormapping
- **line_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported
- **separate_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band_index** – data for which eigenvalue belongs to which band (needed for line_plot and separate_bands)
- **copy_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the *multi_scatter_plot()* call

masci_tools.vis.plot_methods.**plot_colortable**(*colors*, *title*, *sort_colors=False*, *emptycols=0*)

Plot a legend of named colors.

Reference: https://matplotlib.org/3.1.0/gallery/color/named_colors.html

Parameters

- **colors** (Dict) – a dict color_name : color_value (hex str, rgb tuple, ...)
- **title** (str) – plot title
- **sort_colors** (bool) – True: sort legend entries not by dict position, but by color hue, staturation, value.
- **emptycols** (int) –

Returns

figure

masci_tools.vis.plot_methods.**plot_convergence**(*iteration*, *distance*, *total_energy*, ***, *data=None*, *saveas_energy='energy_convergence'*, *saveas_distance='distance_convergence'*, *axis_energy=None*, *axis_distance=None*, *xlabel='Iteration'*, *ylabel_energy='Total energy difference [Htr]'*, *ylabel_distance='Distance [me/bohr^3]'*, *title_energy='Total energy difference over scf-Iterations'*, *title_distance='Convergence (log)'*, *copy_data=False*, *drop_last_iteration=False*, ***kwargs*)

Plot the total energy differences versus the scf iteration and plot the distance of the density versus iterations.

Parameters

- **iteration** – data for the number of iterations
- **distance** – data of distances
- **total_energy** – data of total energies
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)

- **xlabel** – str, label for the x-axis of both plots
- **saveas_energy** – str, filename for the energy convergence plot
- **axis_energy** – Axes object for the energy convergence plot
- **title_energy** – str, title for the energy convergence plot
- **ylabel_energy** – str, label for the y-axis for the energy convergence plot
- **saveas_distance** – str, filename for the distance plot
- **axis_distance** – Axes object for the distance plot
- **title_distance** – str, title for the distance plot
- **ylabel_distance** – str, label for the y-axis for the distance plot
- **copy_data** – bool if True the data argument is copied
- **drop_last_iteration** – bool if True the last iteration is dropped for the distance plot

Other Kwargs will be passed on to all `multiple_scatterplots()` calls

masci_tools.vis.plot_methods.**plot_convergence_results**(*iteration*, *distance*, *total_energy*, *, *saveas1='t_energy_convergence'*, *axis1=None*, *saveas2='distance_convergence'*, *axis2=None*, ***kwargs*)

DEPRECATED Plot the total energy versus the scf iteration and plot the distance of the density versus iterations.

Parameters

- **iteration** – array for the number of iterations
- **distance** – array of distances
- **total_energy** – array of total energies
- **saveas1** – str, filename for the energy convergence plot
- **axis1** – Axes object for the energy convergence plot
- **saveas2** – str, filename for the distance plot
- **axis2** – Axes object for the distance plot

Other Kwargs will be passed on to all `single_scatterplot()` calls

masci_tools.vis.plot_methods.**plot_convergence_results_m**(*iterations*, *distances*, *total_energies*, *, *modes*, *nodes=None*, *saveas1='t_energy_convergence'*, *saveas2='distance_convergence'*, *axis1=None*, *axis2=None*, ***kwargs*)

DEPRECATED Plot the total energy versus the scf iteration and plot the distance of the density versus iterations.

Parameters

- **iterations** – array for the number of iterations
- **distances** – array of distances
- **total_energies** – array of total energies
- **modes** – list of convergence modes (if 'force' is in the list the last distance is removed)
- **saveas1** – str, filename for the energy convergence plot
- **axis1** – Axes object for the energy convergence plot

- **saveas2** – str, filename for the distance plot
- **axis2** – Axes object for the distance plot

Other Kwargs will be passed on to all `multiple_scatterplots()` calls

`masci_tools.vis.plot_methods.`**`plot_convex_hull2d`**(*hull*, **, title='Convex Hull', xlabel='Atomic Percentage', ylabel='Formation energy / atom [eV]', saveas='convex_hull', axis=None, **kwargs*)

Plot method for a 2d convex hull diagram

Parameters

- **hull** – pyhull.Convexhull #scipy.spatial.ConvexHull
- **axis** – Axes object where to add the plot
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot

Function specific parameters:

param marker_hull
defaults to *marker*, marker type for the hull plot

param markersize_hull
defaults to *markersize*, markersize for the hull plot

param color_hull
defaults to *color*, color for the hull plot

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib functions *plot*

`masci_tools.vis.plot_methods.`**`plot_corelevel_spectra`**(*coreleveldict, natom_typesdict, exp_references=None, scale_to=-1, show_single=True, show_ref=True, energy_range=None, title='', fwhm_g=0.6, fwhm_l=0.1, energy_grid=0.2, peakfunction='voigt', linestyle_spec='-', marker_spec='o', color_spec='k', color_single='g', xlabel='Binding energy [eV]', ylabel='Intensity [arb] (natoms*nelectrons)', saveas=None, xspec=None, alpha_l=1.0, beta_l=1.0, **kwargs*)

Plotting function of corelevel in the form of a spectrum.

Convention: Binding energies are positive!

Args:
coreleveldict: dict of corelevels with a list of corelevel energy of atomstypes # (The given corelevel accounts for a weight (number of electrons for full occupied corelevel) in the plot.) natom_typesdict: dict with number of atom types for each entry

Kwargs:
exp_references: dict with experimental references, will be ploted as vertical lines show_single (bool): plot

all single peaks. scale_to float: the maximum 'intensity' will be scaled to this value (useful for experimental comparisons) title (string): something for labeling fwhm (float): full width half maximum of peaks (gauss, lorentz or voigt_profile) energy_grid (float): energy resolution linetyp_spec : linetype for spectrum peakfunction (string): what the peakfunction should be {'voigt', 'pseudo-voigt', 'lorentz', 'gaus'}

example:
corelevelddict = {u'Be': {'1s1/2' : [-1.0220669053033051, -0.3185614920138805,-0.7924091040092139]}} n_atom_types_Be12Ti = {'Be' : [4,4,4]}

`masci_tools.vis.plot_methods.`**`plot_corelevels`**(*coreleveldict*, *compound=''*, *axis=None*, *saveas='scatterplot'*, ***kwargs*)

Plotting function to visualize corelevels and corelevel shifts

`masci_tools.vis.plot_methods.`**`plot_dos`**(*energy_grid*, *dos_data*, ***, *data=None*, *saveas='dos_plot'*, *energy_label='$E-E_F$ [eV]'*, *dos_label='DOS [1/eV]'*, *title='Density of states'*, *xyswitch=False*, *e_fermi=0*, *copy_data=False*, ***kwargs*)

Plot the provided data for a density of states (not spin-polarized). Can be done horizontally or vertical via the switch *xyswitch*

Parameters

- **energy_grid** – arraylike data for the energy grid of the DOS
- **dos_data** – arraylike data for all the DOS components to plot
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **energy_label** – str, label for the energy-axis
- **dos_label** – str, label for the DOS-axis
- **saveas** – str, filename for the saved plot
- **e_fermi** – float (default 0), place the line for the fermi energy at this value
- **xyswitch** – bool if True, the enrgy axis will be plotted vertically
- **copy_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `multiple_scatterplots()` call

`masci_tools.vis.plot_methods.`**`plot_lattice_constant`**(*scaling*, *total_energy*, ***, *fit_data=None*, *data=None*, *relative=True*, *ref_const=None*, *title='Equation of states'*, *saveas='lattice_constant'*, *axis=None*, *copy_data=False*, ***kwargs*)

Plot a lattice constant versus Total energy Plot also the fit. On the x axis is the scaling, it

Parameters

- **scaling** – arraylike, data for the scaling factor
- **total_energy** – arraylike, data for the total energy
- **fit_data** – arraylike, optional data of fitted data
- **relative** – bool, scaling factor given (True), or lattice constants given?
- **ref_const** – float (optional), or list of floats, lattice constant for scaling 1.0
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)

- **copy_data** – bool if True the data argument will be copied

Function specific parameters:

param marker_fit
defaults to *marker*, marker type for the fit data

param markersize_fit
defaults to *markersize*, markersize for the fit data

param linewidth_fit
defaults to *linewidth*, linewidth for the fit data

param plotlabel_fit
str label for the fit data

Other Kwargs will be passed on to *multiple_scatterplots()*

masci_tools.vis.plot_methods.**plot_one_element_corelv**(*corelevel_dict*, *element*, *compound=''*, *axis=None*, *saveas='scatterplot'*, ***kwargs*)

This routine creates a plot which visualizes all the binding energies of one element (and currently one corelevel) for different atomtypes.

example:
corelevels = { 'W' : { '4f7/2' : [123, 123.3, 123.4 ,123.1], '4f5/2' : [103, 103.3, 103.4, 103.1]}, 'Be' : { '1s': [118, 118.2, 118.4, 118.1, 118.3]}}

masci_tools.vis.plot_methods.**plot_relaxation_results**()

Plot from the result node of a relaxation workflow, All forces of every atom type versus relaxation cycle. Average force of all atom types versus relaxation cycle. Absolute relaxation in Angstroem of every atom type. Relative realxation of every atom type to a reference structure. (if none given use the structure from first relaxation cycle as reference)

masci_tools.vis.plot_methods.**plot_residuen**(*xdata*, *fitdata*, *realdata*, ***, *errors=None*, *xlabel='Energy [eV]'*, *ylabel='cts/s [arb]'*, *title='Residuen'*, *saveas='residuen'*, *hist=True*, *return_residuen_data=True*, ***kwargs*)

Calculates and plots the residuen for given xdata fit results and the real data.

If hist=True also the normed residual distribution is ploted with a normal distribution.

Parameters

- **xdata** – arraylike data for the x-coordinate
- **fitdata** – arraylike fitted data for the y-coordinate
- **realdata** – arraylike data to plot residuen against the fit
- **errors** – dict, can be used to provide errordata for the x and y direction
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **title** – str, title for the plot
- **saveas** – str, filename for the saved plot
- **hist** – bool, if True a normed residual distribution is ploted with a normal distribution.
- **return_residuen_data** – bool, if True in addition to the produced axis object also the residuen data is returned

Special Kwargs:

param hist_kwargs
dict, these arguments will be passed on to the `histogram()` call (if hist=True)

Other Kwargs will be passed on to all `single_scatterplot()` call

masci_tools.vis.plot_methods.**plot_spectral_function**(*kpath*, *energy_grid*, *spectral_function*, *, *data=None*, *special_kpoints=None*, *e_fermi=0*, *xlabel=''*, *ylabel='$E-E_F$ [eV]'*, *title=''*, *saveas='spectral_function'*, *copy_data=False*, ***kwargs*)

Create a colormesh plot of a spectral function

Parameters

- **kpath** – data for the kpoint coordinates
- **energy_grid** – data for the energy grid
- **spectral_function** – 2D data for the spectral function
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **e_fermi** – float (default 0), place the line for the fermi energy at this value
- **special_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **copy_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `colormesh_plot()` call

masci_tools.vis.plot_methods.**plot_spinpol_bands**(*kpath*, *bands_up*, *bands_dn*, *, *size_data=None*, *color_data=None*, *data=None*, *show_spin_pol=True*, *special_kpoints=None*, *e_fermi=0*, *xlabel=''*, *ylabel='$E-E_F$ [eV]'*, *title=''*, *saveas='bandstructure'*, *markersize_min=0.5*, *markersize_scaling=5.0*, *scale_color=True*, *line_plot=False*, *separate_bands=False*, *band_index=None*, *copy_data=False*, ***kwargs*)

Plot the provided data for a bandstrucuture (spin-polarized). Can be used to illustrate weights on bands via *size_data*

Parameters

- **kpath** – arraylike data for the kpoint data
- **bands_up** – arraylike data for the eigenvalues (spin-up)
- **bands_dn** – arraylike data for the eigenvalues (spin-dn)
- **size_data** – arraylike data the weights to emphasize BOTH SPINS (optional)
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot

- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **e_fermi** – float (default 0), place the line for the fermi energy at this value
- **special_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **markersize_min** – minimum value used in scaling points for weight
- **markersize_scaling** – factor used in scaling points for weight
- **show_spin_pol** – bool, if True (default) the two different spin channels will be shown in blue and red by default
- **scale_color** – bool, if True (default) the weight will be additionally shown via a colormapping
- **line_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported
- **separate_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band_index** – data for which eigenvalue belongs to which band (needed for line_plot and separate_bands)
- **copy_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `multi_scatter_plot()` call

masci_tools.vis.plot_methods.**plot_spinpol_dos**(*energy_grid*, *spin_up_data*, *spin_dn_data*, **, data=None*, *saveas='spinpol_dos_plot'*, *energy_label='$E-E_F$ [eV]'*, *dos_label='DOS [1/eV]'*, *title='Density of states'*, *xyswitch=False*, *e_fermi=0*, *spin_dn_negative=True*, *spin_arrows=True*, *copy_data=False*, ***kwargs*)

Plot the provided data for a density of states (spin-polarized). Can be done horizontally or vertical via the switch *xyswitch*

Parameters

- **energy_grid** – arraylike data for the energy grid of the DOS
- **spin_up_data** – arraylike data for all the DOS spin-up components to plot
- **spin_dn_data** – arraylike data for all the DOS spin-down components to plot
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **energy_label** – str, label for the energy-axis
- **dos_label** – str, label for the DOS-axis
- **saveas** – str, filename for the saved plot
- **e_fermi** – float (default 0), place the line for the fermi energy at this value
- **xyswitch** – bool if True, the enrgy axis will be plotted vertically
- **energy_grid_dn** – arraylike data for the energy grid of the DOS of the spin-down component (optional)

- **spin_dn_negative** – bool, if True (default) the spin-down components are plotted downwards
- **spin_arrows** – bool, if True (default) small arrows will be plotted on the left side of the plot indicating the spin directions (if spin_dn_negative is True)
- **copy_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `multiple_scatterplots()` call

masci_tools.vis.plot_methods.**pseudo_voigt_profile**(*x*, *fwhm_g*, *fwhm_l*, *mu*, *mix=0.5*)

Linear combination of gaussian and loretzian instead of convolution

Args:

x: array of floats fwhm_g: FWHM of gaussian fwhm_l: FWHM of Lorentzian mu: Mean mix: ratio of gauss to lorentz, mix* gauss, (1-mix)*Lorentz

masci_tools.vis.plot_methods.**reset_mpl_plot_defaults**()

Reset the defaults for matplotib backend to the hardcoded defaults

Available defaults can be seen in `MatplotlibPlotter`

masci_tools.vis.plot_methods.**save_mpl_defaults**(*filename='plot_mpl_defaults.json'*, *save_complete=False*)

Save the current defaults for the matplotlib backend to a json file.

Parameters

- **filename** – filename, where the defaults should be stored
- **save_complete** – bool if True not only the overwritten user defaults but also the unmodified hardcoded defaults are stored

masci_tools.vis.plot_methods.**set_mpl_plot_defaults**(***kwargs*)

Set defaults for matplotib backend according to the given keyword arguments

Available defaults can be seen in `MatplotlibPlotter`

masci_tools.vis.plot_methods.**show_mpl_plot_defaults**()

Show the currently set defaults for matplotib backend to the hardcoded defaults

Available defaults can be seen in `MatplotlibPlotter`

masci_tools.vis.plot_methods.**single_scatterplot**(*xdata*, *ydata*, ***, *xlabel=''*, *ylabel=''*, *title=''*, *data=None*, *saveas='scatterplot'*, *axis=None*, *xerr=None*, *yerr=None*, *area_curve=0*, *copy_data=False*, ***kwargs*)

Create a standard scatter plot (this should be flexible enough) to do all the basic plots.

Parameters

- **xdata** – str or arraylike, data for the x coordinate
- **ydata** – str or arraylike, data for the y coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **data** – Mapping giving the data for the plot (required if xdata and ydata are str)

- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction
- **area_curve** – if an area plot is made this arguments defines the other enclosing line defaults to 0
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib functions (*errorbar* or *fill_between*)

`masci_tools.vis.plot_methods.`**`surface_plot`**(*xdata*, *ydata*, *zdata*, *, *xlabel=''*, *ylabel=''*, *zlabel=''*, *title=''*, *data=None*, *saveas='surface_plot'*, *axis=None*, *copy_data=False*, ***kwargs*)

Create a standard surface plot

Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **zdata** – arraylike, data for the z coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **zlabel** – str, label written on the z axis
- **title** – str, title of the figure
- **axis** – Axes object, if given the plot will be applied to this object
- **saveas** – str specifying the filename (without file format)
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *plot_surface*

`masci_tools.vis.plot_methods.`**`voigt_profile`**(*x*, *fwhm_g*, *fwhm_l*, *mu*)

Return the Voigt line shape at x with Lorentzian component FWHM fwhm_l and Gaussian component FWHM fwhm_g and mean mu. There is no closed form for the Voigt profile, but it is related to the real part of the Faddeeva function (wofz), which is used here.

`masci_tools.vis.plot_methods.`**`waterfall_plot`**(*xdata*, *ydata*, *zdata*, *, *xlabel=''*, *ylabel=''*, *zlabel=''*, *title=''*, *data=None*, *saveas='waterfallplot'*, *axis=None*, *copy_data=False*, ***kwargs*)

Create a standard waterfall plot

Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **zdata** – arraylike, data for the z coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)

- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **zlabel** – str, label written on the z axis
- **title** – str, title of the figure
- **axis** – Axes object, if given the plot will be applied to this object
- **saveas** – str specifying the filename (without file format)
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *scatter3D*

Bokeh

Here the `masci_tools.vis.parameters.Plotter` subclass for the bokeh plotting backend is defined with default values and many helper methods

class masci_tools.vis.bokeh_plotter.**BokehPlotter**(***kwargs*)

Class for plotting parameters and standard code snippets for plotting with the bokeh backend.

Kwargs in the __init__ method are forwarded to setting default values for the instance

For specific documentation about the parameter/defaults handling refer to `Plotter`.

Below the current defined default values are shown:

Table 2: Plot Parameters

Name	Description	Default value
`figure_kwargs`	Parameters for creating the bokeh figure. Includes things like axis type (x and y), tools plot width/height	`{'tools': 'pan,poly_select,tap,wheel_zoom,box_zoom,redo,undo,reset,save,crosshair,zoom_out,zoom_in', 'y_axis_type': 'linear', 'x_axis_type': 'linear', 'active_inspect': None, 'toolbar_location': 'right'}`
`additional_tools`	tools to add to the tools already specified in `figure_kwargs` (Has to be in the same format)	No Default
`show_tooltips`	Switch whether to add hover tooltips	`True`
`global_tooltips`	Switch whether to add individual (for each renderer) or global hover tooltips.	`False`
`format_tooltips`	Switch whether to enable the processing of formatted strings in tooltips.	`True`

continues on next page

Table 2 – continued from previous page

Name	Description	Default value
tooltips	List of tuples specifying the tooltips. For more information refer to the bokeh documentation. Strings can contain format specifiers with the data keys of the function e.g. '@ ``{x}`. Here the `` ``{x}`` will be replaced by the entry for x. If there are formatting specifications for bokeh they need to be escaped with double curly braces or format_tooltips=False.	[('X', '@{x}'), ('Y', '@{y}')]
additional_tooltips	Tooltips to add to the already defined tooltips (See above)	No Default
axis_linewidth	Linewidth for the lines of the axis	2
label_fontsize	Fontsize for the labels of the axis	18pt
tick_label_fontsize	fontsize for the ticks on the axis	16pt
background_fill_color	Color of the background of the plot	#ffffff
x_axis_formatter	If set this formatter will be used for the ticks on the x-axis	No Default
y_axis_formatter	If set this formatter will be used for the ticks on the y-axis	No Default
x_ticks	Tick specification for the x-axis	No Default
x_ticklabels_overrides	Overrides the labels for the ticks on the x-axis	No Default
y_ticks	Tick specification for the y-axis	No Default
y_ticklabels_overrides	Overrides the labels for the ticks on the y-axis	No Default
x_range_padding	Specifies the amount of padding on the edges of the x-axis	No Default
y_range_padding	Specifies the amount of padding on the edges of the y-axis	No Default
limits	Dict specifying the limits of the axis, e.g {'x': (-5,5)}	No Default
legend_location	Location of the legend inside the plot area	top_right
legend_click_policy	Policy for what happens when labels are clicked in the legend	hide
legend_orientation	Orientation of the legend	vertical
legend_font_size	Fontsize for the labels inside the legend	14pt
legend_outside	If True the legend will be placed outside of the plot area	False
color_palette	Color palette to use for the plot(s)	No Default
color	Specific colors to use for the plot(s)	No Default
legend_label	Labels to use for the legend of the plot(s)	No Default
alpha	Transparency to use for the plot(s)	1.0
name	Name used for identifying elements in the plot (not shown only internally)	No Default
line_color	Color to use for line plot(s)	No Default
line_alpha	Transparency to use for line plot(s)	1.0
line_dash	Dash styles to use for line plot(s)	No Default
line_width	Line width to use for line plot(s)	2.0
marker	Type of marker to use for scatter plot(s)	circle
marker_size	Marker size to use for scatter plot(s)	6
area_plot	If True h(v)area will be used to produce the plot(s)	False
area_vertical	Determines, whether to use harea (False) or varea (True) for area plots	False
fill_alpha	Transparency to use for the area in area plot(s)	1.0
fill_color	Color to use for the area in area plot(s)	No Default
level	Can be used to specified, which elements are fore- or background	No Default
straight_lines	Dict specifying straight help-lines to draw. For example {'vertical': 0, 'horizontal': [-1,1]} will draw a vertical line at 0 and two horizontal at -1 and 1	No Default

continues on next page

Table 2 – continued from previous page

Name	Description	Default value
straight_lines_opt	Color, width, and more options for the help-lines	{'line_color': 'black', 'line_width': 1.0, 'line_dash': 'dashed'}
text_font_size	fontsize for the text glyphs in the plot	10pt
text_font_style	fontstyle for the text glyphs in the plot	normal
text_color	text color for the text glyphs in the plot	black
text_align	text alignment for the text glyphs in the plot	left
text_baseline	text baseline for the text glyphs in the plot	middle
save_plots	If True plots will be saved to file (Configuration beforehand is needed)	False
save_format	Formats to save the plots to, can be single or list of formats (html, png or svg)	html
show	If True bokeh.io.show will be called after the plotting routine	True

add_tooltips(*fig*, *renderers*, *columns=None*, *toggleable=False*)

Add Hover tooltips to the given renderers and figure

Parameters

- **fig** – bokeh figure to apply changes to
- **renderers** – bokeh renderers to activate the tooltips for
- **columns** – namedtuple containing the data keys used for evtl. formatting
- **toggleable** – bool, if True these tooltips will be toggleable in the toolbar

draw_straight_lines(*fig*)

Draw horizontal and vertical lines specified in the lines argument

Parameters

fig – bokeh figure on which to perform the operation

prepare_figure(*title*, *xlabel*, *ylabel*, *figure=None*)

Create a bokeh figure according to the set parameters or modify an existing one

Parameters

- **title** – title of the figure
- **xlabel** – label on the x-axis
- **ylabel** – label on the y-axis
- **figure** – bokeh figure, optional, if given the operations are performed on the object otherwise a new figure is created

Returns

the created or modified bokeh figure

save_plot(*figure*, *saveas*)

Show/save the bokeh figure

Parameters

figure – bokeh figure on which to perform the operation

set_color_palette_by_num_plots()

Set the colormap for the configured number of plots according to the set colormap or color

copied from https://github.com/PatrikHlobil/Pandas-Bokeh/blob/master/pandas_bokeh/plot.py credits to PatrikHlobil modified for use in this Plotter class

set_legend(*fig*)

Set legend options for the figure

Parameters
fig – bokeh figure on which to perform the operation

set_limits(*fig*)

Set limits of the figure

Parameters
fig – bokeh figure on which to perform the operation

Here are general and special bokeh plots to use

masci_tools.vis.bokeh_plots.**bokeh_bands**(*kpath*, *bands=None*, *, *data=None*, *size_data=None*, *color_data=None*, *xlabel=''*, *ylabel='$$E-E_F [eV]$$'*, *title=''*, *special_kpoints=None*, *markersize_min=3.0*, *markersize_scaling=10.0*, *saveas='bands_plot'*, *scale_color=True*, *separate_bands=False*, *line_plot=False*, *band_index=None*, *copy_data=False*, ***kwargs*)

Create an interactive bandstructure plot (non-spinpolarized) with bokeh Can make a simple plot or weight the size and color of the points against a given weight

Parameters

- **kpath** – arraylike or key data for the kpoint data
- **bands** – arraylike or key data for the eigenvalues
- **size_data** – arraylike or key data the weights to emphasize (optional)
- **color_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the bands data (optional) of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis (default no label)
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **special_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **e_fermi** – float, determines, where to put the line for the fermi energy
- **markersize_min** – minimum value used in scaling points for weight
- **markersize_scaling** – factor used in scaling points for weight
- **outfilename** – filename of the output file
- **scale_color** – bool, if True (default) the weight will be additionally shown via a colormapping
- **line_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported

- **separate_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band_index** – data for which eigenvalue belongs to which band (needed for line_plot and separate_bands)
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to *bokeh_multi_scatter()* or *bokeh_line()*

masci_tools.vis.bokeh_plots.**bokeh_dos**(*energy_grid*, *dos_data=None*, **, data=None*, *energy_label='$$E-E_F [eV]$$'*, *dos_label='DOS [1/eV]'*, *title='Density of states'*, *xyswitch=False*, *e_fermi=0*, *saveas='dos_plot'*, *copy_data=False*, ***kwargs*)

Create an interactive dos plot (non-spinpolarized) with bokeh Both horizontal or vertical orientation are possible

Parameters

- **energy_grid** – arraylike or key data for the energy grid
- **spin_up_data** – arraylike or key data for the DOS
- **data** – source for the DOS data (optional) of the plot (pandas Dataframe for example)
- **energy_label** – label for the energy-axis
- **dos_label** – label for the dos-axis
- **title** – title of the figure
- **xyswitch** – bool if True, the energy will be plotted along the y-direction
- **e_fermi** – float, determines, where to put the line for the fermi energy
- **outfilename** – filename of the output file
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to *bokeh_line()*

masci_tools.vis.bokeh_plots.**bokeh_line**(*x*, *y=None*, **, data=None*, *figure=None*, *xlabel='x'*, *ylabel='y'*, *title=''*, *saveas='line'*, *plot_points=False*, *area_curve=0*, *copy_data=False*, *set_default_legend=True*, ***kwargs*)

Create an interactive multi-line plot with bokeh

Parameters

- **x** – arraylike or key for data for the x-axis
- **y** – arraylike or key for data for the y-axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – label for the x-axis
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure
- **outfilename** – filename of the output file
- **plot_points** – bool, if True also plot the points with a scatterplot on top
- **copy_data** – bool, if True the data argument will be copied

- **set_default_legend** – bool if True the data names are used to generate default legend labels

Kwargs will be passed on to `masci_tools.vis.bokeh_plotter.BokehPlotter`. If the arguments are not recognized they are passed on to the bokeh function *line*

masci_tools.vis.bokeh_plots.**bokeh_multi_scatter**(*x*, *y=None*, ***, *data=None*, *figure=None*, *xlabel='x'*, *ylabel='y'*, *title=''*, *saveas='scatter'*, *copy_data=False*, *set_default_legend=True*, ***kwargs*)

Create an interactive scatter (muliple data sets possible) plot with bokeh

Parameters

- **x** – arraylike or key for data for the x-axis
- **y** – arraylike or key for data for the y-axis
- **data** – source for the data of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure
- **outfilename** – filename of the output file
- **copy_data** – bool, if True the data argument will be copied
- **set_default_legend** – bool if True the data names are used to generate default legend labels

Kwargs will be passed on to `masci_tools.vis.bokeh_plotter.BokehPlotter`. If the arguments are not recognized they are passed on to the bokeh function *scatter*

masci_tools.vis.bokeh_plots.**bokeh_scatter**(*x*, *y=None*, ***, *xlabel='x'*, *ylabel='y'*, *title=''*, *figure=None*, *data=None*, *saveas='scatter'*, *copy_data=False*, ***kwargs*)

Create an interactive scatter plot with bokeh

Parameters

- **x** – arraylike or key for data for the x-axis
- **y** – arraylike or key for data for the y-axis
- **data** – source for the data of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure
- **outfilename** – filename of the output file
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.bokeh_plotter.BokehPlotter`. If the arguments are not recognized they are passed on to the bokeh function *scatter*

masci_tools.vis.bokeh_plots.**bokeh_spectral_function**(*kpath*, *energy_grid*, *spectral_function*, *, *data=None*, *special_kpoints=None*, *e_fermi=0*, *xlabel=''*, *ylabel='$$E-E_F [eV]$$'*, *title=''*, *saveas='spectral_function'*, *copy_data=False*, *figure=None*, ***kwargs*)

Create a colormesh plot of a spectral function

Parameters

- **kpath** – data for the kpoint coordinates
- **energy_grid** – data for the energy grid
- **spectral_function** – 2D data for the spectral function
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **e_fermi** – float (default 0), place the line for the fermi energy at this value
- **special_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **copy_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the image call of bokeh

masci_tools.vis.bokeh_plots.**bokeh_spinpol_bands**(*kpath*, *bands_up=None*, *bands_dn=None*, *, *size_data=None*, *color_data=None*, *data=None*, *xlabel=''*, *ylabel='$$E-E_F [eV]$$'*, *title=''*, *special_kpoints=None*, *markersize_min=3.0*, *markersize_scaling=10.0*, *saveas='bands_plot'*, *scale_color=True*, *line_plot=False*, *separate_bands=False*, *band_index=None*, *copy_data=False*, ***kwargs*)

Create an interactive bandstructure plot (spinpolarized) with bokeh Can make a simple plot or weight the size and color of the points against a given weight

Parameters

- **kpath** – arraylike or key data for the kpoint data
- **bands_up** – arraylike or key data for the eigenvalues spin-up
- **bands_dn** – arraylike or key data for the eigenvalues spin-dn
- **size_data** – arraylike or key data the weights to emphasize (optional)
- **color_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the bands data (optional) of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis (default no label)
- **ylabel** – label for the y-axis
- **title** – title of the figure

- **special_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **e_fermi** – float, determines, where to put the line for the fermi energy
- **markersize_min** – minimum value used in scaling points for weight
- **markersize_scaling** – factor used in scaling points for weight
- **outfilename** – filename of the output file
- **scale_color** – bool, if True (default) the weight will be additionally shown via a colormapping
- **line_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported
- **separate_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band_index** – data for which eigenvalue belongs to which band (needed for line_plot and separate_bands)
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to *bokeh_multi_scatter()* or *bokeh_line()*

masci_tools.vis.bokeh_plots.**bokeh_spinpol_dos**(*energy_grid*, *spin_up_data=None*, *spin_dn_data=None*, **, data=None*, *spin_dn_negative=True*, *energy_label='$$E-E_F [eV]$$'*, *dos_label='DOS [1/eV]'*, *title='Density of states'*, *xyswitch=False*, *e_fermi=0*, *spin_arrows=True*, *saveas='dos_plot'*, *copy_data=False*, ***kwargs*)

Create an interactive dos plot (spinpolarized) with bokeh Both horizontal or vertical orientation are possible

Parameters

- **energy_grid** – arraylike or key data for the energy grid
- **spin_up_data** – arraylike or key data for the DOS spin-up
- **spin_dn_data** – arraylike or key data for the DOS spin-dn
- **data** – source for the DOS data (optional) of the plot (pandas Dataframe for example)
- **spin_dn_negative** – bool, if True (default), the spin down components are plotted downwards
- **energy_label** – label for the energy-axis
- **dos_label** – label for the dos-axis
- **title** – title of the figure
- **xyswitch** – bool if True, the energy will be plotted along the y-direction
- **e_fermi** – float, determines, where to put the line for the fermi energy
- **spin_arrows** – bool, if True (default) small arrows will be plotted on the left side of the plot indicating the spin directions (if spin_dn_negative is True)
- **outfilename** – filename of the output file
- **copy_data** – bool, if True the data argument will be copied

Kwargs will be passed on to *bokeh_line()*

masci_tools.vis.bokeh_plots.**get_bokeh_help**(*key*)

Print the description of the given key in the bokeh backend

Available defaults can be seen in *BokehPlotter*

masci_tools.vis.bokeh_plots.**load_bokeh_defaults**(*filename='plot_bokeh_defaults.json'*)

Load defaults for the bokeh backend from a json file.

Parameters

filename – filename,from where the defaults should be taken

masci_tools.vis.bokeh_plots.**matrix_plot**(*text_values*, *x_axis_data*, *y_axis_data*, *positions=None*, *, *color_data=None*, *secondary_color_data=None*, *x_offset=-0.47*, *log_scale=False*, *color_map=None*, *data=None*, *copy_data=False*, *title=''*, *xlabel='x'*, *ylabel='y'*, *saveas='matrix_plot.html'*, *blank_outsiders='both'*, *blank_color='#c4c4c4'*, *figure=None*, *categorical_axis=False*, *categorical_sort_key=None*, *block_size=0.95*, *block_size_pixel=100*, ***kwargs*)

Plot function for an interactive periodic table plot. Heat map and hover tool. source must be a pandas dataframe containing, atom period and group, atomic number and symbol

Parameters

- **values** – data for the text inside each elements box
- **positions** – y positions relative to the middle of the box for each value
- **color_data** – data to display as a heatmap
- **color_map** – color palette to use for the heatmap (default matplotlib plasma)
- **log_scale** – bool, if True the heatmap is done logarithmically
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **saveas** – str, filename for the saved plot
- **blank_outsiders** – either 'both', 'min', 'max' or None, determines, which points outside the color range to color with a default blank color
- **blank_color** – color to replace values outside the color range by
- **include_legend** – if True an additional entry with labels explaing each value entry is added
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure

Additional kwargs are passed on to the label creation for the element box The kwargs *legend_options* and *colorbar_options* can be used to overwrite default values for these regions of the plot

masci_tools.vis.bokeh_plots.**periodic_table_plot**(*values*, *positions=None*, *, *color_data=None*, *log_scale=False*, *color_map=None*, *data=None*, *copy_data=False*, *title=''*, *saveas='periodictable.html'*, *blank_outsiders='both'*, *blank_color='#c4c4c4'*, *include_legend=True*, *figure=None*, ***kwargs*)

Plot function for an interactive periodic table plot. Heat map and hover tool. source must be a pandas dataframe containing, atom period and group, atomic number and symbol

Parameters

- **values** – data for the text inside each elements box
- **positions** – y positions relative to the middle of the box for each value
- **color_data** – data to display as a heatmap
- **color_map** – color palette to use for the heatmap (default matplotlib plasma)
- **log_scale** – bool, if True the heatmap is done logarithmically
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **saveas** – str, filename for the saved plot
- **blank_outsiders** – either 'both', 'min', 'max' or None, determines, which points outside the color range to color with a default blank color
- **blank_color** – color to replace values outside the color range by
- **include_legend** – if True an additional entry with labels explaing each value entry is added
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure

Additional kwargs are passed on to the label creation for the element box The kwargs *legend_options* and *colorbar_options* can be used to overwrite default values for these regions of the plot

`masci_tools.vis.bokeh_plots.`**`plot_convergence`**(*iteration*, *distance*, *total_energy*, *, *data=None*, *saveas_energy='energy_convergence'*, *saveas_distance='distance_convergence'*, *figure_energy=None*, *figure_distance=None*, *xlabel='Iteration'*, *ylabel_energy='Total energy difference [Htr]'*, *ylabel_distance='Distance [me/bohr^3]'*, *title_energy='Total energy difference over scf-Iterations'*, *title_distance='Convergence (log)'*, *copy_data=False*, *drop_last_iteration=False*, ***kwargs*)

Plot the total energy differences versus the scf iteration and plot the distance of the density versus iterations.

Parameters

- **iteration** – data for the number of iterations
- **distance** – data of distances
- **total_energy** – data of total energies
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label for the x-axis of both plots
- **saveas_energy** – str, filename for the energy convergence plot
- **figure_energy** – Axes object for the energy convergence plot
- **title_energy** – str, title for the energy convergence plot
- **ylabel_energy** – str, label for the y-axis for the energy convergence plot
- **saveas_distance** – str, filename for the distance plot
- **figure_distance** – Axes object for the distance plot
- **title_distance** – str, title for the distance plot
- **ylabel_distance** – str, label for the y-axis for the distance plot
- **copy_data** – bool if True the data argument is copied

- **drop_last_iteration** – bool if True the last iteration is dropped for the distance plot

Other Kwargs will be passed on to all *bokeh_line()* calls

masci_tools.vis.bokeh_plots.**plot_convergence_results**(*iteration*, *distance*, *total_energy*, *, *saveas='convergence'*, ***kwargs*)

Plot the total energy versus the scf iteration and plot the distance of the density versus iterations. Uses bokeh_line and bokeh_scatter

Parameters

- **iteration** – list of Int
- **distance** – list of floats
- **show** – bool, if True call show

Total_energy
list of floats

Kwargs will be passed on to *bokeh_line()*

Returns grid
bokeh grid with figures

masci_tools.vis.bokeh_plots.**plot_convergence_results_m**(*iterations*, *distances*, *total_energies*, *, *link=False*, *nodes=None*, *modes=None*, *plot_label=None*, *saveas='convergence'*, ***kwargs*)

Plot the total energy versus the scf iteration and plot the distance of the density versus iterations in a bokeh grid for several SCF results.

Parameters

- **distances** – list of lists of floats
- **iterations** – list of lists of Int
- **link** – bool, optional default=False:
- **nodes** – list of node uuids or pks important for links
- **saveas1** – str, optional default='t_energy_convergence', save first figure as
- **saveas2** – str, optional default='distance_convergence', save second figure as
- **figure_kwargs** – dict, optional default={'width': 600, 'height': 450}, gets parsed to bokeh_line
- **kwargs** – further key-word arguments for bokeh_line

Total_energies
list of lists of floats

Returns grid
bokeh grid with figures

masci_tools.vis.bokeh_plots.**plot_lattice_constant**(*scaling*, *total_energy*, *, *fit_data=None*, *data=None*, *figure=None*, *relative=True*, *ref_const=None*, *title='Equation of states'*, *saveas='lattice_constant'*, *copy_data=False*, ***kwargs*)

Plot a lattice constant versus Total energy Plot also the fit. On the x axis is the scaling, it

Parameters

- **scaling** – arraylike, data for the scaling factor
- **total_energy** – arraylike, data for the total energy
- **fit_data** – arraylike, optional data of fitted data
- **relative** – bool, scaling factor given (True), or lattice constants given?
- **ref_const** – float (optional), or list of floats, lattice constant for scaling 1.0
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **copy_data** – bool if True the data argument will be copied
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure

Function specific parameters:

param marker_fit
defaults to *marker*, marker type for the fit data

param marker_size_fit
defaults to *marker_size*, markersize for the fit data

param line_width_fit
defaults to *line_width*, linewidth for the fit data

param legend_label_fit
str label for the fit data

Other Kwargs will be passed on to *bokeh_line()*

masci_tools.vis.bokeh_plots.**reset_bokeh_plot_defaults**()

Reset the defaults for bokeh backend to the hardcoded defaults

Available defaults can be seen in *BokehPlotter*

masci_tools.vis.bokeh_plots.**save_bokeh_defaults**(*filename='plot_bokeh_defaults.json'*, *save_complete=False*)

Save the current defaults for the matplotlib backend to a json file.

Parameters

- **filename** – filename, where the defaults should be stored
- **save_complete** – bool if True not only the overwritten user defaults but also the unmodified hardcoded defaults are stored

masci_tools.vis.bokeh_plots.**set_bokeh_plot_defaults**(***kwargs*)

Set defaults for bokeh backend according to the given keyword arguments

Available defaults can be seen in *BokehPlotter*

masci_tools.vis.bokeh_plots.**show_bokeh_plot_defaults**()

Show the currently set defaults for bokeh backend

Available defaults can be seen in *BokehPlotter*

6.1.2.2 Calculation tools

This file contains a class to compute the crystalfield coefficients from convoluting the charge density with the potential which produces the crystalfield. This is both compatible with the Yttrium-Analogue approximation and self-consitent calculation of the potential

class masci_tools.tools.cf_calculation.**CFCalculation**(*, *radial_points=4000*, *reference_radius='pot'*, *coefficient_cutoff=0.001*, ***kwargs*)

Class for calculating Crystal Field coefficients using the procedure described in C.E. Patrick, J.B. Staunton: J. Phys.: Condens. Matter 31, 305901 (2019)

Using the formula:

$$B_{lm} = \sqrt{\frac{2l+1}{4\pi}} \int^{R_{MT}} drr^2 V_{lm}(r) n_{4f}(r)$$

The read in quantities are interpolated from logarithmic meshes to equidistant meshes

The function constructs an equidistant mesh between 0 and the muffin tin radius defined in *self.reference_radius* and with *self.radial_points* points

Parameters

- **radial_points** (int) – int, number of radial points in the interpolated mesh
- **reference_radius** (Union[float, Literal['pot', 'cdn']]) – stror float; Either 'pot' or 'cdn' or explicit number. Defines which muffin-tin radius is used for the equidistant mesh. IMPORTANT! If txt files are used the muffin-tin radius has to be provided explicitly
- **coefficient_cutoff** (Optional[float]) – float Defines minimum value that cf coefficients have to have to be considered non-zero

property denNorm

Returns the density normalization

DEPRECATED: Use density_normalization instead

classmethod from_arrays(*charge_density*, *potentials*, *radial_mesh*, ***kwargs*)

Create a CFCalculation instance from arrays

> **Warning:** Using this classmethod will not check that the charge density and potential have the same bravais matrix

Parameters

- **charge_density** (ndarray) – Array for the normed charge density
- **potentials** (dict[tuple[int, int], ndarray]) – Data for the potentials (dict mapping (l,m) tuples to the corresponding data)
- **radial_mesh** (dict[str, ndarray]) – dict for the data for the used radial meshs (keys 'cdn' and 'pot')

Return type
CFCalculation

Other Kwargs are passed on to the constructor

get_charge_density(*interpolated=True*)

Return the charge density and the corresponding radial mesh

Parameters

- **spin** – which spin to return
- **interpolated** – bool, if True the interpolated mesh and density are returned

Return type
tuple[ndarray, ndarray]

get_coefficients(*convention='Stevens'*, *table_fmt=None*)

Performs the integration to obtain the crystal field coefficients If the data was not already interpolated, the interpolation will be performed beforehand

Parameters: :type convention: Literal['Stevens', 'Wybourne']

param convention
str of the convention to use (Stevens or Wybourne)

Return type
list[*CFCoefficient*]

Returns
list of CFCoefficient objects (namedtuple), with all the necessary information

get_potentials(*spin*, *interpolated=True*, *only_nonzero=True*, *complex_data=True*)

Return the potentials and the corresponding radial mesh

Parameters

- **spin** (Literal['up', 'down']) – which spin to return
- **interpolated** (bool) – bool, if True the interpolated mesh and potentials are returned
- **only_nonzero** (bool) – bool, if True only the potentials corresponding to a non-zero coefficient are returned
- **complex_data** (bool) – bool, if False only the real part of the potentials is returned

Return type
tuple[ndarray, dict[tuple[int, int], ndarray]]

interpolate()

Interpolate all quantities to a common equidistant radial mesh

Return type
None

property nonzero_coefficients_lm: list[tuple[int, int]]

Return the lm indices of coefficients that are bigger than the set cutoff

performIntegration(*convert=True*)

DEPRECATED: Use get_coefficients instead

Performs the integration to obtain the crystal field coefficients If the data was not already interpolated, the interpolation will be performed beforehand

Parameters: :type convert:

param convert
bool, converts to Steven's coefficients (if True)

Returns
list of CFCoefficient objects (namedtuple), with all the necessary information

readCDN(**args*, ***kwargs*)

DEPRECATED: Use read_charge_density instead

readPot(**args*, ***kwargs*)

DEPRECATED: Use read_pot

read_charge_density(*file*, *atom_type=None*, *header=0*, *atomic_cdn=True*)

Reads in the normed charge density for the CF coefficient calculation If hdf files are given also the muffin tin radius is read in

Return type
None

Parameters: :type file: Union[str, bytes, Path, PathLike, IO[Any], File]

param file
Expects string filename for the charge density to read in The function expects either HDF files or txt files with the format (rmesh,cdn). The charge density should be given as r^2n(r) and normed to 1

kwargs: :type atom_type: Optional[int]

param atom_type
int, Defines the atom_type to read in (only for HDF files)

read_potential(**files*, *lm_indices=None*, *atom_type=None*, *header=0*, *complex_data=True*)

Reads in the potentials for the CF coefficient calculation If hdf files are given also the muffin tin radius is read in

Parameters

- **args** – Expects string filenames for the potentials to read in The function expects either HDF files or txt files with the format (rmesh,vlmup,vlmdn)
- **lm_indices** (Optional[list[tuple[int, int]]]) – list of tuples, Defines the l and m indices for the given txt files
- **atom_type** (Optional[int]) – int, Defines the atomType to read in (only for HDF files)
- **header** (int) – int, Define how many lines to skip in the beginning of txt file
- **complex_data** (bool) – bool, Define if the data in the text file is complex

Return type
None

Raises:
ValueError: lm indices list length has to match number of files read in

property spin_polarized: bool

Return whether the potentials have a spin-down component

stevens_prefactor(*l*, *m*)

Gives the lm dependent prefactor for conversion between Blm and Alm coefficients

Args: :type l: `int`

param l
int; orbital quantum number

Return type
`float`

Returns
float prefactor for conversion to Steven's Coefficients

validate()

Validate that the object can be used to execute the calculation Checks that the given bravais matrices are equal if given

Return type
`None`

class masci_tools.tools.cf_calculation.**CFCoefficient**(*l: int*, *m: int*, *spin_up: float | complex*, *spin_down: float | complex*, *unit: str*, *convention: str*)

Namedtuple representing an individual crystal field coefficient

convention: `str`

Alias for field number 5

l: `int`

Alias for field number 0

m: `int`

Alias for field number 1

spin_down: `float | complex`

Alias for field number 3

spin_up: `float | complex`

Alias for field number 2

unit: `str`

Alias for field number 4

masci_tools.tools.cf_calculation.**plot_crystal_field_calculation**(*cfcalc*, ***, *saveas='crystal_field_calc'*, *potential_title='Potential'*, *density_title='Density'*, *xlabel='R (Bohr)'*, *potential_ylabel='$Vpot$ (Hartree)'*, *density_ylabel='Density'*, *density_kwargs=None*, *potential=True*, *density=True*, *axis_potential=None*, *axis_density=None*, ***kwargs*)

Plot the given potentials and charge densities used in the given `CFCalculation`

Parameters

- **cfcalc** – CFcalculation containing the data to plot
- **saveas** – str, Define the filename to save the figure
- **potential_title** – Title for the potential subplot
- **density_title** – Title for the charge density subplot
- **xlabel** – label for the x axis of both subplots
- **potential_ylabel** – label for the y axis of the potential subplot
- **density_ylabel** – label for the y axis f the charge density subplot
- **density_kwargs** – dict with keyword argument passed to the plotting of the density
- **figure** – Use this preexisting matplotlib figure to produce the plots

All other Kwargs are passed on to the `multiple_scatterplots()` calls for plotting the potentials

masci_tools.tools.cf_calculation.**plot_crystal_field_potential**(*cfcoeffs*, *, *saveas='crystal_field_potential_areaplot'*, *spin='avg'*, *phi=0.0*, *figure=None*, ***kwargs*)

Plots the angular dependence of the calculated CF potential as well as a plane defined by phi.

Parameters

- **cfcoeffs** – list of CFCoefficients to construct the potential
- **saveas** – str, defines the filename to save the figure
- **spin** – str; Either 'up', 'dn' or 'avg'. Which spin direction to plot ('avg'-> ('up'+'dn')/2.0)
- **phi** – float, defines the phi angle of the plane

Raises

- **AssertionError** – When coefficients are provided as wrong types or in the wrong convention
- **ValueError** – When coefficients are provided in the wrong convention

This module contains utility and functions to work with Green's functions calculated and written to `greensf.hdf` files by fleur

class masci_tools.tools.greensfunction.**GreensFunction**(*element*, *data*, *attributes*)

Class for working with Green's functions calculated by the fleur code

Parameters

- **element** (*GreensfElement*) – *GreensfElement* namedtuple containing the information about the element
- **data** (dict[str, Any]) – datasets dict produced by one of the hdf recipes for reading Green's functions
- **attributes** (dict[str, Any]) – attributes dict produced by one of the hdf recipes for reading Green's functions

energy_dependence(*, *m=None*, *mp=None*, *spin=None*, *imag=True*, *both_contours=False*)

Select data with energy dependence

Parameters

- **m** (Optional[int]) – optional integer magnetic quantum number between -l and l
- **mp** (Optional[int]) – optional integer magnetic quantum number between -lp and lp
- **spin** (Optional[int]) – optional integer spin between 1 and nspins
- **both_contours** (bool) – bool id True the data is not added for both energy contours
- **imag** (bool) – bool if True and both_contours is False the imaginary part:math:*frac{1}{2i}left[Gleft(zright)-Gleft(z^astright)right]* is returned otherwise the real part $\frac{1}{2}\left[G\left(z\right)+G\left(z^{*}\right)\right]$

Return type
ndarray

Returns
numpy array with the selected data

energy_dependence_full_matrix(*imag=True*, *both_contours=False*)

Get the full matrix $N_{\mathrm{spins}}(2l+1)\times N_{\mathrm{spins}}(2l'+1)$

Parameters

- **both_contours** (bool) – bool id True the data is not added for both energy contours
- **imag** (bool) – bool if True and both_contours is False the imaginary part $\frac{1}{2i}\left[G\left(z\right)-G\left(z^{*}\right)\right]$ is returned otherwise the real part $\frac{1}{2}\left[G\left(z\right)+G\left(z^{*}\right)\right]$

Return type
ndarray

Returns
numpy array with the selected data

classmethod fromFile(*file*, *index=None*, ***selection_params*)

Classmethod for creating a *GreensFunction* instance directly from a hdf file

Parameters

- **file** (Any) – path or opened file handle to a greensf.hdf file
- **index** (Optional[int]) – optional int index of the element to read in

Return type
GreensFunction

If index is not given Keyword arguments with the keys being the names of the fields of *GreensfElement* can be given to select the right Green's function. The specification has to match only one element in the file

get_coefficient(*name*, *spin=None*, *radial=False*)

Get the coefficient with the given name from the data attribute

Parameters

- **name** (Literal['sphavg', 'uu', 'ud', 'du', 'dd', 'ulou', 'uulo', 'ulod', 'dulo', 'uloulo']) – name of the coefficient
- **radial** (bool) – if the Green's function is stored by coefficient and radial is True it is multiplied by the corresponding radial function otherwise the scalar product is multiplied
- **spin** (Optional[int]) – integer index of the spin to retrieve

Return type
ndarray

Returns
numpy.ndarray for the given coefficient and spin

moment(*n*, *spin=None*)

Calculate the integral

$$M_n = -\frac{1}{4\pi i}\mathrm{Im}\int_{\mathrm{Contour}} \mathrm{d}z z^n G(z)$$

Parameters

- **n** (int) – power of z in the integral
- **spin** (Optional[int]) – optional integer spin between 1 and nspins

Return type
ndarray

property nspins: int

Return the number of spins of the current element. If mperp is True for the element it is 4 otherwise it is determined by the spins attribute

occupation(*spin=None*)

Calculate the 0-th moment of the green's function

$$n = -\frac{1}{4\pi i}\mathrm{Im}\int_{\mathrm{Contour}} \mathrm{d}z G(z)$$

Note: Only if the energy contour ends at the fermi energy/is correlty weighted to produce occupations, will this function produce occupations

---

Parameters
spin (Optional[int]) – optional integer spin between 1 and nspins

Return type
ndarray

to_global_frame()

Rotate the Green's function into the global real space and spin space frame

Return type
None

to_local_frame()

Rotate the Green's function into the local real space and spin space frame

Return type
None

static to_m_index(*m*)

Convert between magnetic quantum numbers between -l and l to 0 and 2l+1 for easier indexing

Parameters
m (int) – int magnetic quantum number to convert

Return type
int

Returns
converted magnetic quantum number

static to_spin_indices(*spin*)

Convert between spin index (0 to 3) to the corresponding two spin indices (0 or 1)

Parameters
spin (int) – int spin index to convert

Return type
tuple[int, int]

Returns
tuple of spin indices

trace_energy_dependence(*spin=None*, *imag=True*)

Select trace of data with energy dependence

Parameters

- **spin** (Optional[int]) – integer spin between 1 and nspins
- **imag** (bool) – bool if True the imaginary part $\frac{1}{2i}\left[G\left(z\right)-G\left(z^{*}\right)\right]$ is returned otherwise the real part $\frac{1}{2}\left[G\left(z\right)+G\left(z^{*}\right)\right]$

Return type
ndarray

Returns
numpy array with the selected and traced over data

class masci_tools.tools.greensfunction.**GreensfElement**(*l: int*, *lp: int*, *atomType: int*, *atomTypep: int*, *sphavg: bool*, *onsite: bool*, *kresolved: bool*, *contour: int*, *nLO: int*, *atomDiff: np.ndarray*)

Namedtuple representing the high-level information about the Green's functions, i.e. what kind, which atoms, which orbitals

atomDiff: ndarray

Alias for field number 9

atomType: int

Alias for field number 2

atomTypep: int

Alias for field number 3

contour: int

Alias for field number 7

kresolved: bool

Alias for field number 6

l: int

Alias for field number 0

lp: int

Alias for field number 1

nLO: int

Alias for field number 8

onsite: bool

Alias for field number 5

sphavg: bool

Alias for field number 4

class masci_tools.tools.greensfunction.**colors**

Color strings for coloring terminal output

You may need to change color settings in iPython

masci_tools.tools.greensfunction.**intersite_shell_indices**(*elements*, *reference_atom*, *show=False*, *max_shells=None*)

Construct the green's function pairs to calculate the Jij exchange constants for a given reference atom from a list of *GreensfElement*

Parameters

- **elements** (list[*GreensfElement*]) – list of GreenfElements to use
- **reference_atom** (int) – integer of the atom to calculate the Jij's for (correspinds to the i)
- **show** (bool) – if True the elements belonging to a shell are printed in a shell
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type
list[tuple[floating[Any], list[tuple[int, int]]]]

Returns
list of tuples with distance and all indices of pairs in the shell

masci_tools.tools.greensfunction.**intersite_shells**(*greensfunctions*, *reference_atom*, *show=False*, *max_shells=None*)

Construct the green's function pairs to calculate the Jij exchange constants for a given reference atom from a list of given *GreensFunction*

Parameters

- **greensfunctions** (list[*GreensFunction*]) – List of Greens Function to use
- **reference_atom** (int) – integer of the atom to calculate the Jij's for (correspinds to the i)
- **show** (bool) – if True the elements belonging to a shell are printed in a shell
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type
Generator[tuple[floating[Any], *GreensFunction*, *GreensFunction*], None, None]

Returns
flat iterator with distance and the two corresponding *GreensFunction* instances for each Jij calculation

masci_tools.tools.greensfunction.**intersite_shells_from_file**(*hdffile*, *reference_atom*, *show=False*, *max_shells=None*)

Construct the green's function pairs to calculate the Jij exchange constants for a given reference atom from a given greensf.hdf file

Parameters

- **hdffile** (Union[str, bytes, Path, PathLike, IO[Any]]) – filepath or file handle to a greensf.hdf file
- **reference_atom** (int) – integer of the atom to calculate the Jij's for (correspinds to the i)
- **show** (bool) – if True the elements belonging to a shell are printed in a shell
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type
Generator[tuple[floating[Any], *GreensFunction*, *GreensFunction*], None, None]

Returns
flat iterator with distance and the two corresponding *GreensFunction* instances for each Jij calculation

masci_tools.tools.greensfunction.**listElements**(*hdffile*, *show=False*)

Find the green's function elements contained in the given `greens.hdf` file

Parameters

- **hdffile** (Union[str, bytes, Path, PathLike, IO[Any]]) – filepath or file handle to a greensf.hdf file
- **show** (bool) – bool if True the found elements are printed in a table

Return type
list[*GreensfElement*]

Returns
list of *GreensfElement*

masci_tools.tools.greensfunction.**printElements**(*elements*, *index=None*, *mark=None*)

Print the given list of *GreensfElement* in a nice table

Parameters

- **elements** (list[*GreensfElement*]) – list of *GreensfElement* to be printed
- **index** (Optional[list[int]]) – optional list of indices to show instead of the default index in the list
- **mark** (Optional[list[int]]) – optional list of int with elements to emphasize with an arrow and color

Return type
None

masci_tools.tools.greensfunction.**select_element_indices**(*elements*, *show=False*, ***selection_params*)

Select *GreensfElement* objects from a list based on constraints on their values

Parameters

- **elements** (list[*GreensfElement*]) – list of *GreensfElement* to choose from
- **show** (bool) – bool if True the found elements will be printed

The Keyword arguments correspond to the names of the fields and their desired value

Return type
list[int]

Returns
list of the indices matching the criteria

masci_tools.tools.greensfunction.**select_elements**(*greensfunctions*, *show=False*, ***selection_params*)

Select *GreensFunction* objects from a list based on constraints on the values of their underlying *GreensfElement*

Parameters

- **greensfunctions** (list[*GreensFunction*]) – list of *GreensFunction* to choose from
- **show** (bool) – bool if True the found elements will be printed

The Keyword arguments correspond to the names of the fields and their desired value

Return type
Generator[*GreensFunction*, None, None]

Returns
iterator over the matching *GreensFunction*

masci_tools.tools.greensfunction.**select_elements_from_file**(*hdffile*, *show=False*, ***selection_params*)

Construct the green's function matching specified criteria from a given `greensf.hdf` file

Parameters

- **hdffile** (Union[str, bytes, Path, PathLike, IO[Any]]) – file or file path to the `greensf.hdf` file
- **show** (bool) – bool if True the found elements will be printed

The Keyword arguments correspond to the names of the fields and their desired value

Return type
Generator[*GreensFunction*, None, None]

Returns
iterator over the matching *GreensFunction*

This module collects functions for calculating properties with the greens functions calculated by Fleur. At the moment the following are implemented:

- Calculating Heisenberg J_0 (spin stiffness) from onsite Green's functions
- Calculating Heisenberg J_ij exchange constants from intersite Green's functions
- Calculating the hybridization function from onsite Greens functions

masci_tools.tools.greensf_calculations.**calculate_heisenberg_j0**(*greensfunction*, *onsite_delta*, *show=False*)

Calculate spin stiffness J_0 for the given green's function using the formula

$$J_0 = \frac{1}{4\pi}\mathrm{Im}\ \mathrm{Tr_L}\int_{-\infty}^{E_F}\mathrm{dz}\Delta\left(G^{\uparrow}(z) - G^{\downarrow}(z)\right) + \Delta^2 G^{\uparrow}(z)G^{\downarrow}(z)$$

Parameters

- **greensfunction** (*GreensFunction*) – *GreensFunction* to use for the calculation
- **onsite_delta** (float) – onsite exchange splitting to use for the calculation
- **show** (bool) – bool if True additional information about the used Greens functions is printed out

Return type
`float`

Returns
the value of the spin stiffness in meV

`masci_tools.tools.greensf_calculations.calculate_heisenberg_jij(hdffileORgreensfunctions, reference_atom, onsite_delta, max_shells=None)`

Calculate the Heisenberg exchange constants form Green's functions using the formula

$$J_{ij} = \frac{1}{4\pi}\mathrm{Im}\ \mathrm{Tr_L}\int_{-\infty}^{E_F}\mathrm{d}z\Delta_i G_{ij}^{\uparrow}(z)\Delta_j G_{ji}^{\downarrow}(z)$$

Parameters

- **hdffileORgreensfunctions** (Union[str, bytes, Path, PathLike, IO[Any], list[GreensFunction]]) – either pat/file-like object for the `greensf.hdf` file to use or list of *GreensFunction*
- **reference_atom** (int) – integer index of the atom to calculate the Jijs from
- **onsite_delta** (ndarray) – List of floats containing the onsite exchange splitting for each atom type and l-channel
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type
DataFrame

Returns
pandas DataFrame containing all the Jij constants

`masci_tools.tools.greensf_calculations.calculate_heisenberg_tensor(hdffileORgreensfunctions, reference_atom, onsite_delta, max_shells=None)`

Calculate the Heisenberg exchange tensor **J** from Green's functions using the formula

$$J_{ij}^{\alpha\beta} = \frac{1}{4\pi}\mathrm{Im}\ \mathrm{Tr_L}\int_{-\infty}^{E_F}\mathrm{d}z\Delta_i\sigma_\alpha G_{ij}(z)\Delta_j\sigma_\beta G_{ji}(z)$$

for all α and $\beta = x, y, z$.

Parameters

- **hdffileORgreensfunctions** (Union[str, bytes, Path, PathLike, IO[Any], list[GreensFunction]]) – either pat/file-like object for the `greensf.hdf` file to use or list of *GreensFunction*
- **reference_atom** (int) – integer index of the atom to calculate the Jijs from
- **onsite_delta** (ndarray) – List of floats containing the onsite exchange splitting for each atom type and l-channel
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type
DataFrame

Returns
pandas DataFrame containing all the J_xx, J_xy, etc. constants

masci_tools.tools.greensf_calculations.**calculate_hybridization**(*greensfunction*)

Calculate the hybridization function as

$$\Delta(z) = \frac{1}{2 * l + 1} \mathrm{Tr} G^{-1}(z)$$

Return type
ndarray

Returns
numpy array of the hybridization function

masci_tools.tools.greensf_calculations.**decompose_jij_tensor**(*jij_tensor*, *moment_direction*)

Decompose the Heisenberg tensor as calculated by `calculate_heisenberg_tensor()` into three parts

- Isotropic $J = \frac{1}{3}\mathrm{Tr}\left[\mathbf{J}\right]$
- Symmetric traceless $J_S = \frac{1}{2}\left(\mathbf{J} + \mathbf{J}^T\right) - J$
- Antisymmetric $J_A = \frac{1}{2}\left(\mathbf{J} - \mathbf{J}^T\right)$

Parameters
jij_tensor (DataFrame) – Heisenberg tensor

Return type
DataFrame

Returns
tuple of the three aforementioned components

masci_tools.tools.greensf_calculations.**heisenberg_reciprocal**(*qpoints*, *jij_data*, *entry='J_ij'*)

Calculate the fourier transform of an entry for interaction constants

Example for J_{ij}

$$J(\mathbf{q}) \; = \; \sum_{ij} J_{ij} e^{i\mathbf{q}\cdot\mathbf{R}_{ij}}$$

where $\mathbf{R}_{ij}$ is the connecting vector associated with the J_{ij}

Parameters

- **qpoints** (ndarray) – numpy array containing the coordinates of the qpoints
- **jij_data** (DataFrame) – DataFrame generated by the above calculation functions
- **entry** (str) – str of the entry to calculate

Return type
ndarray

Returns
numpy array containing the Fourier transform

6.1.2.3 IO helper functions and file parsers

KKR related IO

In this module you find the kkrparams class that helps defining the KKR input parameters Also some defaults for the parameters are defined.

class masci_tools.io.kkr_params.**kkrparams**(***kwargs*)

Class for creating and handling the parameter input for a KKR calculation Optional keyword arguments are passed to init and stored in values dictionary.

Example usage: params = kkrparams(LMAX=3, BRAVAIS=array([[1,0,0], [0,1,0], [0,0,1]]))

Alternatively values can be set afterwards either individually with
params.set_value('LMAX', 3)

or multiple keys at once with
params.set_multiple_values(EMIN=-0.5, EMAX=1)

Other useful functions

- print the description of a keyword: params.get_description([key]) where [key] is a string for a keyword in params.values
- print a list of mandatory keywords: params.get_all_mandatory()
- print a list of keywords that are set including their value: params.get_set_values()

change_XC_val_kkrimp(*val*)

Convert integer value of KKRhost KEXCOR input to KKRimp XC string input.

fill_keywords_to_inputfile(*is_voro_calc=False*, *output='inputcard'*, *no_check=False*, *verbose=False*)

Fill new inputcard with keywords/values automatically check for input consistency (can be disabled by the no_check input) if is_voro_calc==True change mandatory list to match voronoi code, default is KKRcode

classmethod get_KKRcalc_parameter_defaults(*silent=False*)

set defaults (defined in header of this file) and returns dict, kkrparams_version

get_all_mandatory()

Return a list of mandatory keys

get_description(*key=None*, *search=None*)

Returns description of keyword 'key' If 'key' is None, print all descriptions of all available keywords If 'search' is not None, print all keys+descriptions where the search string is found

get_dict(*group=None*, *subgroup=None*)

Returns values dictionary.

Prints values belonging to a certain group only if the 'group' argument is one of the following: 'lattice', 'chemistry', 'accuracy', 'external fields', 'scf cycle', 'other'

Additionally the subgroups argument allows to print only a subset of all keys in a certain group. The following subgroups are available.

- in 'lattice' group '2D mode', 'shape functions'
- in 'chemistry' group 'Atom types', 'Exchange-correlation', 'CPA mode', '2D mode'
- in 'accuracy' group 'Valence energy contour', 'Semicore energy contour', 'CPA mode', 'Screening clusters', 'Radial solver', 'Ewald summation', 'LLoyd'

get_missing_keys(*use_aiida=False*)

Find list of mandatory keys that are not yet set

get_set_values()

Return a list of all keys/values that are set (i.e. not None)

get_type(*key*)

Extract expected type of 'key' from format info

get_value(*key*)

Gets value of keyword 'key'

is_mandatory(*key*)

Returns mandatory flag (True/False) for keyword 'key'

items()

make kkrparams.items() work

read_keywords_from_inputcard(*inputcard='inputcard'*, *verbose=False*)

Read list of keywords from inputcard and extract values to keywords dict

Example usage

p = kkrparams(); p.read_keywords_from_inputcard('inputcard')

Note

converts '<RBLEFT>', '<RBRIGHT>', 'ZPERIODL', and 'ZPERIODR' automatically to Ang. units!

remove_value(*key*)

Removes value of keyword 'key', i.e. resets to None

set_multiple_values(***kwargs*)

Set multiple values (in example value1 and value2 of keywords 'key1' and 'key2') given as key1=value1, key2=value2

set_value(*key*, *value*, *silent=False*)

Sets value of keyword 'key'

static split_kkr_options(*valtxt*)

Split keywords after fixed length of 8 :type valtxt: :param valtxt: list of strings or single string :returns: List of keywords of maximal length 8

update_to_kkrimp()

Update parameter settings to match kkrimp specification. Sets self.__params_type and calls _update_mandatory_kkrimp()

update_to_voronoi()

Update parameter settings to match voronoi specification. Sets self.__params_type and calls _update_mandatory_voronoi()

Reading of shpefun files of KKR

masci_tools.io.kkr_read_shapefun_info.**read_shapefun**(*path='.'*)

Read vertices of shapefunctions with Zoom into shapefun of a single atom

Author

Philipp Ruessmann

Parameters

path – path where voronoi output is found (optional, defaults to './')

Returns pos
positions of the centers of the shapefunctions

Returns out
dictionary of the vertices of the shapefunctions

Here I collect all functions needed to parse the output of a KKR calculation. These functions do not need aiida and are therefore separated from the actual parser file where parse_kkr_outputfile is called

`masci_tools.io.parsers.kkrparser_functions.check_error_category(err_cat, err_msg, out_dict)`

Check if parser error of the non-critical category (err_cat != 1) are actually consistent and may be discarded.

Parameters

- **err_cat** – the error-category of the error message to be investigated
- **err_msg** – the error-message
- **out_dict** – the dict of results obtained from the parser function

Returns
True/False if message is an error or warning

`masci_tools.io.parsers.kkrparser_functions.get_kmeshinfo(outfile_0init, outfile_000)`

Extract kmesh info from output.0.txt and output.000.txt

`masci_tools.io.parsers.kkrparser_functions.get_lattice_vectors(outfile_0init)`

read direct and reciprocal lattice vectors in internal units (useful for qdos generation)

`masci_tools.io.parsers.kkrparser_functions.get_natom(outfile_0init)`

extract NATYP value from output.0.txt

`masci_tools.io.parsers.kkrparser_functions.get_noco_rms(outfile, debug=False)`

Get average noco rms error

`masci_tools.io.parsers.kkrparser_functions.get_nspin(outfile_0init)`

extract NSPIN value from output.0.txt

`masci_tools.io.parsers.kkrparser_functions.get_orbmom(outfile, natom)`

read orbmom info from outfile and return array (iteration, atom)=orbmom

`masci_tools.io.parsers.kkrparser_functions.get_rms(outfile, outfile2, debug=False)`

Get rms error per atom (both values for charge and spin) and total (i.e. average) value

`masci_tools.io.parsers.kkrparser_functions.get_single_particle_energies(outfile_000)`

extracts single particle energies from outfile_000 (output.000.txt) returns the valence contribution of the single particle energies

`masci_tools.io.parsers.kkrparser_functions.get_spinmom_per_atom(outfile, natom, nonco_out_file=None)`

Extract spin moment information from outfile and nonco_angles_out (if given)

`masci_tools.io.parsers.kkrparser_functions.parse_array_float(outfile, searchstring, splitinfo, replacepair=None, debug=False)`

Search for keyword *searchstring* in *outfile* and extract array of results

Returns: array of results

masci_tools.io.parsers.kkrparser_functions.**parse_kkr_outputfile**(*out_dict*, *outfile*, *outfile_0init*, *outfile_000*, *timing_file*, *potfile_out*, *nonco_out_file*, *outfile_2='output.2.txt'*, *skip_readin=False*, *debug=False*)

Parser method for the kkr outfile. It returns a dictionary with results

masci_tools.io.parsers.kkrparser_functions.**use_BdG**(*outfile_0init*)

extract BdG run info from output.0.txt

masci_tools.io.parsers.kkrparser_functions.**use_newsosol**(*outfile_0init*)

extract NEWSOSOL info from output.0.txt

Everything that is needed to parse the output of a voronoi calculation.

masci_tools.io.parsers.voroparser_functions.**check_voronoi_output**(*potfile*, *outfile*, *delta_emin_safety=0.1*)

Read output from voronoi code and create guess of energy contour

masci_tools.io.parsers.voroparser_functions.**get_valence_min**(*outfile='out_voronoi'*)

Construct minimum of energy contour (between valence band bottom and core states)

masci_tools.io.parsers.voroparser_functions.**parse_voronoi_output**(*out_dict*, *outfile*, *potfile*, *atominfo*, *radii*, *inputfile*, *debug=False*)

Parse output of voronoi calculation and return (success, error_messages_list, out_dict)

Tools for the impurity calculation plugin and its workflows

class masci_tools.io.parsers.kkrimp_parser_functions.**KkrimpParserFunctions**

Class of parser functions for KKRimp calculation

Usage

success, msg_list, out_dict = parse_kkrimp_outputfile().parse_kkrimp_outputfile(out_dict, files)

parse_kkrimp_outputfile(*out_dict*, *file_dict*, *debug=False*)

Main parser function for kkrimp, read information from files in file_dict and fills out_dict :type out_dict: :param out_dict: dictionary that is filled with parsed output of the KKRimp calculation :type file_dict: :param file_dict: dictionary of files that are parsed :returns: success (bool), msg_list(list of error/warning messages of parser), out_dict (filled dict of parsed output) :note: file_dict should contain the following keys

- 'outfile', the std_out of the KKRimp calculation
- 'out_log', the out_log.000.txt file
- 'out_pot', the output potential
- 'out_enersp_at', the out_energysp_per_atom_eV file
- 'out_enertot_at', the out_energytotal_per_atom_eV file
- 'out_timing', the timing file
- 'kkrflex_llyfac', the file for the Lloyd factor
- 'kkrflex_angles', the nonco_angles file for the KKRimp calculation
- 'out_spinmoms', the output spin moments file
- 'out_orbmoms', the output orbital moments file

Fleur related IO

Input/Output Parser

Load both the outxml_parser and inpxml_parser

masci_tools.io.parsers.fleur.**conversion_function**(*func*)

Marks a function as a conversion function, which can be called after performing a parsing task. The function can be specified via the _conversions control key in the task definitions.

Return type
TypeVar(F, bound= Callable[..., Any])

A conversion function has to have the following arguments:

param out_dict
dict with the previously parsed information

param parser_info_out
dict, with warnings, info, errors, ...

and return only the modified output dict

masci_tools.io.parsers.fleur.**inpxml_parser**(*inpxmlfile*, *parser_info_out=None*, *strict=False*, *debug=False*, *base_url=None*)

Parses the given inp.xml file to a python dictionary utilizing the schema defined by the version number to validate and correctly convert to the dictionary

Parameters

- **inpxmlfile** (Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the inp.xml file, opened file handle (in bytes modes i.e. rb) or a xml etree to be parsed
- **parser_info_out** (Optional[dict[str, Any]]) – dict, with warnings, info, errors, ...
- **strict** (bool) – bool if True and no parser_info_out is provided any encountered error will immediately be raised

Return type
dict[str, Any]

Returns
python dictionary with the parsed inp.xml

Raises

- **ValueError** – If the validation against the schema failed, or an irrecoverable error occurred during parsing
- **FileNotFoundError** – If no Schema file for the given version was found

masci_tools.io.parsers.fleur.**outxml_parser**(*outxmlfile*, *parser_info_out=None*, *iteration_to_parse='last'*, *minimal_mode=False*, *additional_tasks=None*, *optional_tasks=None*, *overwrite=False*, *append=False*, *list_return=False*, *strict=False*, *debug=False*, *ignore_validation=False*, *base_url=None*)

Parses the out.xml file to a dictionary based on the version and the given tasks

Parameters

- **outxmlfile** (Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the out.xml file, opened file handle (in bytes modes i.e. rb) or a xml etree to be parsed
- **parser_info_out** (Optional[dict[str, Any]]) – dict, with warnings, info, errors, ...
- **iteration_to_parse** (Union[Literal['all', 'last', 'first'], int]) – either str or int, (optional, default 'last') determines which iteration should be parsed. Accepted are 'all', 'first', 'last' or an index for the iteration
- **minimal_mode** (bool) – bool, if True only total Energy, iteration number and distances are parsed
- **additional_tasks** (Optional[dict[str, dict[str, Any]]]) – dict to define custom parsing tasks. For detailed explanation See *default_parse_tasks*.
- **overwrite** (bool) – bool, if True and keys in additional_tasks collide with defaults The defaults will be overwritten
- **append** (bool) – bool, if True and keys in additional_tasks collide with defaults The inner tasks will be written into the dict. If inner keys collide they are overwritten
- **optional_tasks** (Optional[Iterable[str]]) – Iterable of strings, defines additional tasks to perform. See *default_parse_tasks* for examples.
- **list_return** (bool) – bool, if True one-item lists in the output dict are not converted to simple values
- **strict** (bool) – bool if True and no parser_info_out is provided any encountered error will immediately be raised
- **debug** (bool) – bool if True additional information is printed out in the logs
- **ignore_validation** (bool) – bool, if True schema validation errors are only logged

Return type
dict[str, Any]

Returns
python dictionary with the information parsed from the out.xml

Raises

- **ValueError** – If the validation against the schema failed, or an irrecoverable error occurred during parsing
- **FileNotFoundError** – If no Schema file for the given version was found
- **KeyError** – If an unknown task is encountered

masci_tools.io.parsers.fleur.**register_migration**(*base_version*, *target_version*)

Decorator to add migration for task definition dictionary to the _TaskParser class The function should only take the dict of task definitions as an argument

Parameters

- **base_version** (str) – str of the version, from which the migration starts
- **target_version** (str | list[str]) – str or list of str with the versions that work after the migration has been performed

Return type
Callable[[TypeVar(F, bound= Callable[..., Any])], TypeVar(F, bound= Callable[..., Any])]

Inputgenerator related IO

This module contains functionality for writing input files for the input generator of fleur

class masci_tools.io.fleur_inpgen.**AtomDictProperties**

TypedDict for the atom properties

class masci_tools.io.fleur_inpgen.**Kinds**

TypedDict for the kinds

masci_tools.io.fleur_inpgen.**read_inpgen_file**(*file*, *convert_to_angstroem=True*)

Method which reads in an inpgen input file and parses the structure and name lists information.

Parameters

- **file** (Union[str, bytes, Path, PathLike, IO[Any]]) – path to the file to read or opened file handle
- **convert_to_angstroem** (bool) – bool if True the bravais matrix (and atom positions) are converted to angstroem

Return type

tuple[Optional[ndarray], list[*AtomSiteProperties*], tuple[bool, bool, bool], dict[str, Any]]

Returns

tuple of bravais matrix, atom sites, periodic boundary conditions and parameters

masci_tools.io.fleur_inpgen.**write_inpgen_file**(*cell*, *atom_sites*, *kinds=None*, *return_contents=False*, *file='inpgen.in'*, *pbc=(True, True, True)*, *input_params=None*, *significant_figures_cell=9*, *significant_figures_positions=10*, *significant_figures_magnetic_moments=4*, *convert_from_angstroem=True*)

Write an input file for the fleur inputgenerator 'inpgen' from given inputs

Parameters

- **cell** (ndarray | list[list[float]]) – 3x3 arraylike. The bravais matrix of the structure, in Angstrom by default
- **atom_sites** (Union[Sequence[*AtomSiteProperties*], Sequence[tuple[list[float], str, str]], Sequence[*AtomDictProperties*]]) – either list of a dict containing the keys absolute 'position' in Angstrom (default) and 'kind_name', i.e

```
[{'position': (0.0, 0.0, -1.0545708047819), 'kind_name': 'Fe123'},
 {'position': (1.4026317387183, 1.9836207751336, 0.0), 'kind_name':
↪'Pt'},
 {'position': (0.0, 0.0, 1.4026318234924), 'kind_name': 'Pt'}]
```

  In this case the argument kinds is required. The other possibility is a list of tuples of the form of *AtomSiteProperties*

- **kinds** (Optional[Iterable[*Kinds*]]) – a list of kind information containing the keys symbols, weights, mass, name i.e.

```
[{'symbols': ('Fe',), 'weights': (1.0,), 'mass': 55.845, 'name':
↪'Fe123'},
 {'symbols': ('Pt',), 'weights': (1.0,), 'mass': 195.084, 'name
↪': 'Pt'}]
```

Required when atom_sites is a list of dicts

- **file** (Union[str, bytes, Path, PathLike, IO[Any]]) – Path or filehandle where the file should be written to. Defaults to 'inpgen.in' in the current folder.
- **pbc** (tuple[bool, bool, bool]) – tuple of boolean length 3, optional, Periodic boundary conditions of the structure. Defaults to (True, True, True).
- **input_params** (Optional[dict[str, Any]]) – Optional dict containing further namelist which should be written to the file. Defaults to None.
- **significant_figures_cell** (int) – int, how many decimal places should be written for the bravais matrix (default: 9)
- **significant_figures_positions** (int) – int, how many decimal places should be written for the atom positions (default: 10)
- **convert_from_angstroem** (bool) – optional boolean, if True the positions and elements of the bravais matrix are converted to bohr from Angstroem

Raises
ValueError – If some input is wrong or inconsistent.

Return type
Optional[str]

Comments: This was extracted out of aiida-fleur for more general use, the datastructures stayed very close to what aiida provides (to_raw()), it may not yet be convenient for all usecases. I.e data so far has to be given in Angstrom and will be converted to fleur units. # This could be made optional

Functions for modifying the input file

This module contains a class for organizing and grouping changes to a input file of fleur in a robust way.

Essentially a low-level version of the FleurinpModifier in aiida_fleur.

Note: The docstrings for the setter methods are generated from their actual implementations in the *xml* modules via a pre-commit hook. Changes in the docstrings here will be overwritten

class masci_tools.io.fleurxmlmodifier.**FleurXMLModifier**(*validate_signatures=True*)

Class for grouping and organizing changes to a inp.xml file of fleur via the xml setting methods in *xml_setters_names* and *xml_setters_basic*

The basic usage is shown below

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fmode = FleurXMLModifier()

#Add changes by calling the methods on this class
#(names correspond to the setting methods in the xml_setters modules)
```

(continues on next page)

(continued from previous page)

```
#They are not modifying a input file directly
#Instead all the tasks are collected and performed in one go

fmode.set_inpchanges({'Kmax': 4.0}) #Set Kmax to 4.0
fmode.shift_value({'Gmax': 5.0}) #Add 5 to the current value of Gmax

#Set the local orbital configuration on all iron atoms to '3s 3p'
fmode.set_species('all-Fe', {'lo': [{'n':3, 'l': 's', 'type': 'SCLO'},
                                    {'n':3, 'l': 'p', 'type': 'SCLO'}]})

#To undo the last change call undo
#fmode.undo()

#revert_all=True resets all added tasks
#fmode.undo(revert_all=True)

#To apply the changes to an input file use the modify_xmlfile method
new_xmltree, additional_files = fmode.modify_xmlfile('/path/to/input/file/inp.xml')
```

add_number_to_attrib(**args*, ***kwargs*)

Appends a *add_number_to_attrib()* to the list of tasks that will be done on the xmltree.

Adds a given number to the attribute value in a xmltree specified by the name of the attribute and optional further specification If there are no nodes under the specified xpath an error is raised

Parameters

- **name** – the attribute name to change
- **number_to_add** – number to add/multiply with the old attribute value
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **mode** – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add abs/absolute* adds the old value and *number_to_add*
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

Return type
None

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

add_number_to_first_attrib(**args*, ***kwargs*)

Appends a *add_number_to_first_attrib()* to the list of tasks that will be done on the xmltree.

Adds a given number to the first occurrence of an attribute value in a xmltree specified by the name of the attribute and optional further specification If there are no nodes under the specified xpath an error is raised

Parameters

- **name** – the attribute name to change
- **number_to_add** – number to add/multiply with the old attribute value
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **mode** – str (either *rel*/*relative* or *abs*/*absolute*). *rel*/*relative* multiplies the old value with *number_to_add* *abs*/*absolute* adds the old value and *number_to_add*
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
None

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

add_task_list(*task_list*)

Add a list of tasks to be added

Parameters
task_list (list[tuple[str, dict[str, Any]]]) – list of tuples first index is the name of the method second is defining the arguments by keyword in a dict

Return type
None

align_nmmpmat_to_sqa(**args*, ***kwargs*)

Appends a *align_nmmpmat_to_sqa()* to the list of tasks that will be done on the xmltree.

Align the density matrix with the given SQA of the associated species

Parameters

- **species_name** – string, name of the species you want to change
- **orbital** – integer or string ('all'), orbital quantum number of the LDA+U procedure to be modified
- **phi_before** – float or list of floats, angle (radian), values for phi for the previous alignment of the density matrix

- **theta_before** – float or list of floats, angle (radian), values for theta for the previous alignment of the density matrix
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type
None

classmethod apply_modifications(*xmltree*, *nmmp_lines*, *modification_tasks*, *validate_changes=True*, *adjust_version_for_dev_version=True*)

Applies given modifications to the fleurinp lxml tree. It also checks if a new lxml tree is validated against schema. Does not rise an error if inp.xml is not validated, simple prints a message about it.

Parameters

- **xmltree** (_ElementTree) – a lxml tree to be modified (IS MODIFIED INPLACE)
- **nmmp_lines** (Optional[list[str]]) – a n_mmp_mat file to be modified (IS MODIFIED INPLACE)
- **modification_tasks** (list[*ModifierTask*]) – a list of modification tuples
- **validate_changes** (bool) – bool optional (default True), if True after all tasks are performed both the xmltree and nmmp_lines are checked for consistency
- **adjust_version_for_dev_version** (bool) – bool optional (default True), if True and the schema_dict and file version differ, e.g. a development version is used the version is temporarily modified to swallow the validation error that would occur

Return type
tuple[_ElementTree, Optional[list[str]]]

Returns
a modified lxml tree and a modified n_mmp_mat file

changes()

Prints out all changes currently registered on this instance

Return type
list[*ModifierTask*]

clone_species(**args*, ***kwargs*)

Appends a *clone_species()* to the list of tasks that will be done on the xmltree.

Method to create a new species from an existing one with evtl. modifications

For reference of the changes dictionary look at *set_species()*

Parameters

- **species_name** – string, name of the specie you want to clone Has to correspond to one single species (no 'all'/'all-<search_string>')
- **new_name** – new name of the cloned species
- **changes** – a optional python dict specifying what you want to change.

Return type
None

`create_tag`(**args*, ***kwargs*)

Appends a `create_tag()` to the list of tasks that will be done on the xmltree.

This method creates a tag with a uniquely identified xpath under the nodes of its parent. If there are no nodes evaluated the subtags can be created with *create_parents=True*

The tag is always inserted in the correct place if a order is enforced by the schema

Parameters

- **tag** – str of the tag to create or etree Element or string representing the XML element with the same name to insert
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details
- **create_parents** – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type
None

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

`delete_att`(**args*, ***kwargs*)

Appends a `delete_att()` to the list of tasks that will be done on the xmltree.

This method deletes a attribute with a uniquely identified xpath.

Parameters

- **name** – str of the attribute to delete
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a attribute. By default all nodes are used.

Return type
None

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

delete_tag(**args*, ***kwargs*)

Appends a *delete_tag()* to the list of tasks that will be done on the xmltree.

This method deletes a tag with a uniquely identified xpath.

Parameters

- **tag** – str of the tag to delete
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a tag. By default all nodes are used.

Return type
None

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

classmethod fromList(*task_list*, **args*, ***kwargs*)

Instantiate the FleurXMLModifier from a list of tasks to be added immediately

Parameters
task_list (list[tuple[str, dict[str, Any]]]) – list of tuples first index is the name of the method second is defining the arguments by keyword in a dict

Other arguments are passed on to the __init__ method

Return type
FleurXMLModifier

Returns
class with the task list instantiated

get_avail_actions()

Returns the allowed functions from FleurXMLModifier

Return type
dict[str, Callable]

modify_xmlfile(*original_inpxmlfile*, *original_nmmp_file=None*, *validate_changes=True*, *adjust_version_for_dev_version=True*, *keep_inpgen_comments=True*)

Applies the registered modifications to a given inputfile

Parameters

- **original_inpxmlfile** (Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the inp.xml file, opened file handle or a xml etree to be parsed
- **original_nmmp_file** (Union[str, bytes, Path, PathLike, IO[Any], list[str], None]) – path or list of str to a corresponding density matrix file

Raises
ValueError – if the parsing of the input file

Return type
tuple[_ElementTree, dict[str, str]]

Returns
a modified xmltree and if existent a modified density matrix file

replace_tag(**args*, ***kwargs*)

Appends a *replace_tag()* to the list of tasks that will be done on the xmltree.

This method deletes a tag with a uniquely identified xpath.

Parameters

- **tag** – str of the tag to replace
- **element** – etree Element or string representing the XML element to replace the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** – int or list of int. Which occurrence of the parent nodes to replace a tag. By default all nodes are used.

Return type
None

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

rotate_nmmpmat(**args*, ***kwargs*)

Appends a *rotate_nmmpmat()* to the list of tasks that will be done on the xmltree.

Rotate the density matrix with the given angles phi and theta

Parameters

- **species_name** – string, name of the species you want to change
- **orbital** – integer or string ('all'), orbital quantum number of the LDA+U procedure to be modified
- **phi** – float, angle (radian), by which to rotate the density matrix
- **theta** – float, angle (radian), by which to rotate the density matrix

- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type
None

set_atomgroup(**args*, ***kwargs*)

Appends a *set_atomgroup()* to the list of tasks that will be done on the xmltree.

Method to set parameters of an atom group of the fleur inp.xml file.

Parameters

- **changes** – a python dict specifying what you want to change.
- **position** – position of an atom group to be changed. If equals to 'all', all species will be changed
- **species** – atom groups, corresponding to the given species will be changed
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
None

changes is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'changes': {'nocoParams': {'beta': val}}
```

set_atomgroup_label(**args*, ***kwargs*)

Appends a *set_atomgroup_label()* to the list of tasks that will be done on the xmltree.

This method calls *set_atomgroup()* method for a certain atom species that corresponds to an atom with a given label.

Parameters

- **atom_label** – string, a label of the atom which specie will be changed. 'all' to change all the species
- **changes** – a python dict specifying what you want to change.

Return type
None

changes is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'changes': {'nocoParams': {'beta': val}}
```

set_attrib_value(**args*, ***kwargs*)

Appends a *set_attrib_value()* to the list of tasks that will be done on the xmltree.

Sets an attribute in a xmltree to a given value, specified by its name and further specifications. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are

converted automatically according to the types of the attribute with `convert_to_xml()` if they are not *str* already.

Parameters

- **name** – the attribute name to set
- **value** – value or list of values to set
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** – bool optional (default False), if True the tag is created if is missing

Return type
`None`

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

set_complex_tag(**args*, ***kwargs*)

Appends a `set_complex_tag()` to the list of tasks that will be done on the xmltree.

Function to correctly set tags/attributes for a given tag. Goes through the attributedict and decides based on the schema_dict, how the corresponding key has to be handled. The tag is specified via its name and evtl. further specification

Supports:

- attributes
- tags with text only
- simple tags, i.e. only attributes (can be optional single/multiple)
- complex tags, will recursively create/modify them

Parameters

- **tag_name** – name of the tag to set
- **changes** – Keys in the dictionary correspond to names of tags and the values are the modifications to do on this tag (attributename, subdict with changes to the subtag, . . .)
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details

- **create** – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

Return type
None

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

set_first_attrib_value(**args*, ***kwargs*)

Appends a `set_first_attrib_value()` to the list of tasks that will be done on the xmltree.

Sets the first occurrence of an attribute in a xmltree to a given value, specified by its name and further specifications. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with `convert_to_xml()` if they are not *str* already.

Parameters

- **name** – the attribute name to set
- **value** – value or list of values to set
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details
- **create** – bool optional (default False), if True the tag is created if is missing

Return type
None

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

set_first_text(**args*, ***kwargs*)

Appends a `set_first_text()` to the list of tasks that will be done on the xmltree.

Sets the text the first occurrence of a tag in a xmltree to a given value, specified by the name of the tag and further specifications. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with `convert_to_xml()` if they are not *str* already.

Parameters

- **tag_name** – str name of the tag, where the text should be set
- **text** – value or list of values to set
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create** – bool optional (default False), if True the tag is created if is missing

Return type
None

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

set_inpchanges(**args*, ***kwargs*)

Appends a *set_inpchanges()* to the list of tasks that will be done on the xmltree.

This method sets all the attribute and texts provided in the change_dict.

The first occurrence of the attribute/tag is set

Parameters

- **changes** – dictionary {attrib_name : value} with all the wanted changes.
- **path_spec** – dict, with ggf. necessary further specifications for the path of the attribute

Return type
None

An example of changes:

```
changes = {
    'itmax' : 1,
    'l_noco': True,
    'ctail': False,
    'l_ss': True
}
```

set_kpath(**args*, ***kwargs*)

Appends a *set_kpath()* to the list of tasks that will be done on the xmltree.

Sets a k-path directly into inp.xml as a alternative kpoint set with purpose 'bands'

> **Warning:** This method is only supported for input versions before the Max5 release

Parameters

- **kpath** – a dictionary with kpoint name as key and k point coordinate as value
- **count** – number of k-points
- **gamma** – bool that controls if the gamma-point should be included in the k-point mesh

Return type
None

set_kpointlist(**args*, ***kwargs*)

Appends a *set_kpointlist()* to the list of tasks that will be done on the xmltree.

Explicitly create a kPointList from the given kpoints and weights. This routine will add the specified kPointList with the given name.

> **Warning:** For input versions Max4 and older **all** keyword arguments are not valid (*name*, *kpoint_type*, *special_labels*, *switch* and *overwrite*)

Parameters

- **kpoints** – list or array containing the **relative** coordinates of the kpoints
- **weights** – list or array containing the weights of the kpoints
- **name** – str for the name of the list, if not given a default name is generated
- **kpoint_type** – str specifying the type of the kPointList ('path', 'mesh', 'spex', 'tria', ...)
- **special_labels** – dict mapping indices to labels. The labels will be inserted for the kpoints corresponding to the given index
- **switch** – bool, if True the kPointlist will be used by Fleur when starting the next calculation
- **overwrite** – bool, if True and a kPointlist with the given name already exists it will be overwritten

Return type
None

set_kpointmesh(**args*, ***kwargs*)

Appends a *set_kpointmesh()* to the list of tasks that will be done on the xmltree.

Create a kpoint mesh using spglib

for details see `get_stabilized_reciprocal_mesh()`

Parameters

- **mesh** – list-like woth three elements, giving the size of the kpoint set in each direction
- **use_symmetry** – bool if True the available symmetry operations in the inp.xml will be used to reduce the kpoint set otherwise only the identity matrix is used
- **name** – Name of the created kpoint list. If not given a name is generated
- **switch** – bool if True the kpoint list is direclty set as the used set
- **overwrite** – if True and a kpoint list of the given name already exists it will be overwritten
- **shift** – shift the center of the kpint set
- **time_reversal** – bool if True time reversal symmetry will be used to reduce the kpoint set
- **map_to_first_bz** – bool if True the kpoints are mapped into the [0,1] interval

Return type
None

set_kpointpath(**args*, ***kwargs*)

Appends a *set_kpointpath()* to the list of tasks that will be done on the xmltree.

Create a kpoint list for a bandstructure calculation (using ASE kpath generation)

The path can be defined explictly (see `bandpath()`) or derived from the unit cell

Parameters

- **path** – str, list of str or None defines the path to interpolate (for syntax `bandpath()`)
- **nkpts** – int number of kpoints in the path
- **density** – float number of kpoints per Angstroem
- **name** – Name of the created kpoint list. If not given a name is generated
- **switch** – bool if True the kpoint list is direclty set as the used set
- **overwrite** – if True and a kpoint list of the given name already exists it will be overwritten
- **special_points** – dict mapping names to coordinates for special points to use

Return type
`None`

set_nkpts(**args*, ***kwargs*)

Appends a *set_nkpts()* to the list of tasks that will be done on the xmltree.

Sets a k-point mesh directly into inp.xml

> **Warning:** This method is only supported for input versions before the Max5 release

Parameters

- **count** – number of k-points
- **gamma** – bool that controls if the gamma-point should be included in the k-point mesh

Return type
`None`

set_nmmpmat(**args*, ***kwargs*)

Appends a *set_nmmpmat()* to the list of tasks that will be done on the xmltree.

Routine sets the block in the n_mmp_mat file specified by species_name, orbital and spin to the desired density matrix

Parameters

- **species_name** – string, name of the species you want to change
- **orbital** – integer, orbital quantum number of the LDA+U procedure to be modified
- **spin** – integer, specifies which spin block should be modified
- **state_occupations** – list, sets the diagonal elements of the density matrix and everything else to zero
- **denmat** – matrix, specify the density matrix explicitly
- **phi** – float, optional angle (radian), by which to rotate the density matrix before writing it

- **theta** – float, optional angle (radian), by which to rotate the density matrix before writing it
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type
None

set_simple_tag(**args*, ***kwargs*)

Appends a *set_simple_tag()* to the list of tasks that will be done on the xmltree.

Sets one or multiple *simple* tag(s) in an xmltree. A simple tag can only hold attributes and has no subtags. The tag is specified by its name and further specification If the tag can occur multiple times all existing tags are DELETED and new ones are written. If the tag only occurs once it will automatically be created if its missing.

Parameters

- **tag_name** – str name of the tag to modify/set
- **changes** – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to {'attributename': attributevalue}
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create_parents** – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

Return type
None

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

set_species(**args*, ***kwargs*)

Appends a *set_species()* to the list of tasks that will be done on the xmltree.

Method to set parameters of a species tag of the fleur inp.xml file.

Parameters

- **species_name** – string, name of the specie you want to change Can be name of the species, 'all' or 'all-<string>' (sets species with the string in the species name)
- **changes** – a python dict specifying what you want to change.
- **create** – bool, if species does not exist create it and all subtags?

- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises
ValueError – if species name is non existent in inp.xml and should not be created. also if other given tags are garbage. (errors from eval_xpath() methods)

Return xmltree
xml etree of the new inp.xml

Return type
None

changes is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a MT radius it can be done via:

```
changes = {'mtSphere' : {'radius' : 2.2}}
```

Another example:

```
'changes': {'special': {'socscale': 0.0}}
```

that switches SOC terms on a sertain specie. `mtSphere`, `atomicCutoffs`, `energyParameters`, `lo`, `electronConfig`, `nocoParams`, `ldaU` and `special` keys are supported. To find possible keys of the inner dictionary please refer to the FLEUR documentation flapw.de

set_species_label(**args*, ***kwargs*)

Appends a *set_species_label()* to the list of tasks that will be done on the xmltree.

This method calls *set_species()* method for a certain atom species that corresponds to an atom with a given label

Parameters

- **atom_label** – string, a label of the atom which specie will be changed. 'all' to change all the species
- **changes** – a python dict specifying what you want to change.
- **create** – bool, if species does not exist create it and all subtags?

Return type
None

set_text(**args*, ***kwargs*)

Appends a *set_text()* to the list of tasks that will be done on the xmltree.

Sets the text on tags in a xmltree to a given value, specified by the name of the tag and further specifications. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with *convert_to_xml()* if they are not *str* already.

Parameters

- **tag_name** – str name of the tag, where the text should be set
- **text** – value or list of values to set
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** – bool optional (default False), if True the tag is created if is missing

Return type
None

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

set_xcfunctional(**args*, ***kwargs*)

Appends a `set_xcfunctional()` to the list of tasks that will be done on the xmltree.

Set the Exchange Correlation potential tag

Setting a inbuilt XC functional .. code-block:: python

set_xcfunctional(xmltree, schema_dict, 'vwn')

Setting a LibXC XC functional .. code-block:: python

set_xcfunctional(xmltree, schema_dict, {'exchange': 'lda_x', 'correlation':"lda_c_xalpha"}, libxc=True)

Parameters

- **xc_functional** – str or dict. If str it is the name of a inbuilt XC functional. If it is a dict it specifies either the name or id for LibXC functionals for the keys *'exchange', 'correlation', 'etot_exchange' and 'etot_correlation'*
- **xc_functional_options** – dict with further general changes to the *xcFunctional* tag
- **libxc** – bool if True the functional is a LibXC functional

Return type
None

shift_value(**args*, ***kwargs*)

Appends a `shift_value()` to the list of tasks that will be done on the xmltree.

Shifts numerical values of attributes directly in the inp.xml file.

The first occurrence of the attribute is shifted

Parameters

- **changes** – a python dictionary with the keys to shift and the shift values.
- **mode** – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with the given value *abs/absolute* adds the old value and the given value
- **path_spec** – dict, with ggf. necessary further specifications for the path of the attribute

Return type
None

An example of changes:

```
changes = {'itmax' : 1, 'dVac': -0.123}
```

shift_value_species_label(**args*, ***kwargs*)

Appends a *shift_value_species_label()* to the list of tasks that will be done on the xmltree.

Shifts the value of an attribute on a species by label if atom_label contains 'all' then applies to all species

Parameters

- **atom_label** – string, a label of the atom which specie will be changed. 'all' if set up all species
- **attribute_name** – name of the attribute to change
- **number_to_add** – value to add or to multiply by
- **mode** – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add abs/absolute* adds the old value and *number_to_add*

Return type
None

Kwargs if the attribute_name does not correspond to a unique path:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

switch_kpointset(**args*, ***kwargs*)

Appends a *switch_kpointset()* to the list of tasks that will be done on the xmltree.

Switch the used k-point set

> **Warning:** This method is only supported for input versions after the Max5 release

Parameters
list_name – name of the kPoint set to use

Return type
None

switch_species(**args*, ***kwargs*)

Appends a *switch_species()* to the list of tasks that will be done on the xmltree.

Method to switch the species of an atom group of the fleur inp.xml file.

Parameters

- **new_species_name** – name of the species to switch to
- **position** – position of an atom group to be changed. If equals to 'all', all species will be changed
- **species** – atom groups, corresponding to the given species will be changed
- **clone** – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with *clone_species()*

- **changes** – changes to do if the species is cloned
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
None

switch_species_label(**args*, ***kwargs*)

Appends a *switch_species_label()* to the list of tasks that will be done on the xmltree.

Method to switch the species of an atom group of the fleur inp.xml file based on a label of a contained atom

Parameters

- **atom_label** – string, a label of the atom which group will be changed. 'all' to change all the groups
- **new_species_name** – name of the species to switch to
- **clone** – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with *clone_species()*
- **changes** – changes to do if the species is cloned

Return type
None

property task_list: list[tuple[str, dict[str, Any]]]

Return the current changes in a format accepted by *add_task_list()* and *fromList()*

undo(*revert_all=False*)

Cancels the last change or all of them

Parameters
revert_all (bool) – set True if need to cancel all the changes, False if the last one.

Return type
list[*ModifierTask*]

xml_create_tag(**args*, ***kwargs*)

Appends a *xml_create_tag()* to the list of tasks that will be done on the xmltree.

This method evaluates an xpath expression and creates a tag in a xmltree under the returned nodes. If there are no nodes under the specified xpath an error is raised.

The tag is appended by default, but can be inserted at a certain index (*place_index*) or can be inserted according to a given order of tags

Parameters

- **xpath** – a path where to place a new tag
- **element** – a tag name, etree Element or string representing the XML element to be created
- **place_index** – defines the place where to put a created tag
- **tag_order** – defines a tag order
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.
- **correct_order** – bool, if True (default) and a tag_order is given, that does not correspond to the given order in the xmltree (only order wrong no unknown tags) it will be corrected and a warning is given This is necessary for some edge cases of the xml schemas of fleur

- **several** – bool, if True multiple tags od the given name are allowed

Raises
ValueError – If the insertion failed in any way (tag_order does not match, failed to insert, ...)

Return type
None

xml_delete_att(**args*, ***kwargs*)

Appends a *xml_delete_att()* to the list of tasks that will be done on the xmltree.

Deletes an attribute in the XML tree

Parameters

- **xpath** – a path to the attribute to be deleted
- **name** – the name of an attribute to delete
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type
None

xml_delete_tag(**args*, ***kwargs*)

Appends a *xml_delete_tag()* to the list of tasks that will be done on the xmltree.

Deletes a tag in the XML tree.

Parameters

- **xpath** – a path to the tag to be deleted
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type
None

xml_replace_tag(**args*, ***kwargs*)

Appends a *xml_replace_tag()* to the list of tasks that will be done on the xmltree.

Replace XML tags by a given tag on the given XML tree

Parameters

- **xpath** – a path to the tag to be replaced
- **element** – an Element or string representing the Element to replace the found tags with
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type
None

xml_set_attrib_value_no_create(**args*, ***kwargs*)

Appends a *xml_set_attrib_value_no_create()* to the list of tasks that will be done on the xmltree.

Sets an attribute in a xmltree to a given value. By default the attribute will be set on all nodes returned for the specified xpath.

Parameters

- **xpath** – a path where to set the attributes
- **name** – the attribute name to set
- **value** – value or list of values to set (if not str they will be converted with *str(value)*)
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

Raises
ValueError – If the lengths of attribv or occurrences do not match number of nodes

Return type
None

`xml_set_text_no_create`(**args*, ***kwargs*)

Appends a *xml_set_text_no_create()* to the list of tasks that will be done on the xmltree.

Sets the text of a tag in a xmltree to a given value. By default the text will be set on all nodes returned for the specified xpath.

Parameters

- **xpath** – a path where to set the text
- **text** – value or list of values to set (if not str they will be converted with *str(value)*)
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

Raises
ValueError – If the lengths of text or occurrences do not match number of nodes

Return type
None

class masci_tools.io.fleurxmlmodifier.**ModifierTask**(*name*, *args*, *kwargs*)

args: tuple[Any, ...]

Alias for field number 1

kwargs: dict[str, Any]

Alias for field number 2

name: str

Alias for field number 0

Functions/Classes for loading/validating fleur XML files

This module provides the classes for easy access to information from the fleur input and output xsd schema files

masci_tools.io.parsers.fleur_schema.schema_dict.**F**

Generic Type variable for callable

alias of TypeVar('F', bound=Callable[[...], Any])

exception masci_tools.io.parsers.fleur_schema.schema_dict.**IncompatibleSchemaVersions**

Exception raised when it is known that a given output version and input version cannot be compiled into a complete fleur output xml schema

class masci_tools.io.parsers.fleur_schema.schema_dict.**InputSchemaDict**(**args*, *xmlschema=None*, ***kwargs*)

This class contains information parsed from the FleurInputSchema.xsd

The keys contain the following information:

inp_version
Version string of the input schema represented in this object

tag_paths
simple xpath expressions to all valid tag names Multiple paths or ambiguous tag names are parsed as a list

_basic_types
Parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions (Only used in the schema construction itself)

attrib_types
All possible base types for all valid attributes. If multiple are possible a list, with 'string' always last (if possible)

simple_elements
All elements with simple types and their type definition with the additional attributes

unique_attribs
All attributes and their paths, which occur only once and have a unique path

unique_path_attribs
All attributes and their paths, which have a unique path but occur in multiple places

other_attribs
All attributes and their paths, which are not in 'unique_attribs' or 'unique_path_attribs'

omitt_contained_tags
All tags, which only contain a list of one other tag

tag_info
For each tag (path), the valid attributes and tags (optional, several, order, simple, text)

classmethod fromPath(*path*)

load the FleurInputSchema dict for the specified FleurInputSchema file

Parameters
path (PathLike) – path to the input schema file

Return type
InputSchemaDict

Returns
InputSchemaDict object with the information for the provided file

classmethod fromVersion(*version*, *logger=None*, *no_cache=False*)

load the FleurInputSchema dict for the specified version

Parameters

- **version** (str) – str with the desired version, e.g. '0.33'
- **logger** (Optional[Logger]) – logger object for warnings, errors and information, ...

Return type
InputSchemaDict

Returns
InputSchemaDict object with the information for the provided version

property inp_version: `tuple[int, int]`

Returns the input version as an integer for comparisons (> or <)

exception `masci_tools.io.parsers.fleur_schema.schema_dict.`**`NoPathFound`**

Exception raised when no path is found for a given tag/attribute

exception `masci_tools.io.parsers.fleur_schema.schema_dict.`**`NoUniquePathFound`**

Exception raised when no unique path is found for a given tag/attribute

class `masci_tools.io.parsers.fleur_schema.schema_dict.`**`OutputSchemaDict`**`(`**args*, *xmlschema=None*, ***kwargs*`)`

This object contains information parsed from the FleurOutputSchema.xsd

The keys contain the following information:

out_version
Version string of the output schema represented in this class

input_tag
Name of the element containing the fleur input

iteration_tags
Names of the elements that can contain all iteration tags

tag_paths
simple xpath expressions to all valid tag names not in an iteration Multiple paths or ambiguous tag names are parsed as a list

iteration_tag_paths
simple relative xpath expressions to all valid tag names inside an iteration. Multiple paths or ambiguous tag names are parsed as a list

_basic_types
Parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions (Only used in the schema construction itself)

_input_basic_types
Part of the parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions from the input schema (Only used in the schema construction itself)

attrib_types
All possible base types for all valid attributes. If multiple are possible a list, with 'string' always last (if possible)

simple_elements
All elements with simple types and their type definition with the additional attributes

unique_attribs
All attributes and their paths, which occur only once and have a unique path outside of an iteration

unique_path_attribs
All attributes and their paths, which have a unique path but occur in multiple places outside of an iteration

other_attribs
All attributes and their paths, which are not in 'unique_attribs' or 'unique_path_attribs' outside of an iteration

iteration_unique_attribs
All attributes and their relative paths, which occur only once and have a unique path inside of an iteration

iteration_unique_path_attribs
All attributes and their relative paths, which have a unique path but occur in multiple places inside of an iteration

iteration_other_attribs
All attributes and their relative paths, which are not in 'unique_attribs' or 'unique_path_attribs' inside of an iteration

omitt_contained_tags
All tags, which only contain a list of one other tag

tag_info
For each tag outside of an iteration (path), the valid attributes and tags (optional, several, order, simple, text)

iteration_tag_info
For each tag inside of an iteration (relative path), the valid attributes and tags (optional, several, order, simple, text)

`classmethod fromPath`(*path*, *inp_path=None*, *inpschema_dict=None*)

load the FleurOutputSchema dict for the specified paths

Parameters

- **path** (PathLike) – path to the FleurOutputSchema file
- **inp_path** (Optional[PathLike]) – path to the FleurInputSchema file (defaults to same folder as path)

Return type
OutputSchemaDict

Returns
OutputSchemaDict object with the information for the provided files

`classmethod fromVersion`(*version*, *inp_version=None*, *logger=None*, *no_cache=False*)

load the FleurOutputSchema dict for the specified version

Parameters

- **version** (str) – str with the desired version, e.g. '0.33'
- **inp_version** (Optional[str]) – str with the desired input version, e.g. '0.33' (defaults to version)
- **logger** (Optional[Logger]) – logger object for warnings, errors and information, ...

Return type
OutputSchemaDict

Returns
OutputSchemaDict object with the information for the provided versions

`property inp_version: tuple[int, int]`

Returns the input version as an integer for comparisons (> or <)

iteration_attrib_xpath(*name*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the attribute and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **name** (str) – str, name of the attribute
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other
- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *iteration_tag_xpath()*
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath to the tag with the given attribute

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

iteration_tag_xpath(*name*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

property out_version: tuple[int, int]

Returns the output version as an integer for comparisons (> or <)

relative_iteration_attrib_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

Tries to find a unique relative path from the schema_dict based on the given name of the attribute name of the root, from which the path should be relative and additional further specifications

Parameters

- **schema_dict** – dict, containing all the path information and more
- **name** (str) – str, name of the attribute
- **root_tag** (str) – str, name of the tag from which the path should be relative
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other
- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *relative_iteration_tag_xpath()*
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

relative_iteration_tag_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found

- *NoUniquePathFound* – If multiple paths matching the criteria are found

class masci_tools.io.parsers.fleur_schema.schema_dict.**SchemaDict**(**args*, *xmlschema=None*, ***kwargs*)

Base class for schema dictionaries. Is locked on initialization with *freeze()*. Holds a reference to the xmlSchema for validating files.

Also provides interfaces for utility functions

Parameters
xmlschema (Optional[XMLSchema]) – etree.XMLSchema object for validating files

All other arguments are passed on to *LockableDict*

attrib_xpath(*name*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*)

Tries to find a unique path from the schema_dict based on the given name of the attribute and additional further specifications

Parameters

- **name** (str) – str, name of the attribute
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other
- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *tag_xpath()*

Return type
str

Returns
str, xpath to the tag with the given attribute

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

classmethod clear_cache()

Remove all stored entries in the schema dictionary cache

Return type
None

relative_attrib_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*)

Tries to find a unique relative path from the schema_dict based on the given name of the attribute name of the root, from which the path should be relative and additional further specifications

Parameters

- **schema_dict** – dict, containing all the path information and more
- **name** (str) – str, name of the attribute
- **root_tag** (str) – str, name of the tag from which the path should be relative

- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other
- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *relative_tag_xpath()*

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

relative_tag_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*)

Tries to find a unique relative path from the schema_dict based on the given name of the tag name of the root, from which the path should be relative and additional further specifications

Parameters

- **name** (str) – str, name of the tag
- **root_tag** (str) – str, name of the tag from which the path should be relative
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path

Return type
str

Returns
str, xpath for the given tag

Raises
ValueError – If no unique path could be found

tag_info(*name*, *contains=None*, *not_contains=None*, *parent=False*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications and returns the tag_info entry for this tag

Parameters

- **schema_dict** – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path

- **parent** (bool) – bool, if True the tag_info for the parent of the tag is returned

Return type
TagInfo

Returns
dict, tag_info for the found xpath

tag_xpath(*name*, *contains=None*, *not_contains=None*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications

Parameters

- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

validate(*xmltree*, *logger=None*, *header=''*)

Validate the given XML tree against the schema

Parameters

- **xmltree** (_ElementTree) – XML tree to validate
- **logger** (Optional[Logger]) – Logger to relay evlt warnings/errors

Return type
None

class masci_tools.io.parsers.fleur_schema.schema_dict.**SchemaDictDispatch**(**args*, ***kwargs*)

Protocol representing function decorated by the schema_dict_version_dispatch decorator

register(*min_version=Ellipsis*, *max_version=Ellipsis*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

Return type
Callable[[TypeVar(F, bound= Callable[..., Any])], TypeVar(F, bound= Callable[..., Any])]

masci_tools.io.parsers.fleur_schema.schema_dict.**list_available_versions**(*output_schema*)

List the available versions for the schema

Parameters
output_schema (bool) – bool, if True search for FleurOutputSchema.xsd otherwise FleurInputSchema.xsd

Return type
`list[str]`

Returns
list version string of the available versions

`masci_tools.io.parsers.fleur_schema.schema_dict.`**`schema_dict_version_dispatch`**(*output_schema=False*)

Decorator for creating variations of functions based on the inp/out version of the schema_dict. All functions here need to have the signature:

```
def f(node, schema_dict, *args, **kwargs):
    pass
```

So schema_dict is the second positional argument

Inspired by singledispatch in the functools module

Return type
`Callable[[TypeVar(F, bound= Callable[..., Any])], SchemaDictDispatch[TypeVar(F, bound= Callable[..., Any])]]`

This module provides easy functions for loading a input/output xml file of fleur and providing a parsed xml etree together with its corresponding schema dict

`masci_tools.io.fleur_xml.`**`FleurXMLContext`**(*etree_or_element*, *schema_dict*, *constants=None*, *logger=None*)

Contextmanager to hold the relevant datastructures for evaluating values from XML files

This holds:

Return type
`Generator[_EvalContext, None, None]`

The following methods are available:

- *`_EvalContext.attribute()`*: Evaluate attribute values (has an additional argument *default* to provide values for missing attributes)
- *`_EvalContext.text()`*: Evaluate text of tags
- *`_EvalContext.all_attributes()`*: Evaluate all attributes of a tag
- *`_EvalContext.parent_attributes()`*: Evaluate attribute values of parent of given tag
- *`_EvalContext.single_value()`*: Evaluate *value* and *unit* attribute of tag
- *`_EvalContext.tag_exists()`*: Evaluate whether a given tag exists
- *`_EvalContext.number_nodes()`*: Evaluate how many elements of the tag are present
- *`_EvalContext.attribute_exists()`*: Evaluate whether an attribute exists
- *`_EvalContext.simple_xpath()`*: Evaluate the simple xpath expression for a given tag
- *Nested Context* *`_EvalContext.find()`*: Find the first occurrence of the tag and provide a nested context to that element
- *Nested Context* *`_EvalContext.iter()`*: Find all occurrences of the tag and provide a nested context to these elements when iterated over
- *Nested Context* *`_EvalContext.nested()`*: Explicitly inherit all the context except the XML tree to a new context with the XML element replaced by the given argument

Example Usage:

```
from masci_tools.io.fleur_xml import load_inpxml, FleurXMLContext
xmltree, schema_dict = load_inpxml('/path/to/inp.xml')

with FleurXMLContext(xmltree, schema_dict) as root:
    spins = root.attribute('jspins')
    noco = root.attribute('l_noco', default=False)

    #Not nesting the context we need to specify which elements are meant
    mt_radii = root.attribute('radius', contains='species')

    #Nesting using find
    with root.find('atomspecies') as all_species:
        mt_radii = all_species.attribute('radius')

    #Nesting using iter
    mt_radii = []
    for species in root.iter('species'):
        mt_radii.append(species.attribute('radius'))
```

class masci_tools.io.fleur_xml.**_EvalContext**(*etree_or_element*, *schema_dict*, *constants=None*, *logger=None*)

Contextmanager to hold the relevant datastructures for evaluating values from XML files

This holds:

all_attributes(*name*, ***kwargs*)

Alias for *evaluate_tag()*

Parameters

- **name** (str) – str, name of the tag
- **subtags** – optional bool, if True the subtags of the given tag are evaluated
- **text** – optional bool, if True the text of the tag is also parsed
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type

Any

Kwargs:

param contains

str, this string has to be in the final path

param not_contains

str, this string has to NOT be in the final path

param only_required

bool (optional, default False), if True only required attributes are parsed

param ignore
list of str (optional), attributes not to parse

param list_return
if True, the returned quantity is always a list even if only one element is in it

param strict_missing_error
if True, and no logger is given an error is raised if any attribute is not found

`attribute`(*name*, *default=None*, ***kwargs*)

Alias for `evaluate_attribute()`

Parameters

- **name** (`str`) – str, name of the attribute
- **default** (`Optional[Any]`) – value to return, if there are no values found
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

param list_return
if True, the returned quantity is always a list even if only one element is in it

param optional
bool, if True and no logger given none or an empty list is returned

Return type
`Any`

Returns
list or single value, converted in convert_xml_attribute

`attribute_exists`(*name*, ***kwargs*)

Alias for `attrib_exists()`

Parameters

- **name** (`str`) – str, name of the attribute
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

Return type
bool

Returns
bool, True if any tag with the attribute exists

find(*name*, ***kwargs*)

Finds the first element for the given name and constraints and gives a nested context for this element, i.e. inheriting the schema_dict, constants and logger

Example:

```
with FleurXMLContext(xmltree, schema_dict) as root:
    #Operations happen on the root here
    jspins = root.attribute('jspins')
    with root.find('species') as species:
        #Operations in this block happen on the first species node
        radius = species.attribute('radius')
```

Parameters

- **name** (str) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
_GeneratorContextManager[*_EvalContext*]

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

iter(*name*, ***kwargs*)

Finds all elements for the given name and constraints and gives nested contexts for these elements to be iterated over, i.e. inheriting the schema_dict, constants and logger

Example:

```
with FleurXMLContext(xmltree, schema_dict) as root:
    #Operations happen on the root here
    jspins = root.attribute('jspins')
    for species in root.iter('species'):
        #Operations happen on one species node in each iteration of the
        #for loop. The order is the same as they appear in the XML file
        radius = species.attribute('radius')
```

Parameters

- **name** (str) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
Generator[*_EvalContext*, None, None]

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

nested(*etree_or_element*)

Create a nested context from the current one inheriting the schema_dict, constants and logger only replacing the XML element

Example of explicit usage:

```
with FleurXMLContext(xmltree, schema_dict) as root:
    #Operations happen on the root here
    jspins = root.attribute('jspins')

    all_species_tag = root.simple_xpath('atomspecies')
    with root.nested(all_species_tag) as all_species:
        #Operations happen on the 'atomSpecies' tag
        mt_radii = all_species.attribute('radius')
```

Parameters
etree_or_element (Union[_Element, _ElementTree]) – Element to use for evaluation in the nested context

Return type
Generator[*_EvalContext*, None, None]

number_nodes(*name*, ***kwargs*)

Alias of *get_number_of_nodes()*

Parameters

name (str) – str, name of the tag

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param iteration_path
bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters
Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
int

Returns
number of nodes of the given tag

parent_attributes(*name*, ***kwargs*)

Alias for *evaluate_parent_tag()*

Parameters

- **name** (str) – str, name of the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param only_required
bool (optional, default False), if True only required attributes are parsed

param ignore
list of str (optional), attributes not to parse

param list_return
if True, the returned quantity is always a list even if only one element is in it

param strict_missing_error
if True, and no logger is given an error is raised if any attribute is not found

Return type
`Any`

Returns
dict, with attribute values converted via convert_xml_attribute

`simple_xpath`(*name*, ***kwargs*)

Alias for `eval_simple_xpath()`

Parameters

- **name** (`str`) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details
- **list_return** – bool, if True a list is always returned

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
`list[_Element] | _Element`

Returns
etree Elements obtained via the simple xpath expression

`single_value`(*name*, ***kwargs*)

Alias for `evaluate_single_value_tag()`

Parameters

- **name** (`str`) – str, name of the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param only_required
bool (optional, default False), if True only required attributes are parsed

param ignore
list of str (optional), attributes not to parse

param list_return
if True, the returned quantity is always a list even if only one element is in it

param strict_missing_error
if True, and no logger is given an error is raised if any attribute is not found

param iteration_path
bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters
Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
Any

Returns
value and unit, both converted in convert_xml_attribute

tag_exists(*name*, ***kwargs*)

Alias for *tag_exists()*

Parameters

- **name** (str) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
bool

Returns
bool, True if any tag exists

text(*name*, ***kwargs*)

Alias for *evaluate_text()*

Parameters

- **name** (str) – str, name of the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param list_return
if True, the returned quantity is always a list even if only one element is in it

param optional
bool, if True and no logger given none or an empty list is returned

Return type
`Any`

Returns
list or single value, converted in convert_xml_text

`masci_tools.io.fleur_xml.get_constants(xmltree, schema_dict, logger=None)`

Reads in the constants defined in the inp.xml and returns them combined with the predefined constants from fleur as a dictionary

Parameters

- **root** – root of the etree of the inp.xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema_dictionary of the version of the file to read (inp.xml or out.xml)
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
`dict[str, float]`

Returns
a python dictionary with all defined constants

`masci_tools.io.fleur_xml.load_inpxml(inpxmlfile, logger=None, base_url=None, **kwargs)`

Loads a inp.xml file for fleur together with its corresponding schema dictionary

Parameters
inpxmlfile (Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the inp.xml file, opened file handle (in bytes modes i.e. rb) or a xml etree to be parsed

Return type
`tuple[_ElementTree, InputSchemaDict]`

Returns
parsed xmltree of the inpxmlfile and the schema dictionary for the corresponding input version

`masci_tools.io.fleur_xml.load_outxml(outxmlfile, logger=None, base_url=None, **kwargs)`

Loads a out.xml file for fleur together with its corresponding schema dictionary. Also returns whether the XML file had to be parsed with *recover=True*

Parameters
outxmlfile (Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the out.xml file, opened file handle (in bytes modes i.e. rb) or a xml etree to be parsed

Return type
`tuple[_ElementTree, OutputSchemaDict]`

Returns
parsed xmltree of the outxmlfile and the schema dictionary for the corresponding output version

`masci_tools.io.fleur_xml.load_outxml_and_check_for_broken_xml(outxmlfile, logger=None, base_url=None, **kwargs)`

Loads a out.xml file for fleur together with its corresponding schema dictionary. Also returns whether the XML file had to be parsed with *recover=True*

Parameters
outxmlfile (Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the out.xml file, opened file handle (in bytes modes i.e. rb) or a xml etree to be parsed

Return type
tuple[_ElementTree, *OutputSchemaDict*, bool]

Returns
parsed xmltree of the outxmlfile and the schema dictionary for the corresponding output version and bool indicating whether the outxml is broken

Helper functions for the `n_mmp_mat` file

Simple IO routines for creating text for nmmp_mat files

`masci_tools.io.io_nmmpmat.format_nmmpmat(denmat)`

Format a given 7x7 complex numpy array into the format for the n_mmp_mat file

Results in list of 14 strings. Every 2 lines correspond to one row in array Real and imaginary parts are formatted with 20.13f in alternating order

Parameters
denmat (ndarray) – numpy array (7x7) and complex for formatting

Raises
ValueError – If denmat has wrong shape or datatype

Return type
list[str]

Returns
list of str formatted in lines for the n_mmp_mat file

`masci_tools.io.io_nmmpmat.read_nmmpmat_block(nmmp_lines, block_index)`

Convert 14 line block of given nmmp_lines into 7x7 complex numpy array

Parameters

- **nmmplines** – list of lines in the n_mmp_mat file
- **block_index** (int) – int specifying which 14 line block to convert

Return type
ndarray

Returns
7x7 complex numpy array of the numbers in the given block

`masci_tools.io.io_nmmpmat.rotate_nmmpmat_block(denmat, orbital, phi=None, theta=None, inverse=False)`

Rotate the given 7x7 complex numpy array with the d-wigner matrix corresponding to the given orbital and angles

Parameters

- **denmat** (ndarray) – complex numpy array of shape 7x7
- **orbital** (int) – int of the orbital for the current block
- **phi** (Optional[float]) – float, angle (radian), by which to rotate the density matrix
- **theta** (Optional[float]) – float, angle (radian), by which to rotate the density matrix

Return type
ndarray

Returns
denmat rotated by the d-wigner matrix

masci_tools.io.io_nmmpmat.**write_nmmpmat**(*orbital*, *denmat*, *phi=None*, *theta=None*, *inverse=False*)

Generate list of str for n_mmp_mat file from given numpy array

Parameters

- **orbital** (int) – int of the orbital for the current block
- **denmat** (ndarray) – complex numpy array of shape (2*orbital+1 x 2*orbital+1) with the wanted occupations
- **phi** (Optional[float]) – float, angle (radian), by which to rotate the density matrix
- **theta** (Optional[float]) – float, angle (radian), by which to rotate the density matrix

Return type
list[str]

Returns
list of str formatted in lines for the n_mmp_mat file

masci_tools.io.io_nmmpmat.**write_nmmpmat_from_orbitals**(*orbital*, *orbital_occupations*, *phi=None*, *theta=None*, *inverse=False*)

Generate list of str for n_mmp_mat file from orbital occupations

orbital occupations are provided in the following order (expressed as the spherical harmonics since it can be used for all orbitals):

- Y_l^0
- $\frac{1}{\sqrt{2}}\left(Y_l^{-1} + Y_l^1\right)$
- $\frac{i}{\sqrt{2}}\left(Y_l^{-1} - Y_l^1\right)$
- $\frac{1}{\sqrt{2}}\left(Y_l^{-2} + Y_l^2\right)$
- $\frac{i}{\sqrt{2}}\left(Y_l^{-2} - Y_l^2\right)$
- and so on …

Parameters

- **orbital** (int) – int of the orbital for the current block
- **orbital_occupations** (list[float]) – list like with length 2*orbital+1 with the occupations of the orbitals

- **phi** (Optional[float]) – float, angle (radian), by which to rotate the density matrix
- **theta** (Optional[float]) – float, angle (radian), by which to rotate the density matrix

Return type
list[str]

Returns
list of str formatted in lines for the *n_mmp_mat* file

masci_tools.io.io_nmmpmat.**write_nmmpmat_from_states**(*orbital*, *state_occupations*, *phi=None*, *theta=None*, *inverse=False*)

Generate list of str for n_mmp_mat file from diagonal occupations

Parameters

- **orbital** (int) – int of the orbital for the current block
- **state_occupations** (list[float]) – list like with length 2*orbital+1 with the occupations of the diagonals
- **phi** (Optional[float]) – float, angle (radian), by which to rotate the density matrix
- **theta** (Optional[float]) – float, angle (radian), by which to rotate the density matrix

Return type
list[str]

Returns
list of str formatted in lines for the n_mmp_mat file

General HDF5 parser

This module contains a generic HDF5 reader

class masci_tools.io.parsers.hdf5.reader.**AttribTransformation**(*name*, *attrib_name*, *args*, *kwargs*)

args: tuple[Any, ...]
Alias for field number 2

attrib_name: str
Alias for field number 1

kwargs: dict[str, Any]
Alias for field number 3

name: str
Alias for field number 0

class masci_tools.io.parsers.hdf5.reader.**HDF5LimitedTransformation**

class masci_tools.io.parsers.hdf5.reader.**HDF5Reader**(*file*, *move_to_memory=True*, *filename='UNKNOWN'*)

Class for reading in data from hdf5 files using a specified recipe

Parameters

- **file** (Union[str, bytes, Path, PathLike, IO[Any]]) – filepath to hdf file or opened file handle (mode 'rb')

- **move_to_memory** (bool) – bool if True after reading and transforming the data all leftover h5py.Datasets are moved into np.arrays
- **filename** (str) – Name of the file. Only used for logging. If not given and the file provides the information extract it from there

The recipe is passed to the *HDF5Reader.read()* method and consists of a dict specifying which attributes and datasets to read in and how to transform them

Each attribute/dataset entry corresponds to one entry point in the given *.hdf* file Available transformations can either be found in *transforms* or can be defined by the user with the *hdf5_transformation()* decorator

Basic Usage:

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
import masci_tools.io.parsers.hdf5.recipes as recipes

#This example shows the usage for producing data from a bandstructure calculation
#in Fleur
with HDF5Reader('/path/to/hdf/banddos.hdf') as h5reader:
    data, attributes = h5reader.read(recipe=recipes.FleurBands)
print(data, attributes)
```

read(*recipe=None*)

Extracts datasets from HDF5 file, transforms them and puts all into a namedtuple.

Parameters
recipe (Optional[*HDF5Recipe*]) – dict with the format given in *recipes*

Return type
tuple[dict[str, Any], dict[str, Any]]

Returns
two dicts with the datasets/attributes read in and transformed according to the recipe

class masci_tools.io.parsers.hdf5.reader.**HDF5Recipe**

class masci_tools.io.parsers.hdf5.reader.**HDF5Transformation**

class masci_tools.io.parsers.hdf5.reader.**Transformation**(*name*, *args*, *kwargs*)

args: tuple[Any, ...]

Alias for field number 1

kwargs: dict[str, Any]

Alias for field number 2

name: str

Alias for field number 0

This module defines commonly used recipes for the *HDF5Reader*

Available are:

- Recipe for bandstructure calculations with Fleur
- Recipes for almost all DOS calculation modes of Fleur

A Recipe is a python dictionary in a specific format.

A Template Example:

```
from masci_tools.io.parser.hdf5.readers import Transformation, AttribTransformation

RecipeExample = {
    'datasets': {
        'example_dataset': {
            'h5path': '/path/in/hdf/file',
            'transforms': [Transformation(name='get_first_element')]
        },
        'example_attrib_transform': {
            'h5path': '/other/path/in/hdf/file',
            'transforms': [AttribTransformation(name='multiply_by_attribute', attrib_
↪name='example_attribute')]
        }
    },
    'attributes': {
        'example_attribute': {
            'h5path':
            '/path/in/hdf/file',
            'transforms':
            [Transformation(name='get_attribute', args=('attribName',)),
             Transformation(name='get_first_element')]
        }
    }
}
```

The Recipe consists of two sections 'datasets' and 'attributes'. All data from these two sections will be returned in separate python dictionaries by the *HDF5Reader* class

Each entry in those sections has to have a *h5path* entry, which will specify the dataset to initially read from the hdf file. Then each entry can define a entry *transforms* with a list of the namedtuples imported at the top of the code example. These corresponds to function calls to functions in *transforms* to transform the read in data

Entries in the *attributes* section are read and transformed first and can subsequently be used in transformations for the *datasets*. These correpsond to the transforms created with the *AttribTransformation* namedtuple instead of *Transformation*.

`masci_tools.io.parsers.hdf5.recipes.`**`bands_recipe_format`**(*group*, *simple=False*)

Format for bandstructure calculations retrieving weights from the given group

Parameters

- **group** (`Literal['Local', 'jDOS', 'Orbcomp', 'MCD']`) – str of the group the weights should be taken from
- **simple** (`bool`) – bool, if True no additional weights are retrieved with the produced recipe

Return type
HDF5Recipe

Returns
dict of the recipe to retrieve a bandstructure calculation

`masci_tools.io.parsers.hdf5.recipes.`**`dos_recipe_format`**(*group*)

Format for denisty of states calculations retrieving the DOS from the given group

Parameters
group (`Literal['Local', 'jDOS', 'Orbcomp', 'MCD']`) – str of the group the DOS should be taken from

Return type
HDF5Recipe

Returns
dict of the recipe to retrieve a DOS calculation

masci_tools.io.parsers.hdf5.recipes.**get_fleur_bands_specific_weights**(*weight_name*, *group='Local'*)

Recipe for bandstructure calculations only retrieving one additional weight besides the eigenvalues and kpath

Parameters

- **weight_name** (str | list[str]) – key or list of keys of the weight(s) to retrieve
- **group** (Literal['Local', 'jDOS', 'Orbcomp', 'MCD']) – optional str (default Local) name of the group from where to take the weights

Return type
HDF5Recipe

Returns
dict of the recipe to retrieve a simple bandstructure plus the one specified weight

Collection of predefined transformations for the *HDF5Reader* class

All Transformation have to be able to handle (or fail gracefully with a clear error) for the following 3 cases:

1. The dataset is still a h5py.Dataset and might need to be transformed to a numpy array
2. The dataset is a numpy array
3. The dataset is a dict. This is needed to read arbitrary child dataset, where not all labels are known. Two options can be chosen apply the transformation to all keys in the dict or throw an error

masci_tools.io.parsers.hdf5.transforms.**F**

Generic Callable type

alias of TypeVar('F', bound=Callable[[...], Any])

exception masci_tools.io.parsers.hdf5.transforms.**HDF5TransformationError**

masci_tools.io.parsers.hdf5.transforms.**add_partial_sums**(*dataset*, *attribute_value*, *pattern_format*, *make_set=False*, *replace_format=None*)

Add entries to the dataset dict (Only available for dict datasets) with sums over entries containing a given pattern formatted with a attribute_value

Used for example in the FleurBands recipe to calculate total atom weights with the pattern_format *'MT:{}'.format* and the atomtype as the attribute_value

Parameters

- **dataset** – dataset to transform
- **attribute_value** – value to multiply by (attribute value passed in from *_transform_dataset*)
- **pattern_format** – callable returning a formatted string This will be called with every entry in the attribute_value list
- **replace_format** – callable returning a formatted string This will be called with every entry in the attribute_value list

Returns
dataset with new entries containing the sums over entries matching the given pattern

masci_tools.io.parsers.hdf5.transforms.**add_partial_sums_fixed**(*dataset*, *patterns*, *replace_entries=None*)

Add entries to the dataset dict (Only available for dict datasets) with sums over entries containing a given pattern

Used for example in the FleurBands recipe to calculate total atom weights with the patterns *['MT:1', 'MT:2', ...]*

Parameters

- **dataset** – dataset to transform
- **patterns** – list of str to sum entries over
- **replace_entries** – list of str under which to enter the entries back

Returns
dataset with new entries containing the sums over entries matching the given pattern

masci_tools.io.parsers.hdf5.transforms.**apply_lambda**(*dataset*, *lambda_func*)

Applies a given lambda function to the dataset
This should be used with care. One possible example is converting to a boolean with lambda x: x==1

Parameters

- **dataset** – dataset to transform
- **lambda_func** – lambda function to apply to the dataset

Returns
return value of the lambda function

masci_tools.io.parsers.hdf5.transforms.**attributes**(*dataset*)

Extracts all attributes of the dataset

Parameters
dataset – dataset to transform

Returns
dict with all the set attributes on the dataset

masci_tools.io.parsers.hdf5.transforms.**calculate_norm**(*dataset*, *between_neighbours=False*)

Calculate norms on the given dataset. Calculates the norm of each row in the dataset

Parameters

- **dataset** – dataset to transform
- **between_neighbours** – bool, if True the distance between subsequent entries in the dataset is calculated

Returns
norms of the given dataset

masci_tools.io.parsers.hdf5.transforms.**convert_to_complex_array**(*dataset*)

Converts the given dataset of real numbers into dataset of complex numbers. This follows the convention of how complex numbers are normally written out by Fleur (last index 0 real part, last index 1 imag part)

Parameters
dataset – dataset to transform

Returns
dataset with complex values

`masci_tools.io.parsers.hdf5.transforms.convert_to_str(dataset, join=False)`

Converts the given dataset to a numpy array of type string

Parameters

- **dataset** – dataset to transform
- **join** – bool if True the result will be joined together

Returns
numpy array of dtype str

`masci_tools.io.parsers.hdf5.transforms.cumulative_sum(dataset, beginning_zero=True)`

Calculate the cumulative sum of the dataset

Parameters
dataset – dataset to transform

Returns
cumulative sum of the dataset

`masci_tools.io.parsers.hdf5.transforms.flatten_array(dataset, order='C')`

Flattens the given dataset to one dimensional array. Copies the array !!

Parameters

- **dataset** – dataset to transform
- **order** – str $\{C, F, A, K\}$ flatten in column major or row-major order (see numpy.flatten documentation)

Returns
flattened dataset

`masci_tools.io.parsers.hdf5.transforms.get_all_child_datasets(group, ignore=None, contains=None)`

Get all datasets contained in the given group

Parameters

- **group** – h5py object to extract from
- **ignore** – str or iterable of str (optional). These keys will be ignored
- **contains** – str or iterable of str (optional). This phrase has to be in the key

Returns
a dict with the contained dataset entered with their names as keys

`masci_tools.io.parsers.hdf5.transforms.get_attribute(dataset, attribute_name)`

Extracts a specified attribute's value.

Parameters

- **dataset** – dataset to transform
- **attribute_name** – str of the attribute to extract from the dataset

Returns
value of the attribute on the dataset

`masci_tools.io.parsers.hdf5.transforms.get_first_element(dataset)`

Get the first element of the dataset.

Parameters
dataset – dataset to transform

Returns
first element of the dataset

masci_tools.io.parsers.hdf5.transforms.**get_name**(*dataset*, *full_path=False*)

Get the name of the dataset.

Parameters

- **dataset** – dataset to get the shape
- **full_path** – bool, if True the full path to the dataset is returned

Returns
name of the dataset

masci_tools.io.parsers.hdf5.transforms.**get_shape**(*dataset*)

Get the shape of the dataset.

Parameters
dataset – dataset to get the shape

Returns
shape of the dataset

masci_tools.io.parsers.hdf5.transforms.**hdf5_transformation**(*, *attribute_needed*)

Decorator for registering a function as a transformation functions on the *HDF5Reader* class

Parameters
attribute_needed (bool) – bool if True this function takes a previously processed attribute value and is therefore only available for the entries in datasets

Return type
Callable[[TypeVar(F, bound= Callable[..., Any])], TypeVar(F, bound= Callable[..., Any])]

masci_tools.io.parsers.hdf5.transforms.**index_dataset**(*dataset*, *index*)

Get the n-th element of the dataset.

Parameters
dataset – dataset to transform

Returns
first element of the dataset

masci_tools.io.parsers.hdf5.transforms.**merge_subgroup_datasets**(*group*, *ignore=None*, *contains=None*, *ignore_group=None*, *contains_group=None*, *stack_results=True*, *sort_key=None*)

Get all datasets contained in the given group

Parameters

- **group** – h5py object to extract from
- **ignore_group** – str or iterable of str (optional). These keys will be ignored
- **contains_group** – str or iterable of str (optional). This phrase has to be in the key

- **ignore** – str or iterable of str (optional). These keys of the datasets in the subgroup will be ignored
- **contains** – str or iterable of str (optional). This phrase has to be in the key of the datasets in the subgroup
- **stack_results** – bool if True the resulting list of datasets will be used to construct one numpy array

Returns
a dict with the contained dataset of the subgroups of the given group entered with their names as keys

`masci_tools.io.parsers.hdf5.transforms.move_to_memory(dataset)`

Moves the given dataset to memory, if it's not already there Creates numpy arrays for each dataset it finds

Parameters
dataset – dataset to transform

Returns
dataset with h5py.Datasets converted to numpy arrays

`masci_tools.io.parsers.hdf5.transforms.multiply_array(dataset, matrix, transpose=False)`

Multiply the given dataset with a matrix

Parameters

- **dataset** – dataset to multiply
- **matrix** – matrix to multiply by
- **transpose** – bool, if True the given matrix is transposed

Returns
dataset multiplied with the given matrix

`masci_tools.io.parsers.hdf5.transforms.multiply_by_attribute(dataset, attribute_value, transpose=False)`

Multiply the given dataset with a previously parsed attribute, either scalar or matrix like

Parameters

- **dataset** – dataset to transform
- **attribute_value** – value to multiply by (attribute value passed in from *_transform_dataset*)

Only relevant for matrix multiplication: :type transpose:

param transpose
bool if True the Matrix order is transposed before multiplying

Returns
dataset multiplied with the given attribute_value

`masci_tools.io.parsers.hdf5.transforms.multiply_scalar(dataset, scalar_value)`

Multiply the given dataset with a scalar_value

Parameters

- **dataset** – dataset to transform
- **scalar_value** – value to multiply the dataset by

Returns
the dataset multiplied by the scalar if it is a dict all entries are multiplied

masci_tools.io.parsers.hdf5.transforms.**periodic_elements**(*dataset*)

Converts the given dataset (int or list of ints)
To the atomic symbols corresponding to the atomic number

Parameters
dataset – dataset to transform

Returns
str or array of str with the atomic elements

masci_tools.io.parsers.hdf5.transforms.**repeat_array**(*dataset*, *n_repeats*)

Use numpy.repeat to repeat each element in array n-times

Parameters

- **dataset** – dataset to transform
- **n_repeats** – int, time to repeat each element

Returns
dataset with elements repeated n-times

masci_tools.io.parsers.hdf5.transforms.**repeat_array_by_attribute**(*dataset*, *attribute_value*)

Use numpy.repeat to repeat each element in array n-times (given by attribute_value)

Parameters

- **dataset** – dataset to transform
- **attribute_shape** – int, time to repeat the elements in the given array

Returns
dataset with elements repeated n-times

masci_tools.io.parsers.hdf5.transforms.**shift_by_attribute**(*dataset*, *attribute_value*, *negative=False*)

Shift the dataset by the given value of the attribute

Parameters

- **dataset** – dataset to transform
- **attribute_value** – value to shift the dataset by
- **negative** – bool, if True the scalar_value will be subtracted

Returns
the dataset shifted by the scalar if it is a dict all entries are shifted

masci_tools.io.parsers.hdf5.transforms.**shift_dataset**(*dataset*, *scalar_value*, *negative=False*)

Shift the dataset by the given scalar_value

Parameters

- **dataset** – dataset to transform
- **scalar_value** – value to shift the dataset by
- **negative** – bool, if True the scalar_value will be subtracted

Returns
the dataset shifted by the scalar if it is a dict all entries are shifted

`masci_tools.io.parsers.hdf5.transforms.slice_dataset(dataset, slice_arg)`

Slice the dataset with the given slice argument.

Parameters

- **dataset** – dataset to transform
- **slice_arg** – slice to apply to the dataset

Returns
first element of the dataset

`masci_tools.io.parsers.hdf5.transforms.split_array(dataset, suffixes=None, name=None)`

Split the arrays in a dataset into multiple entries by their first index

If the dataset is a dict the entries will be split up. If the dataset is not a dict a dict is created with the dataset entered under *name* and this will be split up

Parameters

- **dataset** – dataset to transform
- **suffix** – Optional list of str to use for suffixes for the split up entries. by default it is the value of the first index of the original array
- **name** – str for the case of the dataset not being a dict. Key for the entry in the new dict for the original dataset. The returned dataset will only contain the split up entries
- **dataset** – dict with the entries split up

`masci_tools.io.parsers.hdf5.transforms.stack_datasets(dataset, axis=0, sort_key=None)`

Stack the entries in the given dict dataset along the given axis

Parameters

- **dataset** – dict dataset to transform
- **axis** – int along which axis should be stacked

Returns
the array resulting from stacking all entries in the dictionary

`masci_tools.io.parsers.hdf5.transforms.sum_over_dict_entries(dataset, overwrite_dict=False, entries=None, dict_entry='sum', entry_format=None)`

Sum the datasets contained in the given dict dataset

Parameters

- **dataset** – dataset to transform
- **overwrite_dict** – bool if True, the result will overwrite the dictionary if False it is entered under *sum* in the dict

Returns
dataset with summed entries

`masci_tools.io.parsers.hdf5.transforms.tile_array(dataset, n_repeats)`

Use numpy.tile to repeat array n-times

Parameters

- **dataset** – dataset to transform
- **attribute_shape** – int, time sto repeat the given array

Returns
dataset repeated n-times

masci_tools.io.parsers.hdf5.transforms.**tile_array_by_attribute**(*dataset*, *attribute_value*)

Use numpy.tile to repeat array n-times (given by attribute_value)

Parameters

- **dataset** – dataset to transform
- **attribute_shape** – int, time sto repeat the given array

Returns
dataset repeated n-times

Definition of default parsing tasks for fleur out.xml

This module contains the dictionary with all defined tasks for the outxml_parser. The entries in the TASK_DEFINITION dict specify how to parse specific attributes tags.

This needs to be maintained if the specifications do not work for a new schema version because of changed attribute names for example.

Each entry in the TASK_DEFINITION dict can contain a series of keys, which by default correspond to the keys in the output dictionary

The following keys are expected in each entry:

param parse_type
str, defines which methods to use when extracting the information

param subdict
str, if present the parsed values are put into this key in the output dictionary

param overwrite_last
bool, if True no list is inserted and each entry overwrites the last

If *parse_type* is not equal to xmlGetter the following key is required:

param path_spec
dict with all the arguments that should be passed to tag_xpath or attrib_xpath to get the correct path

param kwargs
additional arguments to pass to the parsing function

In the case of xmlGetter the following keys are allowed:

param name
name of the function in *masci_tools.util.xml.xml_getters* (required)

param result_names
list of str defining the keys under which to enter the outputs of the function

For the allAttribs parse_type there are more keys that can appear:

param base_value
str, optional. If given the attribute with this name will be inserted into the key from the task_definition all other keys are formatted as {task_key}_{attribute_name}

param overwrite
list of str, these attributes will not create a list and overwrite any value that might be there

param flat
bool, if False the dict parsed from the tag is inserted as a dict into the correspondin key if True the values will be extracted and put into the output dictionary with the format {task_key}_{attribute_name}

Each task entry can have additional keys to specify, when to perform the task. These are denoted with underscores in their names and are all optional:

param _general
bool, default False. If True the parsing is not performed for each iteration on the iteration node but beforehand and on the root node

param _modes
list of tuples, sets conditions for the keys in fleur_modes to perform the task .e.g. [('jspins', 2), ('soc', True)] means only perform this task for a magnetic soc calculation

param _minimal
bool, default False, denotes task to perform when minimal_mode=True is passed to the parser

param _special
bool, default False, If true these tasks are not added by default and need to be added manually

param _conversions
list of str, gives the names of functions in fleur_outxml_conversions to perform after parsing

The following keys are special at the moment:

- `fleur_modes` specifies how to identify the type of the calculation (e.g. SOC, magnetic, lda+u) this is used to determine, whether additional things should be parsed

Following is the current specification of tasks

```
from masci_tools.util.parse_utils import Conversion

__working_out_versions__ = {'0.34', '0.35', '0.36', '0.37'}
__base_version__ = '0.34'

TASKS_DEFINITION = {
    #--------Definitions for general info from outfile (start, endtime, number_
↪iterations)--------
    'general_out_info': {
        '_general': True,
        '_minimal': True,
        '_conversions': [Conversion(name='calculate_walltime')],
        'creator_name': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'version',
                'not_contains': 'git'
            }
        },
        'creator_target_architecture': {
```

(continues on next page)

(continued from previous page)

```
            'parse_type': 'text',
            'path_spec': {
                'name': 'targetComputerArchitectures'
            }
        },
        'output_file_version': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'fleurOutputVersion'
            }
        },
        'number_of_iterations': {
            'parse_type': 'numberNodes',
            'path_spec': {
                'name': 'iteration'
            }
        },
        'number_of_atoms': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'nat'
            }
        },
        'number_of_atom_types': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'ntype'
            }
        },
        'number_of_kpoints': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'count',
                'contains': 'numericalParameters'
            }
        },
        'start_date': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'startDateAndTime'
            },
            'flat': False,
            'kwargs': {
                'ignore': ['zone'],
            }
        },
        'end_date': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'endDateAndTime'
            },
            'flat': False,
```

(continues on next page)

(continued from previous page)

```
            'kwargs': {
                'ignore': ['zone'],
            }
        }
    },
    #--------Defintions for general info from input section of outfile (kmax, symmetries, ..)--------
    'general_inp_info': {
        '_general': True,
        '_minimal': True,
        'title': {
            'parse_type': 'text',
            'path_spec': {
                'name': 'comment'
            }
        },
        'kmax': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'Kmax'
            }
        },
        'gmax': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'Gmax'
            }
        },
        'number_of_spin_components': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'jspins'
            }
        },
        'number_of_symmetries': {
            'parse_type': 'numberNodes',
            'path_spec': {
                'name': 'symOp'
            }
        },
        'number_of_species': {
            'parse_type': 'numberNodes',
            'path_spec': {
                'name': 'species'
            }
        },
        'film': {
            'parse_type': 'exists',
            'path_spec': {
                'name': 'filmPos'
            }
        },
```

(continues on next page)

(continued from previous page)

```
    },
    #--------Defintions for lda+u info from input section (species, ldau tags)--------
    'ldau_info': {
        '_general': True,
        '_modes': [('ldau', True)],
        '_conversions': [Conversion(name='convert_ldau_definitions')],
        'parsed_ldau': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'ldaU',
                'contains': 'species'
            },
            'subdict': 'ldau_info',
            'flat': False,
            'kwargs': {
                'only_required': True
            }
        },
        'ldau_species': {
            'parse_type': 'parentAttribs',
            'path_spec': {
                'name': 'ldaU',
                'contains': 'species'
            },
            'subdict': 'ldau_info',
            'flat': False,
            'kwargs': {
                'only_required': True
            }
        }
    },
    'ldahia_info': {
        '_general': True,
        '_modes': [('ldahia', True)],
        '_conversions': [Conversion(name='convert_ldahia_definitions')],
        'parsed_ldahia': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'ldaHIA',
                'contains': 'species'
            },
            'subdict': 'ldahia_info',
            'flat': False,
        },
        'ldahia_species': {
            'parse_type': 'parentAttribs',
            'path_spec': {
                'name': 'ldaHIA',
                'contains': 'species'
            },
            'subdict': 'ldahia_info',
            'flat': False,
```

(continues on next page)

(continued from previous page)

```
            'kwargs': {
                'only_required': True
            }
        }
    },
    #--------Defintions for relaxation info from input section (bravais matrix, atompos)
    #--------for Bulk and film
    'relax_info': {
        '_general': True,
        '_modes': [
            ('relax', True),
        ],
        '_conversions': [Conversion(name='convert_relax_info')],
        'parsed_relax_info': {
            'parse_type': 'xmlGetter',
            'name': 'get_structuredata',
            'kwargs': {
                'convert_to_angstroem': False,
                'include_relaxations': False
            }
        }
    },
    'hubbard1_distances': {
        '_general': True,  #General because not every iteration contains a hubbard1␣
↪iteration
        '_minimum_version': '0.35',
        '_minimal': True,
        '_modes': [('ldahia', True)],
        'occupation_distance': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'occupationDistance'
            },
            'kwargs': {
                'iteration_path': True,
                'list_return': True
            },
            'subdict': 'ldahia_info',
        },
        'element_distance': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'elementDistance'
            },
            'kwargs': {
                'iteration_path': True,
                'list_return': True
            },
            'subdict': 'ldahia_info',
        }
    },
    #----General iteration tasks
```

(continues on next page)

(continued from previous page)

```
    # iteration number
    # total energy (only total or also contributions, also lda+u correction)
    # distances (nonmagnetic and magnetic, lda+u density matrix)
    # charges (total, interstitial, mt sphere)
    # fermi energy and bandgap
    # magnetic moments
    # orbital magnetic moments
    # forces
    'iteration_number': {
        '_minimal': True,
        'number_of_iterations_total': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'overallNumber'
            },
            'overwrite_last': True,
        }
    },
    'total_energy': {
        '_minimal':
        True,
        '_conversions': [
            Conversion(name='convert_htr_to_ev', kwargs={
                'name': 'energy_hartree',
                'converted_name': 'energy',
            }),
            Conversion(name='convert_htr_to_ev',
                       kwargs={
                           'name': 'ts_energy_hartree',
                           'converted_name': 'ts_energy',
                           'pop': True
                       })
        ],
        'energy_hartree': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'value',
                'tag_name': 'freeEnergy'
            }
        },
        'energy_hartree_units': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'units',
                'tag_name': 'totalEnergy'
            },
            'overwrite_last': True,
        },
        'ts_energy_hartree': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'value',
```

(continues on next page)

(continued from previous page)

```
                'tag_name': 'tkbtimesentropy'
            }
        }
    },
    'distances': {
        '_minimal': True,
        'density_convergence': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'distance',
                'tag_name': 'chargeDensity'
            }
        },
        'density_convergence_units': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'units',
                'tag_name': 'densityConvergence',
            },
            'overwrite_last': True,
        }
    },
    'magnetic_distances': {
        '_minimal': True,
        '_modes': [('jspin', 2)],
        'overall_density_convergence': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'distance',
                'tag_name': 'overallChargeDensity'
            }
        },
        'spin_density_convergence': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'distance',
                'tag_name': 'spinDensity'
            }
        }
    },
    'total_energy_contributions': {
        'sum_of_eigenvalues': {
            'parse_type': 'singleValue',
            'path_spec': {
                'name': 'sumOfEigenvalues'
            },
            'kwargs': {
                'only_required': True
            }
        },
        'energy_core_electrons': {
            'parse_type': 'singleValue',
```

(continues on next page)

(continued from previous page)

```
            'path_spec': {
                'name': 'coreElectrons',
                'contains': 'sumOfEigenvalues'
            },
            'kwargs': {
                'only_required': True
            }
        },
        'energy_valence_electrons': {
            'parse_type': 'singleValue',
            'path_spec': {
                'name': 'valenceElectrons'
            },
            'kwargs': {
                'only_required': True
            }
        },
        'charge_den_xc_den_integral': {
            'parse_type': 'singleValue',
            'path_spec': {
                'name': 'chargeDenXCDenIntegral'
            },
            'kwargs': {
                'only_required': True
            }
        },
    },
    'ldau_energy_correction': {
        '_modes': [('ldau', True)],
        'ldau_energy_correction': {
            'parse_type': 'singleValue',
            'path_spec': {
                'name': 'dftUCorrection'
            },
            'subdict': 'ldau_info',
            'kwargs': {
                'only_required': True
            }
        },
    },
    'nmmp_distances': {
        '_minimal': True,
        '_modes': [('ldau', True)],
        'density_matrix_distance': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'distance',
                'contains': 'ldaUDensityMatrixConvergence'
            },
            'subdict': 'ldau_info'
        },
    },
```

(continues on next page)

(continued from previous page)

```
    'fermi_energy': {
        'fermi_energy': {
            'parse_type': 'singleValue',
            'path_spec': {
                'name': 'FermiEnergy'
            },
        }
    },
    'bandgap': {
        '_modes': [('bz_integration', 'hist')],
        'bandgap': {
            'parse_type': 'singleValue',
            'path_spec': {
                'name': 'bandgap'
            },
        }
    },
    'magnetic_moments': {
        '_modes': [('jspin', 2)],
        'magnetic_moments': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'magneticMoment'
            },
            'base_value': 'moment',
            'kwargs': {
                'ignore': ['atomType'],
            }
        }
    },
    'orbital_magnetic_moments': {
        '_modes': [('jspin', 2), ('soc', True)],
        'orbital_magnetic_moments': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'orbMagMoment'
            },
            'base_value': 'moment',
            'kwargs': {
                'ignore': ['atomType'],
            }
        }
    },
    'global_magnetic_moments': {
        '_minimum_version': '0.35',
        '_modes': [('noco', True)],
        'magnetic_vec_moments': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'vec',
                'contains': 'magnetic',
                'tag_name': 'globalMagMoment'
```

(continues on next page)

(continued from previous page)

```
            }
        }
    },
    'magnetic_moment': {
        '_minimum_version': '0.36',
        '_modes': [('jspin', 2), ('noco', False)],
        'magnetic_vec_moments': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'vec',
                'contains': 'magneticMomentsIn',
                'not_contains': 'local'
            }
        }
    },
    'forces': {
        '_minimal': True,
        '_modes': [('relax', True)],
        '_conversions': [Conversion(name='convert_forces')],
        'force_units': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'units',
                'tag_name': 'totalForcesOnRepresentativeAtoms'
            },
            'overwrite_last': True
        },
        'parsed_forces': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'forceTotal'
            },
            'flat': False,
            'kwargs': {
                'only_required': True
            }
        }
    },
    'charges': {
        '_conversions': [Conversion(name='calculate_total_magnetic_moment')],
        'spin_dependent_charge': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'spinDependentCharge',
                'contains': 'allElectronCharges',
                'not_contains': 'fixed'
            },
            'kwargs': {
                'only_required': True
            }
        },
        'total_charge': {
```

(continues on next page)

(continued from previous page)

```
            'parse_type': 'singleValue',
            'path_spec': {
                'name': 'totalCharge',
                'contains': 'allElectronCharges',
                'not_contains': 'fixed'
            },
            'kwargs': {
                'only_required': True
            }
        }
    },
    #-------Tasks for forcetheorem Calculations
    # DMI, JIJ, MAE, SSDISP
    'forcetheorem_dmi': {
        '_special': True,
        'dmi_force': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'Entry',
                'contains': 'DMI'
            }
        },
        'dmi_force_so': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'allAtoms',
                'contains': 'DMI'
            }
        },
        'dmi_force_qs': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'qpoints',
                'contains': 'Forcetheorem_DMI'
            }
        },
        'dmi_force_angles': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'Angles',
                'contains': 'Forcetheorem_DMI'
            }
        },
        'dmi_force_units': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'units',
                'contains': 'Forcetheorem_DMI'
            }
        }
    },
    'forcetheorem_ssdisp': {
```

(continues on next page)

(continued from previous page)

```
        '_special': True,
        'spst_force': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'Entry',
                'contains': 'SSDISP'
            }
        },
        'spst_force_qs': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'qvectors',
                'contains': 'Forcetheorem_SSDISP'
            }
        },
        'spst_force_units': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'units',
                'contains': 'Forcetheorem_SSDISP'
            }
        }
    },
    'forcetheorem_mae': {
        '_special': True,
        'mae_force': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'Angle',
                'contains': 'MAE'
            }
        },
        'mae_force_units': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'units',
                'contains': 'Forcetheorem_MAE'
            }
        }
    },
    'forcetheorem_jij': {
        '_special': True,
        'jij_force': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'Config',
                'contains': 'JIJ'
            }
        },
        'jij_force_units': {
            'parse_type': 'attrib',
            'path_spec': {
```

(continues on next page)

(continued from previous page)

```
                'name': 'units',
                'contains': 'Forcetheorem_JIJ'
            }
        }
    },
    'torques': {
        '_minimum_version': '0.35',  #Typo torgue/torque before
        '_optional': True,
        'torque_x': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'sigma_x',
                'contains': 'noncollinearTorque'
            }
        },
        'torque_y': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'sigma_y',
                'contains': 'noncollinearTorque'
            }
        }
    },
    'noco_angles': {
        '_general': True,
        '_optional': True,
        'noco_alpha': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'alpha',
                'tag_name': 'nocoParams',
                'contains': 'Group'
            }
        },
        'noco_beta': {
            'parse_type': 'attrib',
            'path_spec': {
                'name': 'beta',
                'tag_name': 'nocoParams',
                'contains': 'Group'
            }
        }
    },
    'corelevels': {
        '_optional': True,
        'corestates': {
            'parse_type': 'allAttribs',
            'path_spec': {
                'name': 'coreStates'
            },
            'kwargs': {
                'subtags': True,
```

(continues on next page)

(continued from previous page)

```
            },
            'flat': False
        }
    }
}
```

In this module migration functions for the task definitions are collected

`masci_tools.io.parsers.fleur.task_migrations.`**`migrate_033_to_031`**(*definition_dict*)

Migrate definitions for MaX5 release to MaX4 release

Return type
`dict`[`str`, `dict`[`str`, `Any`]]

Changes:

- LDA+U density matrix distance output did not exist

`masci_tools.io.parsers.fleur.task_migrations.`**`migrate_034_to_033`**(*definition_dict*)

Migrate definitions for MaX5 bugfix release to MaX5 release

Return type
`dict`[`str`, `dict`[`str`, `Any`]]

Changes:

- forcetheorem units attribute did not exist (get from 'sumValenceSingleParticleEnergies')

6.1.2.4 Commandline interface (CLI)

masci_tools

CLI for the *masci-tools* library.

```
masci_tools [OPTIONS] COMMAND [ARGS]...
```

Options

`-v, --version`

Show the version and exit.

convert-inpgen

Convert the given file to an fleur inpgen file

```
masci_tools convert-inpgen [OPTIONS] INPUT_FILE OUTPUT_FILE
```

Options

-c, --converter <converter>

Which library is used to read in the given file format

Options

ase | pymatgen

Arguments

INPUT_FILE

Required argument

OUTPUT_FILE

Required argument

fleur-schema

Commands related to the Fleur XML Schemas

```
masci_tools fleur-schema [OPTIONS] COMMAND [ARGS]...
```

add

Adds a new xml schema file to the folder in *masci_tools/io/parsers/fleur_schema* corresponding to its version number

```
masci_tools fleur-schema add [OPTIONS] SCHEMA_FILE
```

Options

--overwrite

Overwrite any existing schema-file

--branch <branch>

If the file does not exist the branch can be specified in the fleur git

--api-key <api_key>

API key for access to the Iff Gitlab instance

--test-xml-file <test_xml_file>

Example xmlfile for this schema version to test the file parser against

--from-git

Add the schema from the fleur git repository

Arguments

SCHEMA_FILE

Required argument

list

Show the available fleur schemas

```
masci_tools fleur-schema list [OPTIONS]
```

pull

Pull the default XML schema files from the iffgit and store them in the subfolder of *masci_tools/io/parsers/fleur_schema* corresponding to its version number

```
masci_tools fleur-schema pull [OPTIONS] BRANCH
```

Options

--api-key <api_key>

API key for access to the Iff Gitlab instance

--test-xml-file <test_xml_file>

Example xmlfile for this schema version to test the file parser against

Arguments

BRANCH

Required argument

validate-input

Validate the given inp.xml file against the Fleur schema stored for the version of the input

```
masci_tools fleur-schema validate-input [OPTIONS] XML_FILE
```

Arguments

XML_FILE

Required argument

validate-output

Validate the given out.xml file against the Fleur schema stored for the version of the output

```
masci_tools fleur-schema validate-output [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

inpxml

Tool for converting inp.xml files to different versions

```
masci_tools inpxml [OPTIONS] COMMAND [ARGS]...
```

convert

Convert the given XML_FILE file to version TO_VERSION

XML_FILE is the file to convert TO_VERSION is the file version of the finale input file

```
masci_tools inpxml convert [OPTIONS] XML_FILE TO_VERSION
```

Options

-o, --output-file <output_file>
Name of the output file

--overwrite
If the flag is given and the file already exists it is overwritten

Arguments

XML_FILE
Required argument

TO_VERSION
Required argument

generate-conversion

Generate the conversions from FROM_VERSION to TO_VERSION

FROM_VERSION is the file version of the initial input file TO_VERSION is the file version of the finale input file

```
masci_tools inpxml generate-conversion [OPTIONS] FROM_VERSION TO_VERSION
```

Options

--show, --no-show

Show a summary of the conversion at the end

Arguments

FROM_VERSION

Required argument

TO_VERSION

Required argument

show-conversion

Show the actions for an already created conversion from FROM_VERSION to TO_VERSION

FROM_VERSION is the file version of the initial input file TO_VERSION is the file version of the finale input file

```
masci_tools inpxml show-conversion [OPTIONS] FROM_VERSION TO_VERSION
```

Arguments

FROM_VERSION

Required argument

TO_VERSION

Required argument

parse

Commands for parsing information from KKR/Fleur files

```
masci_tools parse [OPTIONS] COMMAND [ARGS]...
```

all-attribs

Parse all attributes of the specified tag from the given xml file

```
masci_tools parse all-attribs [OPTIONS] XML_FILE
```

Options

-n, --name <name>

-c, --contains <contains>

-nc, --not-contains <not_contains>

--subtags

--text

Arguments

XML_FILE

Required argument

attrib

Parse the specified attribute from the given xml file

```
masci_tools parse attrib [OPTIONS] XML_FILE
```

Options

-n, --name <name>

-c, --contains <contains>

-nc, --not-contains <not_contains>

-t, --tag <tag>

Arguments

XML_FILE

Required argument

cell

Parse the unit cell definition of the given xml file

```
masci_tools parse cell [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

constants

Parse the mathematical constants used in the given xml-file

```
masci_tools parse constants [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

fleur-modes

Parse the Fleur modes of the given xml file

```
masci_tools parse fleur-modes [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

inp-file

Parse the Fleur inp.xml into a python dictionary

```
masci_tools parse inp-file [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

kpoints

Parse the used kpoints from the given xml-file

```
masci_tools parse kpoints [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

nkpts

Extract the number of kpoints used in the given xml file

```
masci_tools parse nkpts [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

number-nodes

Return how often the specified tag occurs in the given xml file

```
masci_tools parse number-nodes [OPTIONS] XML_FILE
```

Options

-n, --name <name>

-c, --contains <contains>

-nc, --not-contains <not_contains>

Arguments

XML_FILE
Required argument

out-file

Parse the Fleur out.xml into a python dictionary

```
masci_tools parse out-file [OPTIONS] XML_FILE
```

Options

--ignore-validation

Arguments

XML_FILE
Required argument

parameters

Parse the calculation parameters of the given xml file

```
masci_tools parse parameters [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

parent-attribs

Parse all attributes of the parent of the specified tag from the given xml file

```
masci_tools parse parent-attribs [OPTIONS] XML_FILE
```

Options

-n, --name <name>

-c, --contains <contains>

-nc, --not-contains <not_contains>

Arguments

XML_FILE
Required argument

relaxation

Parse the relaxation information for the given xml file

```
masci_tools parse relaxation [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

structure

Parse the structure information in the given Fleur xml file

```
masci_tools parse structure [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

symmetry

Parse the symmetry information for the given xml file

```
masci_tools parse symmetry [OPTIONS] XML_FILE
```

Arguments

XML_FILE
Required argument

tag-exists

Return whether the specified tag exists in the given xml file

```
masci_tools parse tag-exists [OPTIONS] XML_FILE
```

Options

-n, --name <name>

-c, --contains <contains>

-nc, --not-contains <not_contains>

Arguments

XML_FILE

Required argument

text

Parse the text of the specified tag from the given xml file

```
masci_tools parse text [OPTIONS] XML_FILE
```

Options

-n, --name <name>

-c, --contains <contains>

-nc, --not-contains <not_contains>

Arguments

XML_FILE

Required argument

plot

Commands for visualizing data

```
masci_tools plot [OPTIONS] COMMAND [ARGS]...
```

fleur-bands

Plot bandstructures from the banddos.hdf file from Fleur

```
masci_tools plot fleur-bands [OPTIONS] BANDDOS_FILE
```

Options

-w, --weight <weight>

--backend <backend>

Options
matplotlib | mpl | bokeh

--save

--show

-r, --recipe <recipe>

Options
FleurBands | FleurOrbcompBands | FleurjDOSBands | FleurSimpleBands | FleurMCDBands

Arguments

BANDDOS_FILE
Required argument

fleur-dos

Plot density of states from the banddos.hdf file from Fleur

```
masci_tools plot fleur-dos [OPTIONS] BANDDOS_FILE
```

Options

--total <total>

--interstitial <interstitial>

--atoms <atoms>

--l_resolved <l_resolved>

--backend <backend>

Options
matplotlib | mpl | bokeh

--save

--show

-r, --recipe <recipe>

Options
FleurDOS | FleurORBCOMP | FleurJDOS | FleurMCD

Arguments

BANDDOS_FILE
Required argument

6.1.2.5 Utility Functions/Classes

Custom Datatypes

This module defines subclasses of UserDict and UserList to be able to prevent unintended modifications

masci_tools.util.lockable_containers.**LockContainer**(*lock_object*)

Contextmanager for temporarily locking a lockable object. Object is unfrozen when exiting with block

Parameters
lock_object (*LockableList*[Any] | *LockableDict*[Any, Any]) – lockable container (not yet frozen)

Return type
Iterator[None]

class masci_tools.util.lockable_containers.**LockableDict**(**args*, *recursive=True*, ***kwargs*)

Subclass of UserDict, which can prevent modifications to itself. Raises *RuntimeError* if modification is attempted.

Use *LockableDict.freeze()* to enforce. *LockableDict.get_unlocked()* returns a copy of the locked object with builtin lists and dicts

Parameters
recursive (bool) – bool if True (default) all subitems (lists or dicts) are converted into their lockable counterparts

All other args or kwargs will be passed on to initialize the *UserDict*

IMPORTANT NOTE:
This is not a direct subclass of dict. So isinstance(a, dict) will be False if a is an LockableDict

freeze()

Freezes the object. This prevents further modifications

Return type
None

get_unlocked()

Get copy of object with builtin lists and dicts

Return type
dict[TypeVar(S), TypeVar(T_co, covariant=True)]

property locked: bool

Returns whether the object is locked

class masci_tools.util.lockable_containers.**LockableList**(**args*, *recursive=True*, ***kwargs*)

Subclass of UserList, which can prevent modifications to itself. Raises *RuntimeError* if modification is attempted.

Use `LockableList.freeze()` to enforce. `LockableList.get_unlocked()` returns a copy of the locked object with builtin lists and dicts

Parameters

recursive (bool) – bool if True (default) all subitems (lists or dicts) are converted into their lockable counterparts

All other args or kwargs will be passed on to initialize the *UserList*

IMPORTANT NOTE:

This is not a direct subclass of list. So isinstance(a, list) will be False if a is an LockableList

append(*item*)

S.append(value) – append value to the end of the sequence

Return type

None

clear()

Clear the list

Return type

None

extend(*other*)

S.extend(iterable) – extend sequence by appending elements from the iterable

Return type

None

freeze()

Freezes the object. This prevents further modifications

Return type

None

get_unlocked()

Get copy of object with builtin lists and dicts

Return type

list[TypeVar(T)]

insert(*i*, *item*)

S.insert(index, value) – insert value before index

Return type

None

property locked: bool

Returns whether the object is locked

pop(*i=-1*)

return the value at index i (default last) and remove it from list

Return type

TypeVar(T)

remove(*item*)

S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

Return type
None

reverse()

S.reverse() – reverse *IN PLACE*

Return type
None

masci_tools.util.lockable_containers.**S**

Type variable for the key type of the dictionary

alias of TypeVar('S')

masci_tools.util.lockable_containers.**T**

Type variable for the value type of the list

alias of TypeVar('T')

masci_tools.util.lockable_containers.**T_co**

Type variable for the value type of the dictionary

alias of TypeVar('T_co', covariant=True)

This module defines a small helper class to make case insensitive dictionary lookups available naturally

class masci_tools.util.case_insensitive_dict.**CaseInsensitiveDict**(**args*, *upper=False*, *recursive=True*, ***kwargs*)

Dict with case insensitive lookup. Used in Schema dicts to make finding paths for tags and attributes easier. Does not preserve the case of the inserted key. Does not support case insensitive lookups in nested dicts Subclass of *masci_tools.util.lockable_containers.LockableDict*. So can be frozen via the`freeze()` method

Parameters
upper (bool) – bool if True the method *upper()* will be used instead of *lower()* to normalize keys

All other args or kwargs will be passed on to initialize the *UserDict*

IMPORTANT NOTE:
This is not a direct subcalss of dict. So isinstance(a, dict) will be False if a is an CaseInsensitiveDict

class masci_tools.util.case_insensitive_dict.**CaseInsensitiveFrozenSet**(*iterable=None*, *upper=False*)

Frozenset (i.e. immutable set) with case insensitive membership tests. Used in Schema dicts in *tag_info* entries to make flexible classification easy Preserves the case of the entered keys (*original_case()* returns the case of the first encounter)

Parameters
iterable (Optional[Iterable[TypeVar(T_co, covariant=True)]]) – iterable only containing str

Note:
There might be subtle differences to expected behaviour with the methods __radd__, __ror__, and so on

difference(**others*)

Return the difference of two or more sets as a new set.

(i.e. all elements that are in this set but not the others.)

Return type
CaseInsensitiveFrozenSet[TypeVar(T_co, covariant=True)]

intersection(**others*)

Return the intersection of two sets as a new set.

(i.e. all elements that are in both sets.)

Return type
CaseInsensitiveFrozenSet[TypeVar(T_co, covariant=True)]

isdisjoint(*other*)

Return True if two sets have a null intersection.

Return type
bool

issubset(*other*)

Report whether another set contains this set.

Return type
bool

issuperset(*other*)

Report whether this set contains another set.

Return type
bool

symmetric_difference(*other*)

Return the symmetric difference of two sets as a new set.

(i.e. all elements that are in exactly one of the sets.)

Return type
CaseInsensitiveFrozenSet[TypeVar(T_co, covariant=True)]

union(**others*)

Return the union of sets as a new set.

(i.e. all elements that are in either set.)

Return type
CaseInsensitiveFrozenSet[Union[TypeVar(T_co, covariant=True), TypeVar(_S)]]

masci_tools.util.case_insensitive_dict.**S**

Generic Type

alias of TypeVar('S')

masci_tools.util.case_insensitive_dict.**T_co**

Generic Type

alias of TypeVar('T_co', covariant=True)

This module defines some aliases used in typing

masci_tools.util.typing.**FileLike**

Type used for functions accepting file-like objects, i.e. handles or file paths

alias of Union[str, bytes, Path, PathLike, IO[Any]]

masci_tools.util.typing.**TXPathLike**

Type for xpath expressions

alias of TypeVar('TXPathLike', bound=Union[str, bytes, XPath, *XPathBuilder*])

masci_tools.util.typing.**XMLFileLike**

Type used for functions accepting xml-file-like objects, i.e. handles or file paths or already parsed xml objects

alias of Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]

masci_tools.util.typing.**XMLLike**

Type used for functions accepting xml objects from lxml

alias of Union[_Element, _ElementTree]

masci_tools.util.typing.**XPathLike**

Type for xpath expressions

alias of Union[str, bytes, XPath, *XPathBuilder*]

Common XML utility

Common functions for parsing input/output files or XMLschemas from FLEUR

masci_tools.util.xml.common_functions.**T**

Generic Type

alias of TypeVar('T')

masci_tools.util.xml.common_functions.**abs_to_rel_xpath**(*xpath*, *new_root*)

Convert a given xpath to be relative from a tag appearing in the original xpath.

Parameters

- **xpath** (str) – str of the xpath to convert
- **new_root** (str) – str of the tag from which the new xpath should be relative

Return type
str

Returns
str of the relative xpath

masci_tools.util.xml.common_functions.**add_tag**(*xpath*, *tag*)

Add tag to xpath

Note: etree.XPath objects could lose context in here, i.e. non-default options passed at init

Parameters

- **xpath** (TypeVar(TXPathLike, bound= Union[str, bytes, XPath, *XPathBuilder*])) – xpath to change
- **tag** (str) – str of the tag to add

Return type
TypeVar(TXPathLike, bound= Union[str, bytes, XPath, *XPathBuilder*])

Returns
xpath with the form {old_xpath}/tag

`masci_tools.util.xml.common_functions.check_complex_xpath(node, base_xpath, complex_xpath)`

Check that the given complex xpath produces a subset of the results for the simple xpath

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – root node of an etree or an etree
- **base_xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – str of the xpath without complex syntax
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – str of the xpath to check

Raises
ValueError – If the complex_xpath does not produce a subset of the results of the base_xpath

Return type
None

`masci_tools.util.xml.common_functions.clear_xml(tree)`

Removes comments and executes xinclude tags of an xml tree.

Parameters
tree (_ElementTree) – an xml-tree which will be processed

Return type
tuple[_ElementTree, set[str]]

Returns
cleared_tree, an xmltree without comments and with replaced xinclude tags

`masci_tools.util.xml.common_functions.contains_tag(xpath, tag)`

Return whether a given xpath contains a given tag This assumes that predicates of xpaths can't be nested since otherwise the regex for removing them could fail

This function will only return True if one of the tags exactly matches the tag argument not if one tag contains the given name in it's name

Parameters

- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – xpath expression
- **tag** (str) – tag to check for

Return type
bool

Returns
whether a tag is contained in the xpath

`masci_tools.util.xml.common_functions.eval_xpath(node, xpath, logger=None, list_return=False, namespaces=None, **variables)`

Tries to evaluate an xpath expression. If it fails it logs it. If a absolute path is given (starting with '/') and the tag of the node does not match the root. It will try to find the tag in the path and convert it into a relative path

Parameters

- **node** – root node of an etree
- **xpath** – xpath expression (relative, or absolute)

- **logger** – logger object for logging warnings, errors, if not provided all errors will be raised
- **list_return** – if True, the returned quantity is always a list even if only one element is in it
- **namespaces** – dict, passed to namespaces argument in xpath call

Returns
text, attribute or a node list

masci_tools.util.xml.common_functions.**get_inpgen_comments**(*xmltree*)

Get the XML comment element appended after the root of the inp.xml file

These contain at the moment the inpgen command line and the content of the inpgen file

Parameters
xmltree (_ElementTree) – representation of the inp.xml

Return type
list[_Element]

Returns
list of XML comments, which appear after the fleurInput tag

masci_tools.util.xml.common_functions.**get_xml_attribute**(*node*, *attributename*, *logger=None*)

Get an attribute value from a node.

Parameters

- **node** (_Element) – a node from etree
- **attributename** (str) – a string with the attribute name.
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised

Return type
Optional[str]

Returns
either attributevalue, or None

masci_tools.util.xml.common_functions.**is_valid_tag**(*tag*)

Return whether the given string is a valid XML tag name

Parameters
tag (str) – tag to check

Return type
bool

masci_tools.util.xml.common_functions.**normalize_xmllike**(*xmllike*)

Returns the root of the xmltree

Return type
_Element

masci_tools.util.xml.common_functions.**process_xpath_argument**(*simple_xpath*, *complex_xpath*, *filters*)

Process the simple and complex Xpath expressions and given filters Used for unifying the logic for all xml setters/evaluators using these arguments

Parameters

- **simple_xpath** (str | bytes | XPath) – THe simple XPath (no predicates) expression to base the paths on
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – Optional XPath given with no restrictions
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
Union[str, bytes, XPath, *XPathBuilder*]

Returns
Complex XPath expression

masci_tools.util.xml.common_functions.**readd_inpgen_comments**(*xmltree*, *comments*)

Add the given comments after the fleurInput tag of the inp.xml

These contain at the moment the inpgen command line and the content of the inpgen file

Parameters

- **xmltree** (_ElementTree) – representation of the inp.xml
- **comments** (list[_Element]) – list of XML comments

Return type
_ElementTree

masci_tools.util.xml.common_functions.**reverse_xinclude**(*xmltree*, *schema_dict*, *included_tags*, ***kwargs*)

DEPRECATED ALIAS: Moved to masci_tools.util.schema_dict_util

Split the xmltree back up according to the given included tags. The original xmltree will be returned with the corresponding xinclude tags and the included trees are returned in a dict mapping the inserted filename to the extracted tree

Tags for which no known filename is known are returned under unknown-1.xml, … The following tags have known filenames:

- *relaxation*: `relax.xml`
- *kPointLists*: `kpts.xml`
- *symmetryOperations*: `sym.xml`
- *atomSpecies*: `species.xml`
- *atomGroups*: `atoms.xml`

Additional mappings can be given in the keyword arguments

Parameters

- **xmltree** – an xml-tree which will be processed
- **schema_dict** – Schema dictionary containing all the necessary information
- **included_tags** – Iterable of str, containing the names of the tags to be excluded

Returns
xmltree with the inseerted xinclude tags and a dict mapping the filenames to the excluded trees

Raises
ValueError – if the tag can not be found in the given xmltree

masci_tools.util.xml.common_functions.**serialize_xml_objects**(*args*, *kwargs*)

Convert every XML element/tree in the given args/kwargs to string using `lxml.etree.tostring()`

Parameters

- **args** (tuple[Any, ...]) – positional arguments
- **kwargs** (dict[str, Any]) – keyword arguments

Return type
tuple[tuple[Any, ...], dict[str, Any]]

masci_tools.util.xml.common_functions.**split_off_attrib**(*xpath*)

Splits off attribute of the given xpath (part after @)

Note: etree.XPath objects could lose context in here, i.e. non-default options passed at init

Parameters
xpath (TypeVar(TXPathLike, bound= Union[str, bytes, XPath, *XPathBuilder*])) – xpath to split up

Return type
tuple[TypeVar(TXPathLike, bound= Union[str, bytes, XPath, *XPathBuilder*]), str]

masci_tools.util.xml.common_functions.**split_off_tag**(*xpath*)

Splits off the last part of the given xpath

Note: etree.XPath objects could lose context in here, i.e. non-default options passed at init

Parameters
xpath (TypeVar(TXPathLike, bound= Union[str, bytes, XPath, *XPathBuilder*])) – xpath to split up

Return type
tuple[TypeVar(TXPathLike, bound= Union[str, bytes, XPath, *XPathBuilder*]), str]

masci_tools.util.xml.common_functions.**validate_xml**(*xmltree*, *schema*, *error_header='File does not validate'*)

Checks a given xmltree against a schema and produces a nice error message with all the validation errors collected

Parameters

- **xmltree** (_ElementTree) – xmltree of the file to validate
- **schema** (XMLSchema) – etree.XMLSchema to validate against
- **error_header** (str) – str to lead a evtl error message with

Raises
etree.DocumentInvalid if the schema does not validate

Return type
None

Common functions for converting types to and from XML files

masci_tools.util.xml.converters.**convert_fleur_electronconfig**(*econfig_element*)

Convert electronConfig tag to eConfig string

Return type
str

masci_tools.util.xml.converters.**convert_fleur_lo**(*loelements*, *allow_special_los=True*)

Converts lo xml elements from the inp.xml file into a lo string for the inpgen

Return type
str

masci_tools.util.xml.converters.**convert_from_fortran_bool**(*stringbool*)

Converts a string in this case ('T', 'F', or 't', 'f') to True or False

Parameters
stringbool (str | bool) – a string ('t', 'f', 'F', 'T')

Return type
bool

Returns
boolean (either True or False)

masci_tools.util.xml.converters.**convert_from_fortran_complex**(*number_str*)

Converts a string of the form (float,float) to a complex number

Parameters
number_str (str) – string to convert

Return type
complex

Returns
complex number

masci_tools.util.xml.converters.**convert_from_xml**(*xmlstring*, *schema_dict*, *name*, *text=False*, *constants=None*, *logger=None*, *list_return=False*)

Tries to converts a given string to the types specified in the schema_dict. First succeeded conversion will be returned

If no logger is given and a attribute cannot be converted an error is raised

Parameters

- **stringattribute** – str, Attribute to convert.
- **schema_dict** (*SchemaDict*) – Schema dictionary containing all the information
- **name** (str) – name of the attribute or element
- **text** (bool) – bool, decides whether to take the definitions for text or attributes
- **constants** (Optional[dict[str, float]]) – dict, of constants defined in fleur input
- **logger** (Optional[Logger]) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list_return** (bool) – if True, the returned quantity is always a list even if only one element is in it

Return type
tuple[Union[int, float, bool, str, complex, list[Union[int, float, bool, str, complex]], list[Union[int, float, bool, str, complex, list[Union[int, float, bool, str, complex]]]]], bool]

Returns
The converted value of the first successful conversion

masci_tools.util.xml.converters.**convert_from_xml_explicit**(*xmlstring*, *definitions*, *constants=None*, *logger=None*, *list_return=False*)

Tries to converts a given string to the types given in definitions. First succeeded conversion will be returned

If no logger is given and a attribute cannot be converted an error is raised

Parameters

- **stringattribute** – str, Attribute to convert.
- **definitions** (list[*AttributeType*]) – list of *AttributeType* definitions
- **constants** (Optional[dict[str, float]]) – dict, of constants defined in fleur input
- **logger** (Optional[Logger]) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list_return** (bool) – if True, the returned quantity is always a list even if only one element is in it

Return type
tuple[Union[int, float, bool, str, complex, list[Union[int, float, bool, str, complex]], list[Union[int, float, bool, str, complex, list[Union[int, float, bool, str, complex]]]]], bool]

Returns
The converted value of the first successful conversion

masci_tools.util.xml.converters.**convert_from_xml_single_values**(*xmlstring*, *possible_types*, *constants=None*, *logger=None*)

Tries to converts a given string attribute to the types given in possible_types. First succeeded conversion will be returned

If no logger is given and a attribute cannot be converted an error is raised

Parameters

- **stringattribute** – str, Attribute to convert.
- **possible_types** (tuple[Literal['int', 'switch', 'string', 'float', 'float_expression', 'complex'], ...]) – list of str What types it will try to convert to
- **constants** (Optional[dict[str, float]]) – dict, of constants defined in fleur input
- **logger** (Optional[Logger]) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list_return** – if True, the returned quantity is always a list even if only one element is in it

Return type
tuple[list[Union[int, float, bool, str, complex]], bool]

Returns
The converted value of the first successful conversion

masci_tools.util.xml.converters.**convert_str_version_number**(*version_str*)

Convert the version number as a integer for easy comparisons

Parameters
version_str (str) – str of the version number, e.g. '0.33'

Return type
tuple[int, int]

Returns
tuple of ints representing the version str

masci_tools.util.xml.converters.**convert_to_fortran_bool**(*boolean*)

Converts a Boolean as string to the format defined in the input

Parameters
boolean (bool | str) – either a boolean or a string ('True', 'False', 'F', 'T')

Return type
Literal['T', 'F']

Returns
a string (either 't' or 'f')

masci_tools.util.xml.converters.**convert_to_xml**(*value*, *schema_dict*, *name*, *text=False*, *logger=None*, *list_return=False*)

Tries to converts a given string to the types specified in the schema_dict. First succeeded conversion will be returned

If no logger is given and a attribute cannot be converted an error is raised

Parameters

- **stringattribute** – str, Attribute to convert.
- **schema_dict** (*SchemaDict*) – Schema dictionary containing all the information
- **name** (str) – name of the attribute or element
- **text** (bool) – bool, decides whether to take the definitions for text or attributes
- **constants** – dict, of constants defined in fleur input
- **logger** (Optional[Logger]) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list_return** (bool) – if True, the returned quantity is always a list even if only one element is in it

Return type
tuple[str | list[str], bool]

Returns
The converted value of the first successful conversion

masci_tools.util.xml.converters.**convert_to_xml_explicit**(*value*, *definitions*, *logger=None*, *float_format='.10'*, *list_return=False*)

Tries to convert a given list of values to str for a xml file based on the definitions (length and type). First succeeded conversion will be returned

Parameters

- **textvalue** – value to convert
- **definitions** (list[*AttributeType*]) – list of *AttributeType* definitions
- **logger** (Optional[Logger]) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list_return** (bool) – if True, the returned quantity is always a list even if only one element is in it

Return type
tuple[str | list[str], bool]

Returns
The converted value of the first successful conversion

`masci_tools.util.xml.converters.convert_to_xml_single_values(value, possible_types, logger=None, float_format='.10')`

Tries to converts a given attributevalue to a string for a xml file according to the types given in possible_types. First succeeded conversion will be returned

Parameters

- **value** (Union[Any, Iterable[Any]]) – value to convert.
- **possible_types** (tuple[Literal['int', 'switch', 'string', 'float', 'float_expression', 'complex'], ...]) – list of str What types it will try to convert from
- **logger** (Optional[Logger]) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list_return** – if True, the returned quantity is always a list even if only one element is in it

Return type
tuple[list[str], bool]

Returns
The converted str of the value of the first successful conversion

This module contains Classes for building complex XPath expressions based on general attribute conditions from simple XPath expressions

`masci_tools.util.xml.xpathbuilder.FilterType`

Type for filters argument for XPathBuilder

alias of Dict[str, Any]

`class masci_tools.util.xml.xpathbuilder.XPathBuilder(simple_path, filters=None, compile_path=False, strict=False, **kwargs)`

Class for building a complex xpath (restricted to adding filters) from a simple xpath expression

Note: passing in an etree.XPath object will not respect the options passed into it. Only the kwargs in __init__ are used to compile the path if compile_path=True

Note: Filters/Constraints (or predicates like they are called for XPaths) can either be added by providing the `filters` argument in the constructor or by calling the *add_filter()* method.

The `filters` argument is a dictionary with the tag names, where to apply the condition, as keys and the condition as values while the *add_filter()* method takes these as it's two arguments. The tag name has to be a part of the original simple xpath expression. The condition is a dictionary with one key specifying the kind of condition and the value for the condition. The condition can also be the name of an attribute or path, in which case the value can be another condition dictionary. The following conditions operators i.e. keys in the dictionary are supported:

- =/==: equal to
- `!=`: not equal to
- <: less than
- >: greater than
- <=: less than or equal to
- >=: greater than or equal to
- `contains`: attribute/tag contains the given value (case sensitive)
- `not-contains`: attribute/tag does not contains the given value
- `starts-with`: attribute/tag starts with the given value (case sensitive)
- `ends-with`: attribute/tag ends with the given value (case sensitive)
- `index`: Select tags based on their index in the parent tag (either explicit index or another condition)
- `has`: Select tags based on the presence of the given attribute/tag
- `has-not`: Select tags based on the absence of the given attribute/tag
- `number-nodes`: Compute the number of nodes in the previous path and select based on further criteria
- `and`: Provide multiple conditions in a list joined by `and`
- `or`: Provide multiple conditions in a list joined by `or`
- `in`: Select tags if the value of the path is in a given list of values
- `not-in`: Select tags if the value of the path is not in a given list of values
- `<string>`: All other strings are interpreted as paths to attributes/tags specifying conditions on their value
- `<tuple of paths>`: Multiple strings are interpreted as multiple node sets, which are joined with |

Example:

```
from masci_tools.util.xml.xpathbuilder import XPathBuilder

# XPath selecting all lo tags for SCLO type LOs and Iron species
xpath = XPathBuilder('/fleurInput/atomSpecies/species/lo',
                     filters = {'species': {
                                    'name': {'contains': 'Fe'},
                                },
                                'lo': {
                                    'type': 'SCLO'
                                }
                            })
```

Parameters

- **simple_path** (str | bytes | XPath) – basic simple XPath expression to start from
- **filters** (Optional[dict[str, Dict[str, Any]]]) – dictionary with filters
- **compile_path** (bool) – bool if True the path property will be compiled as etree.XPath
- **strict** (bool) – bool if True the __str__ conversion will raise an error

Other Kwargs will be passed on to the etree.XPath compilation if `compile_path=True`

add_filter(*tag*, *conditions*)

Add a filter to the filters dictionary

Parameters

- **tag** (str) – str name of the tag name to add a filter to
- **conditions** (Union[Dict[str, Any], Any]) – dictionary specifying the filter

Return type
None

append_tag(*tag*)

Append another tag to the end of the simple xpath expression

Parameters
tag (str) – str name of the tag to append

Return type
None

get_predicate(*tag*, *condition*, *compound=False*, *path='.'*, *process_path=False*)

Construct the predicate for the given tag and condition

Parameters

- **tag** (str) – str name of the tag
- **condition** (Any) – condition specified, either dict or single value
- **compound** (bool) – bool if True the enclosing condition is a compound condition, forbidding any other compound condition
- **path** (str | tuple[str, ...]) – path, to which to apply the condition
- **process_path** (bool) – bool if True the path will taken apart into its components and the components will be checked with XPath variables

Return type
str

property path: str | lxml.etree.XPath

Property for constructing the complex Xpath

process_condition(*tag*, *operator*, *content*, *path*, *process_path=False*)

Process the condition for the given tag and condition

Parameters

- **tag** (str) – str name of the tag
- **operator** (str) – operator for condition
- **content** (Any) – content of condition

- **path** (str | tuple[str, ...]) – path, to which to apply the condition
- **process_path** (bool) – bool if True the path will taken apart into its components and the components will be checked with XPath variables

Return type
str

strip_off_tag()

Strip off the last tag of the simple xpath expression

Return type
str

XML Setter functions

Functions for modifying the xml input file of Fleur utilizing the schema dict and as little knowledge of the concrete xpaths as possible

masci_tools.util.xml.xml_setters_names.**add_number_to_attrib**(*xmltree*, *schema_dict*, *name*, *number_to_add*, *complex_xpath=None*, *filters=None*, *mode='absolute'*, *occurrences=None*, ***kwargs*)

Adds a given number to the attribute value in a xmltree specified by the name of the attribute and optional further specification If there are no nodes under the specified xpath an error is raised

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **name** (str) – the attribute name to change
- **number_to_add** (Any) – number to add/multiply with the old attribute value
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **mode** (Literal['abs', 'absolute', 'rel', 'relative']) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

Return type
Union[_Element, _ElementTree]

Returns
xmltree with shifted attribute

masci_tools.util.xml.xml_setters_names.**add_number_to_first_attrib**(*xmltree*, *schema_dict*, *name*, *number_to_add*, *complex_xpath=None*, *filters=None*, *mode='absolute'*, ***kwargs*)

Adds a given number to the first occurrence of an attribute value in a xmltree specified by the name of the attribute and optional further specification If there are no nodes under the specified xpath an error is raised

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **name** (str) – the attribute name to change
- **number_to_add** (Any) – number to add/multiply with the old attribute value
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **mode** (Literal['abs', 'absolute', 'rel', 'relative']) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

Return type
Union[_Element, _ElementTree]

Returns
xmltree with shifted attribute

masci_tools.util.xml.xml_setters_names.**clone_species**(*xmltree*, *schema_dict*, *species_name*, *new_name*, *changes=None*)

Method to create a new species from an existing one with evtl. modifications

For reference of the changes dictionary look at *set_species()*

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **species_name** (str) – string, name of the specie you want to clone Has to correspond to one single species (no 'all'/'all-<search_string>')
- **new_name** (str) – new name of the cloned species
- **changes** (Optional[dict[str, Any]]) – a optional python dict specifying what you want to change.

Returns xmltree
xml etree of the new inp.xml

Return type
Union[_Element, _ElementTree]

masci_tools.util.xml.xml_setters_names.**create_tag**(*xmltree*, *schema_dict*, *tag*, *complex_xpath=None*, *filters=None*, *create_parents=False*, *occurrences=None*, ***kwargs*)

This method creates a tag with a uniquely identified xpath under the nodes of its parent. If there are no nodes evaluated the subtags can be created with *create_parents=True*

The tag is always inserted in the correct place if a order is enforced by the schema

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **tag** (QName | str | _Element) – str of the tag to create or etree Element or string representing the XML element with the same name to insert
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create_parents** (bool) – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xmltree with created tags

masci_tools.util.xml.xml_setters_names.**delete_att**(*xmltree*, *schema_dict*, *name*, *complex_xpath=None*, *filters=None*, *occurrences=None*, ***kwargs*)

This method deletes a attribute with a uniquely identified xpath.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **name** (str) – str of the attribute to delete
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to delete a attribute. By default all nodes are used.

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

Return type
Union[_Element, _ElementTree]

Returns
xmltree with deleted attributes

masci_tools.util.xml.xml_setters_names.**delete_tag**(*xmltree*, *schema_dict*, *tag_name*, *complex_xpath=None*, *filters=None*, *occurrences=None*, ***kwargs*)

This method deletes a tag with a uniquely identified xpath.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the tag to delete

- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to delete a tag. By default all nodes are used.

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xmltree with deleted tags

masci_tools.util.xml.xml_setters_names.**replace_tag**(*xmltree*, *schema_dict*, *tag_name*, *element*, *complex_xpath=None*, *filters=None*, *occurrences=None*, ***kwargs*)

This method deletes a tag with a uniquely identified xpath.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the tag to replace
- **element** (str | _Element) – etree Element or string representing the XML element to replace the tag
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to replace a tag. By default all nodes are used.

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xmltree with replaced tags

masci_tools.util.xml.xml_setters_names.**set_atomgroup**(*xmltree*, *schema_dict*, *changes*, *position=None*, *species=None*, *filters=None*)

Method to set parameters of an atom group of the fleur inp.xml file.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **changes** (dict[str, Any]) – a python dict specifying what you want to change.
- **position** (Union[int, Literal['all'], None]) – position of an atom group to be changed. If equals to 'all', all species will be changed
- **species** (Optional[str]) – atom groups, corresponding to the given species will be changed
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
Union[_Element, _ElementTree]

Returns
xml etree of the new inp.xml

changes is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'changes': {'nocoParams': {'beta': val}}
```

masci_tools.util.xml.xml_setters_names.**set_atomgroup_label**(*xmltree*, *schema_dict*, *atom_label*, *changes*)

This method calls *set_atomgroup()* method for a certain atom species that corresponds to an atom with a given label.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **atom_label** (str) – string, a label of the atom which specie will be changed. 'all' to change all the species
- **changes** (dict[str, Any]) – a python dict specifying what you want to change.

Return type
Union[_Element, _ElementTree]

Returns
xml etree of the new inp.xml

changes is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'changes': {'nocoParams': {'beta': val}}
```

masci_tools.util.xml.xml_setters_names.**set_attrib_value**(*xmltree*, *schema_dict*, *name*, *value*, *complex_xpath=None*, *filters=None*, *occurrences=None*, *create=False*, ***kwargs*)

Sets an attribute in a xmltree to a given value, specified by its name and further specifications. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **name** (str) – the attribute name to set
- **value** (Any) – value or list of values to set
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set attribute

masci_tools.util.xml.xml_setters_names.**set_complex_tag**(*xmltree*, *schema_dict*, *tag_name*, *changes*, *complex_xpath=None*, *filters=None*, *create=False*, ***kwargs*)

Function to correctly set tags/attributes for a given tag. Goes through the attributedict and decides based on the schema_dict, how the corresponding key has to be handled. The tag is specified via its name and evtl. further specification

Supports:

- attributes

- tags with text only
- simple tags, i.e. only attributes (can be optional single/multiple)
- complex tags, will recursively create/modify them

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (str) – name of the tag to set
- **changes** (dict[str, Any]) – Keys in the dictionary correspond to names of tags and the values are the modifications to do on this tag (attributename, subdict with changes to the subtag, ...)
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create** (bool) – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xmltree with changes to the complex tag

masci_tools.util.xml.xml_setters_names.**set_first_attrib_value**(*xmltree*, *schema_dict*, *name*, *value*, *complex_xpath=None*, *filters=None*, *create=False*, ***kwargs*)

Sets the first occurrence of an attribute in a xmltree to a given value, specified by its name and further specifications. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **name** (str) – the attribute name to set
- **value** (Any) – value or list of values to set

- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set attribute

masci_tools.util.xml.xml_setters_names.**set_first_text**(*xmltree*, *schema_dict*, *tag_name*, *text*, *complex_xpath=None*, *filters=None*, *create=False*, ***kwargs*)

Sets the text the first occurrence of a tag in a xmltree to a given value, specified by the name of the tag and further specifications. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (str) – str name of the tag, where the text should be set
- **text** (Any) – value or list of values to set
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set text

masci_tools.util.xml.xml_setters_names.**set_inpchanges**(*xmltree*, *schema_dict*, *changes*, *path_spec=None*)

This method sets all the attribute and texts provided in the change_dict.

The first occurrence of the attribute/tag is set

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **changes** (dict[str, Any]) – dictionary {attrib_name : value} with all the wanted changes.
- **path_spec** (Optional[dict[str, Any]]) – dict, with ggf. necessary further specifications for the path of the attribute

An example of changes:

```
changes = {
    'itmax' : 1,
    'l_noco': True,
    'ctail': False,
    'l_ss': True
}
```

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**set_kpath**(*xmltree*, *schema_dict*, *kpath*, *count*, *gamma=False*)

Sets a k-path directly into inp.xml as a alternative kpoint set with purpose 'bands'

Warning: This method is only supported for input versions before the Max5 release

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **kpath** (dict[str, Iterable[float]]) – a dictionary with kpoint name as key and k point coordinate as value
- **count** (int) – number of k-points

- **gamma** (bool) – bool that controls if the gamma-point should be included in the k-point mesh

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**set_kpath_max4**(*xmltree*, *schema_dict*, *kpath*, *count*, *gamma=False*)

Sets a k-path directly into inp.xml as a alternative kpoint set with purpose 'bands'

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **kpath** (dict[str, Iterable[float]]) – a dictionary with kpoint name as key and k point coordinate as value
- **count** (int) – number of k-points
- **gamma** (bool) – bool that controls if the gamma-point should be included in the k-point mesh

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**set_kpointlist**(*xmltree*, *schema_dict*, *kpoints*, *weights*, *name=None*, *kpoint_type='path'*, *special_labels=None*, *switch=False*, *overwrite=False*, *additional_attributes=None*)

Explicitly create a kPointList from the given kpoints and weights. This routine will add the specified kPointList with the given name.

> **Warning:** For input versions Max4 and older **all** keyword arguments are not valid (*name*, *kpoint_type*, *special_labels*, *switch* and *overwrite*)

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **kpoints** (Iterable[Iterable[float]]) – list or array containing the **relative** coordinates of the kpoints
- **weights** (Iterable[float]) – list or array containing the weights of the kpoints
- **name** (Optional[str]) – str for the name of the list, if not given a default name is generated
- **kpoint_type** (Literal['path', 'mesh', 'tria', 'tria-bulk', 'spex-mesh']) – str specifying the type of the kPointList ('path', 'mesh', 'spex', 'tria', ...)

- **special_labels** (Optional[dict[int, str]]) – dict mapping indices to labels. The labels will be inserted for the kpoints corresponding to the given index
- **switch** (bool) – bool, if True the kPointlist will be used by Fleur when starting the next calculation
- **overwrite** (bool) – bool, if True and a kPointlist with the given name already exists it will be overwritten

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**set_kpointlist_max4**(*xmltree*, *schema_dict*, *kpoints*, *weights*, *name=None*, *kpoint_type='path'*, *special_labels=None*, *switch=False*, *overwrite=False*, *additional_attributes=None*)

Explicitly create a kPointList from the given kpoints and weights. This routine is specific to input versions Max4 and before and will replace any existing kPointCount, kPointMesh, ... with the specified kPointList

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **kpoints** (Iterable[Iterable[float]]) – list or array containing the **relative** coordinates of the kpoints
- **weights** (Iterable[float]) – list or array containing the weights of the kpoints

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**set_kpointmesh**(*xmltree*, *schema_dict*, *mesh*, *name=None*, *use_symmetries=True*, *switch=False*, *overwrite=False*, *shift=None*, *time_reversal=True*, *map_to_first_bz=True*)

Create a kpoint mesh using spglib

for details see `get_stabilized_reciprocal_mesh()`

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **mesh** (Collection[int]) – list-like woth three elements, giving the size of the kpoint set in each direction
- **use_symmetry** – bool if True the available symmetry operations in the inp.xml will be used to reduce the kpoint set otherwise only the identity matrix is used
- **name** (Optional[str]) – Name of the created kpoint list. If not given a name is generated

- **switch** (bool) – bool if True the kpoint list is direclty set as the used set
- **overwrite** (bool) – if True and a kpoint list of the given name already exists it will be overwritten
- **shift** (Optional[Iterable[float]]) – shift the center of the kpint set
- **time_reversal** (bool) – bool if True time reversal symmetry will be used to reduce the kpoint set
- **map_to_first_bz** (bool) – bool if True the kpoints are mapped into the [0,1] interval

Return type
Union[_Element, _ElementTree]

Returns
xmltree with a created kpoint path

masci_tools.util.xml.xml_setters_names.**set_kpointpath**(*xmltree*, *schema_dict*, *path=None*, *nkpts=None*, *density=None*, *name=None*, *switch=False*, *overwrite=False*, *special_points=None*)

Create a kpoint list for a bandstructure calculation (using ASE kpath generation)

The path can be defined explictly (see bandpath()) or derived from the unit cell

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **path** (Union[str, list[str], None]) – str, list of str or None defines the path to interpolate (for syntax bandpath())
- **nkpts** (Optional[int]) – int number of kpoints in the path
- **density** (Optional[float]) – float number of kpoints per Angstroem
- **name** (Optional[str]) – Name of the created kpoint list. If not given a name is generated
- **switch** (bool) – bool if True the kpoint list is direclty set as the used set
- **overwrite** (bool) – if True and a kpoint list of the given name already exists it will be overwritten
- **special_points** (Optional[dict[str, Iterable[float]]]) – dict mapping names to coordinates for special points to use

Return type
Union[_Element, _ElementTree]

Returns
xmltree with a created kpoint path

masci_tools.util.xml.xml_setters_names.**set_nkpts**(*xmltree*, *schema_dict*, *count*, *gamma=False*)

Sets a k-point mesh directly into inp.xml

> **Warning:** This method is only supported for input versions before the Max5 release

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **count** (int) – number of k-points
- **gamma** (bool) – bool that controls if the gamma-point should be included in the k-point mesh

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**set_nkpts_max4**(*xmltree*, *schema_dict*, *count*, *gamma=False*)

Sets a k-point mesh directly into inp.xml specific for inputs of version Max4

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **count** (int) – number of k-points
- **gamma** (bool) – bool that controls if the gamma-point should be included in the k-point mesh

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**set_simple_tag**(*xmltree*, *schema_dict*, *tag_name*, *changes*, *complex_xpath=None*, *filters=None*, *create_parents=False*, ***kwargs*)

Sets one or multiple *simple* tag(s) in an xmltree. A simple tag can only hold attributes and has no subtags. The tag is specified by its name and further specification If the tag can occur multiple times all existing tags are DELETED and new ones are written. If the tag only occurs once it will automatically be created if its missing.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (str) – str name of the tag to modify/set
- **changes** (list[dict[str, Any]] | dict[str, Any]) – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to {'attributename': attributevalue}
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create_parents** (bool) – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set simple tags

masci_tools.util.xml.xml_setters_names.**set_species**(*xmltree*, *schema_dict*, *species_name*, *changes*, *filters=None*, *create=False*)

Method to set parameters of a species tag of the fleur inp.xml file.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **species_name** (str) – string, name of the specie you want to change Can be name of the species, 'all' or 'all-<string>' (sets species with the string in the species name)
- **changes** (dict[str, Any]) – a python dict specifying what you want to change.
- **create** (bool) – bool, if species does not exist create it and all subtags?
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises
ValueError – if species name is non existent in inp.xml and should not be created. also if other given tags are garbage. (errors from eval_xpath() methods)

Return xmltree
xml etree of the new inp.xml

Return type
Union[_Element, _ElementTree]

changes is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a MT radius it can be done via:

```
changes = {'mtSphere' : {'radius' : 2.2}}
```

Another example:

```
'changes': {'special': {'socscale': 0.0}}
```

that switches SOC terms on a sertain specie. `mtSphere`, `atomicCutoffs`, `energyParameters`, `lo`, `electronConfig`, `nocoParams`, `ldaU` and `special` keys are supported. To find possible keys of the inner dictionary please refer to the FLEUR documentation flapw.de

masci_tools.util.xml.xml_setters_names.**set_species_label**(*xmltree*, *schema_dict*, *atom_label*, *changes*, *create=False*)

This method calls *set_species()* method for a certain atom species that corresponds to an atom with a given label

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **atom_label** (str) – string, a label of the atom which specie will be changed. 'all' to change all the species
- **changes** (dict[str, Any]) – a python dict specifying what you want to change.
- **create** (bool) – bool, if species does not exist create it and all subtags?

Return type
Union[_Element, _ElementTree]

Returns
xml etree of the new inp.xml

masci_tools.util.xml.xml_setters_names.**set_text**(*xmltree*, *schema_dict*, *tag_name*, *text*, *complex_xpath=None*, *filters=None*, *occurrences=None*, *create=False*, ***kwargs*)

Sets the text on tags in a xmltree to a given value, specified by the name of the tag and further specifications. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (str) – str name of the tag, where the text should be set
- **text** (Any) – value or list of values to set
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set text

`masci_tools.util.xml.xml_setters_names.set_xcfunctional(xmltree, schema_dict, xc_functional, xc_functional_options=None, libxc=False)`

Set the Exchange Correlation potential tag

Setting a inbuilt XC functional .. code-block:: python

set_xcfunctional(xmltree, schema_dict, 'vwn')

Setting a LibXC XC functional .. code-block:: python

set_xcfunctional(xmltree, schema_dict, {'exchange': 'lda_x', 'correlation':"lda_c_xalpha"}, libxc=True)

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – XML tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xc_functional** (str | dict[str, int | str]) – str or dict. If str it is the name of a inbuilt XC functional. If it is a dict it specifies either the name or id for LibXC functionals for the keys *'exchange', 'correlation', 'etot_exchange' and 'etot_correlation'*
- **xc_functional_options** (Optional[dict[str, Any]]) – dict with further general changes to the *xcFunctional* tag
- **libxc** (bool) – bool if True the functional is a LibXC functional

Return type
Union[_Element, _ElementTree]

Returns
an xmltree with modified xcFunctional tag

`masci_tools.util.xml.xml_setters_names.shift_value(xmltree, schema_dict, changes, mode='absolute', path_spec=None)`

Shifts numerical values of attributes directly in the inp.xml file.

The first occurrence of the attribute is shifted

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **changes** (dict[str, Any]) – a python dictionary with the keys to shift and the shift values.
- **mode** (Literal['abs', 'absolute', 'rel', 'relative']) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with the given value *abs/absolute* adds the old value and the given value
- **path_spec** (Optional[dict[str, Any]]) – dict, with ggf. necessary further specifications for the path of the attribute

Return type
Union[_Element, _ElementTree]

Returns
a xml tree with shifted values

An example of changes:

```
changes = {'itmax' : 1, 'dVac': -0.123}
```

masci_tools.util.xml.xml_setters_names.**shift_value_species_label**(*xmltree*, *schema_dict*, *atom_label*, *attribute_name*, *number_to_add*, *mode='absolute'*, ***kwargs*)

Shifts the value of an attribute on a species by label if atom_label contains 'all' then applies to all species

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **atom_label** (str) – string, a label of the atom which specie will be changed. 'all' if set up all species
- **attribute_name** (str) – name of the attribute to change
- **number_to_add** (Any) – value to add or to multiply by
- **mode** (Literal['abs', 'absolute', 'rel', 'relative']) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*

Kwargs if the attribute_name does not correspond to a unique path:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
Union[_Element, _ElementTree]

Returns
xml etree of the new inp.xml

masci_tools.util.xml.xml_setters_names.**switch_kpointset**(*xmltree*, *schema_dict*, *list_name*)

Switch the used k-point set

> **Warning:** This method is only supported for input versions after the Max5 release

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **list_name** (str) – name of the kPoint set to use

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**switch_kpointset_max4**(*xmltree*, *schema_dict*, *list_name*)

Sets a k-point mesh directly into inp.xml specific for inputs of version Max4

> **Warning:** This method is only supported for input versions after the Max5 release

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml tree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **list_name** (str) – name of the kPoint set to use

Return type
Union[_Element, _ElementTree]

Returns
an xmltree of the inp.xml file with changes.

masci_tools.util.xml.xml_setters_names.**switch_species**(*xmltree*, *schema_dict*, *new_species_name*, *position=None*, *species=None*, *filters=None*, *clone=False*, *changes=None*)

Method to switch the species of an atom group of the fleur inp.xml file.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **new_species_name** (str) – name of the species to switch to
- **position** (Union[int, Literal['all'], None]) – position of an atom group to be changed. If equals to 'all', all species will be changed
- **species** (Optional[str]) – atom groups, corresponding to the given species will be changed
- **clone** (bool) – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with *clone_species()*
- **changes** (Optional[dict[str, Any]]) – changes to do if the species is cloned
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
Union[_Element, _ElementTree]

Returns
xml etree of the new inp.xml

masci_tools.util.xml.xml_setters_names.**switch_species_label**(*xmltree*, *schema_dict*, *atom_label*, *new_species_name*, *clone=False*, *changes=None*)

Method to switch the species of an atom group of the fleur inp.xml file based on a label of a contained atom

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **atom_label** (str) – string, a label of the atom which group will be changed. 'all' to change all the groups
- **new_species_name** (str) – name of the species to switch to
- **clone** (bool) – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with *clone_species()*
- **changes** (Optional[dict[str, Any]]) – changes to do if the species is cloned

Return type
Union[_Element, _ElementTree]

Returns
xml etree of the new inp.xml

This module contains useful methods for initializing or modifying a n_mmp_mat file for LDA+U

class masci_tools.util.xml.xml_setters_nmmpmat.**LDAUElement**(*species: str*, *orbital: int*, *group_index: int*)

Contains the important information needed to locate the associated density matrix blocks

group_index: int

Alias for field number 2

orbital: int

Alias for field number 1

species: str

Alias for field number 0

masci_tools.util.xml.xml_setters_nmmpmat.**align_nmmpmat_to_sqa**(*xmltree*, *nmmplines*, *schema_dict*, *species_name='all'*, *orbital='all'*, *phi_before=0.0*, *theta_before=0.0*, *filters=None*)

Align the density matrix with the given SQA of the associated species

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **nmmplines** (list[str]) – list of lines in the n_mmp_mat file
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **species_name** (str) – string, name of the species you want to change
- **orbital** (int | str) – integer or string ('all'), orbital quantum number of the LDA+U procedure to be modified

- **phi_before** (float | list[float]) – float or list of floats, angle (radian), values for phi for the previous alignment of the density matrix
- **theta_before** (float | list[float]) – float or list of floats, angle (radian), values for theta for the previous alignment of the density matrix
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type
list[str]

Returns
list with modified nmmplines

masci_tools.util.xml.xml_setters_nmmpmat.**rotate_nmmpmat**(*xmltree*, *nmmplines*, *schema_dict*, *species_name*, *orbital*, *phi*, *theta*, *inverse=False*, *filters=None*)

Rotate the density matrix with the given angles phi and theta

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **nmmplines** (list[str]) – list of lines in the n_mmp_mat file
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **species_name** (str) – string, name of the species you want to change
- **orbital** (int | str) – integer or string ('all'), orbital quantum number of the LDA+U procedure to be modified
- **phi** (float) – float, angle (radian), by which to rotate the density matrix
- **theta** (float) – float, angle (radian), by which to rotate the density matrix
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type
list[str]

Returns
list with modified nmmplines

masci_tools.util.xml.xml_setters_nmmpmat.**set_nmmpmat**(*xmltree*, *nmmplines*, *schema_dict*, *species_name*, *orbital*, *spin*, *state_occupations=None*, *orbital_occupations=None*, *denmat=None*, *phi=None*, *theta=None*, *inverse=False*, *align_to_sqa=False*, *filters=None*)

Routine sets the block in the n_mmp_mat file specified by species_name, orbital and spin to the desired density matrix

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **nmmplines** (Optional[list[str]]) – list of lines in the n_mmp_mat file
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **species_name** (str) – string, name of the species you want to change
- **orbital** (int) – integer, orbital quantum number of the LDA+U procedure to be modified
- **spin** (int) – integer, specifies which spin block should be modified
- **state_occupations** (Optional[list[float]]) – list, sets the diagonal elements of the density matrix and everything else to zero
- **denmat** (Optional[ndarray]) – matrix, specify the density matrix explicitly
- **phi** (Optional[float]) – float, optional angle (radian), by which to rotate the density matrix before writing it
- **theta** (Optional[float]) – float, optional angle (radian), by which to rotate the density matrix before writing it
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type
list[str]

Returns
list with modified nmmplines

masci_tools.util.xml.xml_setters_nmmpmat.**validate_nmmpmat**(*xmltree*, *nmmplines*, *schema_dict*)

Checks that the given nmmp_lines is valid with the given xmltree

Checks that the number of blocks is as expected from the inp.xml and each block does not contain non-zero elements outside their size given by the orbital quantum number in the inp.xml. Additionally the occupations, i.e. diagonal elements are checked that they are in between 0 and the maximum possible occupation

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **nmmplines** (Optional[list[str]]) – list of lines in the n_mmp_mat file

Raises
ValueError – if any of the above checks are violated.

Return type
None

Functions for modifying the xml input file of Fleur with explicit xpath arguments These can still use the schema dict for finding information about the xpath

masci_tools.util.xml.xml_setters_xpaths.**eval_xpath_create**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *create_parents=False*, *occurrences=None*, *number_nodes=1*)

Evaluates and xpath and creates tag if the result is empty

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to place a new tag
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **create_parents** (bool) – bool optional (default False), if True also the parents of the tag are created if they are missing
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to create a tag if the tag is missing. By default all nodes are used.
- **list_return** – if True, the returned quantity is always a list even if only one element is in it
- **number_nodes** (int) – how many identical nodes to create

Return type
list[_Element]

Returns
list of nodes from the result of the xpath expression

masci_tools.util.xml.xml_setters_xpaths.**xml_add_number_to_attrib**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *name*, *number_to_add*, *mode='absolute'*, *occurrences=None*)

Adds a given number to the attribute value in a xmltree. By default the attribute will be shifted on all nodes returned for the specified xpath. If there are no nodes under the specified xpath an error is raised

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the attributes
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (str) – the attribute name to change
- **number_to_add** (Any) – number to add/multiply with the old attribute value
- **mode** (Literal['abs', 'absolute', 'rel', 'relative']) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.

Raises

- **ValueError** – If the attribute is unknown or cannot be float or int
- **ValueError** – If the evaluation of the old values failed
- **ValueError** – If a float result is written to a integer attribute

Return type
Union[_Element, _ElementTree]

Returns
xmltree with shifted attribute

masci_tools.util.xml.xml_setters_xpaths.**xml_add_number_to_first_attrib**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *name*, *number_to_add*, *mode='absolute'*)

Adds a given number to the first occurrence of a attribute value in a xmltree. If there are no nodes under the specified xpath an error is raised

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the attributes
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (str) – the attribute name to change
- **number_to_add** (Any) – number to add/multiply with the old attribute value
- **mode** (Literal['abs', 'absolute', 'rel', 'relative']) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add abs/absolute* adds the old value and *number_to_add*

Raises

- **ValueError** – If the attribute is unknown or cannot be float or int
- **ValueError** – If the evaluation of the old values failed
- **ValueError** – If a float result is written to a integer attribute

Return type
Union[_Element, _ElementTree]

Returns
xmltree with shifted attribute

masci_tools.util.xml.xml_setters_xpaths.**xml_create_tag_schema_dict**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *element*, *create_parents=False*, *number_nodes=1*, *occurrences=None*)

This method evaluates an xpath expression and creates a tag in a xmltree under the returned nodes. If there are no nodes evaluated the subtags can be created with *create_parents=True*

The tag is always inserted in the correct place if a order is enforced by the schema

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to place a new tag
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **element** (QName | str | _Element) – a tag name or etree Element or string representing the XML element to be created
- **create_parents** (bool) – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.
- **number_nodes** (int) – how many identical nodes to create

Raises

ValueError – If the nodes are missing and *create_parents=False*

Return type

Union[_Element, _ElementTree]

Returns

xmltree with created tags

masci_tools.util.xml.xml_setters_xpaths.**xml_set_attrib_value**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *name*, *value*, *occurrences=None*, *create=False*)

Sets an attribute in a xmltree to a given value. By default the attribute will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the attributes
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (str) – the attribute name to set
- **value** (Any) – value or list of values to set
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Raises

- ValueError – If the conversion to string failed
- ValueError – If the tag is missing and *create=False*

- **ValueError** – If the name is not allowed on the base_xpath

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set attribute

masci_tools.util.xml.xml_setters_xpaths.**xml_set_complex_tag**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *changes*, *create=False*)

Recursive function to correctly set tags/attributes for a given tag and it's subtags. Goes through the changes dictionary and decides based on the schema_dict, how the corresponding key has to be handled.

Supports:

- attributes
- tags with text only
- simple tags, i.e. only attributes (can be optional single/multiple)
- complex tags, will recursively create/modify them

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the attributes
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **tag_name** – name of the tag to set
- **changes** (dict[str, Any]) – Keys in the dictionary correspond to names of tags and the values are the modifications to do on this tag (attributename, subdict with changes to the subtag, . . .)
- **create** (bool) – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

Return type
Union[_Element, _ElementTree]

Returns
xmltree with changes to the complex tag

masci_tools.util.xml.xml_setters_xpaths.**xml_set_first_attrib_value**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *name*, *value*, *create=False*)

Sets the first occurrence attribute in a xmltree to a given value. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input

- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the attribute
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (str) – the attribute name to set
- **value** (Any) – value or list of values to set
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Raises

- **ValueError** – If the conversion to string failed
- **ValueError** – If the tag is missing and *create=False*
- **ValueError** – If the attributename is not allowed on the base_xpath

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set attribute

masci_tools.util.xml.xml_setters_xpaths.**xml_set_first_text**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *text*, *create=False*)

Sets the text on the first occurrence of a tag in a xmltree to a given value. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the text
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **text** (Any) – value or list of values to set
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Raises

- **ValueError** – If the conversion to string failed
- **ValueError** – If the tag is missing and *create=False*

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set text

masci_tools.util.xml.xml_setters_xpaths.**xml_set_simple_tag**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *tag_name*, *changes*, *create_parents=False*)

Sets one or multiple *simple* tag(s) in an xmltree. A simple tag can only hold attributes and has no subtags. If the tag can occur multiple times all existing tags are DELETED and new ones are written. If the tag only occurs once it will automatically be created if its missing.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the attributes
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **tag_name** (str) – name of the tag to set
- **changes** (list[dict[str, Any]] | dict[str, Any]) – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to {'name': value}
- **create_parents** (bool) – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set simple tags

masci_tools.util.xml.xml_setters_xpaths.**xml_set_text**(*xmltree*, *schema_dict*, *xpath*, *base_xpath*, *text*, *occurrences=None*, *create=False*)

Sets the text on tags in a xmltree to a given value. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with *convert_to_xml()* if they are not *str* already.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the text
- **base_xpath** (str) – path where to place a new tag without complex syntax ([] conditions and so on)
- **text** (Any) – value or list of values to set
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (bool) – bool optional (default False), if True the tag is created if is missing

Raises

- **ValueError** – If the conversion to string failed
- **ValueError** – If the tag is missing and *create=False*

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set text

Basic functions for modifying the xml input file of Fleur. These functions DO NOT have the ability to create missing tags on the fly. This functionality is added on top in *xml_setters_xpaths* since we need the schema dictionary to do these operations robustly

masci_tools.util.xml.xml_setters_basic.**xml_create_tag**(*xmltree*, *xpath*, *element*, *place_index=None*, *tag_order=None*, *occurrences=None*, *correct_order=True*, *several=True*)

This method evaluates an xpath expression and creates a tag in a xmltree under the returned nodes. If there are no nodes under the specified xpath an error is raised.

The tag is appended by default, but can be inserted at a certain index (*place_index*) or can be inserted according to a given order of tags

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to place a new tag
- **element** (QName | str | _Element) – a tag name, etree Element or string representing the XML element to be created
- **place_index** (Optional[int]) – defines the place where to put a created tag
- **tag_order** (Optional[list[str]]) – defines a tag order
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.
- **correct_order** (bool) – bool, if True (default) and a tag_order is given, that does not correspond to the given order in the xmltree (only order wrong no unknown tags) it will be corrected and a warning is given This is necessary for some edge cases of the xml schemas of fleur
- **several** (bool) – bool, if True multiple tags od the given name are allowed

Raises
ValueError – If the insertion failed in any way (tag_order does not match, failed to insert, . . .)

Return type
Union[_Element, _ElementTree]

Returns
xmltree with created tags

masci_tools.util.xml.xml_setters_basic.**xml_delete_att**(*xmltree*, *xpath*, *name*, *occurrences=None*)

Deletes an attribute in the XML tree

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path to the attribute to be deleted
- **name** (str) – the name of an attribute to delete
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type
Union[_Element, _ElementTree]

Returns
xmltree with deleted attribute

masci_tools.util.xml.xml_setters_basic.**xml_delete_tag**(*xmltree*, *xpath*, *occurrences=None*)

Deletes a tag in the XML tree.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path to the tag to be deleted
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type
Union[_Element, _ElementTree]

Returns
xmltree with deleted tag

masci_tools.util.xml.xml_setters_basic.**xml_replace_tag**(*xmltree*, *xpath*, *element*, *occurrences=None*)

Replace XML tags by a given tag on the given XML tree

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path to the tag to be replaced
- **element** (str | _Element) – an Element or string representing the Element to replace the found tags with
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type
Union[_Element, _ElementTree]

Returns
xmltree with replaced tag

masci_tools.util.xml.xml_setters_basic.**xml_set_attrib_value_no_create**(*xmltree*, *xpath*, *name*, *value*, *occurrences=None*)

Sets an attribute in a xmltree to a given value. By default the attribute will be set on all nodes returned for the specified xpath.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the attributes
- **name** (str) – the attribute name to set
- **value** (Any) – value or list of values to set (if not str they will be converted with *str(value)*)
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.

Raises
ValueError – If the lengths of attribv or occurrences do not match number of nodes

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set attribute

masci_tools.util.xml.xml_setters_basic.**xml_set_text_no_create**(*xmltree*, *xpath*, *text*, *occurrences=None*)

Sets the text of a tag in a xmltree to a given value. By default the text will be set on all nodes returned for the specified xpath.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – an xmltree that represents inp.xml
- **xpath** (Union[str, bytes, XPath, *XPathBuilder*]) – a path where to set the text
- **text** (Any) – value or list of values to set (if not str they will be converted with *str(value)*)
- **occurrences** (Union[int, Iterable[int], None]) – int or list of int. Which occurrence of the node to set. By default all are set.

Raises
ValueError – If the lengths of text or occurrences do not match number of nodes

Return type
Union[_Element, _ElementTree]

Returns
xmltree with set text

XML Getter functions

This module provides functions to extract distinct parts of the fleur xml files for easy versioning and reuse

masci_tools.util.xml.xml_getters.**get_cell**(*xmltree*, *schema_dict*, *logger=None*, *convert_to_angstroem=True*)

Get the Bravais matrix from the given fleur xml file. In addition a list determining in, which directions there are periodic boundary conditions in the system.

> **Warning:** Only the explicit definition of the Bravais matrix is supported. Old inputs containing the *latnam* definitions are not supported

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **logger** (Optional[Logger]) – logger object for logging warnings, errors
- **convert_to_angstroem** (bool) – bool if True the bravais matrix is converted to angstroem

Return type
tuple[ndarray, tuple[bool, bool, bool]]

Returns
numpy array of the bravais matrix and list of boolean values for periodic boundary conditions

masci_tools.util.xml.xml_getters.**get_fleur_modes**(*xmltree*, *schema_dict*, *logger=None*)

Determine the calculation modes of fleur for the given xml file. Calculation modes are things that change the produced files or output in the out.xml files

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
dict[str, Any]

Returns
dictionary with all the extracted calculation modes

The following modes are inspected:

- *jspin*: How many spins are considered in the calculation
- *noco*: Is the calculation non-collinear?
- *soc*: Is spin-orbit coupling included?
- *relax*: Is the calculation a structure relaxation?
- *spex*: Special mode for GW/Spex calculations
- *force_theorem*: Is a Force theorem calculation performed?
- *film*: Is the structure a film system
- *ldau*: Is LDA+U included?
- *dos*: Is it a density of states calculation?
- *band*: Is it a bandstructure calculation?
- *bz_integration*: How is the integration over the Brillouin-Zone performed?

masci_tools.util.xml.xml_getters.**get_kpoints_data**(**args*, ***kwargs*)

RENAMED TO get_kpointsdata

Return type
tuple[list[list[float]] | dict[str, list[list[float]]], list[float] | dict[str, list[float]], ndarray, tuple[bool, bool, bool]]

masci_tools.util.xml.xml_getters.**get_kpointsdata**(*xmltree*, *schema_dict*, *name=None*, *index=None*, *only_used=False*, *logger=None*, *convert_to_angstroem=True*)

Get the kpoint sets defined in the given fleur xml file.

> **Warning:** For file versions before Max5 arguments *name*, *index* and *only_used* have no effect

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **name** (Optional[str]) – str, optional, if given only the kpoint set with the given name is returned
- **index** (Optional[int]) – int, optional, if given only the kpoint set with the given index is returned

- **only_used** (bool) – bool if True only the kpoint list used in the calculation is returned
- **logger** (Optional[Logger]) – logger object for logging warnings, errors
- **convert_to_angstroem** (bool) – bool if True the bravais matrix is converted to angstroem

Return type
tuple[list[list[float]] | dict[str, list[list[float]]], list[float] | dict[str, list[float]], ndarray, tuple[bool, bool, bool]]

Returns
tuple containing the kpoint information

The tuple contains the following entries:

1. **kpoints**
   dict or list (list if there is only one kpoint set), containing the coordinates of the kpoints
2. **weights**
   dict or list (list if there is only one kpoint set), containing the weights of the kpoints
3. **cell**
   numpy array, bravais matrix of the given system
4. **pbc**
   list of booleans, determines in which directions periodic boundary conditions are applicable

masci_tools.util.xml.xml_getters.**get_kpointsdata_max4**(*xmltree*, *schema_dict*, *name=None*, *index=None*, *only_used=False*, *logger=None*, *convert_to_angstroem=True*)

Get the kpoint sets defined in the given fleur xml file.

Note: This function is specific to file version before and including the Max4 release of fleur

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **logger** (Optional[Logger]) – logger object for logging warnings, errors
- **convert_to_angstroem** (bool) – bool if True the bravais matrix is converted to angstroem
- **only_used** (bool) – (Has no effect for Max4) bool if True only the kpoint list used in the calculation is returned
- **name** (Optional[str]) – (Has no effect for Max4)
- **index** (Optional[int]) – (Has no effect for Max4)

Return type
tuple[list[list[float]], list[float], ndarray, tuple[bool, bool, bool]]

Returns
tuple containing the kpoint information

The tuple contains the following entries:

1. **kpoints**
   list containing the coordinates of the kpoints

2. **weights**
   list containing the weights of the kpoints
3. **cell**
   numpy array, bravais matrix of the given system
4. **pbc**
   list of booleans, determines in which directions periodic boundary conditions are applicable

`masci_tools.util.xml.xml_getters.get_nkpts(xmltree, schema_dict, logger=None)`

Get the number of kpoints that will be used in the calculation specified in the given fleur XMl file.

> **Warning:** For file versions before Max5 only kPointList or kPointCount tags will work. However, for kPointCount there is no real guarantee that for every occasion it will correspond to the number of kpoints. So a warning is written out

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
int

Returns
int with the number of kpoints

`masci_tools.util.xml.xml_getters.get_nkpts_max4(xmltree, schema_dict, logger=None)`

Get the number of kpoints that will be used in the calculation specified in the given fleur XMl file. Version specific for Max4 versions or older

> **Warning:** For file versions before Max5 only kPointList or kPointCount tags will work. However, for kPointCount there is no real guarantee that for every occasion it will correspond to the number of kpoints. So a warning is written out

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
int

Returns
int with the number of kpoints

`masci_tools.util.xml.xml_getters.get_parameter_data(*args, **kwargs)`

RENAMED TO get_parameterdata

Return type
dict[str, Any]

masci_tools.util.xml.xml_getters.**get_parameterdata**(*xmltree*, *schema_dict*, *inpgen_ready=True*, *write_ids=True*, *extract_econfig=False*, *allow_special_los=True*, *logger=None*)

This routine returns an python dictionary produced from the inp.xml file, which contains all the parameters needed to setup a new inp.xml from a inpgen input file to produce the same input (for parameters that the inpgen can control)

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **inpgen_ready** (bool) – Bool, return a dict which can be inputted into inpgen while setting atoms
- **write_ids** (bool) – Bool, if True the atom ids are added to the atom namelists
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
dict[str, Any]

Returns
dict, which will lead to the same inp.xml (in case if other defaults, which can not be controlled by input for inpgen, were changed)

masci_tools.util.xml.xml_getters.**get_relaxation_information**(*xmltree*, *schema_dict*, *logger=None*)

Get the relaxation information from the given fleur XML file. This includes the current displacements, energy and posforce evolution

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
dict[str, Any]

Returns
dict with the relaxation information

Raises
ValueError – If no relaxation section is included in the xml tree

masci_tools.util.xml.xml_getters.**get_relaxation_information_pre029**(*xmltree*, *schema_dict*, *logger=None*)

Get the relaxation information from the given fleur XML file. This includes the current displacements, energy and posforce evolution

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree

- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
dict[str, Any]

Returns
dict with the relaxation information

Raises
ValueError – If no relaxation section is included in the xml tree

masci_tools.util.xml.xml_getters.**get_special_kpoints**(*xmltree*, *schema_dict*, *name=None*, *index=None*, *only_used=False*, *logger=None*)

Extract the labeled special kpoints from the given kpointlist

> **Warning:** Only implemented for versions starting with Max5

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **name** (Optional[str]) – str, optional, if given only the kpoint set with the given name is returned
- **index** (Optional[int]) – int, optional, if given only the kpoint set with the given index is returned
- **only_used** (bool) – bool if True only the kpoint list used in the calculation is returned
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
list[tuple[int, str]] | dict[str, list[tuple[int, str]]]

Returns
list of tuples (index, label) for multiple kpoint sets a dict with the names containing the list of tuples is returned

masci_tools.util.xml.xml_getters.**get_special_kpoints_max4**(*xmltree*, *schema_dict*, *name=None*, *index=None*, *only_used=False*, *logger=None*)

Extract the labeled special kpoints from the given kpointlist

> **Warning:** Only implemented for versions starting with Max5

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **name** (Optional[str]) – str, optional, if given only the kpoint set with the given name is returned

- **index** (Optional[int]) – int, optional, if given only the kpoint set with the given index is returned
- **only_used** (bool) – bool if True only the kpoint list used in the calculation is returned
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
list[tuple[int, str]] | dict[str, list[tuple[int, str]]]

Returns
list of tuples (index, label) for multiple kpoint sets a dict with the names containing the list of tuples is returned

masci_tools.util.xml.xml_getters.**get_structure_data**(**args*, ***kwargs*)

RENAMED TO get_structuredata

Return type
tuple[list[*AtomSiteProperties*], ndarray, tuple[bool, bool, bool]]

masci_tools.util.xml.xml_getters.**get_structuredata**(*xmltree*, *schema_dict*, *include_relaxations=True*, *convert_to_angstroem=True*, *normalize_kind_name=True*, *extract_magnetic_moments=True*, *logger=None*, ***kwargs*)

Get the structure defined in the given fleur xml file.

> **Warning:** Only the explicit definition of the Bravais matrix is supported. Old inputs containing the *latnam* definitions are not supported

> **Warning:** In versions `0.5.0` or later the output of the atom sites was restructured to be more interoperable with other IO functions (e.g. *write_inpgen_file()*) The new format returns a list of *AtomSiteProperties* instead of the list of tuples (position, symbol)
>
> For better compatibility this output is not default in `0.5.0` but instead is enabled by `site_namedtuple=True` and a DeprecationWarning is given when this argument is `False`.

Note: In versions `0.5.0` or later the returned atom positions correspond to the relaxed structure if a `relaxation` section is present in the xmltree

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **include_relaxations** (bool) – bool if True and a relaxation section is included the resulting positions correspond to the relaxed structure
- **logger** (Optional[Logger]) – logger object for logging warnings, errors
- **convert_to_angstroem** (bool) – bool if True the bravais matrix is converted to angstroem

- **extract_magnetic_moments** (bool) – bool, if True (default) the magnetic moments are also extracted and put onto the *atom_data* output

Return type
tuple[list[*AtomSiteProperties*], ndarray, tuple[bool, bool, bool]]

Returns
tuple containing the structure information

The tuple contains the following entries:

1. **atom_data**
   list of (named)tuples containing the absolute positions and symbols of the atoms
2. **cell**
   numpy array, bravais matrix of the given system
3. **pbc**
   list of booleans, determines in which directions periodic boundary conditions are applicable

Changed in version 0.7.0: The default for *site_namedtuple* is set to *True*

Changed in version 0.10.0: The argument *site_namedtuple* was deprecated. The old output is no longer supported. If the argument *site_namedtuple* is passed a deprecation warning is shown

masci_tools.util.xml.xml_getters.**get_symmetry_information**(*xmltree*, *schema_dict*, *logger=None*)

Get the symmetry information from the given fleur XML file. This includes the rotation matrices and shifts defined in the `symmetryOperations` tag.

Note: Only the explicit definition of the used symmetry operations in the xml file is supported.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – etree representing the fleur xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – schema dictionary corresponding to the file version of the xmltree
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
tuple[list[ndarray], list[ndarray]]

Returns
tuple of the rotations and their respective shifts

Raises
ValueError – If no symmetryOperations section is included in the xml tree

Basic IO helper functions

Here commonly used functions that do not need aiida-stuff (i.e. can be tested without a database) are collected.

class masci_tools.io.common_functions.**AtomSiteProperties**(*position: list[float], symbol: str, kind: str, magnetic_moment: Literal['up', 'down'] | float | list[float] | None = None*)

namedtuple used for input output of atom sites

kind: str

Alias for field number 2

magnetic_moment: Union[Literal['up', 'down'], float, list[float], None]

Alias for field number 3

position: list[float]

Alias for field number 0

symbol: str

Alias for field number 1

masci_tools.io.common_functions.**_TVectorType**

Generic type variable for atom position types

alias of TypeVar('_TVectorType', bound=Union[Tuple[float, float, float], List[float], ndarray])

masci_tools.io.common_functions.**abs_to_rel**(*vector, cell*)

Converts a position vector in absolute coordinates to relative coordinates.

Parameters

- **vector** (TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])) – list or np.array of length 3, vector to be converted
- **cell** (list[list[float]] | ndarray) – Bravais matrix of a crystal 3x3 Array, List of list or np.array

Return type
TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])

Returns
list of length 3 of scaled vector, or False if vector was not length 3

masci_tools.io.common_functions.**abs_to_rel_f**(*vector, cell, pbc*)

Converts a position vector in absolute coordinates to relative coordinates for a film system.

Parameters

- **vector** (TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])) – list or np.array of length 3, vector to be converted
- **cell** (list[list[float]] | ndarray) – Bravais matrix of a crystal 3x3 Array, List of list or np.array
- **pbc** (tuple[bool, bool, bool]) – Boundary conditions, List or Tuple of 3 Boolean

Return type
TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])

Returns
list of length 3 of scaled vector, or False if vector was not length 3

masci_tools.io.common_functions.**angles_to_vec**(*magnitude*, *theta*, *phi*)

convert (magnitude, theta, phi) to (x,y,z)

theta/phi need to be in radians!

Input can be single number, list of numpy.ndarray data Returns x,y,z vector

Return type
ndarray

masci_tools.io.common_functions.**camel_to_snake**(*name*)

Converts camelCase to snake_case variable names Used in the Fleur parser to convert attribute names from the xml files

Return type
str

masci_tools.io.common_functions.**convert_to_fortran**(*val*, *quote_strings=True*)

Parameters
val (Any) – the value to be read and converted to a Fortran-friendly string.

Return type
str

masci_tools.io.common_functions.**convert_to_fortran_string**(*string*)

converts some parameter strings to the format for the inpgen :type string: str :param string: some string :rtype: str :returns: string in right format (extra "" if not already present)

masci_tools.io.common_functions.**convert_to_pystd**(*value*)

Recursively convert numpy datatypes to standard python, needed by aiida-core.

Return type
Any

Usage:
converted = convert_to_pystd(to_convert)

where *to_convert* can be a dict, array, list, or single valued variable

masci_tools.io.common_functions.**fac**(*n*)

Returns the factorial of n

Return type
int

masci_tools.io.common_functions.**filter_out_empty_dict_entries**(*dict_to_filter*)

Filter out entries in a given dict that correspond to empty values. At the moment this is empty lists, dicts and None

Parameters
dict_to_filter (dict) – dict to filter

Return type
dict

Returns
dict without empty entries

masci_tools.io.common_functions.**find_symmetry_relation**(*from_pos*, *to_pos*, *rotations*, *shifts*, *cell*, *relative_pos=False*, *film=False*)

Find symmetry relation between the given vectors. This functions assumes that a symmetry relation exists otherwise an error is raised

Parameters

- **from_pos** (Union[Tuple[float, float, float], List[float], ndarray]) – vector to rotate
- **to_pos** (Union[Tuple[float, float, float], List[float], ndarray]) – vector to rotate to
- **rotations** (list[ndarray]) – list of np.arrays with the given symmetry rotations
- **shifts** (list[ndarray]) – list of np.arrays with the given shifts for the symmetry operations
- **cell** (list[list[float]] | ndarray) – Bravais matrix of a crystal 3x3 Array, List of list or np.array
- **relative_pos** (bool) – bool if True the given vectors are assuemd to be in internal coordinates
- **film** (bool) – bool if True the vectors are assumed to be film coordinates

Return type
tuple[ndarray, ndarray]

Returns
tuple of rotation and shift mapping `from_pos` to `to_pos`

Raises
ValueError – If no symmetry relation is found

masci_tools.io.common_functions.**get_corestates_from_potential**(*potfile='potential'*)

Read core states from potential file

Return type
tuple[list, list, list]

masci_tools.io.common_functions.**get_ef_from_potfile**(*potfile*)

extract fermi energy from potfile

Return type
float

masci_tools.io.common_functions.**get_highest_core_state**(*nstates*, *energies*, *lmoments*)

Find highest lying core state from list of core states, needed to find and check energy contour

Return type
tuple[int, float, str]

masci_tools.io.common_functions.**get_outfile_txt**(*outfile*)

Get the content of a file In case the outfile is a file handle, we just roll it back and read everything in again. For an ordinary file path we open the file in a context manager and then read it.

masci_tools.io.common_functions.**get_pauli_matrix**(*direction*, *alpha=0.0*, *beta=0.0*)

Get the pauli matrix with additional rotation applied

Parameters

- **direction** (Literal['x', 'y', 'z']) – str (x,y or z) for which pauli matrix to return

- **alpha** (float) – angle in radians
- **beta** (float) – angle in radians

Return type
ndarray

masci_tools.io.common_functions.**get_spin_rotation**(*alpha*, *beta*)

Get matrix to rotate the spin frame by the given angles alpha/beta

Parameters

- **alpha** (float) – angle in radians
- **beta** (float) – angle in radians

Return type
ndarray

masci_tools.io.common_functions.**get_wigner_matrix**(*l*, *alpha*, *beta*, *gamma=0.0*, *inverse=False*)

Produces the wigner rotation matrix for the density matrix

Parameters

- **l** (int) – int, orbital quantum number
- **alpha** (float) – float, angle (radian) corresponds to euler angle alpha
- **beta** (float) – float, angle (radian) corresponds to euler angle beta
- **gamma** (float) – float, angle (radian) corresponds to euler angle gamma

Return type
ndarray

masci_tools.io.common_functions.**interpolate_dos**(*dosfile*, *return_original=False*)

interpolation function copied from complexdos3 fortran code

Principle of DOS here: Two-point contour integration for DOS in the middle of the two points. The input DOS and energy must be complex. Parameter deltae should be of the order of magnitude of eim:

```
      <-2*deltae->   _
           /\        |     DOS=(n(1)+n(2))/2 + (n(1)-n(2))*eim/deltae
          /  \       |
        (1)  (2)   2*i*eim=2*i*pi*Kb*Tk
        /      \     |
       /        \    |
------------------------ (Real E axis)
```

Parameters
input – either absolute path of 'complex.dos' file or file handle to it

Returns
E_Fermi, numpy array of interpolated dos

Note
output units are in Ry!

masci_tools.io.common_functions.**is_sequence**(*arg*)

Checks if arg is a sequence

Return type
bool

masci_tools.io.common_functions.**open_general**(*filename_or_handle*, *iomode=None*)

Open a file directly from a path or use a file handle if that is given. Also take care of closed files by reopenning them. This is intended to be used like this:

```
with open_general(outfile) as f:
    txt = f.readlines()
```

Return type
IO[Any]

masci_tools.io.common_functions.**rel_to_abs**(*vector*, *cell*)

Converts a position vector in internal coordinates to absolute coordinates in Angstrom.

Parameters

- **vector** (TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])) – list or np.array of length 3, vector to be converted
- **cell** (list[list[float]] | ndarray) – Bravais matrix of a crystal 3x3 Array, List of list or np.array

Return type
TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])

Returns
list of length 3 of scaled vector, or False if vector was not length 3

masci_tools.io.common_functions.**rel_to_abs_f**(*vector*, *cell*)

Converts a position vector in internal coordinates to absolute coordinates in Angstrom for a film structure (2D).

Return type
TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])

masci_tools.io.common_functions.**skipHeader**(*seq*, *n*)

Iterate over a sequence skipping the first n elements

Return type
Generator[Any, None, None]

Args:
seq (iterable): Iterable sequence n (int): Number of Elements to skip in the beginning of the sequence

Yields:
item: Elements in seq after the first n elements

masci_tools.io.common_functions.**vec_to_angles**(*vec*)

converts vector (x,y,z) to (magnitude, theta, phi)

Return type
tuple[ndarray, ndarray, ndarray] | tuple[float, float, float]

Small utility functions for inspecting hdf files and converting the complete file structure into a python dictionary

masci_tools.io.hdf5_util.**h5dump**(*file*, *group='/'*)

Shows the overall filestructure of an hdf file Goes through all groups and subgroups and prints the attributes or the shape and datatype of the datasets

Parameters
filepath – path to the hdf file

Return type
None

masci_tools.io.hdf5_util.**hdfList**(*name*, *obj*)

Print the name of the current object (indented to create a nice tree structure)

Also prints attribute values and dataset shapes and datatypes

Return type
None

masci_tools.io.hdf5_util.**read_groups**(*hdfdata*, *flatten=False*)

Recursive function to read a hdf datastructure and extract the datasets and attributes

Parameters

- **hdfdata** (Group) – current hdf group to process
- **flatten** (bool) – bool, if True the dictionary will be flattened (does not check for lost information)

Return type
tuple[dict[str, Any], dict[str, Any]]

Returns
two dictionaries, one with the datasets the other with the attributes in the file

masci_tools.io.hdf5_util.**read_hdf_simple**(*file*, *flatten=False*)

Reads in an hdf file and returns its context in a nested dictionary

Parameters

- **filepath** – path or filehandle to the hdf file
- **flatten** (bool) – bool, if True the dictionary will be flattened (does not check for lost information)

Return type
tuple[dict[str, Any], dict[str, Any]]

Returns
two dictionaries, one with the datasets the other with the attributes in the file

Non unique group attribute or dataset names will be overwritten in the return dict

Logging Utility

This module defines useful utility for logging related functionality

class masci_tools.util.logging_util.**DictHandler**(*log_dict*, *ignore_unknown_levels=False*, ***kwargs*)

Custom Handler for the logging module inserting logging messages into a given dictionary.

Messages are grouped into list under the names of the error categories. Keyword arguments can be used to modify the keys for the different levels

emit(*record*)

Emit a record.

Return type
None

class masci_tools.util.logging_util.**OutParserLogAdapter**(*logger*, *extra=None*)

This adapter expects the passed in dict-like object to have a 'iteration' key, whose value is prepended as [Iteration i] to the message

process(*msg*, *kwargs*)

Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword args or both. Return the message and kwargs modified (or not) to suit your needs.

Normally, you'll only need to override this one method in a LoggerAdapter subclass for your specific needs.

Return type
tuple[str, dict[str, Any]]

Fleur parser utility

This module contains helper functions for extracting information easily from the schema_dicts defined for the Fleur input/output

Also provides convenient functions to use just a attribute name for extracting the attribute from the right place in the given etree

masci_tools.util.schema_dict_util.**attrib_exists**(*node*, *schema_dict*, *name*, *logger=None*, *iteration_path=False*, *filters=None*, ***kwargs*)

Evaluates whether the attribute exists in the xmltree based on the given name and additional further specifications with the available type information

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*SchemaDict*) – dict, containing all the path information and more
- **name** (str) – str, name of the attribute
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised
- **iteration_path** (bool) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

Return type
`bool`

Returns
bool, True if any tag with the attribute exists

`masci_tools.util.schema_dict_util.`**`ensure_relaxation_xinclude`**(*xmltree*, *schema_dict*)

Ensure that the xinclude tag for the `relax.xml` is added if no `relaxation` is present in the `inp.xml`

Parameters

- **xmltree** (`_ElementTree`) – an xml-tree which will be processed
- **schema_dict** (*`InputSchemaDict`*) – Schema dictionary containing all the necessary information

Return type
`None`

Returns
xmltree, which either contains the relaxation section or a xinclude tag

`masci_tools.util.schema_dict_util.`**`eval_simple_xpath`**(*node*, *schema_dict*, *name*, *logger=None*, *iteration_path=False*, *filters=None*, *list_return=False*, ***kwargs*)

Evaluates a simple xpath expression of the tag in the xmltree based on the given name and additional further specifications with the available type information

Parameters

- **node** (`Union[_Element, _ElementTree, XPathElementEvaluator]`) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*`SchemaDict`*) – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See *`XPathBuilder`* for details
- **list_return** (`bool`) – bool, if True a list is always returned

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

Return type
`_Element` | `list[_Element]`

Returns
etree Elements obtained via the simple xpath expression

`masci_tools.util.schema_dict_util.evaluate_attribute(node, schema_dict, name, constants=None, logger=None, complex_xpath=None, filters=None, iteration_path=False, **kwargs)`

Evaluates the value of the attribute based on the given name and additional further specifications with the available type information

Parameters

- **node** (`Union[_Element, _ElementTree, XPathElementEvaluator]`) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*`SchemaDict`*) – dict, containing all the path information and more
- **name** (`str`) – str, name of the attribute
- **constants** (`Optional[dict[str, float]]`) – dict, contains the defined constants
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised
- **complex_xpath** (`Union[str, bytes, XPath,` *`XPathBuilder`*`, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See *`XPathBuilder`* for details

Kwargs:

param tag_name
str, name of the tag where the attribute should be parsed

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param exclude
list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

param list_return
if True, the returned quantity is always a list even if only one element is in it

param optional
bool, if True and no logger given none or an empty list is returned

Return type
Any

Returns
list or single value, converted in convert_xml_attribute

`masci_tools.util.schema_dict_util.evaluate_parent_tag(node, schema_dict, name, constants=None, logger=None, complex_xpath=None, iteration_path=False, filters=None, **kwargs)`

Evaluates all attributes of the parent tag based on the given name and additional further specifications with the available type information

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*SchemaDict*) – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **constants** (Optional[dict[str, float]]) – dict, contains the defined constants
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (bool) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param only_required
bool (optional, default False), if True only required attributes are parsed

param ignore
list of str (optional), attributes not to parse

param list_return
if True, the returned quantity is always a list even if only one element is in it

param strict_missing_error
if True, and no logger is given an error is raised if any attribute is not found

Return type
Any

Returns
dict, with attribute values converted via convert_xml_attribute

masci_tools.util.schema_dict_util.**evaluate_single_value_tag**(*node*, *schema_dict*, *name*, *constants=None*, *logger=None*, *complex_xpath=None*, ***kwargs*)

Evaluates the value and unit attribute of the tag based on the given name and additional further specifications with the available type information

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*SchemaDict*) – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **constants** (Optional[dict[str, float]]) – dict, contains the defined constants
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param only_required
bool (optional, default False), if True only required attributes are parsed

param ignore
list of str (optional), attributes not to parse

param list_return
if True, the returned quantity is always a list even if only one element is in it

param strict_missing_error
if True, and no logger is given an error is raised if any attribute is not found

param iteration_path
bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters
Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
Any

Returns
value and unit, both converted in convert_xml_attribute

masci_tools.util.schema_dict_util.**evaluate_tag**(*node*, *schema_dict*, *name*, *constants=None*, *logger=None*, *subtags=False*, *text=True*, *complex_xpath=None*, *iteration_path=False*, *filters=None*, ***kwargs*)

Evaluates all attributes of the tag based on the given name and additional further specifications with the available type information

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*SchemaDict*) – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **constants** (Optional[dict[str, float]]) – dict, contains the defined constants
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised
- **subtags** (bool) – optional bool, if True the subtags of the given tag are evaluated
- **text** (bool) – optional bool, if True the text of the tag is also parsed
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (bool) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param only_required
bool (optional, default False), if True only required attributes are parsed

param ignore
list of str (optional), attributes not to parse

param list_return
if True, the returned quantity is always a list even if only one element is in it

param strict_missing_error
if True, and no logger is given an error is raised if any attribute is not found

Return type
Any

Returns
dict, with attribute values converted via convert_xml_attribute

masci_tools.util.schema_dict_util.**evaluate_text**(*node*, *schema_dict*, *name*, *constants=None*, *logger=None*, *complex_xpath=None*, *iteration_path=False*, *filters=None*, ***kwargs*)

Evaluates the text of the tag based on the given name and additional further specifications with the available type information

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*SchemaDict*) – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **constants** (Optional[dict[str, float]]) – dict, contains the defined constants
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised
- **complex_xpath** (Union[str, bytes, XPath, *XPathBuilder*, None]) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (bool) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param list_return
if True, the returned quantity is always a list even if only one element is in it

param optional
bool, if True and no logger given none or an empty list is returned

Return type
Any

Returns
list or single value, converted in convert_xml_text

masci_tools.util.schema_dict_util.**get_number_of_nodes**(*node*, *schema_dict*, *name*, *logger=None*, ***kwargs*)

Evaluates the number of occurrences of the tag in the xmltree based on the given name and additional further specifications with the available type information

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*SchemaDict*) – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param iteration_path
bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters
Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
int

Returns
number of nodes for the given tag

masci_tools.util.schema_dict_util.**read_constants**(*root*, *schema_dict*, *logger=None*)

Reads in the constants defined in the inp.xml and returns them combined with the predefined constants from fleur as a dictionary

Parameters

- **root** (Union[_Element, _ElementTree, XPathElementEvaluator]) – root of the etree of the inp.xml file
- **schema_dict** (*SchemaDict*) – schema_dictionary of the version of the file to read (inp.xml or out.xml)
- **logger** (Optional[Logger]) – logger object for logging warnings, errors

Return type
dict[str, float]

Returns
a python dictionary with all defined constants

masci_tools.util.schema_dict_util.**reverse_xinclude**(*xmltree*, *schema_dict*, *included_tags*, ***kwargs*)

Split the xmltree back up according to the given included tags. The original xmltree will be returned with the corresponding xinclude tags and the included trees are returned in a dict mapping the inserted filename to the extracted tree

Tags for which no known filename is known are returned under unknown-1.xml, … The following tags have known filenames:

- *relaxation*: `relax.xml`
- *kPointLists*: `kpts.xml`
- *symmetryOperations*: `sym.xml`
- *atomSpecies*: `species.xml`
- *atomGroups*: `atoms.xml`

Additional mappings can be given in the keyword arguments

Parameters

- **xmltree** (_ElementTree) – an xml-tree which will be processed

- **schema_dict** (*InputSchemaDict*) – Schema dictionary containing all the necessary information
- **included_tags** (Iterable[str]) – Iterable of str, containing the names of the tags to be excluded

Return type
tuple[_ElementTree, dict[PathLike | str, _ElementTree]]

Returns
xmltree with the inseerted xinclude tags and a dict mapping the filenames to the excluded trees

Raises
ValueError – if the tag can not be found in the given xmltree

masci_tools.util.schema_dict_util.**tag_exists**(*node*, *schema_dict*, *name*, *logger=None*, ***kwargs*)

Evaluates whether the tag exists in the xmltree based on the given name and additional further specifications with the available type information

Parameters

- **node** (Union[_Element, _ElementTree, XPathElementEvaluator]) – etree Element, on which to execute the xpath evaluations
- **schema_dict** (*SchemaDict*) – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **logger** (Optional[Logger]) – logger object for logging warnings, errors, if not provided all errors will be raised

Kwargs:

param contains
str, this string has to be in the final path

param not_contains
str, this string has to NOT be in the final path

param iteration_path
bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters
Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type
bool

Returns
bool, True if any nodes with the path exist

This module contains the functions necessary to parse mathematical expressions with predefined constants given in the inp.xml file of Fleur

exception masci_tools.util.fleur_calculate_expression.**MissingConstant**

Exception raised when a constant appearing in a expression is not defined

masci_tools.util.fleur_calculate_expression.**calculate_expression**(*expression*, *constants=None*)

Recursively evaluates the given expression string with the given defined constants

Parameters

- **expression** (str | float | int) – str containing the expression to be parsed
- **constants** (Optional[dict[str, float]]) – dict with all defined constants (predefined in the Fleur code or defined in the inp.xml)

Return type
float | int

Returns
float value of the given expression string

masci_tools.util.fleur_calculate_expression.**calculate_expression_partial**(*expression*, *constants=None*, *prevCommand=None*)

Recursively evaluates the given expression string with the given defined constants and returns the unevaluated part of the expression

Parameters

- **expression** (str | float | int) – str containing the expression to be parsed
- **constants** (Optional[dict[str, float]]) – dict with all defined constants (predefined in the Fleur code or defined in the inp.xml)
- **prevCommand** (Optional[str]) – str, which gives the command before the beginning of the current block if it is given the calculation is stopped, when a command is encountered, which should be exectuted after prevCommand (order of operations)

Return type
tuple[float | int, str]

Returns
float value of the given expression string

masci_tools.util.fleur_calculate_expression.**evaluate_bracket**(*expression*, *constants*)

Evaluates the bracket opened at the start of the expression

Parameters

- **expression** (str) – expression to be parsed
- **constants** (dict[str, float]) – dict with defined constants

Return type
tuple[float | int, str]

Returns
value of the expression inside the brackets and remaining string of the expression after the corresponding closed bracket

masci_tools.util.fleur_calculate_expression.**get_first_number**(*expression*)

Reads the number in the beginning of the expression string. This number can begin with a sign +-, a number or the decimal point

Parameters
expression (str) – str of the expression

Return type
tuple[float, str]

Returns
float value of the number in the beginning and the string of the remaining expression

masci_tools.util.fleur_calculate_expression.**get_first_string**(*expression*)

Reads the letter string in the beginning of the expression string.

Parameters
expression (str) – str of the expression

Return type
tuple[str, str]

Returns
letter string in the beginning and the string of the remaining expression

This module contains custom conversion functions for the outxml_parser, which cannot be handled by the standard parsing framework

masci_tools.io.parsers.fleur.outxml_conversions.**calculate_total_magnetic_moment**(*out_dict*, *logger*)

Calculate the the total magnetic moment per cell

Parameters
out_dict (dict[str, Any]) – dict with the already parsed information

Return type
dict[str, Any]

masci_tools.io.parsers.fleur.outxml_conversions.**calculate_walltime**(*out_dict*, *logger*)

Calculate the walltime from start and end time

Parameters

- **out_dict** (dict[str, Any]) – dict with the already parsed information
- **logger** (Logger) – logger object for logging warnings, errors, if not provided all errors will be raised

Return type
dict[str, Any]

masci_tools.io.parsers.fleur.outxml_conversions.**convert_forces**(*out_dict*, *logger*)

Convert the parsed forces from a iteration

Parameters
out_dict (dict[str, Any]) – dict with the already parsed information

Return type
dict[str, Any]

masci_tools.io.parsers.fleur.outxml_conversions.**convert_htr_to_ev**(*out_dict*, *name*, *converted_name=None*, *pop=False*, *add_unit=True*, *logger=None*)

Convert value from htr to eV

Return type
dict[str, Any]

masci_tools.io.parsers.fleur.outxml_conversions.**convert_ldahia_definitions**(*out_dict*, *logger*)

Convert the parsed information from LDA+U into a more readable dict

ldau_info has keys for each species with LDA+U ({species_name}/{atom_number}) and this in turn contains a dict with the LDA+U definition for the given orbital (spdf)

Parameters
out_dict (dict[str, Any]) – dict with the already parsed information

Return type
dict[str, Any]

masci_tools.io.parsers.fleur.outxml_conversions.**convert_ldau_definitions**(*out_dict*, *logger*)

Convert the parsed information from LDA+U into a more readable dict

ldau_info has keys for each species with LDA+U ({species_name}/{atom_number}) and this in turn contains a dict with the LDA+U definition for the given orbital (spdf)

Parameters
out_dict (dict[str, Any]) – dict with the already parsed information

Return type
dict[str, Any]

masci_tools.io.parsers.fleur.outxml_conversions.**convert_relax_info**(*out_dict*, *logger*)

Convert the general relaxation information

Parameters
out_dict (dict[str, Any]) – dict with the already parsed information

Return type
dict[str, Any]

Functions for expanding/splitting or converting electron configuration strings

masci_tools.util.econfig.**convert_fleur_config_to_econfig**(*fleurconf_str*, *keep_spin=False*)

'[Kr] (4d3/2) (4d5/2) (4f5/2) (4f7/2)' -> '[Kr] 4d10 4f14', or '[Kr] 4d3/2 4d5/2 4f5/2 4f7/2'

for now only use for coreconfig, it will fill all orbitals, since it has no information on the filling.

Parameters

- **fleurconf_str** (str) – string of the electron config like it is read from the inp.xml
- **keep_spin** (bool) – bool if True the spin indices will be kept in the converted string

Return type
str

Returns
string of the electron config to be used in the inpgen

masci_tools.util.econfig.**get_coreconfig**(*element*, *full=False*)

returns the econfiguration as a string of an element.

Parameters

- **element** (str | int) – element string
- **full** (bool) – a bool if True the econfig without [He]… is returned

Return type
Optional[str]

Returns
coreconfig string

masci_tools.util.econfig.**get_econfig**(*element*, *full=False*)

returns the econfiguration as a string of an element.

Parameters

- **element** (str | int) – element string
- **full** (bool) – a bool if True the econfig without [He]… is returned

Return type
Optional[str]

Returns
a econfig string

`masci_tools.util.econfig.rek_econ(econfigstr)`

recursive routine to return a full econfig '[Xe] 4f14 | 5d10 6s2 6p4' -> '1s 2s … 4f14 | 5d10 6s2 6p4'

Parameters
econfigstr (str) – electron config string to expand

Return type
Optional[str]

Returns
expanded econfig string

6.1.2.6 Basic Fleur Schema parser functions

Load all fleur schema related functions

`class masci_tools.io.parsers.fleur_schema.AttributeType(base_type: str, length: int | Literal['unbounded'] | None)`

Type for describing the types of attributes/text

`exception masci_tools.io.parsers.fleur_schema.IncompatibleSchemaVersions`

Exception raised when it is known that a given output version and input version cannot be compiled into a complete fleur output xml schema

`class masci_tools.io.parsers.fleur_schema.InputSchemaDict(*args, xmlschema=None, **kwargs)`

This class contains information parsed from the FleurInputSchema.xsd

The keys contain the following information:

inp_version
Version string of the input schema represented in this object

tag_paths
simple xpath expressions to all valid tag names Multiple paths or ambiguous tag names are parsed as a list

_basic_types
Parsed definitions of all simple Types with their respective base type (int, float, …) and evtl. length restrictions (Only used in the schema construction itself)

attrib_types
All possible base types for all valid attributes. If multiple are possible a list, with 'string' always last (if possible)

simple_elements
All elements with simple types and their type definition with the additional attributes

unique_attribs
All attributes and their paths, which occur only once and have a unique path

unique_path_attribs
All attributes and their paths, which have a unique path but occur in multiple places

other_attribs
All attributes and their paths, which are not in 'unique_attribs' or 'unique_path_attribs'

omitt_contained_tags
All tags, which only contain a list of one other tag

tag_info
For each tag (path), the valid attributes and tags (optional, several, order, simple, text)

classmethod fromPath(*path*)

load the FleurInputSchema dict for the specified FleurInputSchema file

Parameters
path (PathLike) – path to the input schema file

Return type
InputSchemaDict

Returns
InputSchemaDict object with the information for the provided file

classmethod fromVersion(*version*, *logger=None*, *no_cache=False*)

load the FleurInputSchema dict for the specified version

Parameters

- **version** (str) – str with the desired version, e.g. '0.33'
- **logger** (Optional[Logger]) – logger object for warnings, errors and information, ...

Return type
InputSchemaDict

Returns
InputSchemaDict object with the information for the provided version

property inp_version: tuple[int, int]

Returns the input version as an integer for comparisons (> or <)

exception masci_tools.io.parsers.fleur_schema.**NoPathFound**

Exception raised when no path is found for a given tag/attribute

exception masci_tools.io.parsers.fleur_schema.**NoUniquePathFound**

Exception raised when no unique path is found for a given tag/attribute

class masci_tools.io.parsers.fleur_schema.**OutputSchemaDict**(**args*, *xmlschema=None*, ***kwargs*)

This object contains information parsed from the FleurOutputSchema.xsd

The keys contain the following information:

out_version
Version string of the output schema represented in this class

input_tag
Name of the element containing the fleur input

iteration_tags
Names of the elements that can contain all iteration tags

tag_paths
simple xpath expressions to all valid tag names not in an iteration Multiple paths or ambiguous tag names are parsed as a list

iteration_tag_paths
simple relative xpath expressions to all valid tag names inside an iteration. Multiple paths or ambiguous tag names are parsed as a list

_basic_types
Parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions (Only used in the schema construction itself)

_input_basic_types
Part of the parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions from the input schema (Only used in the schema construction itself)

attrib_types
All possible base types for all valid attributes. If multiple are possible a list, with 'string' always last (if possible)

simple_elements
All elements with simple types and their type definition with the additional attributes

unique_attribs
All attributes and their paths, which occur only once and have a unique path outside of an iteration

unique_path_attribs
All attributes and their paths, which have a unique path but occur in multiple places outside of an iteration

other_attribs
All attributes and their paths, which are not in 'unique_attribs' or 'unique_path_attribs' outside of an iteration

iteration_unique_attribs
All attributes and their relative paths, which occur only once and have a unique path inside of an iteration

iteration_unique_path_attribs
All attributes and their relative paths, which have a unique path but occur in multiple places inside of an iteration

iteration_other_attribs
All attributes and their relative paths, which are not in 'unique_attribs' or 'unique_path_attribs' inside of an iteration

omitt_contained_tags
All tags, which only contain a list of one other tag

tag_info
For each tag outside of an iteration (path), the valid attributes and tags (optional, several, order, simple, text)

iteration_tag_info
For each tag inside of an iteration (relative path), the valid attributes and tags (optional, several, order, simple, text)

`classmethod fromPath`(*path*, *inp_path=None*, *inpschema_dict=None*)

load the FleurOutputSchema dict for the specified paths

Parameters

- **path** (PathLike) – path to the FleurOutputSchema file
- **inp_path** (Optional[PathLike]) – path to the FleurInputSchema file (defaults to same folder as path)

Return type
OutputSchemaDict

Returns
OutputSchemaDict object with the information for the provided files

classmethod fromVersion(*version*, *inp_version=None*, *logger=None*, *no_cache=False*)

load the FleurOutputSchema dict for the specified version

Parameters

- **version** (str) – str with the desired version, e.g. '0.33'
- **inp_version** (Optional[str]) – str with the desired input version, e.g. '0.33' (defaults to version)
- **logger** (Optional[Logger]) – logger object for warnings, errors and information, ...

Return type
OutputSchemaDict

Returns
OutputSchemaDict object with the information for the provided versions

property inp_version: tuple[int, int]

Returns the input version as an integer for comparisons (> or <)

iteration_attrib_xpath(*name*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the attribute and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **name** (str) – str, name of the attribute
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other
- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *iteration_tag_xpath()*
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath to the tag with the given attribute

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

iteration_tag_xpath(*name*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

property out_version: tuple[int, int]

Returns the output version as an integer for comparisons (> or <)

relative_iteration_attrib_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

Tries to find a unique relative path from the schema_dict based on the given name of the attribute name of the root, from which the path should be relative and additional further specifications

Parameters

- **schema_dict** – dict, containing all the path information and more
- **name** (str) – str, name of the attribute
- **root_tag** (str) – str, name of the tag from which the path should be relative
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other
- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *relative_iteration_tag_xpath()*
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

relative_iteration_tag_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **iteration_tag** (str) – name of the tag containing the iteration information

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

class masci_tools.io.parsers.fleur_schema.**SchemaDict**(**args*, *xmlschema=None*, ***kwargs*)

Base class for schema dictionaries. Is locked on initialization with *freeze()*. Holds a reference to the xmlSchema for validating files.

Also provides interfaces for utility functions

Parameters
xmlschema (Optional[XMLSchema]) – etree.XMLSchema object for validating files

All other arguments are passed on to *LockableDict*

attrib_xpath(*name*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*)

Tries to find a unique path from the schema_dict based on the given name of the attribute and additional further specifications

Parameters

- **name** (str) – str, name of the attribute
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other

- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *tag_xpath()*

Return type
str

Returns
str, xpath to the tag with the given attribute

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

classmethod clear_cache()

Remove all stored entries in the schema dictionary cache

Return type
None

relative_attrib_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*)

Tries to find a unique relative path from the schema_dict based on the given name of the attribute name of the root, from which the path should be relative and additional further specifications

Parameters

- **schema_dict** – dict, containing all the path information and more
- **name** (str) – str, name of the attribute
- **root_tag** (str) – str, name of the tag from which the path should be relative
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **exclude** (Optional[Iterable[str]]) – list of str, here specific types of attributes can be excluded valid values are: settable, settable_contains, other
- **tag_name** (Optional[str]) – str, if given this name will be used to find a path to a tag with the same name in *relative_tag_xpath()*

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

relative_tag_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*)

Tries to find a unique relative path from the schema_dict based on the given name of the tag name of the root, from which the path should be relative and additional further specifications

Parameters

- **name** (str) – str, name of the tag

- **root_tag** (str) – str, name of the tag from which the path should be relative
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path

Return type
str

Returns
str, xpath for the given tag

Raises
ValueError – If no unique path could be found

tag_info(*name*, *contains=None*, *not_contains=None*, *parent=False*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications and returns the tag_info entry for this tag

Parameters

- **schema_dict** – dict, containing all the path information and more
- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path
- **parent** (bool) – bool, if True the tag_info for the parent of the tag is returned

Return type
TagInfo

Returns
dict, tag_info for the found xpath

tag_xpath(*name*, *contains=None*, *not_contains=None*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications

Parameters

- **name** (str) – str, name of the tag
- **contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to be in the final path
- **not_contains** (Union[str, Iterable[str], None]) – str or list of str, this string has to NOT be in the final path

Return type
str

Returns
str, xpath for the given tag

Raises

- *NoPathFound* – If no path matching the criteria could be found

- *NoUniquePathFound* – If multiple paths matching the criteria are found

validate(*xmltree*, *logger=None*, *header=''*)

Validate the given XML tree against the schema

Parameters

- **xmltree** (_ElementTree) – XML tree to validate
- **logger** (Optional[Logger]) – Logger to relay evlt warnings/errors

Return type
None

masci_tools.io.parsers.fleur_schema.**list_available_versions**(*output_schema*)

List the available versions for the schema

Parameters
output_schema (bool) – bool, if True search for FleurOutputSchema.xsd otherwise FleurInputSchema.xsd

Return type
list[str]

Returns
list version string of the available versions

masci_tools.io.parsers.fleur_schema.**schema_dict_version_dispatch**(*output_schema=False*)

Decorator for creating variations of functions based on the inp/out version of the schema_dict. All functions here need to have the signature:

```
def f(node, schema_dict, *args, **kwargs):
    pass
```

So schema_dict is the second positional argument

Inspired by singledispatch in the functools module

Return type
Callable[[TypeVar(F, bound= Callable[..., Any])], *SchemaDictDispatch*[TypeVar(F, bound= Callable[..., Any])]]

functions to extract information about the fleur schema input or output

class masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**AttributeType**(*base_type: str*, *length: int* | *Literal['unbounded']* | *None*)

Type for describing the types of attributes/text

class masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**TagInfo**

Dict representing the entries for the tag information.

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**convert_str_version_number**(*version_str*)

Convert the version number as a integer for easy comparisons

Parameters
version_str (str) – str of the version number, e.g. '0.33'

Return type
tuple[int, int]

Returns
tuple of ints representing the version str

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**eval_single_string_attribute**(*xmlschema xpath, **variables*)

Wrapper around the xpath calls in this module. Makes sure the return value is a single string (not cached)

Parameters

- **xmlschema_evaluator** – etree.XPathEvaluator for the schema
- **xpath** – str, xpath expression to evaluate

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**extract_attribute_types**(*xmlschema_evalu **kwargs*)

Determine the required type of all attributes

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
CaseInsensitiveDict[str, list[*AttributeType*]]

Returns
possible types of the attributes in a dictionary, if multiple types are possible a list is inserted for the tag

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**extract_text_types**(*xmlschema_evaluator, **kwargs*)

Determine the required type of all elements with text

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
CaseInsensitiveDict[str, list[*AttributeType*]]

Returns
possible types of the attributes in a dictionary, if multiple types are possible a list is inserted for the tag

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_basic_types**(*xmlschema_evaluator, input_basic_types=None, **kwargs*)

find all types, which can be traced back directly to a base_type

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
dict[str, list[*AttributeType*]]

Returns
dictionary with type names and their corresponding type_definition meaning a dictionary with possible base types and evtl. length restriction

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_input_tag**(*xmlschema_evaluator*, ***kwargs*)

Returns the tag for the input type element of the outxmlschema

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
str

Returns
name of the element with the type 'FleurInputType'

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_iteration_tags**(*xmlschema_evaluator*, ***kwargs*)

Returns the tags that can contain the information from a SCF iteration

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
CaseInsensitiveFrozenSet[str]

Returns
set of tag names that contain elements from the group 'GeneralIterationType'

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_omittable_tags**(*xmlschema_evaluator*, ***kwargs*)

find tags with no attributes and, which are only used to mask a list of one other possible tag (e.g. atomSpecies)

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
list[str]

Returns
list of tags, containing only a sequence of one allowed tag

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_other_attribs**(*xmlschema_evaluator*, ***kwargs*)

Determine all other attributes not contained in settable or settable_contains

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
CaseInsensitiveDict[str, list[str]]

Returns
dictionary with all attributes and the corresponding list of paths to the tag

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_root_tag**(*xmlschema_evaluator*, ***kwargs*)

Returns the tag for the root element of the xmlschema

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
str

Returns
name of the single element defined in the first level of the schema

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_tag_info**(*xmlschema_evaluator*, ***kwargs*)

Get all important information about the tags

- allowed attributes
- contained tags (simple (only attributes), optional (with default values), several, order, text tags)

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
dict[str, *TagInfo*]

Returns
dictionary with the tag information

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_tag_paths**(*xmlschema_evaluator*, ***kwargs*)

Determine simple xpaths to all possible tags

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
CaseInsensitiveDict[str, list[str] | str]

Returns
possible paths of all tags in a dictionary, if multiple paths are possible a list is inserted for the tag

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_text_tags**(*xmlschema_evaluator*, ***kwargs*)

find all elements, who can contain text

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – xmltree representing the schema

Return type
CaseInsensitiveFrozenSet[str]

Returns
dictionary with tags and their corresponding type_definition meaning a dictionary with possible base types and evtl. length restriction

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_unique_attribs**(*xmlschema_evaluator*, ***kwargs*)

Determine all attributes, which can be set through set_inpchanges in aiida_fleur Meaning ONE possible path and no tags in the path with maxOccurs!=1

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
CaseInsensitiveDict[str, str]

Returns
dictionary with all settable attributes and the corresponding path to the tag

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**get_unique_path_attribs**(*xmlschema_evalu* ***kwargs*)

Determine all attributes, with multiple possible path that do have at least one path with all contained tags maxOccurs!=1

Parameters
xmlschema_evaluator (XPathDocumentEvaluator) – etree.XPathEvaluator for the schema

Return type
CaseInsensitiveDict[str, list[str]]

Returns
dictionary with all attributes and the corresponding list of paths to the tag

masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.**type_order**(*type_def*)

Key function for sorting the type definitions to avoid conflicts

Sorted by base_type first (bool before int, string at the end) and then by length in ascending order (unbounded last)

Parameters
type_def (*AttributeType*) – definition to be sorted

Return type
tuple[int, float]

6.1.2.7 Defined constants

Here we collect physical constants which are used throughout the code. That way we ensure consistency.

Note: For masci-tools versions after v0.4.6, the constants values used in the KKR functions for conversion between Angstrom and Bohr radius, and electron Volt and Rydberg, have been replaced by the NIST values by default. Prior to that, two different versions had been in use. If you need to work with KKR calculations / `aiida-kkr` workchains performed with these constants versions, you can switch to these older KKR constants versions by setting the environment variable `MASCI_TOOLS_USE_OLD_CONSTANTS` prior to loading masci-tools. During interpreter runtime, the chosen version cannot be switched.

- For KKR constants versions used starting from masci-tools v0.4.7, more specifically starting from commit 66953f8, Apr 28, 2021, do not set `MASCI_TOOLS_USE_OLD_CONSTANTS`.
- For KKR constants versions used in masci-tools v0.4.0-dev7 - v0.4.6, more specifically starting from commit c171563, Feb 16, 2021, to prior to commit 66953f8, Apr 28, 2021, set `MASCI_TOOLS_USE_OLD_CONSTANTS` to `'interim'`.
- For KKR constants versions used prior to masci-tools v0.4.0-dev7, more specifically prior to commit c171563, Feb 16, 2021, set `MASCI_TOOLS_USE_OLD_CONSTANTS` to `'old'` or `'True'`.

```
# NIST https://physics.nist.gov/cgi-bin/cuu/Value?hrev
HTR_TO_EV = 27.211386245988  #(53)
RY_TO_EV = 13.605693122994  #(26)
BOHR_A = 0.5291772108
```

(continues on next page)

(continued from previous page)

```
HTR_TO_KELVIN = 315_775.02480407
#Scipy bohr 5.29177210903e-11 m
#Scipy htr 27.211386245988 eV
# NIST BOHR 0.529177210903 #(80)
#https://physics.nist.gov/cgi-bin/cuu/Value?bohrrada0

#KKR constants versions. See module docstring for details.
_MASCI_TOOLS_USE_OLD_CONSTANTS = os.environ.get('MASCI_TOOLS_USE_OLD_CONSTANTS', None)
if _MASCI_TOOLS_USE_OLD_CONSTANTS and _MASCI_TOOLS_USE_OLD_CONSTANTS.lower() in ['old',
↪'true']:
    ANG_BOHR_KKR = 1.8897261254578281
    RY_TO_EV_KKR = 13.605693009
elif _MASCI_TOOLS_USE_OLD_CONSTANTS and _MASCI_TOOLS_USE_OLD_CONSTANTS.lower() in [
↪'interim']:
    ANG_BOHR_KKR = 1.8897261249935897
    RY_TO_EV_KKR = RY_TO_EV
else:
    #Set the constants to the NIST values
    ANG_BOHR_KKR = 1.8897261246257702
    RY_TO_EV_KKR = RY_TO_EV

#Fleur
#htr_eV   = 27.21138602
#bohr=0.5291772108
#bohrtocm=0.529177e-8
#pymatgen uses scipy.constants
#ase: Bohr 0.5291772105638411
#Hartree 27.211386024367243
#Rydberg 13.605693012183622
#1/Bohr
#1.8897261258369282
#aiida-core units:
#bohr_to_ang = 0.52917720859

#Predefined constants in the Fleur Code (These are accepted in the inp.xml)
FLEUR_DEFINED_CONSTANTS = {
    'Pi': np.pi,
    'Deg': 2 * np.pi / 360.0,
    'Ang': 1.8897261247728981,
    'nm': 18.897261247728981,
    'pm': 0.018897261247728981,
    'Bohr': 1.0,
    'Htr': 1.0,
    'eV': 1.0 / HTR_TO_EV,
    'Ry': 0.5
}

PERIODIC_TABLE_ELEMENTS = {
    0: {  # This is for empty spheres etc.
        'mass': 1.00000,
        'name': 'Unknown',
        'symbol': 'X'
```

(continues on next page)

(continued from previous page)

```
    },
    1: {
        'mass': 1.00794,
        'name': 'Hydrogen',
        'symbol': 'H',
        'econfig': '1s1',
        'fleur_default_econfig': '| 1s1',
        'lo': '',
        'rmt': 0.65,
        'lmax': '',
        'jri': 981,
        'soc': False,
        'mag': False
    },
    2: {
        'mass': 4.002602,
        'name': 'Helium',
        'symbol': 'He',
        'econfig': '1s2',
        'fleur_default_econfig': '| 1s2',
        'lo': '',
        'rmt': 1.2,
        'lmax': '',
        'jri': 981
    },
    3: {
        'mass': 6.941,
        'name': 'Lithium',
        'symbol': 'Li',
        'econfig': '1s2 | 2s1',
        'fleur_default_econfig': '1s2 | 2s1',
        'lo': '',
        'rmt': 2.13,
        'lmax': '',
        'jri': 981
    },
    4: {
        'mass': 9.012182,
        'name': 'Beryllium',
        'symbol': 'Be',
        'econfig': '1s2 | 2s2',
        'fleur_default_econfig': '1s2 | 2s2',
        'lo': '',
        'rmt': 1.87,
        'lmax': '',
        'jri': 981
    },
    5: {
        'mass': 10.811,
        'name': 'Boron',
        'symbol': 'B',
        'econfig': '1s2 | 2s2 2p1',
```

(continues on next page)

(continued from previous page)

```
        'fleur_default_econfig': '1s2 | 2s2 2p1',
        'lo': '',
        'rmt': 1.4,
        'lmax': '',
        'jri': 981
    },
    6: {
        'mass': 12.0107,
        'name': 'Carbon',
        'symbol': 'C',
        'econfig': '[He] 2s2 | 2p2',
        'fleur_default_econfig': '[He] 2s2 | 2p2',
        'lo': '',
        'rmt': 1.2,
        'lmax': '',
        'jri': 981
    },
    7: {
        'mass': 14.0067,
        'name': 'Nitrogen',
        'symbol': 'N',
        'econfig': '[He] 2s2 | 2p3',
        'fleur_default_econfig': '[He] 2s2 | 2p3',
        'lo': '',
        'rmt': 1.0,
        'lmax': '',
        'jri': 981
    },
    8: {
        'mass': 15.9994,
        'name': 'Oxygen',
        'symbol': 'O',
        'econfig': '[He] 2s2 | 2p4',
        'fleur_default_econfig': '[He] 2s2 | 2p4',
        'lo': '',
        'rmt': 1.1,
        'lmax': '',
        'jri': 981
    },
    9: {
        'mass': 18.9984032,
        'name': 'Fluorine',
        'symbol': 'F',
        'econfig': '[He] 2s2 | 2p5',
        'fleur_default_econfig': '[He] 2s2 | 2p5',
        'lo': '',
        'rmt': 1.2,
        'lmax': '',
        'jri': 981
    },
    10: {
        'mass': 20.1797,
```

(continues on next page)

(continued from previous page)

```
        'name': 'Neon',
        'symbol': 'Ne',
        'econfig': '[He] 2s2 | 2p6',
        'fleur_default_econfig': '[He] 2s2 | 2p6',
        'lo': '',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    11: {
        'mass': 22.98977,
        'name': 'Sodium',
        'symbol': 'Na',
        'econfig': '[He] 2s2 | 2p6 3s1',
        'fleur_default_econfig': '[He] | 2s2 2p6 3s1',
        'lo': '2s 2p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    12: {
        'mass': 24.305,
        'name': 'Magnesium',
        'symbol': 'Mg',
        'econfig': '[He] 2s2 | 2p6 3s2',
        'fleur_default_econfig': '[He] 2s2 | 2p6 3s2',
        'lo': '2p',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
    13: {
        'mass': 26.981538,
        'name': 'Aluminium',
        'symbol': 'Al',
        'econfig': '[He] 2s2 2p6 | 3s2 3p1',
        'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p1',
        'lo': '',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    14: {
        'mass': 28.0855,
        'name': 'Silicon',
        'symbol': 'Si',
        'econfig': '[He] 2s2 2p6 | 3s2 3p2',
        'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p2',
        'lo': '',
        'rmt': 2.0,
        'lmax': '',
        'jri': 981
```

(continues on next page)

(continued from previous page)

```
    },
    15: {
        'mass': 30.973761,
        'name': 'Phosphorus',
        'symbol': 'P',
        'econfig': '[He] 2s2 2p6 | 3s2 3p3',
        'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p3',
        'lo': '',
        'rmt': 1.9,
        'lmax': '',
        'jri': 981
    },
    16: {
        'mass': 32.065,
        'name': 'Sulfur',
        'symbol': 'S',
        'econfig': '[He] 2s2 2p6 | 3s2 3p4',
        'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p4',
        'lo': '',
        'rmt': 1.7,
        'lmax': '',
        'jri': 981
    },
    17: {
        'mass': 35.453,
        'name': 'Chlorine',
        'symbol': 'Cl',
        'econfig': '[He] 2s2 2p6 | 3s2 3p5',
        'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p5',
        'lo': '',
        'rmt': 1.7,
        'lmax': '',
        'jri': 981
    },
    18: {
        'mass': 39.948,
        'name': 'Argon',
        'symbol': 'Ar',
        'econfig': '[He] 2s2 2p6 | 3s2 3p6',
        'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p6',
        'lo': '',
        'rmt': 1.8,
        'lmax': '',
        'jri': 981
    },
    19: {
        'mass': 39.0983,
        'name': 'Potassium',
        'symbol': 'K',
        'econfig': '[Ne] 3s2 | 3p6 4s1',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s1',
        'lo': '3s 3p',
```

(continues on next page)

(continued from previous page)

```
        'rmt': 2.0,
        'lmax': '',
        'jri': 981
    },
    20: {
        'mass': 40.078,
        'name': 'Calcium',
        'symbol': 'Ca',
        'econfig': '[Ne] 3s2 | 3p6 4s2',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2',
        'lo': '3s 3p',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
    21: {
        'mass': 44.955912,
        'name': 'Scandium',
        'symbol': 'Sc',
        'econfig': '[Ne] 3s2 3p6 | 4s2 3d1',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d1',
        'lo': '3s 3p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    22: {
        'mass': 47.867,
        'name': 'Titanium',
        'symbol': 'Ti',
        'econfig': '[Ne] | 3s2 3p6 4s2 3d2',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d2',
        'lo': '3s 3p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    23: {
        'mass': 50.9415,
        'name': 'Vanadium',
        'symbol': 'V',
        'econfig': '[Ne] 3s2 3p6 | 4s2 3d3',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d3',
        'lo': '3s 3p',
        'rmt': 1.9,
        'lmax': '',
        'jri': 981
    },
    24: {
        'mass': 51.9961,
        'name': 'Chromium',
        'symbol': 'Cr',
```

(continues on next page)

(continued from previous page)

```
        'econfig': '[Ne] 3s2 3p6 | 4s1 3d5',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s1 3d5',
        'lo': '3s 3p',
        'rmt': 1.8,
        'lmax': '',
        'jri': 981
    },
    25: {
        'mass': 54.938045,
        'name': 'Manganese',
        'symbol': 'Mn',
        'econfig': '[Ne] 3s2 3p6 | 4s2 3d5',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d5',
        'lo': '3s 3p',
        'rmt': 2.0,
        'lmax': '',
        'jri': 981
    },
    26: {
        'mass': 55.845,
        'name': 'Iron',
        'symbol': 'Fe',
        'econfig': '[Ne] 3s2 3p6 | 4s2 3d6',
        'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d6',
        'lo': '3s 3p',
        'rmt': 2.00,
        'lmax': '',
        'jri': 981
    },
    27: {
        'mass': 58.933195,
        'name': 'Cobalt',
        'symbol': 'Co',
        'econfig': '[Ne] 3s2 3p6 | 4s2 3d7',
        'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s2 3d7',
        'lo': '3p',
        'rmt': 1.9,
        'lmax': '',
        'jri': 981
    },
    28: {
        'mass': 58.6934,
        'name': 'Nickel',
        'symbol': 'Ni',
        'econfig': '[Ne] 3s2 3p6 | 4s2 3d8',
        'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s2 3d8',
        'lo': '3p',
        'rmt': 1.9,
        'lmax': '',
        'jri': 981
    },
    29: {
```

(continues on next page)

(continued from previous page)

```
        'mass': 63.546,
        'name': 'Copper',
        'symbol': 'Cu',
        'econfig': '[Ne] 3s2 3p6 |4s1 3d10',
        'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s1 3d10',
        'lo': '3p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    30: {
        'mass': 65.38,
        'name': 'Zinc',
        'symbol': 'Zn',
        'econfig': '[Ne] 3s2 3p6 | 3d10 4s2',
        'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2',
        'lo': '3d',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    31: {
        'mass': 69.723,
        'name': 'Gallium',
        'symbol': 'Ga',
        'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p1',
        'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p1',
        'lo': '3d',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    32: {
        'mass': 72.64,
        'name': 'Germanium',
        'symbol': 'Ge',
        'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p2',
        'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p2',
        'lo': '3d',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    33: {
        'mass': 74.9216,
        'name': 'Arsenic',
        'symbol': 'As',
        'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p3',
        'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p3',
        'lo': '3d',
        'rmt': 2.0,
        'lmax': '',
```

(continues on next page)

(continued from previous page)

```
        'jri': 981
    },
    34: {
        'mass': 78.96,
        'name': 'Selenium',
        'symbol': 'Se',
        'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p4',
        'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p4',
        'lo': '3d',
        'rmt': 2.0,
        'lmax': '',
        'jri': 981
    },
    35: {
        'mass': 79.904,
        'name': 'Bromine',
        'symbol': 'Br',
        'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p5',
        'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p5',
        'lo': '3d',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    36: {
        'mass': 83.798,
        'name': 'Krypton',
        'symbol': 'Kr',
        'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p6',
        'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p6',
        'lo': '3d',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    37: {
        'mass': 85.4678,
        'name': 'Rubidium',
        'symbol': 'Rb',
        'econfig': '[Ar] 3d10 4s2 | 4p6 5s1',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1',
        'lo': '4s 4p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    38: {
        'mass': 87.62,
        'name': 'Strontium',
        'symbol': 'Sr',
        'econfig': '[Ar] 3d10 4s2 | 4p6 5s2',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2',
```

(continues on next page)

(continued from previous page)

```
        'lo': '4s 4p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    39: {
        'mass': 88.90585,
        'name': 'Yttrium',
        'symbol': 'Y',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d1',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d1',
        'lo': '4s 4p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    40: {
        'mass': 91.224,
        'name': 'Zirconium',
        'symbol': 'Zr',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d2',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d2',
        'lo': '4s 4p',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
    41: {
        'mass': 92.90638,
        'name': 'Niobium',
        'symbol': 'Nb',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d4',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d4',
        'lo': '4s 4p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    42: {
        'mass': 95.96,
        'name': 'Molybdenum',
        'symbol': 'Mo',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d5',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d5',
        'lo': '4s 4p',
        'rmt': 2.0,
        'lmax': '',
        'jri': 981
    },
    43: {
        'mass': 98.0,
        'name': 'Technetium',
```

(continues on next page)

(continued from previous page)

```
        'symbol': 'Tc',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d5',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d5',
        'lo': '4s 4p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    44: {
        'mass': 101.07,
        'name': 'Ruthenium',
        'symbol': 'Ru',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d7',
        'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 5s1 4d7',
        'lo': '4p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    45: {
        'mass': 102.9055,
        'name': 'Rhodium',
        'symbol': 'Rh',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d8',
        'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 5s1 4d8',
        'lo': '4p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    46: {
        'mass': 106.42,
        'name': 'Palladium',
        'symbol': 'Pd',
        'econfig': '[Ar] 4s2 3d10 4p6 | 4d10',
        'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 4d10',
        'lo': '4p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    47: {
        'mass': 107.8682,
        'name': 'Silver',
        'symbol': 'Ag',
        'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d10',
        'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d10',
        'lo': '4s 4p',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
```

(continues on next page)

(continued from previous page)

```
    48: {
        'mass': 112.411,
        'name': 'Cadmium',
        'symbol': 'Cd',
        'econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2',
        'fleur_default_econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2',
        'lo': '4d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    49: {
        'mass': 114.818,
        'name': 'Indium',
        'symbol': 'In',
        'econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2 5p1',
        'fleur_default_econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2 5p1',
        'lo': '4d',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    50: {
        'mass': 118.71,
        'name': 'Tin',
        'symbol': 'Sn',
        'econfig': '[Kr] 4d10 | 5s2 5p2',
        'fleur_default_econfig': '[Kr] | 4d10 5s2 5p2',
        'lo': '4d',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
    51: {
        'mass': 121.76,
        'name': 'Antimony',
        'symbol': 'Sb',
        'econfig': '[Kr] 4d10 | 5s2 5p3',
        'fleur_default_econfig': '[Kr] | 4d10 5s2 5p3',
        'lo': '4d',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
    52: {
        'mass': 127.6,
        'name': 'Tellurium',
        'symbol': 'Te',
        'econfig': '[Kr] 4d10 | 5s2 5p4',
        'fleur_default_econfig': '[Kr] | 4d10 5s2 5p4',
        'lo': '4d',
        'rmt': 2.3,
```

(continues on next page)

(continued from previous page)

```
        'lmax': '',
        'jri': 981
    },
    53: {
        'mass': 126.90447,
        'name': 'Iodine',
        'symbol': 'I',
        'econfig': '[Kr] 4d10 | 5s2 5p5',
        'fleur_default_econfig': '[Kr] | 4d10 5s2 5p5',
        'lo': '4d',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    54: {
        'mass': 131.293,
        'name': 'Xenon',
        'symbol': 'Xe',
        'econfig': '[Kr] 4d10 | 5s2 5p6',
        'fleur_default_econfig': '[Kr] | 4d10 5s2 5p6',
        'lo': '4d',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    55: {
        'mass': 132.9054519,
        'name': 'Caesium',
        'symbol': 'Cs',
        'econfig': '[Kr] 4d10 5s2 | 5p6 6s1',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s1',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    56: {
        'mass': 137.327,
        'name': 'Barium',
        'symbol': 'Ba',
        'econfig': '[Kr] 4d10 5s2 | 5p6 6s2',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    57: {
        'mass': 138.90547,
        'name': 'Lanthanum',
        'symbol': 'La',
        'econfig': '[Kr] 4d10 5s2 | 5p6 6s2 5d1',
```

(continues on next page)

(continued from previous page)

```
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 5d1',
        'lo': '5s 5p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    58: {
        'mass': 140.116,
        'name': 'Cerium',
        'symbol': 'Ce',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f1 5d1',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f1 5d1',
        'lo': '5s 5p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    59: {
        'mass': 140.90765,
        'name': 'Praseodymium',
        'symbol': 'Pr',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f3',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6  6s2 4f3',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    60: {
        'mass': 144.242,
        'name': 'Neodymium',
        'symbol': 'Nd',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f4',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f4',
        'lo': '5s 5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    61: {
        'mass': 145.0,
        'name': 'Promethium',
        'symbol': 'Pm',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f5',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f5',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    62: {
        'mass': 150.36,
```

(continues on next page)

(continued from previous page)

```
        'name': 'Samarium',
        'symbol': 'Sm',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f6',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f6',
        'lo': '5s 5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    63: {
        'mass': 151.964,
        'name': 'Europium',
        'symbol': 'Eu',
        'econfig': '[Kr] 4d10 | 4f7 5s2 5p6 6s2',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 4f7 6s2',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    64: {
        'mass': 157.25,
        'name': 'Gadolinium',
        'symbol': 'Gd',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f7 5d1',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f7 5d1',
        'lo': '5s 5p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    65: {
        'mass': 158.92535,
        'name': 'Terbium',
        'symbol': 'Tb',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f9',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f8 5d1',
        'lo': '5s 5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    66: {
        'mass': 162.5,
        'name': 'Dysprosium',
        'symbol': 'Dy',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f10',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f9 5d1',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
```

(continues on next page)

(continued from previous page)

```
    },
    67: {
        'mass': 164.93032,
        'name': 'Holmium',
        'symbol': 'Ho',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f11',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f10 5d1',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    68: {
        'mass': 167.259,
        'name': 'Erbium',
        'symbol': 'Er',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f12',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f11 5d1',
        'lo': '5s 5p',
        'rmt': 2.5,
        'lmax': '',
        'jri': 981
    },
    69: {
        'mass': 168.93421,
        'name': 'Thulium',
        'symbol': 'Tm',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f13',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f12 5d1',
        'lo': '5s 5p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    70: {
        'mass': 173.054,
        'name': 'Ytterbium',
        'symbol': 'Yb',
        'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f14',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f13 5d1',
        'lo': '5s 5p',
        'rmt': 2.6,
        'lmax': '',
        'jri': 981
    },
    71: {
        'mass': 174.9668,
        'name': 'Lutetium',
        'symbol': 'Lu',
        'econfig': '[Kr] 4d10 | 4f14 5s2 5p6 5d1 6s2',
        'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 4f14 6s2 5d1',
        'lo': '5s 5p',
```

(continues on next page)

(continued from previous page)

```
        'rmt': 2.5,
        'lmax': '',
        'jri': 981
    },
    72: {
        'mass': 178.49,
        'name': 'Hafnium',
        'symbol': 'Hf',
        'econfig': '[Kr] 4d10 | 4f14 5s2 5p6 5d2 6s2',
        'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d2',
        'lo': '5s 5p',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
    73: {
        'mass': 180.94788,
        'name': 'Tantalum',
        'symbol': 'Ta',
        'econfig': '[Kr] 4d10 4f14 | 5s2 5p6 5d3 6s2',
        'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d3',
        'lo': '5s 5p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    74: {
        'mass': 183.84,
        'name': 'Tungsten',
        'symbol': 'W',
        'econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d4',
        'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d4',
        'lo': '5s 5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    75: {
        'mass': 186.207,
        'name': 'Rhenium',
        'symbol': 'Re',
        'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d5',
        'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d5',
        'lo': '5s 5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    76: {
        'mass': 190.23,
        'name': 'Osmium',
        'symbol': 'Os',
```

(continues on next page)

(continued from previous page)

```
        'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d6',
        'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d6',
        'lo': '5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    77: {
        'mass': 192.217,
        'name': 'Iridium',
        'symbol': 'Ir',
        'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d7',
        'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d7',
        'lo': '5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    78: {
        'mass': 195.084,
        'name': 'Platinum',
        'symbol': 'Pt',
        'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d8',
        'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d8',
        'lo': '5p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    79: {
        'mass': 196.966569,
        'name': 'Gold',
        'symbol': 'Au',
        'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d9',
        'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d9',
        'lo': '5s 5p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    80: {
        'mass': 200.59,
        'name': 'Mercury',
        'symbol': 'Hg',
        'econfig': '[Kr] 5s2 4d10 4f14 | 5p6 5d10 6s2',
        'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 5p6 | 5d10 6s2',
        'lo': '5d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    81: {
```

(continues on next page)

(continued from previous page)

```
        'mass': 204.3833,
        'name': 'Thallium',
        'symbol': 'Tl',
        'econfig': '[Xe] 4f14 | 5d10 6s2 6p1',
        'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p1',
        'lo': '5d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    82: {
        'mass': 207.2,
        'name': 'Lead',
        'symbol': 'Pb',
        'econfig': '[Xe] 4f14 | 5d10 6s2 6p2',
        'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p2',
        'lo': '5d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    83: {
        'mass': 208.9804,
        'name': 'Bismuth',
        'symbol': 'Bi',
        'econfig': '[Xe] 4f14 | 5d10 6s2 6p3',
        'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p3',
        'lo': '5d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    84: {
        'mass': 209.0,
        'name': 'Polonium',
        'symbol': 'Po',
        'econfig': '[Xe] 4f14 | 5d10 6s2 6p4',
        'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p4',
        'lo': '5d',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    85: {
        'mass': 210.0,
        'name': 'Astatine',
        'symbol': 'At',
        'econfig': '[Xe] 4f14 | 5d10 6s2 6p5',
        'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p5',
        'lo': '5d',
        'rmt': 2.2,
        'lmax': '',
```

(continues on next page)

(continued from previous page)

```
        'jri': 981
    },
    86: {
        'mass': 222.0,
        'name': 'Radon',
        'symbol': 'Rn',
        'econfig': '[Xe] 4f14 | 5d10 6s2 6p6',
        'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p6',
        'lo': '5d',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },  # TODO: after wards not right
    87: {
        'mass': 223.0,
        'name': 'Francium',
        'symbol': 'Fr',
        'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s1',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s1',
        'lo': '6s 6p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    88: {
        'mass': 226.0,
        'name': 'Radium',
        'symbol': 'Ra',
        'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2',
        'lo': '6s 6p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    89: {
        'mass': 227.0,
        'name': 'Actinium',
        'symbol': 'Ac',
        'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1',
        'lo': '6s 6p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    90: {
        'mass': 232.03806,
        'name': 'Thorium',
        'symbol': 'Th',
        'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1 5f1',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1 5f1',
```

(continues on next page)

(continued from previous page)

```
        'lo': '6s 6p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    91: {
        'mass': 231.03588,
        'name': 'Protactinium',
        'symbol': 'Pa',
        'econfig': '[Xe] 4f14  5d10 6s2 | 6p6 7s2 6d1 5f2',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1 5f2',
        'lo': '6s 6p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    92: {
        'mass': 238.02891,
        'name': 'Uranium',
        'symbol': 'U',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6| 7s2 5f4',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f4',
        'lo': '6s 6p',
        'rmt': 2.3,
        'lmax': '',
        'jri': 981
    },
    93: {
        'mass': 237.0,
        'name': 'Neptunium',
        'symbol': 'Np',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f5',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f5',
        'lo': '6s 6p',
        'rmt': 2.1,
        'lmax': '',
        'jri': 981
    },
    94: {
        'mass': 244.0,
        'name': 'Plutonium',
        'symbol': 'Pu',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f6',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f6',
        'lo': '6s 6p',
        'rmt': 2.2,
        'lmax': '',
        'jri': 981
    },
    95: {
        'mass': 243.0,
        'name': 'Americium',
```

(continues on next page)

(continued from previous page)

```
        'symbol': 'Am',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f7',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f7',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    96: {
        'mass': 247.0,
        'name': 'Curium',
        'symbol': 'Cm',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f8',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f8',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    97: {
        'mass': 247.0,
        'name': 'Berkelium',
        'symbol': 'Bk',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f9',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f9',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    98: {
        'mass': 251.0,
        'name': 'Californium',
        'symbol': 'Cf',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f10',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f10',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    99: {
        'mass': 252.0,
        'name': 'Einsteinium',
        'symbol': 'Es',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f11',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f11',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
```

(continues on next page)

(continued from previous page)

```
    100: {
        'mass': 257.0,
        'name': 'Fermium',
        'symbol': 'Fm',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f12',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f12',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    101: {
        'mass': 258.0,
        'name': 'Mendelevium',
        'symbol': 'Md',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f13',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f13',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    102: {
        'mass': 259.0,
        'name': 'Nobelium',
        'symbol': 'No',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f14',
        'lo': '6s 6p',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    103: {
        'mass': 262.0,
        'name': 'Lawrencium',
        'symbol': 'Lr',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d1',
        'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f14 6d1',
        'lo': '6s 6p 5f',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    104: {
        'mass': 267.0,
        'name': 'Rutherfordium',
        'symbol': 'Rf',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d2',
        'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d2',
        'lo': '6p 5f',
        'rmt': 2.4,
```

(continues on next page)

(continued from previous page)

```
        'lmax': '',
        'jri': 981
    },
    105: {
        'mass': 268.0,
        'name': 'Dubnium',
        'symbol': 'Db',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d3',
        'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 5f14 6d3',
        'lo': '6p 5f',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    106: {
        'mass': 271.0,
        'name': 'Seaborgium',
        'symbol': 'Sg',
        'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d4',
        'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 5f14 6d4',
        'lo': '6p 5f',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    107: {
        'mass': 272.0,
        'name': 'Bohrium',
        'symbol': 'Bh',
        'econfig': '[Rn] 7s2 5f14 | 6d5',
        'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 6p6 5f14 | 7s2 6d5',
        'lo': '',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    108: {
        'mass': 270.0,
        'name': 'Hassium',
        'symbol': 'Hs',
        'econfig': '[Rn] 7s2 5f14 | 6d6',
        'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d6',
        'lo': '',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    109: {
        'mass': 276.0,
        'name': 'Meitnerium',
        'symbol': 'Mt',
        'econfig': '[Rn] 7s2 5f14 | 6d7',
```

(continues on next page)

(continued from previous page)

```
        'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d7',
        'lo': '',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    110: {
        'mass': 281.0,
        'name': 'Darmstadtium',
        'symbol': 'Ds',
        'econfig': '[Rn] 7s2 5f14 | 6d8',
        'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d8',
        'lo': '',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    111: {
        'mass': 280.0,
        'name': 'Roentgenium',
        'symbol': 'Rg',
        'econfig': '[Rn] 7s2 5f14 | 6d9',
        'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d9',
        'lo': '',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    112: {
        'mass': 285.0,
        'name': 'Copernicium',
        'symbol': 'Cn',
        'econfig': '[Rn] 7s2 5f14 | 6d10',
        'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d10',
        'lo': '6d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    113: {
        'mass': 285.0,
        'name': 'Nihomium',
        'symbol': 'Nh',
        'econfig': '[Rn] 7s2 5f14 | 6d10 7p1',
        'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p1',
        'lo': '6d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    114: {
        'mass': 289.0,
```

(continues on next page)

(continued from previous page)

```
        'name': 'Flerovium',
        'symbol': 'Fl',
        'econfig': '[Rn] 7s2 5f14 | 6d10 7p2',
        'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p2',
        'lo': '6d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    115: {
        'mass': 0.0,
        'name': 'Mascovium',
        'symbol': 'Mc',
        'econfig': '[Rn] 7s2 5f14 | 6d10 7p3',
        'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p3',
        'lo': '6d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    116: {
        'mass': 293.0,
        'name': 'Livermorium',
        'symbol': 'Lv',
        'econfig': '[Rn] 7s2 5f14 | 6d10 7p4',
        'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p4',
        'lo': '6d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    117: {
        'mass': 0.0,
        'name': 'Tennessine',
        'symbol': 'Ts',
        'econfig': '[Rn] 7s2 5f14 | 6d10 7p5',
        'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p5',
        'lo': '6d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
    },
    118: {
        'mass': 0.0,
        'name': 'Oganesson',
        'symbol': 'Og',
        'econfig': '[Rn] 7s2 5f14 | 6d10 7p6',
        'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p6',
        'lo': '6d',
        'rmt': 2.4,
        'lmax': '',
        'jri': 981
```

(continues on next page)

(continued from previous page)

```
    }
}

ATOMIC_NUMBERS = {data['symbol']: num for num, data in PERIODIC_TABLE_ELEMENTS.items()}
```

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

K

L

M

N

O

P

R

S

T

U

V

W

X

Z