pyAT

0.2

Generated by Doxygen 1.8.17

Chapter 1

Installation

See the `INSTALL.md` document.

1.1 Online documentation

- Check the `Web site`
- Check the `WIKI`

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

Namespace Documentation

5.1 at Namespace Reference

Namespaces

- collective
- integrators
- lattice
- load
- matching
- physics
- plot
- tracking

5.1.1 Detailed Description

```
Python port of the Accelerator Toolbox
```

5.2 at.collective Namespace Reference

Namespaces

- wake_functions
- wake_object

5.2.1 Detailed Description

```
Collective effects
```

5.3 at.collective.wake_functions Namespace Reference

Functions

- def convolve_wakefun (srange, w, sigs)
- def long_resonator_wf (srange, frequency, qfactor, rshunt, beta)
- def transverse_resonator_wf (srange, frequency, qfactor, rshunt, yokoya_factor, beta)
- def transverse_reswall_wf (srange, yokoya_factor, length, rvac, conduct, beta)

5.3.1 Detailed Description

```
Analytical wake functions
```

5.3.2 Function Documentation

5.3.2.1 convolve_wakefun()

```
def at.collective.wake_functions.convolve_wakefun (
              srange,
              w,
              sigs )
```

```
Convolution of a wake function with a pulse of rms
length sigs, this is use to generate a wake potential
that can be added to the output of EM code like GDFIDL
```

5.3.2.2 long_resonator_wf()

```
def at.collective.wake_functions.long_resonator_wf (
              srange,
              frequency,
              qfactor,
              rshunt,
              beta )
```

```
Define the wake function (longitudinal) of a resonator
with the given parameters according to Alex Chao's resonator
model (Eq. 2.82) and definitions of the resonator in HEADTAIL.
```

5.3.2.3 transverse_resonator_wf()

```
def at.collective.wake_functions.transverse_resonator_wf (
                srange,
                frequency,
                qfactor,
                rshunt,
                yokoya_factor,
                beta )
```

Define the wake function (transverse) of a resonator
with the given parameters according to Alex Chao's
resonator model (Eq. 2.82) and definitions of the resonator
in HEADTAIL.

5.3.2.4 transverse_reswall_wf()

```
def at.collective.wake_functions.transverse_reswall_wf (
                srange,
                yokoya_factor,
                length,
                rvac,
                conduct,
                beta )
```

Define the wake function (transverse) of a resistive wall with the given
parameters according to Alex Chao's RW model (2.53) and definitions used in
HEADTAIL

5.4 at.collective.wake_object Namespace Reference

Classes

- class Wake
- class WakeComponent
- class WakeType

5.4.1 Detailed Description

Wake object creation

5.5 at.integrators Namespace Reference

5.5.1 Detailed Description

Integrators for tracking in the Accelerator Toolbox

5.6 at.lattice Namespace Reference

Namespaces

- elements
- lattice_object
- options
- utils

5.6.1 Detailed Description

```
Helper functions for working with AT lattices.

A lattice as understood by pyAT is any sequence of elements.  These functions
are useful for working with these sequences.
```

5.7 at.lattice.elements Namespace Reference

Classes

- class Aperture
- class Collimator
- class Corrector
- class Dipole
- class Drift
- class Element
- class LongElement
- class M66
- class Marker
- class Monitor
- class Multipole
- class Octupole
- class Quadrupole
- class RFCavity
- class Sextupole
- class ThinMultipole
- class Wiggler

Variables

- **Bend** = Dipole
- **CLASS_MAP**

5.7.1 Detailed Description

```
Module to define common elements used in AT.

Each element has a default PassMethod attribute for which it should have the
appropriate attributes.  If a different PassMethod is set, it is the caller's
responsibility to ensure that the appropriate attributes are present.
```

5.7.2 Variable Documentation

5.7.2.1 CLASS_MAP

```
at.lattice.elements.CLASS_MAP
```

Initial value:

```
1 =  dict((k, v) for k, v in locals().items()
2                 if isinstance(v, type) and issubclass(v, Element))
```

5.8 at.lattice.lattice_object Namespace Reference

Classes

- class Lattice

Functions

- def lattice_filter (params, lattice)
- def no_filter (params, elems)
- def type_filter (params, elem_iterator)
- def params_filter (params, elem_iterator, ∗args)

Variables

- int **TWO_PI_ERROR** = 1.E-4
- **divide**
- **invalid**

5.8.1 Detailed Description

```
Lattice object

The methods implemented in this module are internal to the 'lattice' package.
This is necessary to ensure that the 'lattice' package is independent of other
AT packages.

Other Lattice methods are implemented in other AT packages and are available
as soon as the package is imported. The 'tracking' and 'physics' packages are
automatically imported.

As an example, see the at.physics.orbit module
```

5.8.2 Function Documentation

5.8.2.1 lattice_filter()

```
def at.lattice.lattice_object.lattice_filter (
              params,
              lattice )
```

Copy lattice parameters and run through all lattice elements

5.8.2.2 no_filter()

```
def at.lattice.lattice_object.no_filter (
              params,
              elems )
```

Run through all elements without any check

5.8.2.3 params_filter()

```
def at.lattice.lattice_object.params_filter (
              params,
              elem_iterator,
             * args )
```

Run through all elements, looking for energy and periodicity.
Remove the Energy attribute of non-radiating elements

energy is taken from:
    1) The params dictionary
    2) Cavity elements
    3) Any other element

periodicity is taken from:
    1) The params dictionary
    2) Sum of the bending angles of magnets

5.8.2.4 type_filter()

```
def at.lattice.lattice_object.type_filter (
              params,
              elem_iterator )
```

Run through all elements and check element validity.
Analyse elements for radiation state

5.9 at.lattice.options Namespace Reference

Classes

- class _Dst

Variables

- **DConstant** = _Dst()

5.9.1 Detailed Description

```
Global set of constants
```

5.10 at.lattice.utils Namespace Reference

Classes

- class AtError
- class AtWarning

Functions

- def check_radiation (rad)
- def set_radiation (rad)
- def make_copy (copy)
- def uint32_refpts (refpts, n_elements)
- def bool_refpts (refpts, n_elements)
- def checkattr (*args)
- def checktype (eltype)
- def checkname (pattern)
- def get_cells (ring, *args)
- def refpts_iterator (ring, refpts)
- def refpts_count (refpts, n_elements)
- def refpts_len (ring, refpts)
- def get_refpts (ring, key, quiet=True)
- def get_elements (ring, key, quiet=True)
- def get_value_refpts (ring, refpts, var, index=None)
- def set_value_refpts (ring, refpts, var, values, index=None, increment=False, copy=False)
- def get_s_pos (ring, refpts=None)
- def tilt_elem (elem, rots, relative=False)
- def shift_elem (elem, deltax=0.0, deltaz=0.0, relative=False)
- def set_tilt (ring, tilts, relative=False)
- def set_shift (ring, dxs, dzs, relative=False)

5.10.1 Detailed Description

```
Helper functions for working with AT lattices.

A lattice as understood by pyAT is any sequence of elements.  These functions
are useful for working with these sequences.

The refpts allow functions to select points in the lattice, returned values are
given at the entrance of each element specified in refpts;
refpts can be:
- an integer in the range [-len(ring), len(ring)-1] selecting the element
    according to python indexing rules. As a special case, len(ring) is
    allowed and refers to the end of the last element,
- an ordered list of such integers without duplicates,
- a numpy array of booleans of maximum length len(ring)+1, where selected
    elements are True.
```

5.10.2 Function Documentation

5.10.2.1 bool_refpts()

```
def at.lattice.utils.bool_refpts (
              refpts,
              n_elements )
```

```
Return a boolean numpy array of length n_elements + 1 where True elements
are selected. This is used for indexing a lattice using True or False
values.
```

5.10.2.2 check_radiation()

```
def at.lattice.utils.check_radiation (
              rad )
```

```
Function to be used as a decorator for optics functions

If ring is a Lattice object, raises an exception
    if ring.radiation is not rad

If ring is any other sequence, no test is performed
```

5.10.2.3 checkattr()

```
def at.lattice.utils.checkattr (
            * args )

Return a function to be used as a filter. Check the presence or the
value of an attribute. This function can be used to extract from a ring
all elements have a given attribute.

filtfunc = checkattr(attrname)
    returns the function filtfunc such that ok=filtfunc(element) is True if
    the element has a 'attrname' attribute

filtfunc = checkattr(attrname, attrvalue)
    returns the function filtfunc such that ok=filtfunc(element) is True if
    the element has a 'attrname' attribute with the value attrvalue

Examples:

cavs = filter(checkattr('Frequency'), ring)
    returns an iterator over all elements in ring that have a
    'Frequency' attribute

elts = filter(checkattr('K', 0.0), ring)
    returns an iterator over all elements in ring that have a 'K'
    attribute equal to 0.0
```

5.10.2.4 checkname()

```
def at.lattice.utils.checkname (
             pattern )

Return a function to be used as a filter. Check the name of an element.
This function can be used to extract from a ring all elements have a
given FamName attribute.

filtfunc = checkname(pattern)
    returns the function filtfunc such that ok=filtfunc(element) is True
    if the element's FamName matches pattern.

Example:

qps = filter(checkname('QF.*'), ring)
    returns an iterator over all elements
```

5.10.2.5 checktype()

```
def at.lattice.utils.checktype (
             eltype )

Return a function to be used as a filter. Check the type of an element.
This function can be used to extract from a ring all elements have a
given type.

filtfunc = checktype(class)
    returns the function filtfunc such that ok=filtfunc(element) is True
    if the element is an instance of class

Example:

qps = filter(checktype(at.Quadrupole), ring)
    returns an iterator over all quadrupoles in ring
```

5.10.2.6 get_cells()

```
def at.lattice.utils.get_cells (
              ring,
             * args )
```

```
Return a numpy array of booleans, with the same length as ring,
marking all elements satisfying a given condition.

refpts = getcells(ring, filtfunc)
    selects all elements for which the function filtfunc(element)
    returns True

refpts = getcells(ring, attrname)
    selects all elements having a 'attrname' attribute

refpts = getcells(ring, attrname, attrvalue)
    selects all elements having a 'attrname' attribute with value attrvalue

Example:

refpts = getcells(ring, 'Frequency')
    returns a numpy array of booleans where all elements having a
    'Frequency' attribute are True

refpts = getcells(ring, 'K', 0.0)
    returns a numpy array of booleans where all elements having a 'K'
    attribute equal to 0.0 are True
```

5.10.2.7 get_elements()

```
def at.lattice.utils.get_elements (
              ring,
              key,
              quiet = True )
```

```
Get the elements of a family or class (type) from the lattice.

Args:
    ring: lattice from which to retrieve the elements.
    key: can be:
         1) an element instance, will return all elements of the same type
            in the lattice, e.g. key=Drift('d1', 1.0)
         2) an element type, will return all elements of that type in the
            lattice, e.g. key=at.elements.Sextupole
         3) a string to match against elements' FamName, supports Unix
            shell-style wildcards, e.g. key='BPM_*1'
    quiet: if false print information about matched elements for FamName
           matches, defaults to True.

Returns:
    a list of elems matching key
```

5.10.2.8 get_refpts()

```
def at.lattice.utils.get_refpts (
              ring,
              key,
              quiet = True )

Get the elements refpts of a family or class (type) from the lattice.

Args:
    ring: lattice from which to retrieve the elements.
    key: can be:
         1) an element instance, will return all elements of the same type
            in the lattice, e.g. key=Drift('d1', 1.0)
         2) an element type, will return all elements of that type in the
            lattice, e.g. key=at.elements.Sextupole
         3) a string to match against elements' FamName, supports Unix
            shell-style wildcards, e.g. key='BPM_*1'
    quiet: if false print information about matched elements for FamName
           matches, defaults to True.

Returns:
    elems: a list of elems refpts matching key
```

5.10.2.9 get_s_pos()

```
def at.lattice.utils.get_s_pos (
              ring,
              refpts = None )

Return a numpy array corresponding to the s position of the specified
elements.

Args:
    ring: lattice from which to retrieve s position
    refpts: elements at which to return s position. If None, return
            s position at all elements in the ring.
```

5.10.2.10 get_value_refpts()

```
def at.lattice.utils.get_value_refpts (
              ring,
              refpts,
              var,
              index = None )

Get the values of an attribute of an array of elements based on
their refpts

PARAMETERS:
    ring            Lattice description
    refpts          Integer, array of integer or booleans, filter
    var             attribute name

KEYWORDS:
    index=None      index of the value to retrieve if var is an array.
                    If None the full array is retrieved (Default)
```

5.10.2.11 make_copy()

```
def at.lattice.utils.make_copy (
              copy )
```

```
Function to be used as a decorator for optics functions
The decorated function must be defined as:

def func(ring, refpts, *args, **kwargs):
    ...
    return

If copy is False, the function is not modified,
If copy is True, a shallow copy of ring is done, then the elements selected
by refpts are deep-copied, then func is applied to the copy, and the new
ring is returned.
```

5.10.2.12 refpts_count()

```
def at.lattice.utils.refpts_count (
              refpts,
              n_elements )
```

```
Number of reference points
```

5.10.2.13 refpts_iterator()

```
def at.lattice.utils.refpts_iterator (
              ring,
              refpts )
```

```
Return an iterator over all elements in ring identified by refpts.

refpts may be:

1) a integer or a sequence of integers (0 indicating the first element)
2) a sequence of booleans marking the selected elements
3) a callable f such that f(elem) is True for selected elements
```

5.10.2.14 refpts_len()

```
def at.lattice.utils.refpts_len (
              ring,
              refpts )
```

```
Number of reference points
```

5.10.2.15 set_radiation()

```
def at.lattice.utils.set_radiation (
              rad )
```

```
Function to be used as a decorator for optics functions
The decorated function must be defined as:

def func(ring, *args, **kwargs):
    ...
    return

func will be called with a copy of the ring such that its radiation state
is set to rad (no copy is done if it's already the case).
```

5.10.2.16 set_shift()

```
def at.lattice.utils.set_shift (
              ring,
              dxs,
              dzs,
              relative = False )
```

```
Set translations to a list of elements.

ring            sequence of elements to be shifted
dxs             sequence of horizontal displacement as long as ring or
                scalar horizontal displacement applied to all elements
dzs             sequence of vertical displacement as long as ring or
                scalar vertical displacement applied to all elements
relative=False  Displacement relative to the previous alignment
```

5.10.2.17 set_tilt()

```
def at.lattice.utils.set_tilt (
              ring,
              tilts,
              relative = False )
```

```
Set tilts to a list of elements.

ring            sequence of elements to be tilted
tilts           sequence of tilt values as long as ring or
                scalar tilt value applied to all elements
relative=False  Rotation relative to the previous tilt angle
```

5.10.2.18 set_value_refpts()

```
def at.lattice.utils.set_value_refpts (
              ring,
              refpts,
              var,
              values,
              index = None,
              increment = False,
              copy = False )
```

```
Set the values of an attribute of an array of elements based on
their refpts

PARAMETERS:
    ring            Lattice description
    refpts          Integer, array of integer or booleans, filter
    var             attribute name
    values          desired value for the attribute

KEYWORDS:
    index=None      index of the value to change if var is an array.
                    If None the full array is replaced by value (Default)
    increment=False Add values to the initial values.
                    If False the initial value is replaced (Default)
    copy=False      If False, do the modification in-place.
                    If True, returns a shallow copy of ring with new
                    modified elements.
                    CAUTION: a shallow copy means that all non-affected
                    elements are shared with the original lattice.
                    Any further modification will affect in both lattices.
```

5.10.2.19 shift_elem()

```
def at.lattice.utils.shift_elem (
              elem,
              deltax = 0.0,
              deltaz = 0.0,
              relative = False )
```

```
set a new displacement vector to an element.
The ranslation vectors are stored in the T1 and T2 attributes

elem            element to be displaced
deltax          horizontal displacement of the element
deltaz          vertical displacement of the element
relative=False  Displacement relative to the previous alignment
```

5.10.2.20 tilt_elem()

```
def at.lattice.utils.tilt_elem (
              elem,
              rots,
              relative = False )
```

```
set a new tilt angle to an element.
The rotation matrices are stored in the R1 and R2 attributes

R1 = [[ cos(rots) sin(rots)]    R2 = [[cos(rots) -sin(rots)]
      [-sin(rots) cos(rots)]]         [sin(rots)  cos(rots)]]

elem            element to be tilted
rots            tilt angle (in radians).
                rots > 0 corresponds to a corkskew rotation of the element
                looking in the direction of the beam
relative=False  Rotation relative to the previous element rotation
```

5.10.2.21 uint32_refpts()

```
def at.lattice.utils.uint32_refpts (
              refpts,
              n_elements )
```

```
Return a uint32 numpy array with contents as the indices of the selected
elements.  This is used for indexing a lattice using explicit indices.
```

5.11 at.load Namespace Reference

Namespaces

- allfiles
- elegant
- matfile
- reprfile
- tracy
- utils

5.11.1 Detailed Description

```
Import/export AT lattice from/to different formats:
- .mat files
- .m files
```

5.12 at.load.allfiles Namespace Reference

Functions

- def load_lattice (filepath, ∗∗kwargs)
- def save_lattice (ring, filepath, ∗∗kwargs)
- def register_format (extension, load_func=None, save_func=None, descr='')

5.12.1 Detailed Description

```
Generic function to save and load python AT lattices. The format is
determined by the file extension
```

5.12.2 Function Documentation

5.12.2.1 load_lattice()

```
def at.load.allfiles.load_lattice (
              filepath,
             ** kwargs )
```

```
Load a Lattice object from a file

The file format is indicated by the filepath extension.

PARAMETERS
    filepath        name of the file

KEYWORDS
    name            Name of the lattice
                    (default: taken from the file, or '')
    energy          Energy of the lattice
                    (default: taken from the file)
    periodicity     Number of periods
                    (default: taken from the file, or 1)
    *               all other keywords will be set as Lattice attributes

MAT-FILE SPECIFIC KEYWORDS
    mat_key         name of the Matlab variable containing the lattice.
                    Default: Matlab variable name if there is only one,
                    otherwise 'RING'
    check=True      if False, skip the coherence tests
    quiet=False     If True, suppress the warning for non-standard classes
    keep_all=False  if True, keep RingParam elements as Markers

Known extensions are:
```

5.12.2.2 register_format()

```
def at.load.allfiles.register_format (
              extension,
              load_func = None,
              save_func = None,
              descr = '' )

Register format-specific processing functions

PARAMETERS
    extension       File extension string
    load_func       load function (default: None)
    save_func       save_lattice function (default: None)
    descr           File type description
```

5.12.2.3 save_lattice()

```
def at.load.allfiles.save_lattice (
              ring,
              filepath,
             ** kwargs )

Save a Lattice object

The file format is indicated by the filepath extension.

PARAMETERS
    ring            Lattice object
    filepath        name of the file

MAT-FILE SPECIFIC KEYWORDS
    mat_key='RING'  Name of the Matlab variable

Known extensions are:
```

5.13 at.load.elegant Namespace Reference

Functions

- def **create_drift** (name, params, energy, harmonic_number)
- def **create_marker** (name, params, energy, harmonic_number)
- def **create_aperture** (name, params, energy, harmonic_number)
- def **create_quad** (name, params, energy, harmonic_number)
- def **create_sext** (name, params, energy, harmonic_number)
- def **create_oct** (name, params, energy, harmonic_number)
- def **create_multipole** (name, params, energy, harmonic_number)
- def **create_dipole** (name, params, energy, harmonic_number)
- def **create_corrector** (name, params, energy, harmonic_number)
- def **create_cavity** (name, params, energy, harmonic_number)
- def parse_lines (contents)
- def parse_chunk (value, elements, chunks)
- def **expand_elegant** (contents, lattice_key, energy, harmonic_number)
- def **handle_value** (value)
- def elegant_element_from_string (name, element_string, variables)
- def **load_elegant** (filename, **kwargs)

Variables

- dictionary **ELEMENT_MAP**

5.13.1 Detailed Description

```
Load a lattice from an Elegant file (.lte).

This is not complete but can parse the example files that I have.
This parser is quite similar to the Tracy parser in tracy.py.

The Elegant file format is described briefly here:
https://ops.aps.anl.gov/manuals/elegant_latest/elegantse9.html#x113-1120009

It is similar to the MAD-X format, described briefly here:
http://madx.web.cern.ch/madx/

Note that Elegant scales magnet polynomials in a different way
to AT, so the parsed coefficients need to be divided by n! for
the coefficient of order n.
```

5.13.2 Function Documentation

5.13.2.1 elegant_element_from_string()

```
def at.load.elegant.elegant_element_from_string (
              name,
              element_string,
              variables )

Create element from Elegant's string representation.

e.g. drift,l=0.045 => Drift(name, 0.045)
```

5.13.2.2 parse_chunk()

```
def at.load.elegant.parse_chunk (
              value,
              elements,
              chunks )

Parse a non-element part of a lattice file.

That part can reference already-parsed elements and chunks.

if
chunks = {"x": ["a", "b"]}
and
elements = {"a": Marker(...), "b": Quadrupole(...)}

line(a,b) => [a, b]
-x        => [b, a]
2*x       => [a, b, a, b]
```

5.13.2.3 parse_lines()

```
def at.load.elegant.parse_lines (
              contents )
```

```
Return individual lines.

Remove comments and whitespace and convert to lowercase.
```

5.14 at.load.matfile Namespace Reference

Functions

- def matfile_generator (params, mat_file)
- def ringparam_filter (params, elem_iterator, ∗args)
- def load_mat (filename, ∗∗kwargs)
- def mfile_generator (params, m_file)
- def load_m (filename, ∗∗kwargs)
- def load_var (matlat, ∗∗kwargs)
- def matlab_ring (ring)
- def save_mat (ring, filename, mat_key='RING')
- def save_m (ring, filename=None)

5.14.1 Detailed Description

```
Load lattices from Matlab files.
```

5.14.2 Function Documentation

5.14.2.1 load_m()

```
def at.load.matfile.load_m (
              filename,
             ** kwargs )
```

```
Create a lattice object from a matlab m-file

PARAMETERS
    filename        name of a '.m' file

KEYWORDS
    keep_all=False  if True, keep RingParam elements as Markers
    name            Name of the lattice
                    (default: taken from the elements, or '')
    energy          Energy of the lattice
                    (default: taken from the elements)
    periodicity     Number of periods
                    (default: taken from the elements, or 1)
    *               all other keywords will be set as Lattice attributes

OUTPUT
    Lattice object
```

5.14.2.2 load_mat()

```
def at.load.matfile.load_mat (
              filename,
             ** kwargs )
```

```
Create a lattice object from a Matlab mat-file

PARAMETERS
    filename        name of a '.mat' file

KEYWORDS
    mat_key         name of the Matlab variable containing the lattice.
                    Default: Matlab variable name if there is only one,
                    otherwise 'RING'
    check=True      if False, skip the coherence tests
    quiet=False     If True, suppress the warning for non-standard classes
    keep_all=False  if True, keep RingParam elements as Markers
    name            Name of the lattice
                    (default: taken from the lattice, or '')
    energy          Energy of the lattice
                    (default: taken from the elements)
    periodicity     Number of periods
                    (default: taken from the elements, or 1)
    *               all other keywords will be set as Lattice attributes

OUTPUT
    Lattice object
```

5.14.2.3 load_var()

```
def at.load.matfile.load_var (
              matlat,
             ** kwargs )
```

```
Create a lattice from a Matlab cell array
```

5.14.2.4 matfile_generator()

```
def at.load.matfile.matfile_generator (
              params,
              mat_file )
```

```
run through matlab cells and generate AT elements

KEYWORDS
    mat_file        name of the .mat file
    mat_key         name of the Matlab variable containing the lattice.
                    Default: Matlab variable name if there is only one,
                    otherwise 'RING'
    check=True      if False, skip the coherence tests
    quiet=False     If True, suppress the warning for non-standard classes
```

5.14.2.5 matlab_ring()

```
def at.load.matfile.matlab_ring (
              ring )
```

Prepend a RingParam element to a lattice

5.14.2.6 mfile_generator()

```
def at.load.matfile.mfile_generator (
              params,
              m_file )
```

Run through all lines of a m-file and generates AT elements

5.14.2.7 ringparam_filter()

```
def at.load.matfile.ringparam_filter (
              params,
              elem_iterator,
             * args )
```

"
Run through all elements, process and optionally removes RingParam elements

KEYWORDS
    keep_all=False if True, keep RingParam elem_iterator as Markers

5.14.2.8 save_m()

```
def at.load.matfile.save_m (
              ring,
              filename = None )
```

Save a lattice as a Matlab m-file

PARAMETERS
    ring   Lattice object
    filename=None   name of the '.m' file. Default: output on sys.stdout

5.14.2.9 save_mat()

```
def at.load.matfile.save_mat (
              ring,
              filename,
              mat_key = 'RING' )
```

```
Save a Lattice object as a Matlab mat-file

PARAMETERS
    ring            Lattice object
    filename        name of the '.mat' file

KEYWORDS
    mat_key='RING'  Name of the Matlab variable representing the lattice
```

5.15 at.load.reprfile Namespace Reference

Functions

- def load_repr (filename, ∗∗kwargs)
- def save_repr (ring, filename=None)

5.15.1 Detailed Description

```
Text representation of a python AT lattice with each element represented by
its 'repr' string
```

5.15.2 Function Documentation

5.15.2.1 load_repr()

```
def at.load.reprfile.load_repr (
              filename,
             ** kwargs )
```

```
Load a Lattice object stored as a repr-file

PARAMETERS
    filename        name of the '.repr' file

KEYWORDS
    name            Name of the lattice (default: taken from the file)
    energy          Energy of the lattice (default: taken from the file)
    periodicity     Number of periods (default: taken from the file)
    *               all other keywords will be set as Lattice attributes

OUTPUT
    Lattice object
```

5.15.2.2 save_repr()

```
def at.load.reprfile.save_repr (
              ring,
              filename = None )
```

```
Save a Lattice object as a repr-file

PARAMETERS
    ring            Lattice object
    filename=None   name of the '.repr' file. Default: output on sys.stdout
```

5.16 at.load.tracy Namespace Reference

Functions

- def **create_drift** (name, params, variables)
- def **create_marker** (name, params, variables)
- def **create_quad** (name, params, variables)
- def **create_sext** (name, params, variables)
- def **create_dipole** (name, params, variables)
- def **create_corrector** (name, params, variables)
- def **create_multipole** (name, params, variables)
- def **create_cavity** (name, params, variables)
- def **tokenise_expression** (expression)
- def parse_float (expression, variables)
- def parse_lines (contents)
- def **parse_chunk** (value, elements, chunks)
- def **expand_tracy** (contents, lattice_key, harmonic_number)
- def parse_hom (hom_string, variables)
- def **tracy_element_from_string** (name, element_string, variables)
- def **load_tracy** (filename, ∗∗kwargs)

Variables

- dictionary **ELEMENT_MAP**

5.16.1 Detailed Description

```
Load a lattice from a Tracy file (.lat).

This is not complete but can parse the example files that I have.
This parser is quite similar to the Elegant parser in elegant.py.
```

5.16.2 Function Documentation

5.16.2.1 parse_float()

```
def at.load.tracy.parse_float (
              expression,
              variables )
```

Evaluate the provided arithmetic expression substituting variables.

5.16.2.2 parse_hom()

```
def at.load.tracy.parse_hom (
              hom_string,
              variables )
```

Parse 'hom' string from lattice file. Note that PolynomB
is before PolynomA.

hom(3, 1.2, 3.4) => [0, 0, 3.4], [0, 0, 1.2]

5.16.2.3 parse_lines()

```
def at.load.tracy.parse_lines (
              contents )
```

Return individual lines.

Remove comments and whitespace, convert to lowercase, and split on
semicolons.

5.16.3 Variable Documentation

5.16.3.1 ELEMENT_MAP

dictionary at.load.tracy.ELEMENT_MAP

Initial value:

```
1 =  {
2     "drift": create_drift,
3     "bending": create_dipole,
4     "quadrupole": create_quad,
5     "sextupole": create_sext,
6     "multipole": create_multipole,
7     "corrector": create_corrector,
8     "marker": create_marker,
9     "map": create_marker,
10     "beampositionmonitor": create_marker,
11     "cavity": create_cavity,
12 }
```

5.17 at.load.utils Namespace Reference

Classes

- class RingParam

Functions

- def hasattrs (kwargs, *attributes)
- def find_class (elem_dict, quiet=False)
- def element_from_dict (elem_dict, index=None, check=True, quiet=False)
- def element_from_string (elem_string)
- def element_from_m (line)
- def element_to_dict (elem)
- def element_to_m (elem)
- def **find_class_name** (elem_dict, quiet=False)
- def **split_ignoring_parentheses** (string, delimiter)

Variables

- **CLASS_MAPPING** = dict((key, cls.__name__) for (key, cls) in _CLASS_MAP.items())
- **PASS_MAPPING** = dict((key, cls.__name__) for (key, cls) in _PASS_MAP.items())

5.17.1 Detailed Description

```
Conversion utilities for creating pyat elements
```

5.17.2 Function Documentation

5.17.2.1 element_from_dict()

```
def at.load.utils.element_from_dict (
              elem_dict,
              index = None,
              check = True,
              quiet = False )
```

```
return an AT element from a dictionary of attributes
```

5.17.2.2 element_from_m()

```
def at.load.utils.element_from_m (
              line )
```

Generate a AT-element from a line in a m-file

5.17.2.3 element_from_string()

```
def at.load.utils.element_from_string (
              elem_string )
```

Generate an AT-element from its repr string

5.17.2.4 element_to_dict()

```
def at.load.utils.element_to_dict (
              elem )
```

Generate the Matlab structure for a AT element

5.17.2.5 element_to_m()

```
def at.load.utils.element_to_m (
              elem )
```

Generate the Matlab-evaluable string for a AT element

5.17.2.6 find_class()

```
def at.load.utils.find_class (
              elem_dict,
              quiet = False )
```

Attempts to correctly identify the Class of the element from its kwargs.

```
Args:
    elem_dict       The dictionary of keyword arguments passed to the
                    Element constructor.
Keywords:
    quiet=False     If True, suppress the warning for non-standard classes

Returns:
    element_class:  The guessed Class name
```

5.17.2.7 hasattrs()

```
def at.load.utils.hasattrs (
              kwargs,
             * attributes )
```

```
Check if the element would have the specified attribute(s), i.e. if they
are in kwargs; allows checking for multiple attributes in one go.

Args:
    kwargs (dict): The dictionary of keyword arguments passed to the
                   Element constructor.
    attributes (iterable): A list of strings, the attribute names to be
                           checked.

Returns:
    bool: A single boolean, True if the element has any of the specified
          attributes.
```

5.18 at.matching Namespace Reference

Namespaces

- globalfit

5.18.1 Detailed Description

```
matching functions
```

5.19 at.matching.globalfit Namespace Reference

Functions

- def fit_tune (ring, refpts1, refpts2, newval, tol=1.0e-12, dp=0, niter=3)
- def fit_chrom (ring, refpts1, refpts2, newval, tol=1.0e-12, dp=0, niter=3)

5.19.1 Detailed Description

```
Collection of functions to fit various global parameters
such as the tune and the chromaticity
```

5.19.2 Function Documentation

5.19.2.1 fit_chrom()

```
def at.matching.globalfit.fit_chrom (
              ring,
              refpts1,
              refpts2,
              newval,
              tol = 1.0e-12,
              dp = 0,
              niter = 3 )
```

```
Function to fit the chromaticity of the ring, using 2 families
defined by refpts

Args:
    ring: lattice for which the chromaticity needs to be matched
    ring: lattice for which the chromaticity needs to be matched
    refpts1/2: refpts for the 2 families
    newval: new tunes
    tol: tolerance for the matching [default=1.0e-12]
    dp: dp/p at which the values need to be matched [default=0]
    niter: maximum number of iterations to reach tol [default=3]

Typical usage:
at.matching.fit_chrom(ring, refpts1, refpts2, [10,5])
```

5.19.2.2 fit_tune()

```
def at.matching.globalfit.fit_tune (
              ring,
              refpts1,
              refpts2,
              newval,
              tol = 1.0e-12,
              dp = 0,
              niter = 3 )
```

```
Function to fit the tune of the ring, using 2 families defined by
refpts

Args:
    ring: lattice for which the tune needs to be matched
    refpts1/2: refpts for the 2 families
    newval: new tunes
    tol: tolerance for the matching [default=1.0e-12]
    dp: dp/p at which the values need to be matched [default=0]
    niter: maximum number of iterations to reach tol [default=3]

Typical usage:
at.matching.fit_tune(ring, refpts1, refpts2, [0.1,0.25])
```

5.20 at.physics Namespace Reference

Namespaces

- amat
- fastring
- harmonic_analysis
- linear
- matrix
- nonlinear
- orbit
- radiation

5.20.1 Detailed Description

```
Accelerator physics functions
```

5.21 at.physics.amat Namespace Reference

Functions

- def jmat (ind)
- def jmatswap (ind)
- def a_matrix (tt)
- def amat (tt)
- def symplectify (M)
- def get_mode_matrices (a)
- def get_tunes_damp (tt, rr=None)

5.21.1 Detailed Description

5.21.2 Function Documentation

5.21.2.1 a_matrix()

```
def at.physics.amat.a_matrix (
              tt )
```

```
A, eigval = a_matrix(T)
Find the A matrix from one turn map matrix T such that:

           [Rotx  0    0  ]
inv(A)*T*A=[ 0   Rotz  0  ]
           [ 0    0   Rots]

Order it so that it is close to the order of x,y,z
also ensure that positive modes are before negative so that
one has proper symplecticity
B. Nash July 18, 2013

INPUT
T       (m, m)  transfer matrix for 1 turn

OUTPUT
A       (m, m)  A-matrix
eigval  (m,)    vector of Eigen values of T
```

5.21.2.2 amat()

```
def at.physics.amat.amat (
              tt )
```

```
A = amat(T)
Find the A matrix from one turn map matrix T
Provided for backward compatibility, see " A, eigval = a_matrix(T)"

INPUT
T       (m, m)  transfer matrix for 1 turn

OUTPUT
A       (m, m)  A-matrix
```

5.21.2.3 get_mode_matrices()

```
def at.physics.amat.get_mode_matrices (
              a )
```

```
Given a (m, m) A matrix , find the R-matrices of the m/2 normal modes
```

5.21.2.4 get_tunes_damp()

```
def at.physics.amat.get_tunes_damp (
              tt,
              rr = None )
```

```
mode_emit, damping_rates, tunes = get_tunes_damp(T, R)

INPUT
    T                   (m, m) transfer matrix for 1 turn
    R                   (m, m) beam matrix (optional)

    m can be 2 (single plane), 4 (betatron motion) or 6 (full motion)

OUTPUT
    record array with the following fields:
    tunes               (m/2,) tunes of the m/2 normal modes
    damping_rates       (m/2,) damping rates of the m/2 normal modes
    mode_matrices       (m/2, m, m) the R-matrices of the m/2 normal modes
    mode_emittances     Only if R is specified: (m/2,) emittance of each
                        of the m/2 normal modes
```

5.21.2.5 jmat()

```
def at.physics.amat.jmat (
              ind )
```

```
Return the antisymetric block diagonal matrix [[0, 1][-1, 0]]

INPUT
    ind     1, 2 or 3. Matrix dimension

OUTPUT
    jm      block diagonal matrix, (2, 2) or (4, 4) or (6, 6)
```

5.21.2.6 jmatswap()

```
def at.physics.amat.jmatswap (
              ind )
```

```
Modified version of jmat to deal with the swap of the
longitudinal coordinates
```

5.21.2.7 symplectify()

```
def at.physics.amat.symplectify (
              M )
```

```
symplectify makes a matrix more symplectic
follow Healy algorithm as described by McKay
BNL-75461-2006-CP
```

5.22 at.physics.fastring Namespace Reference

Functions

- def fast_ring (ring, split_inds=[])

5.22.1 Detailed Description

```
Functions relating to fast_ring
```

5.22.2 Function Documentation

5.22.2.1 fast_ring()

```
def at.physics.fastring.fast_ring (
              ring,
              split_inds = [] )


Computes a fast ring consisting of:
   -1 RF cavity per distinct frequency
   -6x6 transfer map
   -detuning and chromaticity element
   -quantum diffusion element (for radiation ring)

2 rings are returned one with radiation one without
The original ring is copied such that it is not modified
It is possible to split the original ring in multiple fastrings
using split_inds argument
fr,frrad = at.fast_ring(ring)
fr,frrad = at.fast_ring(ring, split_inds=[100,200])

PARAMETERS
    ring            lattice description

KEYWORDS
    split_inds=[]   List of indexes where to split the ring
```

5.23 at.physics.harmonic_analysis Namespace Reference

Classes

- class HarmonicAnalysis

Functions

- def get_spectrum_harmonic (cent, num_harmonics=20, method='laskar', hann=False)
- def get_tunes_harmonic (cents, method, num_harmonics=20, hann=False, fmin=0, fmax=1)

Variables

- int **PI2I** = 2 ∗ np.pi ∗ complex(0, 1)
- bool **ZERO_PAD_DEF** = False
- bool **HANN_DEF** = False

5.23.1 Detailed Description

```
Original author of HarmonicAnalysis class
Jaime Maria Coello De Portugal - Martinez Vazquez


Written while at CERN circa 2016
This is a reduced version keeping only the key components.
Full code can be found at: https://github.com/pylhc/harpy
```

5.23.2 Function Documentation

5.23.2.1 get_spectrum_harmonic()

```
def at.physics.harmonic_analysis.get_spectrum_harmonic (
              cent,
              num_harmonics = 20,
              method = 'laskar',
              hann = False )

INPUT
cent: centroid motions of the particle
num_harmonics: number of harmonic components to compute (before mask
applied, default=20)
method: laskar or fft [default=laskar]
hann: flag to turn on hanning window [default-> False]

OUTPUT
freq,amp: numpy arrays for the frequency and amplitude
```

5.23.2.2 get_tunes_harmonic()

```
def at.physics.harmonic_analysis.get_tunes_harmonic (
              cents,
              method,
              num_harmonics = 20,
              hann = False,
              fmin = 0,
              fmax = 1 )

INPUT
cents: are the centroid motions of the particles
method: laskar or fft
num_harmonics: number of harmonic components to compute (before mask
applied, default=20)
fmin/fmax: determine the boundaries within which the tune is located
[default 0->1]
hann: flag to turn on hanning window [default-> False]

OUTPUT
tunes: numpy array of length len(cents), max of the spectrum within
fmin:fmax
```

5.24 at.physics.linear Namespace Reference

Functions

- def linopt2 (ring, *args, **kwargs)
- def linopt4 (ring, *args, **kwargs)
- def linopt6 (ring, *args, **kwargs)
- def linopt_auto (ring, *args, **kwargs)
- def get_optics (ring, refpts=None, dp=None, method=linopt6, **kwargs)
- def linopt (ring, dp=0.0, refpts=None, get_chrom=False, **kwargs)
- def avlinopt (ring, dp=0.0, refpts=None, **kwargs)
- def get_tune (ring, method='linopt', dp=None, dct=None, orbit=None, **kwargs)
- def get_chrom (ring, method='linopt', dp=0, dct=None, cavpts=None, **kwargs)

5.24.1 Detailed Description

```
Coupled or non-coupled 4x4 linear motion
```

5.24.2 Function Documentation

5.24.2.1 avlinopt()

```
def at.physics.linear.avlinopt (
              ring,
              dp = 0.0,
              refpts = None,
             ** kwargs )
```

```
Perform linear analysis of a lattice and returns average
beta, dispersion and phase advance

lindata,avebeta,avemu,avedisp,tune,chrom = avlinopt(lattice, dp, refpts)

PARAMETERS
    lattice         lattice description.
    dp=0.0          momentum deviation.
    refpts=None     elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.

KEYWORDS
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    XYStep=1.0e-8   transverse step for numerical computation
    DPStep=1.0E-8   momentum deviation used for computation of
                    chromaticities and dispersion
```

```
OUTPUT
    lindata         linear optics at the points refered to by refpts, if
                    refpts is None an empty lindata structure is returned.
                    See linopt4 for details
    avebeta         Average beta functions [betax,betay] at refpts
    avemu           Average phase advances [mux,muy] at refpts
    avedisp         Average dispersion [Dx,Dx',Dy,Dy'] at refpts
    avespos         Average s position at refpts
    tune            [tune_A, tune_B], linear tunes for the two normal modes
                    of linear motion [1]
    chrom           [ksi_A , ksi_B], chromaticities ksi = d(nu)/(dP/P).

See also linopt4, get_optics
```

5.24.2.2 get_chrom()

```
def at.physics.linear.get_chrom (
              ring,
              method = 'linopt',
              dp = 0,
              dct = None,
              cavpts = None,
             ** kwargs )
```

```
gets the chromaticity using several available methods

PARAMETERS
    ring            lattice description.

KEYWORDS
    dp=None         Ignored if radiation is ON. Momentum deviation.
    dct=None        Ignored if radiation is ON. Path lengthening.
                    If specified, dp is ignored and the off-momentum is
                    deduced from the path lengthening.
    orbit           avoids looking for the closed orbit if already known
                    ((6,) array)
    method='linopt' 'linopt' returns the tunes from the linopt function
                    'laskar' tracks a single particle and computes the
                    tunes with NAFF
    DPStep=1.0E-6   momentum step used for the computation of
                    chromaticities

  for the 'laskar' method only:

    nturns=512      number of turns
    amplitude=1.0E-6 amplitude of oscillation
    remove_dc=False Remove the mean of oscillation data
    num_harmonics   number of harmonic components to compute
                    (before mask applied, default=20)
    fmin/fmax       determine the boundaries within which the tune is
                    located [default 0->1]
    hann=False      flag to turn on Hanning window

OUTPUT
    chromaticities = np.array([Q'x,Q'y])
```

5.24.2.3 get_optics()

```
def at.physics.linear.get_optics (
              ring,
              refpts = None,
              dp = None,
              method = linopt6,
             ** kwargs )
```

```
Perform linear analysis of a fully coupled lattice

elemdata0, beamdata, elemdata = get_optics(lattice, refpts, **kwargs)

PARAMETERS
    lattice         lattice description.
    refpts=None     elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
KEYWORDS
    method=linopt6  Method used for the analysis of the transfer matrix.
                    Can be None at.linopt2, at.linopt4, at.linopt6
                    linopt2:    no longitudinal motion, no H/V coupling,
                    linopt4:    no longitudinal motion, Sagan/Rubin
                                4D-analysis of coupled motion,
                    linopt6:    with or without longitudinal motion, normal
                                mode analysis
    dp=None         Ignored if radiation is ON. Momentum deviation.
    dct=None        Ignored if radiation is ON. Path lengthening.
                    If specified, dp is ignored and the off-momentum is
                    deduced from the path lengthening.
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    get_chrom=False compute chromaticities. Needs computing the tune at
                    2 different momentum deviations around the central one.
    get_w=False     computes chromatic amplitude functions (W) [4].
                    Needs to compute the optics at 2 different momentum
                    deviations around the central one.
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    twiss_in=None   Initial conditions for transfer line optics. Record
                    array as output by linopt, or dictionary. Keys:
                    'R' or 'alpha' and 'beta'   (mandatory)
                    'closed_orbit',             (default 0)
                    'dispersion'                (default 0)
                    If present, the attribute 'R' will be used, otherwise
                    the attributes 'alpha' and 'beta' will be used. All
                    other attributes are ignored.
OUTPUT
    elemdata0       linear optics data at the entrance/end of the ring
    beamdata        lattice properties
    elemdata        linear optics at the points refered to by refpts, if
                    refpts is None an empty elemdata structure is returned.

    elemdata is a record array with fields depending on the
    selected method.
    See the help for linopt6, linopt4, linopt2, linopt_auto.

    beamdata is a record with fields:
    tune            Fractional tunes
    chromaticity    Chromaticities
    damping_time    Damping times [s] (only if radiation is ON)
```

5.24.2.4 get_tune()

```
def at.physics.linear.get_tune (
              ring,
              method = 'linopt',
              dp = None,
              dct = None,
              orbit = None,
             ** kwargs )
```

```
gets the tune using several available methods

PARAMETERS
    ring            lattice description.

KEYWORDS
    dp=None         Ignored if radiation is ON. Momentum deviation.
    dct=None        Ignored if radiation is ON. Path lengthening.
                    If specified, dp is ignored and the off-momentum is
                    deduced from the path lengthening.
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    method='linopt' 'linopt' returns the tunes from the linopt function
                    'fft' tracks a single particle and computes the
                    tunes with fft 'laskar' tracks a single particle
                    and computes the tunes with NAFF

  for the 'fft' and 'laskar' methods only:

    nturns=512      number of turns
    amplitude=1.0E-6 amplitude of oscillation
    remove_dc=False Remove the mean of oscillation data
    num_harmonics   number of harmonic components to compute
                    (before mask applied, default=20)
    fmin/fmax       determine the boundaries within which the tune is
                    located [default 0->1]
    hann=False      flag to turn on Hanning window

OUTPUT
    tunes = np.array([Qx,Qy])
```

5.24.2.5 linopt()

```
def at.physics.linear.linopt (
              ring,
              dp = 0.0,
              refpts = None,
              get_chrom = False,
             ** kwargs )
```

```
Perform linear analysis of a H/V coupled lattice following Sagan/Rubin
4D-analysis of coupled motion

lindata0, tune, chrom, lindata = linopt(lattice, dp, refpts, **kwargs)

PARAMETERS
    lattice         lattice description.
    dp=0.0          momentum deviation.
    refpts=None     elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
```

```
                        selecting the element according to python indexing
                        rules. As a special case, len(ring) is allowed and
                        refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                        len(ring)+1, where selected elements are True.
KEYWORDS
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    get_chrom=False compute chromaticities. Needs computing the tune at
                    2 different momentum deviations around the central one.
    get_w=False     computes chromatic amplitude functions (W) [4].
                    Needs to compute the optics at 2 different momentum
                    deviations around the central one.
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    XYStep=1.0e-8   transverse step for numerical computation
    DPStep=1.0E-6   momentum deviation used for computation of
                    chromaticities and dispersion
    coupled=True    if False, simplify the calculations by assuming
                    no H/V coupling
    twiss_in=None   Initial conditions for transfer line optics. Record
                    array as output by linopt, or dictionary. Keys:
                    'alpha' and 'beta'  (mandatory)
                    'closed_orbit',     (default 0)
                    'dispersion'        (default 0)
                    All other attributes are ignored.
OUTPUT
    lindata0        linear optics data at the entrance of the ring
    tune            [tune_A, tune_B], linear tunes for the two normal modes
                    of linear motion [1]
    chrom           [ksi_A , ksi_B], chromaticities ksi = d(nu)/(dP/P).
                    Only computed if 'get_chrom' is True
    lindata         linear optics at the points refered to by refpts, if
                    refpts is None an empty lindata structure is returned.

    lindata is a record array with fields:
    idx             element index in the ring
    s_pos           longitudinal position [m]
    m44             (4, 4) transfer matrix M from the beginning of ring
                    to the entrance of the element [2]
    closed_orbit    (6,) closed orbit vector
    dispersion      (4,) dispersion vector
    beta            [betax, betay] vector
    alpha           [alphax, alphay] vector
    mu              [mux, muy], betatron phase (modulo 2*pi)
    W               (2,) chromatic amplitude function (only if get_w==True)
    All values given at the entrance of each element specified in refpts.
    In case coupled == True additional outputs are available:
    gamma           gamma parameter of the transformation to eigenmodes
    A               (2, 2) matrix A in [3]
    B               (2, 2) matrix B in [3]
    C               (2, 2) matrix C in [3]
    Field values can be obtained with either
    lindata['idx']    or
    lindata.idx
REFERENCES
    [1] D.Edwards,L.Teng IEEE Trans.Nucl.Sci. NS-20, No.3, p.885-888, 1973
    [2] E.Courant, H.Snyder
    [3] D.Sagan, D.Rubin Phys.Rev.Spec.Top.-Accelerators and beams,
        vol.2 (1999)
    [4] Brian W. Montague Report LEP Note 165, CERN, 1979
```

5.24.2.6 linopt2()

```
def at.physics.linear.linopt2 (
              ring,
```

```
            * args,
            ** kwargs )

Perform linear analysis of an uncoupled lattice

elemdata0, beamdata, elemdata = linopt2(ring, refpts, **kwargs)

PARAMETERS
    lattice         lattice description.
    refpts=None     elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
KEYWORDS
    dp=0.0          momentum deviation.
    dct=None        path lengthening. If specified, dp is ignored and the
                    off-momentum is deduced from the path lengthening.
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    get_chrom=False compute chromaticities. Needs computing the tune at
                    2 different momentum deviations around the central one.
    get_w=False     computes chromatic amplitude functions (W) [4].
                    Needs to compute the optics at 2 different momentum
                    deviations around the central one.
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    XYStep=1.0e-8   transverse step for numerical computation
    DPStep=1.0E-6   momentum deviation used for computation of
                    chromaticities and dispersion
    twiss_in=None   Initial conditions for transfer line optics. Record
                    array as output by linopt, or dictionary. Keys:
                    'alpha' and 'beta'  (mandatory)
                    'closed_orbit',     (default 0)
                    'dispersion'        (default 0)
                    All other attributes are ignored.
OUTPUT
    lindata0        linear optics data at the entrance of the ring
    beamdata        lattice properties
    lindata         linear optics at the points refered to by refpts, if
                    refpts is None an empty lindata structure is returned.

    lindata is a record array with fields:
    s_pos           longitudinal position [m]
    M               (4, 4) transfer matrix M from the beginning of ring
                    to the entrance of the element [2]
    closed_orbit    (6,) closed orbit vector
    dispersion      (4,) dispersion vector
    beta            [betax, betay] vector
    alpha           [alphax, alphay] vector
    mu              [mux, muy], betatron phase (modulo 2*pi)
    W               (2,) chromatic amplitude function (only if get_w==True)
    All values given at the entrance of each element specified in refpts.
    Field values can be obtained with either
    lindata['idx']    or
    lindata.idx

    beamdata is a record with fields:
    tune            Fractional tunes
    chromaticity    Chromaticities, only computed if get_chrom=True

REFERENCES
    [1] D.Edwards,L.Teng IEEE Trans.Nucl.Sci. NS-20, No.3, p.885-888, 1973
    [2] E.Courant, H.Snyder
    [3] D.Sagan, D.Rubin Phys.Rev.Spec.Top.-Accelerators and beams,
        vol.2 (1999)
    [4] Brian W. Montague Report LEP Note 165, CERN, 1979
```

5.24.2.7 linopt4()

```
def at.physics.linear.linopt4 (
              ring,
             * args,
             ** kwargs )
```

```
Perform linear analysis of a H/V coupled lattice following Sagan/Rubin
4D-analysis of coupled motion

elemdata0, beamdata, elemdata = linopt4(lattice, refpts, **kwargs)

PARAMETERS
    lattice         lattice description.
    refpts=None     elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
KEYWORDS
    dp=0.0          momentum deviation.
    dct=None        path lengthening. If specified, dp is ignored and the
                    off-momentum is deduced from the path lengthening.
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    get_chrom=False compute chromaticities. Needs computing the tune at
                    2 different momentum deviations around the central one.
    get_w=False     computes chromatic amplitude functions (W) [4].
                    Needs to compute the optics at 2 different momentum
                    deviations around the central one.
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    XYStep=1.0e-8   transverse step for numerical computation
    DPStep=1.0E-6   momentum deviation used for computation of
                    chromaticities and dispersion
    twiss_in=None   Initial twiss to compute transfer line optics of the
                    type lindata, the initial orbit in twiss_in is ignored,
                    only the beta and alpha are required other quatities
                    set to 0 if absent
    twiss_in=None   Initial conditions for transfer line optics. Record
                    array as output by linopt, or dictionary. Keys:
                    'alpha' and 'beta'  (mandatory)
                    'closed_orbit',     (default 0)
                    'dispersion'        (default 0)
                    All other attributes are ignored.
OUTPUT
    lindata0        linear optics data at the entrance of the ring
    beamdata        lattice properties
    lindata         linear optics at the points refered to by refpts, if
                    refpts is None an empty lindata structure is returned.

    lindata is a record array with fields:
    s_pos           longitudinal position [m]
    M               (4, 4) transfer matrix M from the beginning of ring
                    to the entrance of the element [2]
    closed_orbit    (6,) closed orbit vector
    dispersion      (4,) dispersion vector
    beta            [betax, betay] vector
    alpha           [alphax, alphay] vector
    mu              [mux, muy], betatron phase (modulo 2*pi)
    gamma           gamma parameter of the transformation to eigenmodes [3]
    W               (2,) chromatic amplitude function (only if get_w==True)
    All values given at the entrance of each element specified in refpts.
    Field values can be obtained with either
    lindata['idx']    or
    lindata.idx
```

```
    beamdata is a record with fields:
    tune            Fractional tunes
    chromaticity    Chromaticities, only computed if get_chrom==True

REFERENCES
    [1] D.Edwards,L.Teng IEEE Trans.Nucl.Sci. NS-20, No.3, p.885-888, 1973
    [2] E.Courant, H.Snyder
    [3] D.Sagan, D.Rubin Phys.Rev.Spec.Top.-Accelerators and beams,
        vol.2 (1999)
    [4] Brian W. Montague Report LEP Note 165, CERN, 1979
```

5.24.2.8 linopt6()

```
def at.physics.linear.linopt6 (
              ring,
             * args,
             ** kwargs )


Perform linear analysis of a fully coupled lattice using normal modes

elemdata0, beamdata, elemdata = linopt6(lattice, refpts, **kwargs)

For circular machines, linopt6 analyses
the 4x4 1-turn transfer matrix if radiation is OFF, or
the 6x6 1-turn transfer matrix if radiation is ON.

For a transfer line, The "twiss_in" intput must contain either:
 - a field 'R', as provided by ATLINOPT6, or
  - the fields 'beta' and 'alpha', as provided by linopt and linopt6

PARAMETERS
    lattice         lattice description.
    refpts=None     elements at which data is returned.

KEYWORDS
    dp=None         Ignored if radiation is ON. Momentum deviation.
    dct=None        Ignored if radiation is ON. Path lengthening.
                    If specified, dp is ignored and the off-momentum is
                    deduced from the path lengthening.
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    get_chrom=False compute chromaticities. Needs computing the tune at
                    2 different momentum deviations around the central one.
    get_w=False     compute chromatic amplitude functions (W) [3]. Needs to
                    compute the optics at 2 different momentum deviations
                    around the central one.
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    XYStep=1.0e-8   transverse step for numerical computation
    DPStep=1.0E-6   momentum deviation used for computation of
                    the closed orbit
    twiss_in=None   Initial conditions for transfer line optics. Record
                    array as output by linopt, or dictionary. Keys:
                    'R' or 'alpha' and 'beta'   (mandatory)
                    'closed_orbit',             (default 0)
                    'dispersion'                (default 0)
                    If present, the attribute 'R' will be used, otherwise
                    the attributes 'alpha' and 'beta' will be used. All
                    other attributes are ignored.
    cavpts=None     Cavity location for off-momentum tuning

OUTPUT
    elemdata0       linear optics data at the entrance of the ring
    beamdata        lattice properties
    elemdata        linear optics at the points refered to by refpts, if
```

```
                    refpts is None an empty elemdata structure is returned.

    elemdata is a record array with fields:
    s_pos           longitudinal position [m]
    M               Transfer matrix from the entrance of the line (6, 6)
    closed_orbit    (6,) closed orbit vector
    dispersion      (4,) dispersion vector
    A               A-matrix (6, 6)
    R               R-matrices (3, 6, 6)
    beta            [betax, betay] vector
    alpha           [alphax, alphay] vector
    mu              [mux, muy], betatron phases
    W               (2,) chromatic amplitude function (only if get_w==True)

    All values given at the entrance of each element specified in refpts.
    Field values can be obtained with either
    elemdata['beta']    or
    elemdata.beta

    beamdata is a record with fields:
    tune            Fractional tunes
    chromaticity    Chromaticities, only computed if get_chrom==True
    damping_time    Damping times [s] (only if radiation is ON)

REFERENCES
    [1] Etienne Forest, Phys. Rev. E 58, 2481 - Published 1 August 1998
    [2] Andrzej Wolski, Phys. Rev. ST Accel. Beams 9, 024001 -
        Published 3 February 2006
    [3] Brian W. Montague Report LEP Note 165, CERN, 1979
```

5.24.2.9 linopt_auto()

```
def at.physics.linear.linopt_auto (
              ring,
             * args,
             ** kwargs )

This is a convenience function to automatically switch to the faster
linopt2 in case coupled=False and ring.radiation=False otherwise the
default linopt6 is used

PARAMETERS
    Same as linopt2 or linopt6

KEYWORDS
    coupled = True  If set to False H/V coupling will be ingnored to
                    simplify the calculation, needs radiation OFF

OUTPUT
    elemdata0       linear optics data at the entrance of the ring
    beamdata        lattice properties
    elemdata        linear optics at the points refered to by refpts, if
                    refpts is None an empty elemdata structure is returned.

!!!WARNING!!!       Output varies depending whether linopt2 or linopt6 is
                    called to be used with care
```

5.25 at.physics.matrix Namespace Reference

Functions

- def find_m44 (ring, dp=0.0, refpts=None, dct=None, orbit=None, keep_lattice=False, ∗∗kwargs)
- def find_m66 (ring, refpts=None, orbit=None, keep_lattice=False, ∗∗kwargs)
- def find_elem_m66 (elem, orbit=None, ∗∗kwargs)
- def gen_m66_elem (ring, o4b, o4e)

5.25.1 Detailed Description

```
transfer matrix related functions

A collection of functions to compute 4x4 and 6x6 transfer matrices
```

5.25.2 Function Documentation

5.25.2.1 find_elem_m66()

```
def at.physics.matrix.find_elem_m66 (
              elem,
              orbit = None,
             ** kwargs )
```

```
Numerically find the 6x6 transfer matrix of a single element

INPUT
    elem                AT element

KEYWORDS
    orbit=None          closed orbit at the entrance of the element,
                        default: 0.0
    XYStep=1.e-8        transverse step for numerical computation

OUTPUT
    m66                 (6, 6) transfer matrix
```

5.25.2.2 find_m44()

```
def at.physics.matrix.find_m44 (
              ring,
              dp = 0.0,
              refpts = None,
              dct = None,
              orbit = None,
              keep_lattice = False,
             ** kwargs )
```

```
find_m44 numerically finds the 4x4 transfer matrix of an accelerator
lattice for a particle with relative momentum deviation DP

IMPORTANT!!! find_m44 assumes constant momentum deviation.
PassMethod used for any element in the lattice SHOULD NOT
1.  change the longitudinal momentum dP
    (cavities , magnets with radiation, ...)
2.  have any time dependence (localized impedance, fast kickers, ...)

m44, t = find_m44(lattice, dp=0.0, refpts)
    return 4x4 transfer matrices between the entrance of the first element
    and each element indexed by refpts.
        m44:    full one-turn matrix at the entrance of the first element
```

```
        t:      4x4 transfer matrices between the entrance of the first
                element and each element indexed by refpts:
                (Nrefs, 4, 4) array

Unless an input orbit is introduced, find_m44 assumes that the lattice is
a ring and first finds the closed orbit.

PARAMETERS
    ring            lattice description
    dp              momentum deviation. Defaults to 0
    refpts          elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
                    Defaults to None, if refpts is None an empty array is
                    returned for mstack.

KEYWORDS
    dct=None        path lengthening. If specified, dp is ignored and
                    the off-momentum is deduced from the path lengthening.
    orbit=None      avoids looking for the closed orbit if is already known
                    ((6,) array)
    keep_lattice=False  When True, assume no lattice change since the
                    previous tracking.
    full=False      When True, matrices are full 1-turn matrices at
                    the entrance of each
                    element indexed by refpts.
    orbit=None      Avoids looking for the closed orbit if is already
                    known (6,) array
    XYStep=1.e-8    transverse step for numerical computation

See also find_m66, find_orbit4
```

5.25.2.3 find_m66()

```
def at.physics.matrix.find_m66 (
              ring,
              refpts = None,
              orbit = None,
              keep_lattice = False,
             ** kwargs )


find_m66 numerically finds the 6x6 transfer matrix of an accelerator
lattice by differentiation of lattice_pass near the closed orbit.
find_m66 uses find_orbit6 to search for the closed orbit in 6-D
In order for this to work the ring MUST have a CAVITY element

m66, t = find_m66(lattice, refpts)
    m66:    full one-turn 6-by-6 matrix at the entrance of the
            first element.
    t:      6x6 transfer matrices between the entrance of the first
            element and each element indexed by refpts (nrefs, 6, 6) array.

PARAMETERS
    ring            lattice description
    dp              momentum deviation. Defaults to 0
    refpts          elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
```

```
                      refers to the end of the last element,
                   2) an ordered list of such integers without duplicates,
                   3) a numpy array of booleans of maximum length
                      len(ring)+1, where selected elements are True.
                   Defaults to None, if refpts is None an empty array is
                   returned for mstack.

KEYWORDS
    keep_lattice=False  When True, assume no lattice change since the
                        previous tracking.
    orbit=None          Avoids looking for the closed orbit if is already
                        known (6,) array
    XYStep=1.e-8        transverse step for numerical computation

See also find_m44, find_orbit6
```

5.25.2.4 gen_m66_elem()

```
def at.physics.matrix.gen_m66_elem (
               ring,
               o4b,
               o4e )


converts a ring to a linear 6x6 matrix tracking elemtn
```

5.26 at.physics.nonlinear Namespace Reference

Functions

- def tunes_vs_amp (ring, amp=None, dim=0, window=1, nturns=512)
- def detuning (ring, xm=0.3e-4, ym=0.3e-4, npoints=3, window=1, nturns=512)
- def chromaticity (ring, method='linopt', dpm=0.02, npoints=11, order=3, dp=0, ∗∗kwargs)
- def gen_detuning_elem (ring, orbit=None)

5.26.1 Detailed Description

```
Function to compute quantities related to non-linear optics
```

5.26.2 Function Documentation

5.26.2.1 chromaticity()

```
def at.physics.nonlinear.chromaticity (
              ring,
              method = 'linopt',
              dpm = 0.02,
              npoints = 11,
              order = 3,
              dp = 0,
             ** kwargs )
```

```
This function uses computes the tunes to compute the tune for
the specified momentum offsets. Then it fits this data and returns
the chromaticity up to the given order (npoints>order)
OUTPUT
    (fitx, fity), dpa, qz
```

5.26.2.2 detuning()

```
def at.physics.nonlinear.detuning (
              ring,
              xm = 0.3e-4,
              ym = 0.3e-4,
              npoints = 3,
              window = 1,
              nturns = 512 )
```

```
This function uses tunes_vs_amp to compute the tunes for
the specified amplitudes. Then it fits this data and returns
result for dQx/dx, dQy/dx, dQx/dy, dQy/dy
```

5.26.2.3 gen_detuning_elem()

```
def at.physics.nonlinear.gen_detuning_elem (
              ring,
              orbit = None )
```

```
Generates an element that for detuning with amplitude
```

5.26.2.4 tunes_vs_amp()

```
def at.physics.nonlinear.tunes_vs_amp (
              ring,
              amp = None,
              dim = 0,
              window = 1,
              nturns = 512 )
```

```
Generates a range of tunes for varying x, y, or z amplitudes
```

5.27 at.physics.orbit Namespace Reference

Functions

- def find_orbit4 (ring, dp=0.0, refpts=None, dct=None, orbit=None, keep_lattice=False, ∗∗kwargs)
- def find_sync_orbit (ring, dct=0.0, refpts=None, dp=None, orbit=None, keep_lattice=False, ∗∗kwargs)
- def find_orbit6 (ring, refpts=None, orbit=None, dp=None, dct=None, keep_lattice=False, ∗∗kwargs)
- def find_orbit (ring, refpts=None, ∗∗kwargs)

5.27.1 Detailed Description

```
Closed orbit related functions
```

5.27.2 Function Documentation

5.27.2.1 find_orbit()

```
def at.physics.orbit.find_orbit (
              ring,
              refpts = None,
             ** kwargs )
```

```
find_orbit finds the closed orbit by numerically getting the fixed point
of the one turn map M calculated with lattice_pass.

Depending on the the lattice, find_orbit will:
- use find_orbit6 if ring.radiation is ON,
- use find_sync_orbit if ring.radiation is OFF and dct is specified,
- use find_orbit4 otherwise

PARAMETERS
    ring            Sequence of AT elements
    refpts          elements at which data is returned.

OUTPUT
    orbit0          ((6,) closed orbit vector at the entrance of the
                    1-st element
    orbit           (6, Nrefs) closed orbit vector at each location
                    specified in refpts

KEYWORDS
    dp=0            Momentum deviation, when radiation is OFF
    dct=0            Path lengthening, when radiation ids OFF
    keep_lattice    Assume no lattice change since the previous tracking.
                    Default: False
    guess=None      Initial guess for the closed orbit. It may help
                    convergence.
    orbit=None      Orbit at entrance of the lattice, if known. find_orbit
                    will then propagate it to the selected reference points
    For other keywords, refer to the underlying methods

See also find_orbit4, find_sync_orbit, find_orbit6
```

5.27.2.2 find_orbit4()

```
def at.physics.orbit.find_orbit4 (
              ring,
              dp = 0.0,
              refpts = None,
              dct = None,
              orbit = None,
              keep_lattice = False,
             ** kwargs )
```

```
findorbit4 finds the closed orbit in the 4-d transverse phase
space by numerically solving for a fixed point of the one turn
map M calculated with lattice_pass.

    (X, PX, Y, PY, dP, CT2 ) = M (X, PX, Y, PY, dP, CT1)

under the CONSTANT MOMENTUM constraint dP and with NO constraint
on the 6-th coordinate CT

IMPORTANT!!! findorbit4 imposes a constraint on dP and relaxes
the constraint on the revolution frequency. A physical storage
ring does exactly the opposite: the momentum deviation of a
particle on the closed orbit settles at the value
such that the revolution is synchronous with the RF cavity

            HarmNumber*Frev = Frf

To impose this artificial constraint in find_orbit4, PassMethod
used for any element SHOULD NOT
1.  change the longitudinal momentum dP (cavities , magnets with radiation)
2.  have any time dependence (localized impedance, fast kickers etc)

PARAMETERS
    ring            lattice description (radiation must be OFF)
    dp              momentum deviation. Defaults to 0
    refpts          elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
                    Defaults to None, if refpts is None an empty array is
                    returned for orbit.

OUTPUT
    orbit0          ((6,) closed orbit vector at the entrance of the
                    1-st element (x,px,y,py)
    orbit           (6, Nrefs) closed orbit vector at each location
                    specified in refpts

KEYWORDS
    dct=None        path lengthening. If specified, dp is ignored and
                    the off-momentum is deduced from the path lengthening.
    orbit=None      avoids looking for initial the closed orbit if is
                    already known ((6,) array). find_orbit4 propagates it
                    to the specified refpts.
    guess           (6,) initial value for the closed orbit. It may help
                    convergence. Default: (0, 0, 0, 0, 0, 0)
    keep_lattice    Assume no lattice change since the previous tracking.
                    Default: False
    convergence     Convergence criterion. Default: 1.e-12
    max_iterations  Maximum number of iterations. Default: 20
    XYStep          Step size. Default: DConstant.XYStep

See also find_sync_orbit, find_orbit6.
```

5.27.2.3 find_orbit6()

```
def at.physics.orbit.find_orbit6 (
              ring,
              refpts = None,
              orbit = None,
              dp = None,
              dct = None,
              keep_lattice = False,
             ** kwargs )
```

```
find_orbit6 finds the closed orbit in the full 6-D phase space
by numerically solving  for a fixed point of the one turn
map M calculated with lattice_pass

(X, PX, Y, PY, DP, CT2 ) = M (X, PX, Y, PY, DP, CT1)

with constraint  CT2 - CT1 = C*HarmNumber(1/Frf - 1/Frf0)

IMPORTANT!!! find_orbit6 is a realistic simulation
1.  The Frf frequency in the RF cavities (may be different from Frf0)
    imposes the synchronous condition
    CT2 - CT1 = C*HarmNumber(1/Frf - 1/Frf0)
2.  The algorithm numerically calculates
    6-by-6 Jacobian matrix J6. In order for (J-E) matrix
    to be non-singular it is NECESSARY to use a realistic
    PassMethod for cavities with non-zero momentum kick
    (such as RFCavityPass).
3.  find_orbit6 can find orbits with radiation.
    In order for the solution to exist the cavity must supply
    adequate energy compensation.
    In the simplest case of a single cavity, it must have
    'Voltage' field set so that Voltage > Erad - energy loss per turn
4.  There is a family of solutions that correspond to different RF buckets
    They differ in the 6-th coordinate by C*Nb/Frf. Nb = 1 .. HarmNum-1
5.  The value of the 6-th coordinate found at the cavity gives
    the equilibrium RF phase. If there is no radiation the phase is 0;

PARAMETERS
    ring            lattice description (radiation must be ON)
    refpts          elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
                    Defaults to None, if refpts is None an empty array is
                    returned for orbit.

OUTPUT
    orbit0          ((6,) closed orbit vector at the entrance of the
                    1-st element (x,px,y,py)
    orbit           (6, Nrefs) closed orbit vector at each location
                    specified in refpts

KEYWORDS
    orbit=None      avoids looking for initial the closed orbit if is
                    already known ((6,) array). find_orbit6 propagates it
                    to the specified refpts.
    guess           Initial value for the closed orbit. It may help
                    convergence. The default is computed from the energy
                    loss of the ring
    keep_lattice    Assume no lattice change since the previous tracking.
                    Default: False
    method          Method for energy loss computation
                    (see get_energy_loss)
                    default: ELossMethod.TRACKING
```

```
    cavpts=None     Cavity location. If None, use all cavities.
                    This is used to compute the initial synchronous phase.
    convergence     Convergence criterion. Default: 1.e-12
    max_iterations  Maximum number of iterations. Default: 20
    XYStep          Step size. Default: DConstant.XYStep
    DPStep          Step size. Default: DConstant.DPStep

See also find_orbit4, find_sync_orbit.
```

5.27.2.4 find_sync_orbit()

```
def at.physics.orbit.find_sync_orbit (
              ring,
              dct = 0.0,
              refpts = None,
              dp = None,
              orbit = None,
              keep_lattice = False,
             ** kwargs )


find_sync_orbit finds the closed orbit, synchronous with the RF cavity
and momentum deviation dP (first 5 components of the phase space vector)
% by numerically solving  for a fixed point
% of the one turn map M calculated with lattice_pass

    (X, PX, Y, PY, dP, CT2 ) = M (X, PX, Y, PY, dP, CT1)

under the constraint dCT = CT2 - CT1 = C/Frev - C/Frev0, where
Frev0 = Frf0/HarmNumber is the design revolution frequency
Frev  = (Frf0 + dFrf)/HarmNumber is the imposed revolution frequency

IMPORTANT!!!  find_sync_orbit imposes a constraint (CT2 - CT1) and
dP2 = dP1 but no constraint on the value of dP1, dP2
The algorithm assumes time-independent fixed-momentum ring
to reduce the dimensionality of the problem.

To impose this artificial constraint in find_sync_orbit
PassMethod used for any element SHOULD NOT
1.  change the longitudinal momentum dP (cavities , magnets with radiation)
2.  have any time dependence (localized impedance, fast kickers etc).

PARAMETERS
    ring            lattice description (radiation must be OFF)
    dct             Path length deviation. Default: 0
    refpts          elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
                    Defaults to None, if refpts is None an empty array is
                    returned for orbit.

OUTPUT
    orbit0          ((6,) closed orbit vector at the entrance of the
                    1-st element (x,px,y,py)
    orbit           (6, Nrefs) closed orbit vector at each location
                    specified in refpts

KEYWORDS
    orbit=None      avoids looking for initial the closed orbit if is
                    already known ((6,) array). find_sync_orbit propagates
```

```
                  it to the specified refpts.
    guess         (6,) initial value for the closed orbit. It may help
                  convergence. Default: (0, 0, 0, 0, 0, 0)
    keep_lattice  Assume no lattice change since the previous tracking.
                  Default: False
    convergence   Convergence criterion. Default: 1.e-12
    max_iterations Maximum number of iterations. Default: 20
    XYStep        Step size. Default: DConstant.XYStep

See also find_orbit4, find_orbit6.
```

5.28 at.physics.radiation Namespace Reference

Functions

- def ohmi_envelope (ring, refpts=None, orbit=None, keep_lattice=False)
- def get_radiation_integrals (ring, dp=None, twiss=None, ∗∗kwargs)
- def quantdiffmat (ring, orbit=None)
- def gen_quantdiff_elem (ring, orbit=None)
- def tapering (ring, multipoles=True, niter=1, ∗∗kwargs)

Variables

- list **ENVELOPE_DTYPE**

5.28.1 Detailed Description

```
Radiation and equilibrium emittances
```

5.28.2 Function Documentation

5.28.2.1 gen_quantdiff_elem()

```
def at.physics.radiation.gen_quantdiff_elem (
              ring,
              orbit = None )
```

```
Generates a quantum diffusion element
```

5.28.2.2 get_radiation_integrals()

```
def at.physics.radiation.get_radiation_integrals (
              ring,
              dp = None,
              twiss = None,
             ** kwargs )


Compute the 5 radiation integrals for uncoupled lattices.

PARAMETERS
    ring            lattice description.

KEYWORDS
    twiss=None      linear optics at all points (from linopt). If None,
                    it will be computed.
    dp=0.0          Ignored if radiation is ON. Momentum deviation.
    dct=None        Ignored if radiation is ON. Path lengthening.
                    If specified, dp is ignored and the off-momentum is
                    deduced from the path lengthening.
    method=linopt6  Method used for the analysis of the transfer matrix.
                    See get_optics.
                    linopt6: default
                    linopt2: faster if no longitudinal motion and
                             no H/V coupling,

OUTPUT
    i1, i2, i3, i4, i5
```

5.28.2.3 ohmi_envelope()

```
def at.physics.radiation.ohmi_envelope (
              ring,
              refpts = None,
              orbit = None,
              keep_lattice = False )


Calculate the equilibrium beam envelope in a
circular accelerator using Ohmi's beam envelope formalism [1]

emit0, beamdata, emit = ohmi_envelope(ring[, refpts])

PARAMETERS
    ring            Lattice object.
    refpts=None     elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.

KEYWORDS
    orbit=None          Avoids looking for the closed orbit if it is
                        already known ((6,) array)
    keep_lattice=False  Assume no lattice change since the previous
                        tracking

OUTPUT
    emit0               emittance data at the start/end of the ring
```

```
    beamdata            beam parameters at the start of the ring
    emit                emittance data at the points refered to by refpts,
                        if refpts is None an empty structure is returned.

    emit is a record array with fields:
    r66                 (6, 6) equilibrium envelope matrix R
    r44                 (4, 4) betatron emittance matrix (dpp = 0)
    m66                 (6, 6) transfer matrix from the start of the ring
    orbit6              (6,) closed orbit
    emitXY              (2,) betatron emittance projected on xxp and yyp
    emitXYZ             (3,) 6x6 emittance projected on xxp, yyp, ldp

    beamdata is a record array with fields:
    tunes               tunes of the 3 normal modes
    damping_rates       damping rates of the 3 normal modes
    mode_matrices       R-matrices of the 3 normal modes
    mode_emittances     equilibrium emittances of the 3 normal modes

    Field values can be obtained with either
    emit['r66']    or
    emit.r66

REFERENCES
    [1] K.Ohmi et al. Phys.Rev.E. Vol.49. (1994)
```

5.28.2.4 quantdiffmat()

```
def at.physics.radiation.quantdiffmat (
              ring,
              orbit = None )
```

```
This function computes the diffusion matrix of the whole ring

PARAMETERS
    ring            lattice description.
    orbit=None      initial orbit

OUTPUT
    diffusion matrix (6,6)
```

5.28.2.5 tapering()

```
def at.physics.radiation.tapering (
              ring,
              multipoles = True,
              niter = 1,
             ** kwargs )
```

```
Scales magnet strength with local energy to cancel the closed orbit
and optics errors due to synchrotron radiations. PolynomB is used for
dipoles such that the machine geometry is maintained. This is the ideal
tapering scheme where magnets and multipoles components (PolynomB and
PolynomA) are scaled individually.
!!! WARNING: This method works only for lattices without errors and
corrections: if not all corrections and field errors will also be
scaled !!!
```

```
tapering(ring) or ring.tapering()
PARAMETERS
    ring            lattice description.

KEYWORDS
    multipoles=True scale all multipoles
    method          Method for energy loss computation
                    (see get_energy_loss)
    niter=1         number of iteration
    XYStep=1.0e-8   transverse step for numerical computation
    DPStep=1.0E-6   momentum deviation used for computation of orbit6
```

5.28.3 Variable Documentation

5.28.3.1 ENVELOPE_DTYPE

```
list at.physics.radiation.ENVELOPE_DTYPE
```

Initial value:

```
1 =  [('r66', numpy.float64, (6, 6)),
2                   ('r44', numpy.float64, (4, 4)),
3                   ('m66', numpy.float64, (6, 6)),
4                   ('orbit6', numpy.float64, (6,)),
5                   ('emitXY', numpy.float64, (2,)),
6                   ('emitXYZ', numpy.float64, (3,))]
```

5.29 at.plot Namespace Reference

Namespaces

- generic
- specific
- standalone
- synopt

5.29.1 Detailed Description

```
AT plotting functions
```

5.30 at.plot.generic Namespace Reference

Functions

- def baseplot (ring, plot_function, ∗args, ∗∗kwargs)

Variables

- int **SLICES** = 400

5.30.1 Detailed Description

```
AT generic plotting function
```

5.30.2 Function Documentation

5.30.2.1 baseplot()

```
def at.plot.generic.baseplot (
              ring,
              plot_function,
             * args,
             ** kwargs )
```

```
baseplot divides the region of interest of ring into small elements,
calls the specified function to get the plot data and calls matplotlib
functions to generate the plot.
By default it creates a new figure for the plot, but if provided with
axes objects it can be used as part of a GUI

PARAMETERS
    ring            Lattice object
    plot_function   specific data generating function to be called

    All other positional parameters are sent to the plotting function

    plot_function is called as:

    title, left, right = plot_function(ring, refpts, *args, **kwargs)

    and should return 2 or 3 output:

    title   plot title or None
    left    tuple returning the data for the main (left) axis
       left[0]   y-axis label
       left[1]   xdata: (N,) array (s coordinate)
       left[2]   ydata: iterable of (N,) or (N,M) arrays. Lines from a
                 (N, M) array share the same style and label
       left[3]   labels: (optional) iterable of strings as long as ydata
    right   tuple returning the data for the secondary (right) axis
            (optional)

KEYWORDS
    s_range         lattice range of interest, default: unchanged,
                    initially set to the full cell.
    axes=None       axes for plotting as (primary_axes, secondary_axes)
                    Default: create new axes
    slices=400      Number of slices
    legend=True     Show a legend on the plot
    block=False     if True, block until the figure is closed
    dipole={}       Dictionary of properties overloading the default
                    properties of dipole representation.
                    See 'plot_synopt' for details
    quadrupole={}   Same definition as for dipole
    sextupole={}    Same definition as for dipole
    multipole={}    Same definition as for dipole
    monitor={}      Same definition as for dipole

    All other keywords are sent to the plotting function

RETURN
    left_axes       Main (left) axes
    right_axes      Secondary (right) axes or None
    synopt_axes     Synoptic axes
```

5.31 at.plot.specific Namespace Reference

Functions

- def pldata_beta_disp (ring, refpts, ∗∗kwargs)
- def plot_beta (ring, ∗∗kwargs)
- def pldata_linear (ring, refpts, ∗keys, ∗∗kwargs)
- def plot_linear (ring, ∗keys, ∗∗kwargs)
- def plot_trajectory (ring, r_in, nturns=1, ∗∗kwargs)

5.31.1 Detailed Description

```
AT plotting functions
```

5.31.2 Function Documentation

5.31.2.1 pldata_beta_disp()

```
def at.plot.specific.pldata_beta_disp (
              ring,
              refpts,
             ** kwargs )
```

```
Generates data for plotting beta functions and dispersion
```

5.31.2.2 pldata_linear()

```
def at.plot.specific.pldata_linear (
              ring,
              refpts,
             * keys,
             ** kwargs )
```

```
data extraction function for plotting results of get_optics
```

5.31.2.3 plot_beta()

```
def at.plot.specific.plot_beta (
              ring,
             ** kwargs )


Plot beta functions and dispersion

PARAMETERS
    ring            Lattice object

KEYWORDS
    dp=0.0          Ignored if radiation is ON. Momentum deviation.
    dct=None        Ignored if radiation is ON. Path lengthening.
                    If specified, dp is ignored and the off-momentum is
                    deduced from the path lengthening.
    method=linopt6  Method used for the analysis of the transfer matrix.
                    See get_optics.
                    linopt6: default
                    linopt2: faster if no longitudinal motion and
                             no H/V coupling,
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    ddp=1.0E-8      momentum deviation used for computation of
                    chromaticities and dispersion
    twiss_in=None   Initial conditions for transfer line optics. Record
                    array as output by linopt, or dictionary. Keys:
                    'R' or 'alpha' and 'beta'   (mandatory)
                    'closed_orbit',             (default 0)
                    'dispersion'                (default 0)
                    If present, the attribute 'R' will be used, otherwise
                    the attributes 'alpha' and 'beta' will be used. All
                    other attributes are ignored.
```

5.31.2.4 plot_linear()

```
def at.plot.specific.plot_linear (
              ring,
             * keys,
             ** kwargs )


axleft, axright = plot_linear(ring, left[, right], **keywords
Plot linear optical functions returned by get_optics

PARAMETERS
    ring            Lattice object
    left            Left axis description as a tuple:
                    (key[, indices[, indices]])
                      key:        'beta', 'closed_orbit',...
                      indices:    integer, sequence of integers, or slice
                    The number if sequences of indices is data[key].ndim-1
                    The number of indices is the number of curves to plot.
                    All sequences must have the same length.

        Examples:
          ('beta', [0, 1])              beta_x, beta_z
          ('dispersion', 0)             eta_x
          ('closed_orbit'), [1, 3])     x', z'
          ('m44', 2, 2)                 T33
          ('m44', [0, 0], [0, 1])       T11, T12
```

```
          ('m44', 2, slice(4))          T31, T32, T33, T34
                                        as a single block
          ('m44', [2,2,2,2], [0,1,2,3]) T31, T32, T33, T34

    right           Right axis (optional)

KEYWORDS
    title           Plot title, defaults to "Linear optics"
    dp=0.0          Ignored if radiation is ON. Momentum deviation.
    dct=None        Ignored if radiation is ON. Path lengthening.
                    If specified, dp is ignored and the off-momentum is
                    deduced from the path lengthening.
    method=linopt6  Method used for the analysis of the transfer matrix.
                    See get_optics.
                    linopt6: default
                    linopt2: faster if no longitudinal motion and
                             no H/V coupling,
    orbit           avoids looking for the closed orbit if is already known
                    ((6,) array)
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
    ddp=1.0E-8      momentum deviation used for computation of
                    chromaticities and dispersion
    twiss_in=None   Initial conditions for transfer line optics. Record
                    array as output by linopt, or dictionary. Keys:
                    'R' or 'alpha' and 'beta'   (mandatory)
                    'closed_orbit',             (default 0)
                    'dispersion'                (default 0)
                    If present, the attribute 'R' will be used, otherwise
                    the attributes 'alpha' and 'beta' will be used. All
                    other attributes are ignored.
```

5.31.2.5 plot_trajectory()

```
def at.plot.specific.plot_trajectory (
              ring,
              r_in,
              nturns = 1,
             ** kwargs )


plot a particle's trajectory

PARAMETERS
    ring            Lattice object
    r_in            6xN array: input coordinates of N particles
    nturns=1        Number of turns

KEYWORDS
    keep_lattice    Assume no lattice change since the previous tracking.
                    Defaults to False
```

5.32 at.plot.standalone Namespace Reference

Functions

- def plot_acceptance (ring, ∗args, ∗∗kwargs)

5.32.1 Detailed Description

```
AT plotting functions
```

5.32.2 Function Documentation

5.32.2.1 plot_acceptance()

```
def at.plot.standalone.plot_acceptance (
              ring,
             * args,
             ** kwargs )


Computes the acceptance at repfts observation points
Grid Coordiantes ordering is as follows: CARTESIAN: (x,y), RADIAL/RECURSIVE
(r, theta). Scalar inputs can be used for 1D grid.
The grid can be changed using grid_mode input:
at.GridMode.CARTESIAN: (x,y) grid
at.GridMode.RADIAL: (r,theta) grid
at.GridMode.RECURSIVE: (r,theta) recursive boundary search

Example usage:
ring.plot_acceptance(planes, npoints, amplitudes)
plt.show()

PARAMETERS
PARAMETERS
    ring            ring use for tracking
    planes          max. dimension 2, defines the plane where to search
                    for the acceptance, allowed values are: x,xp,y,yp,dp,ct
    npoints         number of points in each dimension shape (len(planes),)
    amplitudes      max. amplitude  or initial step in RECURSIVE in each
                    dimension
                    shape (len(planes),), for RADIAL/RECURSIVE grid:
                    r = sqrt(x**2+y**2)


KEYWORDS
    acceptance=None tuple containing pre-computed acceptance
                    (boundary, survived, grid)
    nturns=1024     Number of turns for the tracking
    refpts=None     Observation refpts, default start of the machine
    dp=None         static momentum offset
    offset=None     initial orbit, default closed orbit
    bounds=None     Allows to define boundaries for the grid default
                    values are:
                    GridMode.CARTESIAN: ((-1,1),(0,1))
                    GridMode.RADIAL/RECURSIVE: ((0,1),(pi,0))
    grid_mode       at.GridMode.CARTESIAN/RADIAL: track full vector
                    (default) at.GridMode.RECURSIVE: recursive search
    use_mp=False    Use python multiprocessing (patpass, default use
                    lattice_pass). In case multi-processing is not
                    enabled GridMode is forced to
                    RECURSIVE (most efficient in single core)
    divider=2       Value of the divider used in RECURSIVE boundary search
    verbose=True    Print out some inform
    start_method    This parameter allows to change the python
                    multiprocessing start method, default=None uses the
                    python defaults that is considered safe.
                    Available parameters: 'fork', 'spawn', 'forkserver'.
                    Default for linux is fork, default for MacOS and
                    Windows is spawn. fork may used for MacOS to speed-up
```

```
                    the calculation or to solve Runtime Errors, however
                    it is considered unsafe.

OUTPUT
    Returns 3 lists containing the 2D acceptance, the grid that was
    tracked and the particles of the grid that survived. The length
    of the lists=refpts. In case len(refpts)=1 the acceptance, grid,
    survived arrays are returned directly.
```

5.33 at.plot.synopt Namespace Reference

Functions

- def plot_synopt (ring, axes=None, dipole={}, quadrupole={}, sextupole={}, multipole={}, monitor={})

Variables

- **DIPOLE** = dict(label='Dipoles', facecolor=(0.5, 0.5, 1.0))
- **QUADRUPOLE** = dict(label='Quadrupoles', facecolor=(1.0, 0.5, 0.5))
- **SEXTUPOLE** = dict(label='Sextupoles', facecolor=(0.5, 1.0, 0.5))
- **MULTIPOLE** = dict(label='Multipoles', facecolor=(0.25, 0.75, 0.25))
- **MONITOR** = dict(label='Monitors', linestyle=None, marker=10, color='k')

5.33.1 Detailed Description

```
Plot a lattice synoptic
```

5.33.2 Function Documentation

5.33.2.1 plot_synopt()

```
def at.plot.synopt.plot_synopt (
              ring,
              axes = None,
              dipole = {},
              quadrupole = {},
              sextupole = {},
              multipole = {},
              monitor = {} )
```

```
Plot a synoptic of a lattice

PARAMETERS
    ring            Lattice object

KEYWORDS
    s_range=None    plot range, defaults to the full ring
    axes=None       axes for plotting the synoptic. If None, a new
                    figure will be created. Otherwise, a new axes object
                    sharing the same x-axis as the given one is created.
    dipole={}       Dictionary of properties overloading the default
                    properties. If None, dipoles will not be shown.
    quadrupole={}   Same definition as for dipole
    sextupole={}    Same definition as for dipole
    multipole={}    Same definition as for dipole
    monitor={}      Same definition as for dipole

RETURN
    synopt_axes     Synoptic axes
```

5.34 at.tracking Namespace Reference

Namespaces

- particles
- patpass

Variables

- **openmp**
- **mpi**

5.34.1 Detailed Description

```
Tracking functions
```

5.35 at.tracking.particles Namespace Reference

Functions

- def sigma_matrix (ring=None, twiss_in=None, emitx=None, emity=None, blength=None, espread=None)
- def beam (nparts, sigma, orbit=None)

5.35.1 Detailed Description

```
Functions relating to particle generation
```

5.35.2 Function Documentation

5.35.2.1 beam()

```
def at.tracking.particles.beam (
              nparts,
              sigma,
              orbit = None )
```

```
Generates an array of random particles according to the given sigma
matrix

PARAMETERS
    nparts          Number of particles
    sigma           sigma_matrix as calculated by at.sigma_matrix

KEYWORDS
    orbit=None      An orbit can be provided to give a center of
                    mass offset to the distribution

OUTPUT
    particle_dist   a matrix of shape (M, np) where M is shape of
                    sigma matrix
```

5.35.2.2 sigma_matrix()

```
def at.tracking.particles.sigma_matrix (
              ring = None,
              twiss_in = None,
              emitx = None,
              emity = None,
              blength = None,
              espread = None )
```

```
Calculate the correlation matrix to be used for particle generation

PARAMETERS
    ring            Lattice object or list of
                    twiss parameters.
    twiss_in        Data structure containing input
                    twiss parameters.
    emitx           Horizontal emittance [m.rad]
    emity           Vertical emittance [m.rad]
    blength         One sigma bunch length [m]
    espread         One sigma energy spread [dp/p]

OUTPUT
    sigma_matrix    6x6 correlation matrix

If the lattice object is provided with no other
arguments, ohmi_envelope is used to compute the
correlated sigma matrix.

If the lattice object and emittances and longitudinal
parameters are provided, then the 2x2 uncorrelated
matrices are computed for each plane (x,y,z) using
the initial optics computed from ring.get_optics,
and are combined together into the 6x6 matrix.

If the twiss_in is provided alongside the emittances and
longitudinal parameters, then the 2x2 uncorrelated
matrices are computed for each plane and combined
into the 6x6 matrix.
```

5.36 at.tracking.patpass Namespace Reference

Functions

- def **format_results** (results, r_in, losses)
- def patpass (ring, r_in, nturns=1, refpts=None, pool_size=None, start_method=None, ∗∗kwargs)

Variables

- **globring** = None

5.36.1 Detailed Description

```
Simple parallelisation of atpass() using multiprocessing.
```

5.36.2 Function Documentation

5.36.2.1 patpass()

```
def at.tracking.patpass.patpass (
              ring,
              r_in,
              nturns = 1,
              refpts = None,
              pool_size = None,
              start_method = None,
             ** kwargs )
```

```
Simple parallel implementation of atpass().  If more than one particle
is supplied, use multiprocessing to run each particle in a separate
process. In case a single particle is provided or the ring contains
ImpedanceTablePass element, atpass is returned

INPUT:
    ring            lattice description
    r_in:           6xN array: input coordinates of N particles
    nturns:         number of passes through the lattice line
    refpts          elements at which data is returned. It can be:
                    1) an integer in the range [-len(ring), len(ring)-1]
                       selecting the element according to python indexing
                       rules. As a special case, len(ring) is allowed and
                       refers to the end of the last element,
                    2) an ordered list of such integers without duplicates,
                    3) a numpy array of booleans of maximum length
                       len(ring)+1, where selected elements are True.
                    Defaults to None, meaning no refpts, equivelent to
                    passing an empty array for calculation purposes.
    losses          Activate loss maps
    pool_size       number of processes, if None the min(npart,nproc)
                    is used
    start_method    This parameter allows to change the python
                    multiprocessing start method, default=None uses the
                    python defaults that is considered safe.
```

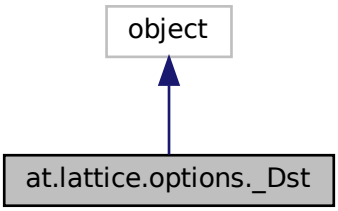
```
                    Available parameters: 'fork', 'spawn', 'forkserver'.
                    Default for linux is fork, default for MacOS and
                    Windows is spawn. fork may used for MacOS to speed-up
                    the calculation or to solve Runtime Errors, however it
                    is considered unsafe.
The following keywords overload the lattice value:
    particle:   circulating particle. Default: lattice.particle if
                existing, otherwise Particle('relativistic')
    energy      lattice energy

If 'energy' is not available, relativistic tracking if forced, rest_energy
is ignored.

 OUTPUT:
    (6, N, R, T) array containing output coordinates of N particles
    at R reference points for T turns.
    If losses ==True: {islost,turn,elem,coord} dictionnary containing
    flag for particles lost (True -> particle lost), turn, element and
    coordinates at which the particle is lost. Set to zero for particles
    that survived
```

Chapter 6

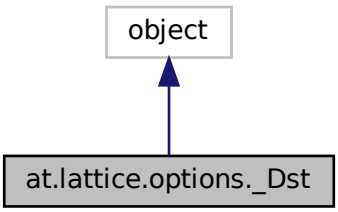
Class Documentation

6.1 at.lattice.options._Dst Class Reference

Inheritance diagram for at.lattice.options._Dst:



Collaboration diagram for at.lattice.options._Dst:



Public Member Functions

- def **__setattr__** (self, name, value)
- def **reset** (self, name)

Static Public Attributes

- int **XYStep** = 3.e-8
- int **DPStep** = 3.e-6
- int **OrbConvergence** = 1.e-12
- int **OrbMaxIter** = 20
- **omp_num_threads** = int(os.environ.get('OMP_NUM_THREADS', '0'))

6.1.1 Detailed Description

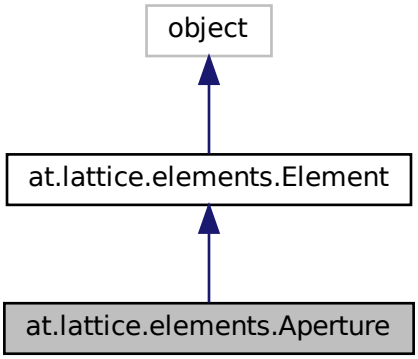
```
Set of constants for AT numerical analysis
```

The documentation for this class was generated from the following file:

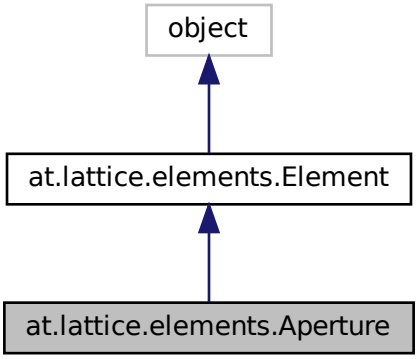
- at/lattice/options.py

6.2 at.lattice.elements.Aperture Class Reference

Inheritance diagram for at.lattice.elements.Aperture:



Collaboration diagram for at.lattice.elements.Aperture:



Public Member Functions

- def **__init__** (self, family_name, limits, ∗∗kwargs)

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = Element.REQUIRED_ATTRIBUTES + ['Limits']

Additional Inherited Members

6.2.1 Detailed Description

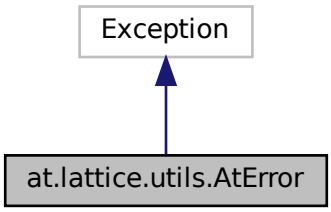
```
pyAT aperture element
```

The documentation for this class was generated from the following file:

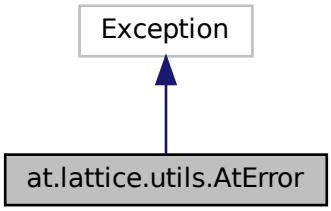
- at/lattice/elements.py

6.3 at.lattice.utils.AtError Class Reference

Inheritance diagram for at.lattice.utils.AtError:



Collaboration diagram for at.lattice.utils.AtError:

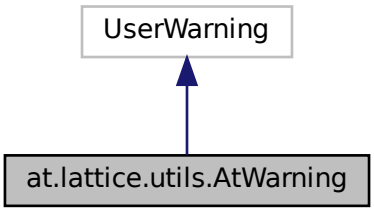


The documentation for this class was generated from the following file:

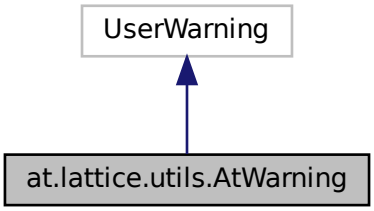
- at/lattice/utils.py

6.4 at.lattice.utils.AtWarning Class Reference

Inheritance diagram for at.lattice.utils.AtWarning:



Collaboration diagram for at.lattice.utils.AtWarning:

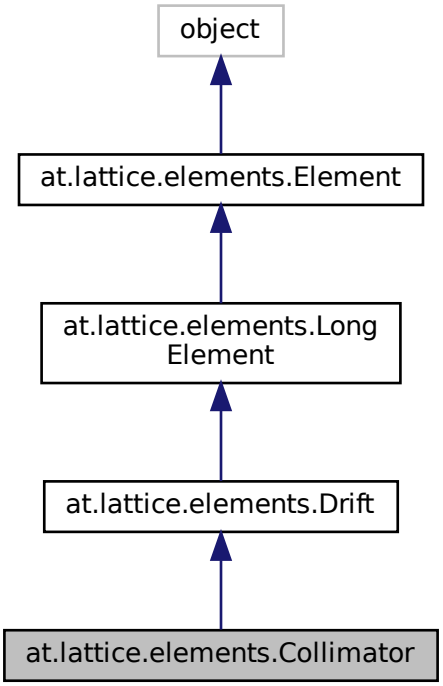


The documentation for this class was generated from the following file:

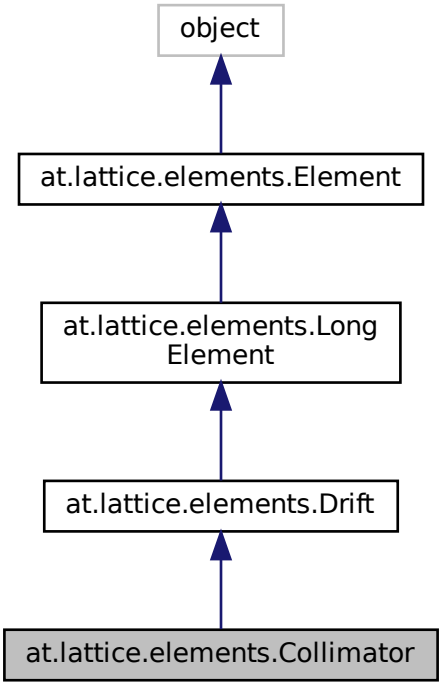
- at/lattice/utils.py

6.5 at.lattice.elements.Collimator Class Reference

Inheritance diagram for at.lattice.elements.Collimator:



Collaboration diagram for at.lattice.elements.Collimator:



Public Member Functions

- def __init__ (self, family_name, length, limits, **kwargs)

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = LongElement.REQUIRED_ATTRIBUTES + ['RApertures']

Additional Inherited Members

6.5.1 Detailed Description

```
pyAT collimator element
```

6.5.2 Constructor & Destructor Documentation

6.5.2.1 __init__()

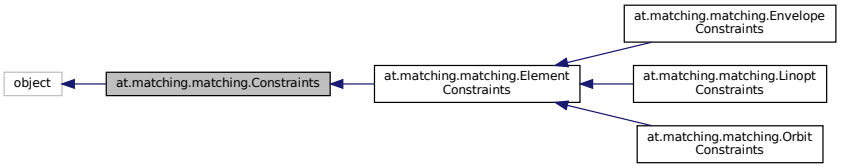
```
def at.lattice.elements.Collimator.__init__ (
              self,
              family_name,
              length,
              limits,
             ** kwargs )

Collimator(FamName, Length, limits, **keywords)
```

The documentation for this class was generated from the following file:

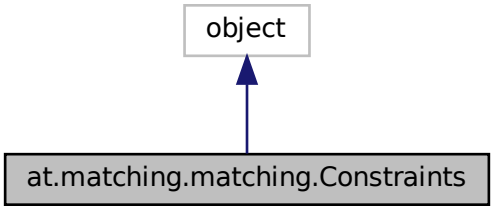
- at/lattice/elements.py

6.6 at.matching.matching.Constraints Class Reference

Inheritance diagram for at.matching.matching.Constraints:



Collaboration diagram for at.matching.matching.Constraints:



Public Member Functions

- def __init__ (self, *args, **kwargs)
- def add (self, fun, target, name=None, weight=1.0, bounds=(0.0, 0.0))
- def values (self, ring)
- def evaluate (self, ring)
- def status (self, ring, initial=None)

Static Public Member Functions

- def header ()

Public Attributes

- **name**
- **fun**
- **target**
- **weight**
- **lbound**
- **ubound**
- **rad**
- **args**
- **kwargs**

6.6.1 Detailed Description

```
Container for generic constraints:
  - a constraint is defined by a user-defined evaluation function.
  - constraints are added to the container with the Constraints.add method

  Example:
      # define an evaluation function for the ring circumference:
      def circ_fun(ring):
          return ring.get_s_pos(len(ring) + 1)

      # define an evaluation function for the momentum compaction factor:
      def mcf_fun(ring):
          return ring.get_mcf()

      # Construct the container:
      cnstrs = Constraints()

      # Add the two constraints:
      cnstrs.add(circ_fun, 850.0)
      cnstrs.add(mcf_fun, 1.0e-4, weight=0.1)
```

6.6.2 Constructor & Destructor Documentation

6.6.2.1 __init__()

```
def at.matching.matching.Constraints.__init__ (
              self,
             * args,
             ** kwargs )


Constraints(*args, **kwargs)
build a generic constraints container.

The positional and keyword parameters are provided to all
the evaluation functions.
```

6.6.3 Member Function Documentation

6.6.3.1 add()

```
def at.matching.matching.Constraints.add (
              self,
              fun,
              target,
              name = None,
              weight = 1.0,
              bounds = (0.0, 0.0) )
```

```
Add a target to the Constraints container

PARAMETERS
    fun           evaluation function. Called as:
          value = fun(ring, *args, **kwargs)
            value is the constrained parameter value
            value may be a scalar or an array.
            the positional and keyword parameters come from
            the Constraints initialisation
    target        desired value.

KEYWORDS
    name=None     name of the constraint. If None, name is generated
          from the name of the evaluation function
    weight=1.0    weight factor: the residual is (value-target)/weight
    bounds=(0,0)  lower and upper bounds. The parameter is constrained
          in the interval [target-low_bound target+up_bound]

The "target", "weight" and "bounds" input must be broadcastable to the
shape of "value".
```

6.6.3.2 evaluate()

```
def at.matching.matching.Constraints.evaluate (
              self,
              ring )
```

```
Return a flattened array of weighted residuals
```

6.6.3.3 header()

```
def at.matching.matching.Constraints.header ( )  [static]
```

```
Header for the display of constraint values and residuals
```

6.6.3.4 status()

```
def at.matching.matching.Constraints.status (
              self,
              ring,
              initial = None )

Return a string giving the actual state of constraints
```

6.6.3.5 values()

```
def at.matching.matching.Constraints.values (
              self,
              ring )

Return the list of actual parameter values
```

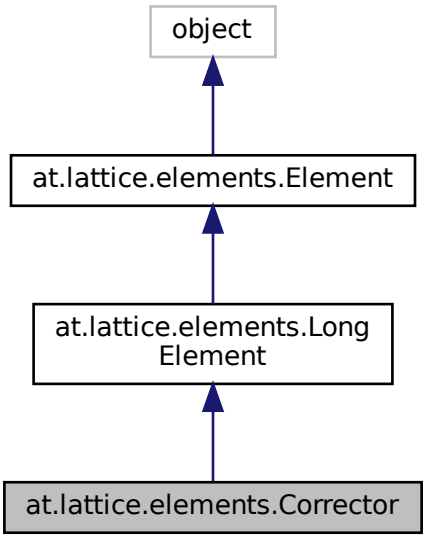
Reimplemented in at.matching.matching.ElementConstraints.

The documentation for this class was generated from the following file:

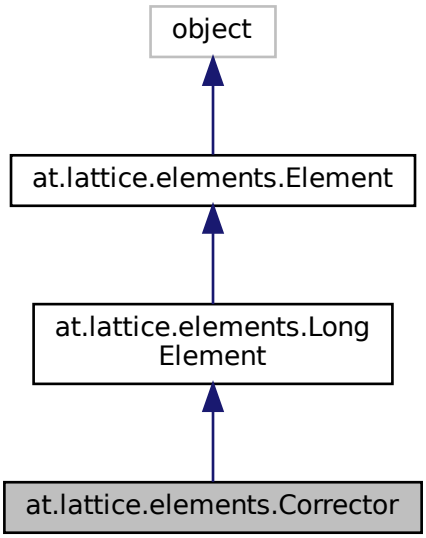
- at/matching/matching.py

6.7 at.lattice.elements.Corrector Class Reference

Inheritance diagram for at.lattice.elements.Corrector:



Collaboration diagram for at.lattice.elements.Corrector:



Public Member Functions

- def **__init__** (self, family_name, length, kick_angle, ∗∗kwargs)

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = LongElement.REQUIRED_ATTRIBUTES + ['KickAngle']

Additional Inherited Members

6.7.1 Detailed Description

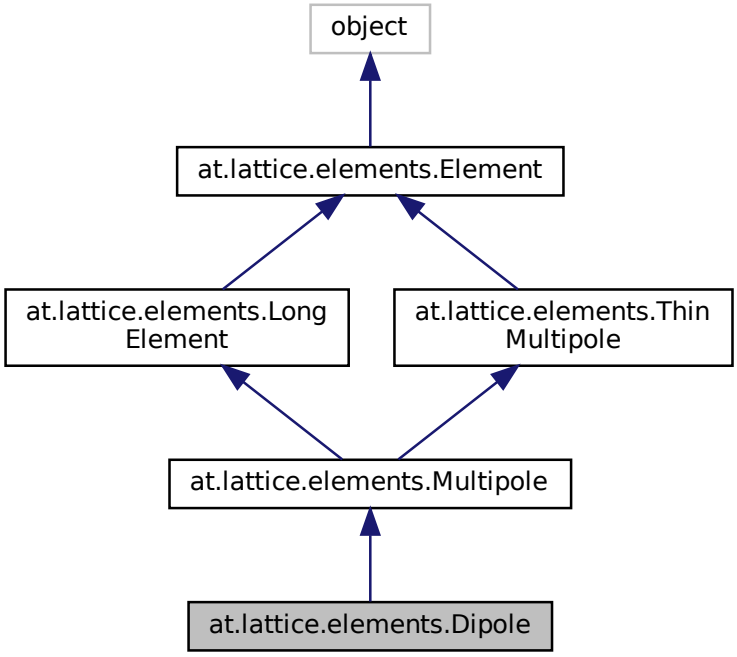
```
pyAT corrector element
```

The documentation for this class was generated from the following file:

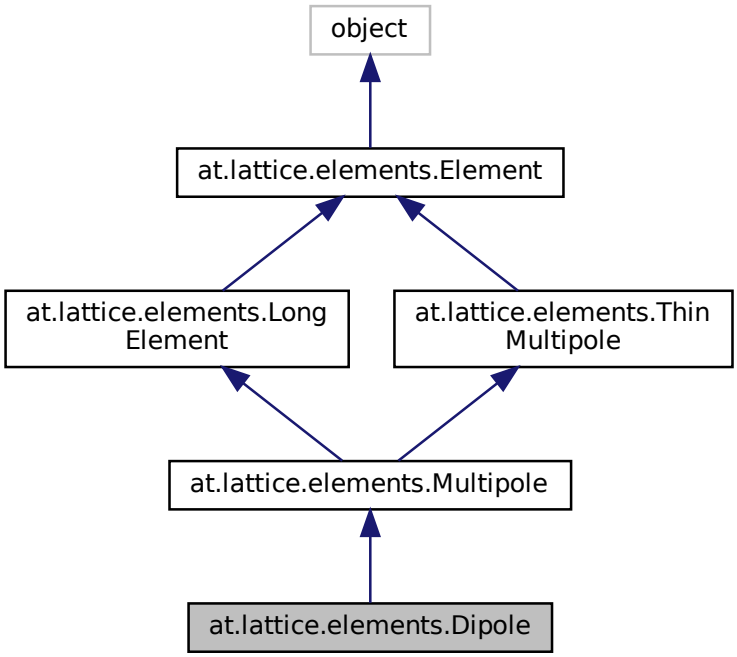
- at/lattice/elements.py

6.8 at.lattice.elements.Dipole Class Reference

Inheritance diagram for at.lattice.elements.Dipole:



Collaboration diagram for at.lattice.elements.Dipole:



Public Member Functions

- def __init__ (self, family_name, length, bending_angle=0.0, k=0.0, **kwargs)
- def **K** (self)
- def **K** (self, strength)

Static Public Attributes

- **REQUIRED_ATTRIBUTES**
- int **DefaultOrder** = 0

Additional Inherited Members

6.8.1 Detailed Description

```
pyAT dipole element
```

6.8.2 Constructor & Destructor Documentation

6.8.2.1 __init__()

```
def at.lattice.elements.Dipole.__init__ (
              self,
              family_name,
              length,
              bending_angle = 0.0,
              k = 0.0,
             ** kwargs )
```

```
Dipole(FamName, Length, bending_angle, Strength=0, **keywords)

Available keywords:
EntranceAngle   entrance angle (default 0.0)
ExitAngle       exit angle (default 0.0)
PolynomB        straight multipoles
PolynomA        skew multipoles
MaxOrder        Number of desired multipoles
NumIntSteps     Number of integration steps (default: 10)
FullGap         Magnet full gap
FringeInt1      Fringe field extension
FringeInt2
FringeBendEntrance  1: legacy version Brown First Order (default)
            2: SOLEIL close to second order of Brown
            3: THOMX
FringeBendExit
FringeQuadEntrance  0: no fringe fiels effect (default)
            1: Lee-Whiting's thin lens limit formula
            2: elegant-like
FringeQuadExit
fringeIntM0     Integrals for FringeQuad method 2
fringeIntP0
KickAngle       Correction deviation angles (H, V)
```

Reimplemented from at.lattice.elements.Multipole.

6.8.3 Member Data Documentation

6.8.3.1 REQUIRED_ATTRIBUTES

```
at.lattice.elements.Dipole.REQUIRED_ATTRIBUTES  [static]
```

Initial value:

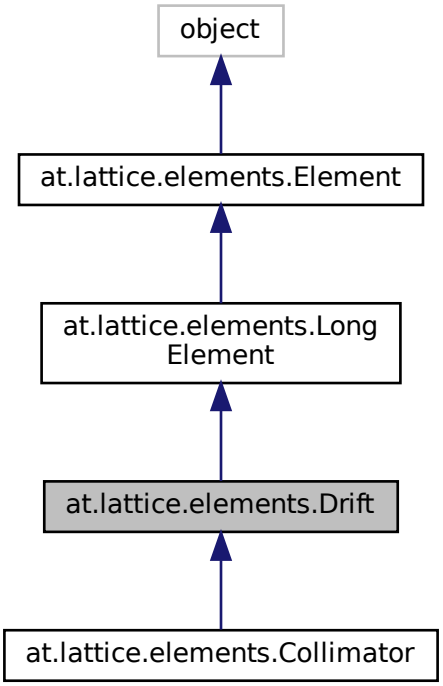
```
=  LongElement.REQUIRED_ATTRIBUTES + ['BendingAngle',
                                                    'K']
```

The documentation for this class was generated from the following file:

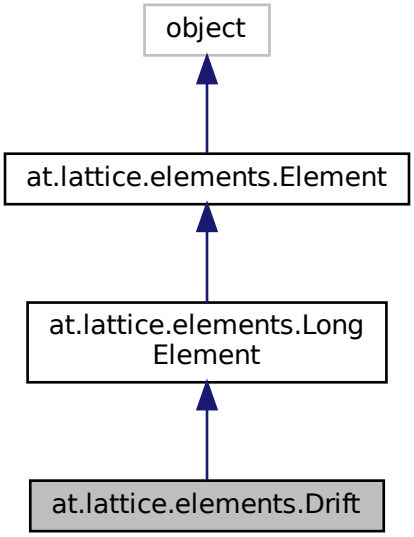
- at/lattice/elements.py

6.9 at.lattice.elements.Drift Class Reference

Inheritance diagram for at.lattice.elements.Drift:



Collaboration diagram for at.lattice.elements.Drift:



Public Member Functions

- def __init__ (self, family_name, length, **kwargs)
- def insert (self, insert_list)

Additional Inherited Members

6.9.1 Detailed Description

```
pyAT drift space element
```

6.9.2 Constructor & Destructor Documentation

6.9.2.1 __init__()

```
def at.lattice.elements.Drift.__init__ (
              self,
              family_name,
              length,
             ** kwargs )

Drift(FamName, Length, **keywords)
```

6.9.3 Member Function Documentation

6.9.3.1 insert()

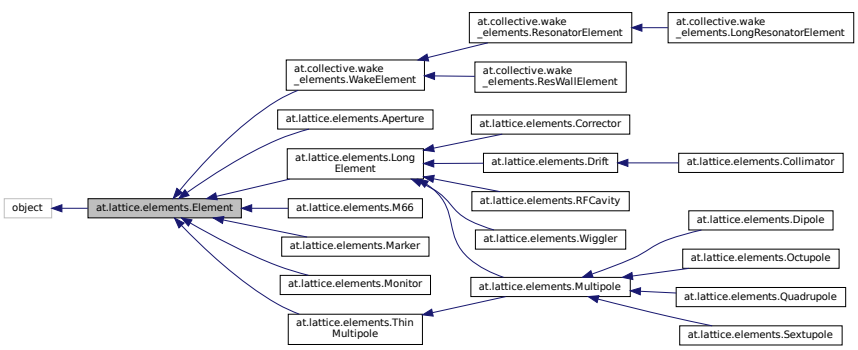
```
def at.lattice.elements.Drift.insert (
              self,
              insert_list )

insert elements inside a drift

arguments:
    insert_list: iterable, each item of insert_list is itself an
         iterable with 2 objects:
             1. the location where the center of the element
                will be inserted, given as a fraction of the
                Drift length.
             2. an element to be inserted at that location. If
                None, the drift will be divided but no element
                will be inserted.

Return a list of elements.

Drifts with negative lengths may be generated if necessary.

Examples:

>>> Drift('dr', 2.0).insert(((0.25, None), (0.75, None)))
[Drift('dr', 0.5), Drift('dr', 1.0), Drift('dr', 0.5)]

>>> Drift('dr', 2.0).insert(((0.0, Marker('m1')), (0.5, Marker('m2'))))
[Marker('m1'), Drift('dr', 1.0), Marker('m2'), Drift('dr', 1.0)]

>>> Drift('dr', 2.0).insert(((0.5, Quadrupole('qp', 0.4, 0.0)),))
[Drift('dr', 0.8), Quadrupole('qp', 0.4), Drift('dr', 0.8)]
```

The documentation for this class was generated from the following file:

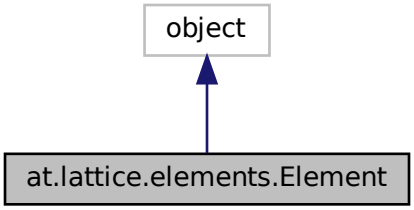
- at/lattice/elements.py

6.10 at.lattice.elements.Element Class Reference

Inheritance diagram for at.lattice.elements.Element:



Collaboration diagram for at.lattice.elements.Element:



Public Member Functions

- def **__init__** (self, family_name, ∗∗kwargs)
- def **__setattr__** (self, key, value)
- def **__str__** (self)
- def **__repr__** (self)
- def equals (self, other)
- def divide (self, frac)
- def update (self, ∗args, ∗∗kwargs)
- def copy (self)
- def deepcopy (self)
- def items (self)

Public Attributes

- **FamName**
- **Length**
- **PassMethod**

Static Public Attributes

- list **REQUIRED_ATTRIBUTES** = ['FamName']

6.10.1 Detailed Description

```
Base of pyat elements
```

6.10.2 Member Function Documentation

6.10.2.1 copy()

```
def at.lattice.elements.Element.copy (
              self )
```

Return a shallow copy of the element

6.10.2.2 deepcopy()

```
def at.lattice.elements.Element.deepcopy (
              self )
```

Return a deep copy of the element

6.10.2.3 divide()

```
def at.lattice.elements.Element.divide (
              self,
              frac )
```

split the element in len(frac) pieces whose length
is frac[i]*self.Length

arguments:
    frac   length of each slice expressed as a fraction of the
            initial length. sum(frac) may differ from 1.

Return a list of elements equivalent to the original.

Example:

```
>>> Drift('dr', 0.5).divide([0.2, 0.6, 0.2])
[Drift('dr', 0.1), Drift('dr', 0.3), Drift('dr', 0.1)]
```

Reimplemented in at.lattice.elements.LongElement.

6.10.2.4 equals()

```
def at.lattice.elements.Element.equals (
              self,
              other )
```

Whether an element is equivalent to another.

This implementation was found to be too slow for the generic
__eq__ method when comparing lattices.

6.10.2.5 items()

```
def at.lattice.elements.Element.items (
              self )
```

Iterates through the data members including slots and properties

6.10.2.6 update()

```
def at.lattice.elements.Element.update (
              self,
             * args,
             ** kwargs )
```

Update the element attributes with the given arguments

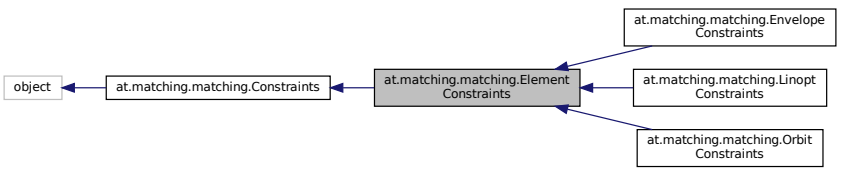
```
update(**kwargs)
update(mapping, **kwargs)
update(iterable, **kwargs)
```

The documentation for this class was generated from the following file:

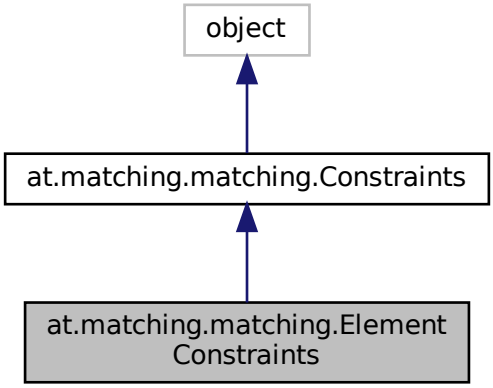
- at/lattice/elements.py

6.11 at.matching.matching.ElementConstraints Class Reference

Inheritance diagram for at.matching.matching.ElementConstraints:



Collaboration diagram for at.matching.matching.ElementConstraints:



Public Member Functions

- def **__init__** (self, ring, ∗args, ∗∗kwargs)
- def **add** (self, fun, target, refpts=None, ∗∗kwargs)
- def values (self, ring)
- def compute (self, ring, ∗args, ∗∗kwargs)

Public Attributes

- **nelems**
- **refs**
- **refpts**

Additional Inherited Members

6.11.1 Detailed Description

```
Base class for position-related constraints: handle the refpoints
of each target
```

6.11.2 Member Function Documentation

6.11.2.1 compute()

```
def at.matching.matching.ElementConstraints.compute (
              self,
              ring,
             * args,
             ** kwargs )

Dummy computation. Compute must return:
- an iterator over local data for each target
- a tuple of global data
```

Reimplemented in at.matching.matching.EnvelopeConstraints, at.matching.matching.OrbitConstraints, and at.matching.matching.LinoptConstraints.

6.11.2.2 values()

```
def at.matching.matching.ElementConstraints.values (
              self,
              ring )

Return the list of actual parameter values
```

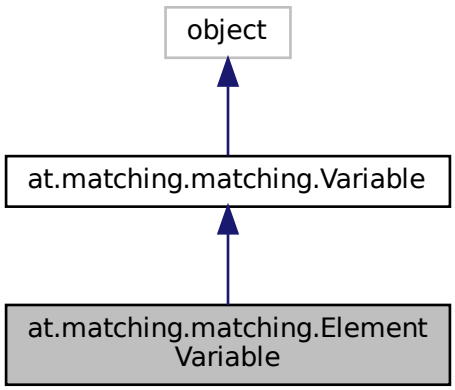
Reimplemented from at.matching.matching.Constraints.

The documentation for this class was generated from the following file:

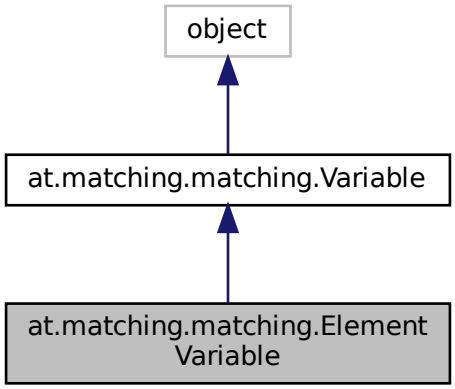
- at/matching/matching.py

6.12 at.matching.matching.ElementVariable Class Reference

Inheritance diagram for at.matching.matching.ElementVariable:



Collaboration diagram for at.matching.matching.ElementVariable:



Public Member Functions

- def **__init__** (self, refpts, attname, index=None, ∗∗kwargs)

Public Attributes

- **refpts**

Additional Inherited Members

6.12.1 Detailed Description

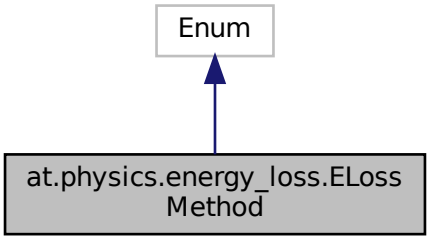
```
An ElementVariable is:
- a scalar attribute or
- an element of an array attribute
of one or several elements of a lattice
```

The documentation for this class was generated from the following file:

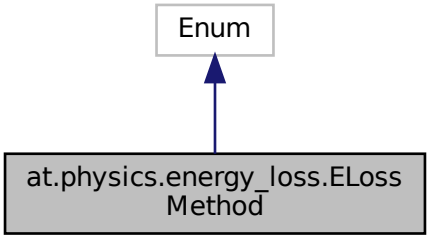
- at/matching/matching.py

6.13 at.physics.energy_loss.ELossMethod Class Reference

Inheritance diagram for at.physics.energy_loss.ELossMethod:



Collaboration diagram for at.physics.energy_loss.ELossMethod:



Static Public Attributes

- int **INTEGRAL** = 1
- int **TRACKING** = 2

6.13.1 Detailed Description

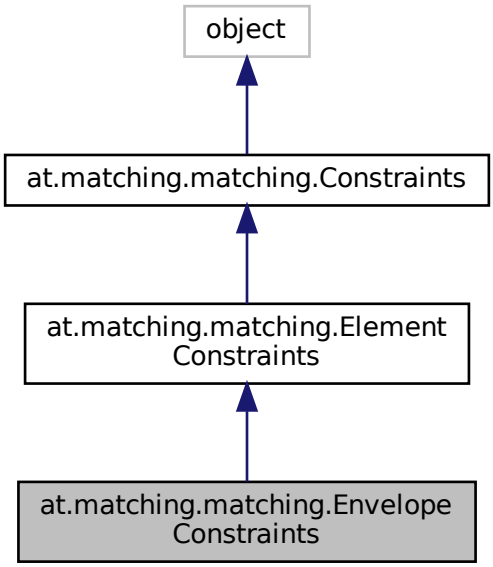
```
Enum class for energy loss methods
```

The documentation for this class was generated from the following file:

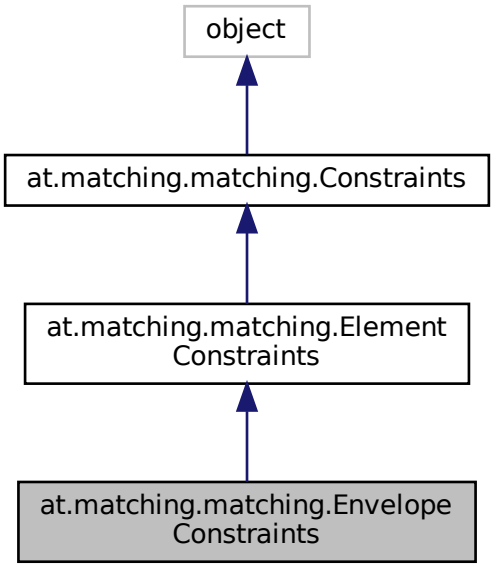
- at/physics/energy_loss.py

6.14 at.matching.matching.EnvelopeConstraints Class Reference

Inheritance diagram for at.matching.matching.EnvelopeConstraints:



Collaboration diagram for at.matching.matching.EnvelopeConstraints:



Public Member Functions

- def __init__ (self, ring)
- def add (self, param, target, refpts=None, index=None, name=None, **kwargs)
- def compute (self, ring, *args, **kwargs)

Additional Inherited Members

6.14.1 Detailed Description

```
Container for envelope constraints:
  - a constraint can be set on any result of at.ohmi_envelope
  - constraints are added to the container with the EnvelopeConstraints.add
    method.

  at.ohmi_envelope is called once before the evaluation of all constraints

  Example:
      cnstrs = EnvelopeConstraints(ring)
```

6.14.2 Constructor & Destructor Documentation

6.14.2.1 __init__()

```
def at.matching.matching.EnvelopeConstraints.__init__ (
              self,
              ring )
```

```
Build a EnvelopeConstraints container
```

6.14.3 Member Function Documentation

6.14.3.1 add()

```
def at.matching.matching.EnvelopeConstraints.add (
              self,
              param,
              target,
              refpts = None,
              index = None,
              name = None,
             ** kwargs )
```

```
Add a target to the EnvelopeConstraints container

PARAMETERS
    param         2 possibilities:
          - parameter name: see at.ohmi_envelope for the
            name of available parameters. In addition to
            local parameters, 'tunes', 'damping_rates',
            'mode_matrices' and 'mode_emittance' are allowed.
          - user-supplied parameter evaluation function:
                value = prm(emit_data, beam_data)
            emit_data contains the emittance data at all the
              specified refpoints
            value is the constrained parameter value
              (scalar or array).
    target        desired value.

KEYWORDS
    refpts=None   location of the constraint. Several locations may be
          given to apply the same constraint at several points.
    index=None    index in the parameter array. If None, the full array
          is used.
    name=None     name of the constraint. If None, name is generated
          from param and index.
    weight=1.0    weight factor: the residual is (value-target)/weight.
    bounds=(0,0)  lower and upper bounds. The parameter is constrained
          in the interval [target-low_bound target+up_bound]

The target, weight and bounds values must be broadcastable to the shape
of value.
```

6.14.3.2 compute()

```
def at.matching.matching.EnvelopeConstraints.compute (
              self,
              ring,
             * args,
             ** kwargs )
```

```
Optics computation before evaluation of all constraints
```

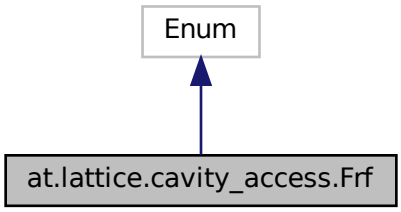
Reimplemented from at.matching.matching.ElementConstraints.

The documentation for this class was generated from the following file:

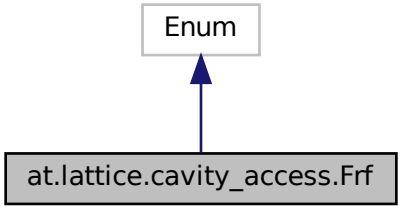
- at/matching/matching.py

6.15 at.lattice.cavity_access.Frf Class Reference

Inheritance diagram for at.lattice.cavity_access.Frf:



Collaboration diagram for at.lattice.cavity_access.Frf:



Static Public Attributes

- string **NOMINAL** = 'nominal'

6.15.1 Detailed Description

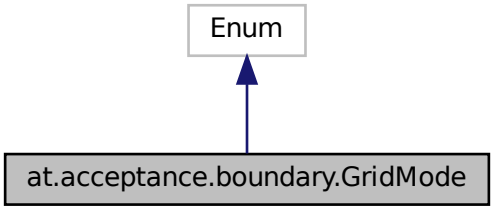
```
Enum class for frequency setting
```

The documentation for this class was generated from the following file:

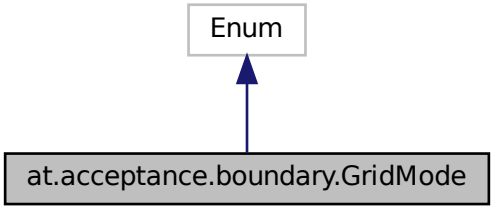
- at/lattice/cavity_access.py

6.16 at.acceptance.boundary.GridMode Class Reference

Inheritance diagram for at.acceptance.boundary.GridMode:



Collaboration diagram for at.acceptance.boundary.GridMode:



Static Public Attributes

- int **RADIAL** = 0
- int **CARTESIAN** = 1
- int **RECURSIVE** = 2

6.16.1 Detailed Description

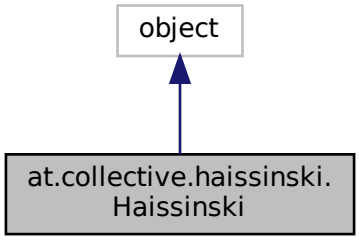
"
```
Class to defined the grid mode use when searching
for the boundary
```

The documentation for this class was generated from the following file:

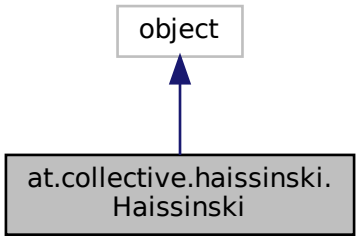
- at/acceptance/boundary.py

6.17 at.collective.haissinski.Haissinski Class Reference

Inheritance diagram for at.collective.haissinski.Haissinski:



Collaboration diagram for at.collective.haissinski.Haissinski:



Public Member Functions

- def **__init__** (self, wake_object, ring, m=12, kmax=1, current=1e-4, numIters=10, eps=1e-10)
- def set_weights (self)
- def precompute_S (self)
- def compute_Smat (self)
- def set_I (self, current)
- def initial_phi (self)
- def Fi (self)
- def **dFi_ij** (self, i, j)
- def dFi_dphij (self)
- def **compute_new_phi** (self)
- def **update** (self)
- def convergence (self)
- def **set_output** (self)
- def **solve** (self)
- def solve_steps (self, currents)

Public Attributes

- **circumference**
- **energy**
- **f_s**
- **nu_s**
- **sigma_e**
- **sigma_l**
- **eta**
- **ga**
- **betrel**
- **numIters**
- **eps**
- **ds**
- **s**
- **wtot_fun**
- **m**
- **npoints**
- **kmax**
- **dq**
- **q_array**
- **dtFlag**
- **weights**
- **Sfun_range**
- **Sfun**
- **Smat**
- **N**
- **lc**
- **phi**
- **phi_0**
- **allFi**
- **alldFi_dphij**
- **pseudo_inv**
- **phi_1**
- **conv**
- **res**
- **l_steps**
- **res_steps**

6.17.1 Detailed Description

```
Class to find the longitudinal distribution in the presence
of a short range wakefield.

This class is a direct implementation of the following paper:
"Numerical solution of the Haïssinski equation for the equilibrium
state of  a stored electron beam", R. Warnock, K.Bane, Phys. Rev. Acc.
and Beams 21, 124401 (2018)

The reader is referred to this paper for a description of the methods.
The equation number of key formula are written next to the relevant
function.

As input:
wake_object is a object that contains an srange and Z array.
ring is a ring instance which is needed for machine parameters
(sigma_l, sigma_e, etc)
```

```
m is the number of points in the full distribution that you want
kmax is the min and max of the range of the distribution.
See equation 27. In units of sigma_z0
current is the bunch current.
numIters is the number of iterations
eps is the convergence criteria.

the functions solve or solve_steps can be used after initialisation

Future developments of this class:
    Adding LR wake or harmonic cavity as done at SOLEIL. Needs
    to be added WITH this class which is just for short range wake.
```

6.17.2 Member Function Documentation

6.17.2.1 compute_Smat()

```
def at.collective.haissinski.Haissinski.compute_Smat (
              self )
```

```
The sampling of the integrated wake potential S
is only made at certain places. So all possibilities
are loaded into a matrix for speed.
```

6.17.2.2 convergence()

```
def at.collective.haissinski.Haissinski.convergence (
              self )
```

```
Equation 32
```

6.17.2.3 dFi_dphij()

```
def at.collective.haissinski.Haissinski.dFi_dphij (
              self )
```

```
Equation 30
```

6.17.2.4 Fi()

```
def at.collective.haissinski.Haissinski.Fi (
              self )
```

Equation 28

6.17.2.5 initial_phi()

```
def at.collective.haissinski.Haissinski.initial_phi (
              self )
```

Simply a gaussian but using the normalised units.
Page 5 top right, in the text.

6.17.2.6 precompute_S()

```
def at.collective.haissinski.Haissinski.precompute_S (
              self )
```

Equation 16

6.17.2.7 set_I()

```
def at.collective.haissinski.Haissinski.set_I (
              self,
              current )
```

Equation 11

6.17.2.8 set_weights()

```
def at.collective.haissinski.Haissinski.set_weights (
              self )
```

Page 7 second paragraph, in the text

6.17.2.9 solve_steps()

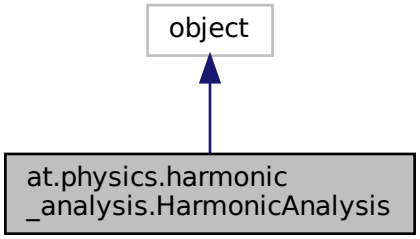
```
def at.collective.haissinski.Haissinski.solve_steps (
              self,
              currents )


INPUT:
    currents  an array of currents to solve. If 0 is given,
      a current of 10uA is used to prevent failure.
```

The documentation for this class was generated from the following file:

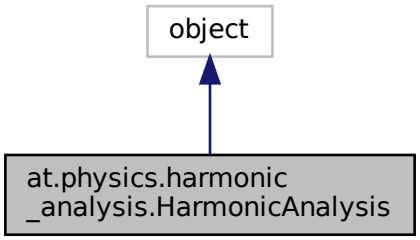
- at/collective/haissinski.py

6.18 at.physics.harmonic_analysis.HarmonicAnalysis Class Reference

Inheritance diagram for at.physics.harmonic_analysis.HarmonicAnalysis:



Collaboration diagram for at.physics.harmonic_analysis.HarmonicAnalysis:



Public Member Functions

- def **__init__** (self, samples, zero_pad=ZERO_PAD_DEF, hann=HANN_DEF)
- def **laskar_method** (self, num_harmonics)
- def **get_signal** (self)
- def **get_coefficient_for_freq** (self, freq)

Public Attributes

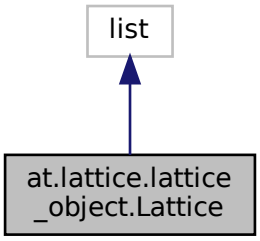
- **closed_orbit**
- **closed_orbit_rms**
- **peak_to_peak**

The documentation for this class was generated from the following file:

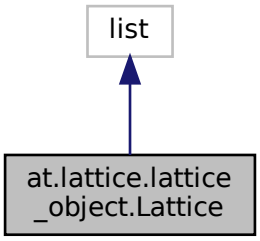
- at/physics/harmonic_analysis.py

6.19 at.lattice.lattice_object.Lattice Class Reference

Inheritance diagram for at.lattice.lattice_object.Lattice:



Collaboration diagram for at.lattice.lattice_object.Lattice:



Public Member Functions

- def __init__ (self, *args, iterator=None, scan=False, **kwargs)
- def **__getitem__** (self, key)
- def **__setitem__** (self, key, values)
- def **__delitem__** (self, key)
- def **__repr__** (self)
- def **__str__** (self)
- def __add__ (self, elems)
- def **__iadd__** (self, elems)
- def __mul__ (self, n)
- def **insert** (self, idx, elem)
- def **extend** (self, elems)
- def **append** (self, elem)
- def attrs (self)
- def uint32_refpts (self, refpts)
- def bool_refpts (self, refpts)
- def rotate (self, n)
- def update (self, *args, **kwargs)
- def copy (self)
- def deepcopy (self)
- def slice (self, size=None, slices=1)
- def attrs_filter (self, params, elem_iterator)
- def s_range (self)
- def **s_range** (self, value)
- def i_range (self)
- def energy (self)
- def **energy** (self, energy)
- def circumference (self)
- def revolution_frequency (self)
- def particle (self)
- def **particle** (self, particle)
- def **harmonic_number** (self)
- def **harmonic_number** (self, value)
- def **gamma** (self)
- def **beta** (self)
- def **BRho** (self)
- def radiation (self)
- def modify_elements (self, elem_modify, copy=True, **kwargs)
- def radiation_on (self, cavity_pass='RFCavityPass', dipole_pass='auto', quadrupole_pass='auto', wiggler_↩ pass='auto', sextupole_pass=None, octupole_pass=None, multipole_pass=None, copy=False)
- def radiation_off (self, cavity_pass='auto', dipole_pass='auto', quadrupole_pass='auto', wiggler_pass='auto', sextupole_pass='auto', octupole_pass='auto', multipole_pass='auto', copy=False)
- def sbreak (self, break_s, break_elems=None, **kwargs)
- def replace (self, refpts, **kwargs)

Public Attributes

- **s_range**

6.19.1 Detailed Description

```
Lattice object
An AT lattice is a sequence of AT elements.
A Lattice accepts extended indexing (as a numpy ndarray).

Lattice attributes;
    name            Name of the lattice
    energy          Particle energy
    periodicity     Number of super-periods to describe the full ring
    particle        Circulating particle
    harmonic_number Harmonic number of the full ring (periodicity x cells)

Lattice(elems, **params)        Create a new lattice object

    INPUT
        elems:                  any iterable of AT elements

    KEYWORDS
        name=''                 Name of the lattice
        energy                  Energy of the lattice
        periodicity=1           Number of periods
        particle='relativistic' Circulating particle. May be
                                'relativistic', 'electron',
                                'positron', 'proton'
                                or a Particle object
        iterator=None           Custom iterator (see below)
        *                       All other keywords will be set as
                                attributes of the Lattice object

To reduce the inter-package dependencies, some methods of the
lattice object are defined in other AT packages, in the module where
the underlying function is implemented.

Custom iterators:
Instead of running through 'elems', the Lattice constructor can use one
or several custom iterators.

Lattice(*args, iterator=it, **params)

    The iterator "it" is called as "it(params, *args)" and must return an
        iterator over AT elements for building the lattice. It must also
        fill the "params" dictionary used to set the Lattice attributes.

        params is the dictionary of lattice parameters. It is initialised
               with the keywords of the lattice constructor. The custom
               iterator may add, remove or mofify parameters.
               Finally, the remaining parameters will be set as Lattice
               attributes.
        *args  all positional arguments of the Lattice constructor are sent
               to the custom iterator.

    An iterator can be:
        - a "generator" which yields elements from scratch.
          Examples: a list, or a file iterator,
        - a "filter" which runs through an input iterator, processes each
          element, possibly adds parameters to the params dictionary
          and yields the processed elements.

    Example of chaining iterators (taken from "load_mat"):

    Lattice(ringparam_filter, matfile_generator, filename
            iterator=params_filter, **params)

    matfile_generator(params, filename)
        opens filename and generates AT elements for each cell of the
        Matlab cell array representing the lattice,

    ringparam_filter(params, matfile_generator, *args)
        runs through matfile_generator(params, *args), looks for RingParam
        elements, fills params with their information and discards them,

    params_filter(params, ringparam_filter, *args)
```

```
        runs through ringparam_filter(params, *args), looks for energy and
        periodicity if not yet defined.
```

6.19.2 Constructor & Destructor Documentation

6.19.2.1 __init__()

```
def at.lattice.lattice_object.Lattice.__init__ (
              self,
             * args,
              iterator = None,
              scan = False,
             ** kwargs )
```

```
Lattice constructor
```

6.19.3 Member Function Documentation

6.19.3.1 __add__()

```
def at.lattice.lattice_object.Lattice.__add__ (
              self,
              elems )
```

```
Add elems, an iterable of AT elements, to the lattice
```

6.19.3.2 __mul__()

```
def at.lattice.lattice_object.Lattice.__mul__ (
              self,
              n )
```

```
Repeats n times the lattice
```

6.19.3.3 attrs()

```
def at.lattice.lattice_object.Lattice.attrs (
              self )
```

Dictionary of lattice attributes

6.19.3.4 attrs_filter()

```
def at.lattice.lattice_object.Lattice.attrs_filter (
              self,
              params,
              elem_iterator )
```

Filter function which duplicates the lattice attributes

6.19.3.5 bool_refpts()

```
def at.lattice.lattice_object.Lattice.bool_refpts (
              self,
              refpts )
```

Return a boolean numpy array of length n_elements + 1 where
True elements are selected.

6.19.3.6 circumference()

```
def at.lattice.lattice_object.Lattice.circumference (
              self )
```

Ring circumference (full ring) [m]

6.19.3.7 copy()

```
def at.lattice.lattice_object.Lattice.copy (
              self )
```

Return a shallow copy of the lattice

6.19.3.8 deepcopy()

```
def at.lattice.lattice_object.Lattice.deepcopy (
              self )
```

Return a deep copy

6.19.3.9 energy()

```
def at.lattice.lattice_object.Lattice.energy (
              self )
```

Lattice energy

6.19.3.10 i_range()

```
def at.lattice.lattice_object.Lattice.i_range (
              self )
```

Range of elements inside the range of interest

6.19.3.11 modify_elements()

```
def at.lattice.lattice_object.Lattice.modify_elements (
              self,
              elem_modify,
              copy = True,
             ** kwargs )
```

Modify selected elements, in-place or in a lattice copy

```
PARAMETERS
    elem_modify         element selection function.

    If elem_modify(elem) returns None, the element is unchanged.
    Otherwise, elem_modify(elem) must return a dictionary of
    attribute name and values, to be set to elem.

RETURNS
    New lattice if copy == True
    None if copy == False

KEYWORDS
    copy=True   If True, return a shallow copy of the lattice. Only the
        modified elements are copied.
        If False, the modification is done in-place
```

6.19.3.12 particle()

```
def at.lattice.lattice_object.Lattice.particle (
              self )
```

Circulating particle

6.19.3.13 radiation()

```
def at.lattice.lattice_object.Lattice.radiation (
              self )
```

If True, at least one element modifies the beam energy

6.19.3.14 radiation_off()

```
def at.lattice.lattice_object.Lattice.radiation_off (
              self,
              cavity_pass = 'auto',
              dipole_pass = 'auto',
              quadrupole_pass = 'auto',
              wiggler_pass = 'auto',
              sextupole_pass = 'auto',
              octupole_pass = 'auto',
              multipole_pass = 'auto',
              copy = False )
```

Turn acceleration and radiation off and return the lattice

```
KEYWORDS
    cavity_pass='IdentityPass'  PassMethod set on cavities
    dipole_pass='auto'          PassMethod set on dipoles
    quadrupole_pass=None        PassMethod set on quadrupoles
    wiggler_pass='auto'         PassMethod set on wigglers
    copy=False  If False, the modification is done in-place,
        If True, return a shallow copy of the lattice. Only the
        radiating elements are copied with PassMethod modified.
        CAUTION: a shallow copy means that all non-radiating
        elements are shared with the original lattice.
        Any further modification will affect in both lattices.

    For PassMethod names, the convention is:
None            no change
'auto'          replace *RadPass by *Pass
anything else   set as it is
```

6.19.3.15 radiation_on()

```
def at.lattice.lattice_object.Lattice.radiation_on (
              self,
              cavity_pass = 'RFCavityPass',
              dipole_pass = 'auto',
              quadrupole_pass = 'auto',
              wiggler_pass = 'auto',
              sextupole_pass = None,
              octupole_pass = None,
              multipole_pass = None,
              copy = False )
```

```
Turn acceleration and radiation on and return the lattice

KEYWORDS
    cavity_pass='RFCavityPass'  PassMethod set on cavities
    dipole_pass='auto'          PassMethod set on dipoles
    quadrupole_pass='auto'      PassMethod set on quadrupoles
    wiggler_pass='auto'         PassMethod set on wigglers
    copy=False  If False, the modification is done in-place,
        If True, return a shallow copy of the lattice. Only the
        radiating elements are copied with PassMethod modified.
        CAUTION: a shallow copy means that all non-radiating
        elements are shared with the original lattice.
        Any further modification will affect in both lattices.

    For PassMethod names, the convention is:
None            no change
'auto'          replace *Pass by *RadPass
anything else   set as the new PassMethod
```

6.19.3.16 replace()

```
def at.lattice.lattice_object.Lattice.replace (
              self,
              refpts,
             ** kwargs )
```

```
Return a shallow copy of the lattice replacing the selected
elements by a deep copy
```

6.19.3.17 revolution_frequency()

```
def at.lattice.lattice_object.Lattice.revolution_frequency (
              self )
```

```
Revolution frequency (fullring) [Hz]
```

6.19.3.18 rotate()

```
def at.lattice.lattice_object.Lattice.rotate (
              self,
              n )
```

```
Return a new lattice rotated left by n elements
```

6.19.3.19 s_range()

```
def at.lattice.lattice_object.Lattice.s_range (
              self )
```

```
Range of interest: [s_min, s_max]. 'None' means the full cell.
```

6.19.3.20 sbreak()

```
def at.lattice.lattice_object.Lattice.sbreak (
              self,
              break_s,
              break_elems = None,
             ** kwargs )
```

```
Insert elements at selected locations in the lattice

PARAMETERS
    break_s:        location or array of locations of breakpoints
    break_elems:    elements to be inserted at breakpoints (array of
            elements as long as break_s or single element
            duplicated as necessary). Default: Marker('sbreak')
RETURNS
    A new lattice with new elements inserted at breakpoints
```

6.19.3.21 slice()

```
def at.lattice.lattice_object.Lattice.slice (
              self,
              size = None,
              slices = 1 )
```

```
Create a new lattice by slicing the range of interest into small
elements

KEYWORDS
    size=None       Length of a slice. Default: computed from the
            range and number of points:
                    sx = (s_max-s_min)/slices.
    slices=1        Number of slices in the specified range. Ignored if
            size is specified. Default: no slicing

RETURN
    New Lattice object
```

6.19.3.22 uint32_refpts()

```
def at.lattice.lattice_object.Lattice.uint32_refpts (
              self,
              refpts )

"Return a uint32 numpy array containing the indices of the selected
elements
```

6.19.3.23 update()

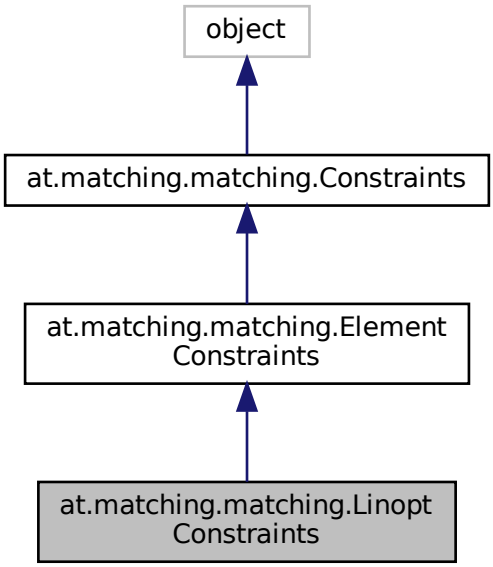
```
def at.lattice.lattice_object.Lattice.update (
              self,
             * args,
             ** kwargs )

Update the element attributes with the given arguments

update(**kwargs)
update(mapping, **kwargs)
update(iterable, **kwargs)
```

The documentation for this class was generated from the following file:

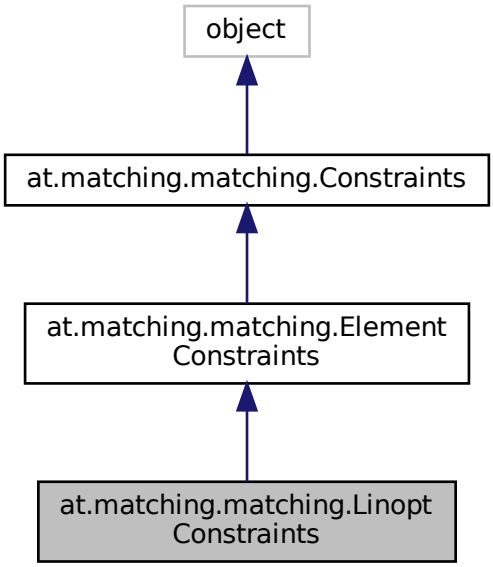
- at/lattice/lattice_object.py

6.20 at.matching.matching.LinoptConstraints Class Reference

Inheritance diagram for at.matching.matching.LinoptConstraints:



Collaboration diagram for at.matching.matching.LinoptConstraints:



Public Member Functions

- def __init__ (self, ring, ∗∗kwargs)
- def add (self, param, target, refpts=None, index=None, name=None, ∗∗kwargs)
- def compute (self, ring, ∗args, ∗∗kwargs)

Public Attributes

- **get_chrom**

Additional Inherited Members

6.20.1 Detailed Description

```
Container for linear optics constraints:
  - a constraint can be set on any result of at.get_optics
  - constraints are added to the container with the LinoptConstraints.add
    method.

  at.get_optics is called once before the evaluation of all constraints

  Example:
      cnstrs = LinoptConstraints(ring, dp=0.01, coupled=False)

      # Add a beta H (beta[0]) constraint at location ref_inj
      cnstrs.add('beta_x_inj', 'beta', 18.0, refpts=ref_inj, index=0)
```

```
# Add a tune constraint
cnstrs.add('tunes', 0.44, index=0, weight=0.01)

# Add a chromaticity constraint (both planes)
cnstrs.add('chroms', [0.0 0.0])

# define a constraint of phase advances between 2 points
def mu_diff(lindata, tune, chrom):
    delta_mu = (lindata[1].mu - lindata[0].mu)/(2*np.pi)
    return delta_mu % 1.0

# Add a H phase advance constraint, giving the desired locations
cnstrs.add(mu_diff, 0.5, refpts=[sf0 sf1], index=0)
```

6.20.2 Constructor & Destructor Documentation

6.20.2.1 __init__()

```
def at.matching.matching.LinoptConstraints.__init__ (
             self,
             ring,
            ** kwargs )
```

```
Build a LinoptConstraints container

KEYWORDS
dp=0.0          momentum deviation.
twiss_in=None   Initial twiss parameters for transfer line optics.
        "lindata" stucture, where only the beta and alpha are
        required and used.
orbit=None      Initial trajectory for transfer line
        ((6,) array)
method=linopt6  Method used for the analysis of the transfer matrix.
        Can be None, at.linopt2, at.linopt4, at.linopt6
        linopt2:    no longitudinal motion, no H/V coupling,
        linopt4:    no longitudinal motion, Sagan/Rubin
                    4D-analysis of coupled motion,
        linopt6:    with or without longitudinal motion, normal
                    mode analysis
```

6.20.3 Member Function Documentation

6.20.3.1 add()

```
def at.matching.matching.LinoptConstraints.add (
             self,
             param,
             target,
             refpts = None,
             index = None,
             name = None,
            ** kwargs )
```

```
Add a target to the LinoptConstraints container

PARAMETERS
    param         2 possibilities:
          - parameter name: see at.linopt for the name of
            available parameters. In addition to local optical
            parameters, 'tunes' and 'chroms' are allowed.
          - user-supplied parameter evaluation function:
                value = param(lindata, tune, chrom)
            lindata contains the optics parameters at all the
              specified refpoints
            value is the constrained parameter value
              (scalar or array).
    target        desired value.

KEYWORDS
    refpts=None   location of the constraint. Several locations may be
          given to apply the same constraint at several points.
    index=None    index in the parameter array. If None, the full array
          is used.
    name=None     name of the constraint. If None, name is generated
          from param and index.
    weight=1.0    weight factor: the residual is (value-target)/weight.
    bounds=(0,0)  lower and upper bounds. The parameter is constrained
          in the interval [target-low_bound target+up_bound]
    UseInteger    Match integer part of mu, much slower as the optics
          calculation is done for all refpts

The target, weight and bounds values must be broadcastable to the shape
of value.
```

6.20.3.2 compute()

```
def at.matching.matching.LinoptConstraints.compute (
              self,
              ring,
             * args,
             ** kwargs )

Optics computation before evaluation of all constraints
```

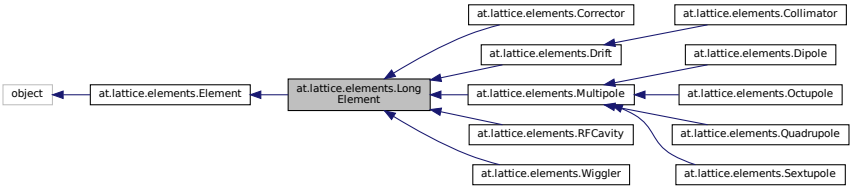
Reimplemented from at.matching.matching.ElementConstraints.

The documentation for this class was generated from the following file:

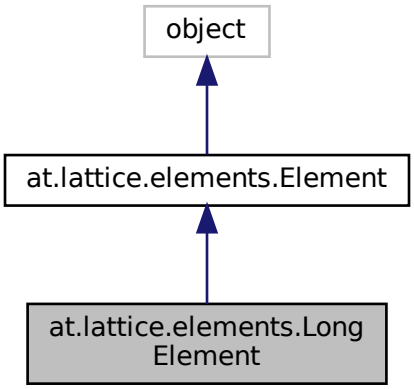
- at/matching/matching.py

6.21 at.lattice.elements.LongElement Class Reference

Inheritance diagram for at.lattice.elements.LongElement:



Collaboration diagram for at.lattice.elements.LongElement:



Public Member Functions

- def **__init__** (self, family_name, length, ∗args, ∗∗kwargs)
- def divide (self, frac)

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = Element.REQUIRED_ATTRIBUTES + ['Length']

Additional Inherited Members

6.21.1 Detailed Description

```
pyAT long element
```

6.21.2 Member Function Documentation

6.21.2.1 divide()

```
def at.lattice.elements.LongElement.divide (
              self,
              frac )
```

```
split the element in len(frac) pieces whose length
is frac[i]*self.Length

arguments:
    frac            length of each slice expressed as a fraction of the
            initial length. sum(frac) may differ from 1.

Return a list of elements equivalent to the original.

Example:

>>> Drift('dr', 0.5).divide([0.2, 0.6, 0.2])
[Drift('dr', 0.1), Drift('dr', 0.3), Drift('dr', 0.1)]
```

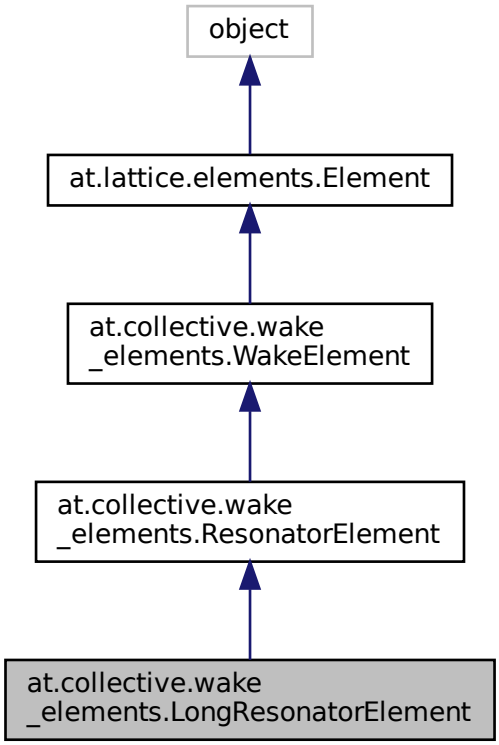
Reimplemented from at.lattice.elements.Element.

The documentation for this class was generated from the following file:

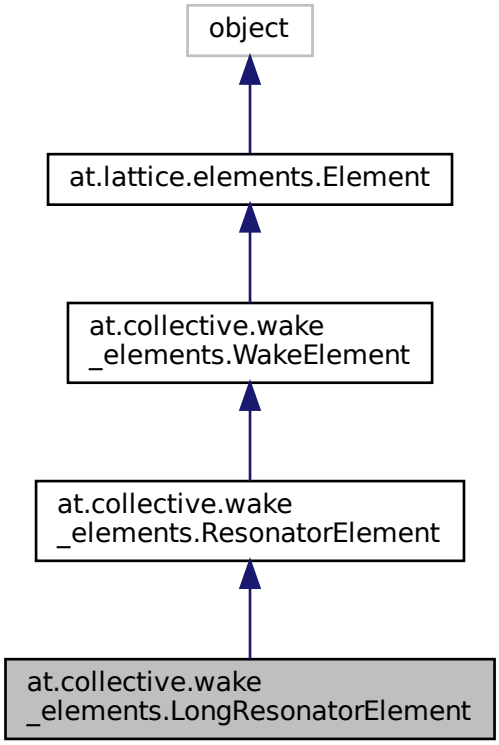
- at/lattice/elements.py

6.22 at.collective.wake_elements.LongResonatorElement Class Reference

Inheritance diagram for at.collective.wake_elements.LongResonatorElement:



Collaboration diagram for at.collective.wake_elements.LongResonatorElement:



Public Member Functions

- def **__init__** (self, family_name, ring, srange, frequency, qfactor, rshunt, ∗∗kwargs)
- def **rebuild_wake** (self)

Additional Inherited Members

6.22.1 Detailed Description

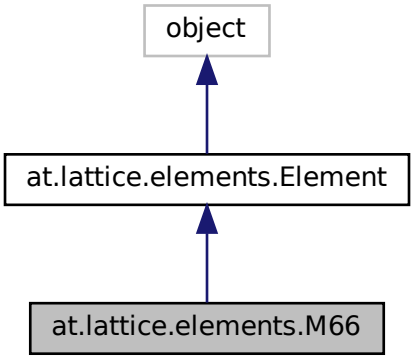
```
Class to generate a longitudinal resonator, inherits from WakeElement
   additional argument are frequency, qfactor, rshunt
```

The documentation for this class was generated from the following file:

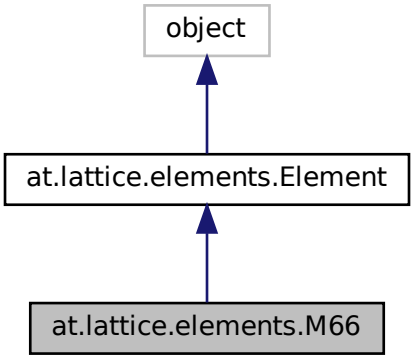
- at/collective/wake_elements.py

6.23 at.lattice.elements.M66 Class Reference

Inheritance diagram for at.lattice.elements.M66:



Collaboration diagram for at.lattice.elements.M66:



Public Member Functions

- def **__init__** (self, family_name, m66=None, ∗∗kwargs)

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = Element.REQUIRED_ATTRIBUTES

Additional Inherited Members

6.23.1 Detailed Description

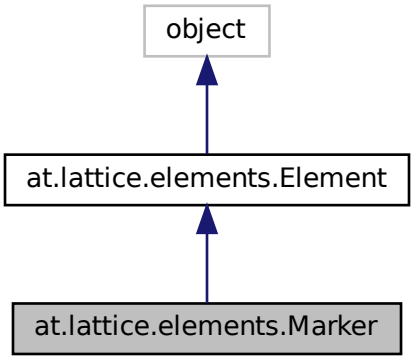
```
Linear (6, 6) transfer matrix
```

The documentation for this class was generated from the following file:

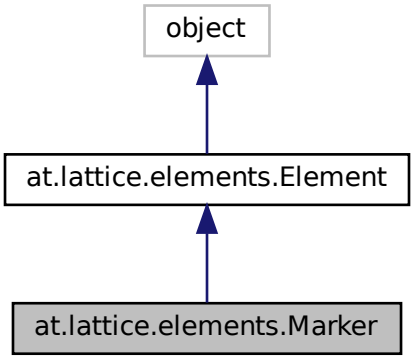
- at/lattice/elements.py

6.24 at.lattice.elements.Marker Class Reference

Inheritance diagram for at.lattice.elements.Marker:



Collaboration diagram for at.lattice.elements.Marker:



Additional Inherited Members

6.24.1 Detailed Description

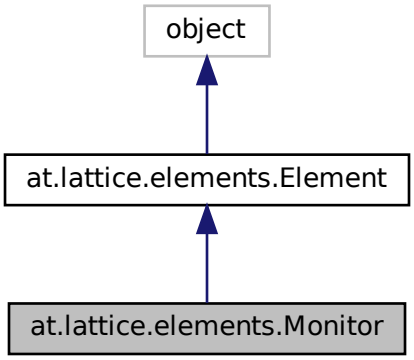
```
pyAT marker element
```

The documentation for this class was generated from the following file:

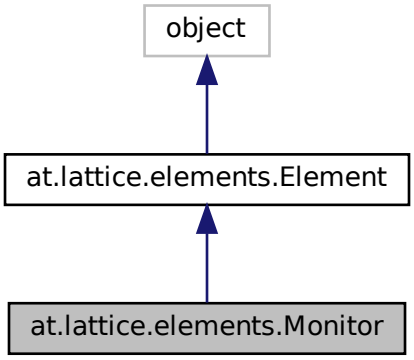
- at/lattice/elements.py

6.25 at.lattice.elements.Monitor Class Reference

Inheritance diagram for at.lattice.elements.Monitor:



Collaboration diagram for at.lattice.elements.Monitor:



Additional Inherited Members

6.25.1 Detailed Description

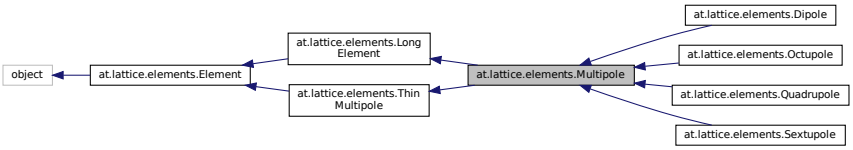
`pyAT monitor element`

The documentation for this class was generated from the following file:

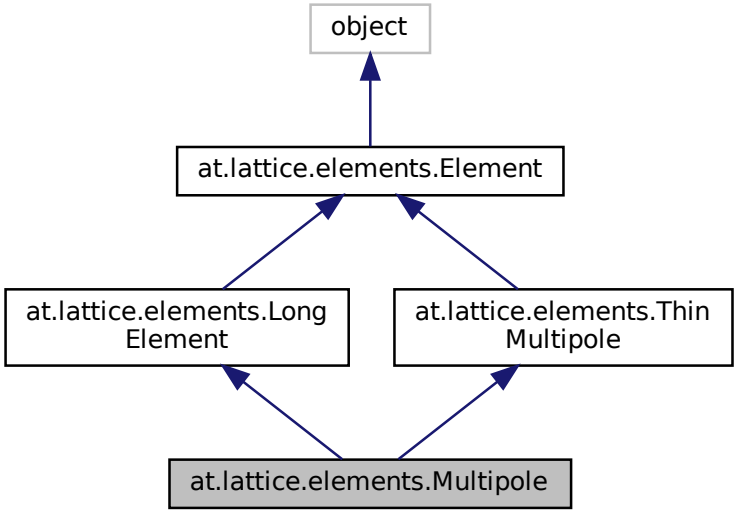
- at/lattice/elements.py

6.26 at.lattice.elements.Multipole Class Reference

Inheritance diagram for at.lattice.elements.Multipole:



Collaboration diagram for at.lattice.elements.Multipole:



Public Member Functions

- def __init__ (self, family_name, length, poly_a, poly_b, ∗∗kwargs)

Static Public Attributes

- **REQUIRED_ATTRIBUTES**

Additional Inherited Members

6.26.1 Detailed Description

```
pyAT multipole element
```

6.26.2 Constructor & Destructor Documentation

6.26.2.1 __init__()

```
def at.lattice.elements.Multipole.__init__ (
              self,
              family_name,
              length,
              poly_a,
              poly_b,
             ** kwargs )
```

```
Multipole(FamName, Length, PolynomA, PolynomB, **keywords)

Available keywords:
MaxOrder        Number of desired multipoles. Default: highest index of
        non-zero polynomial coefficients
NumIntSteps     Number of integration steps (default: 10)
KickAngle       Correction deviation angles (H, V)
```

Reimplemented in at.lattice.elements.Dipole.

6.26.3 Member Data Documentation

6.26.3.1 REQUIRED_ATTRIBUTES

```
at.lattice.elements.Multipole.REQUIRED_ATTRIBUTES  [static]
```

Initial value:

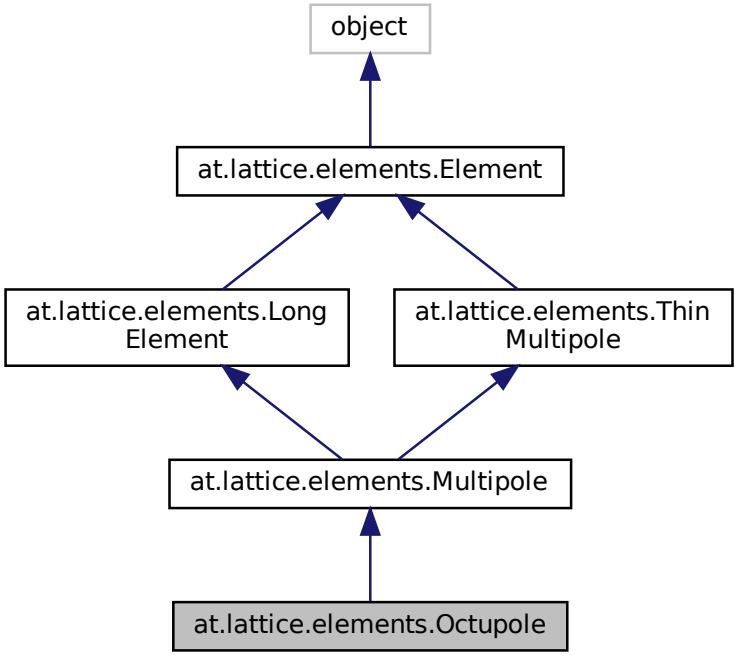
```
=  LongElement.REQUIRED_ATTRIBUTES + ['PolynomA',
                                                    'PolynomB']
```

The documentation for this class was generated from the following file:

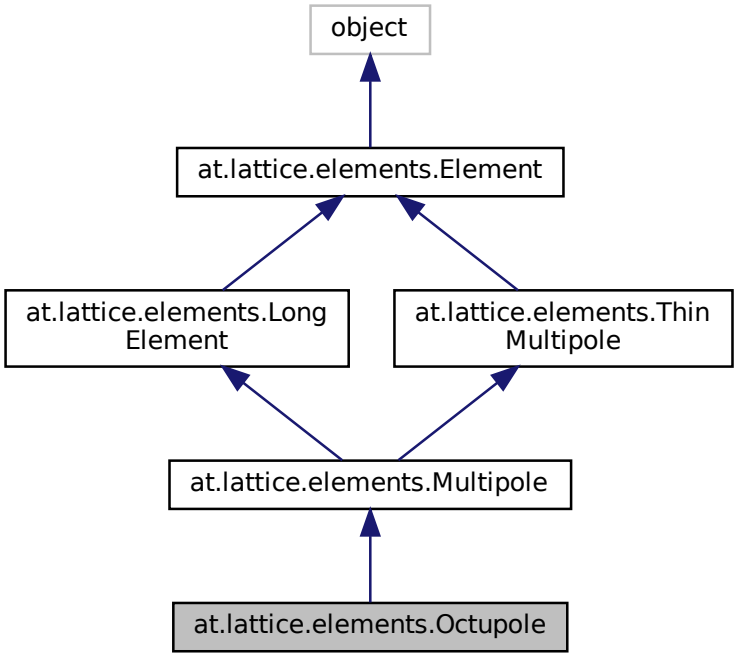
- at/lattice/elements.py

6.27 at.lattice.elements.Octupole Class Reference

Inheritance diagram for at.lattice.elements.Octupole:



Collaboration diagram for at.lattice.elements.Octupole:



Static Public Attributes

- **REQUIRED_ATTRIBUTES** = Multipole.REQUIRED_ATTRIBUTES
- int **DefaultOrder** = 3

Additional Inherited Members

6.27.1 Detailed Description

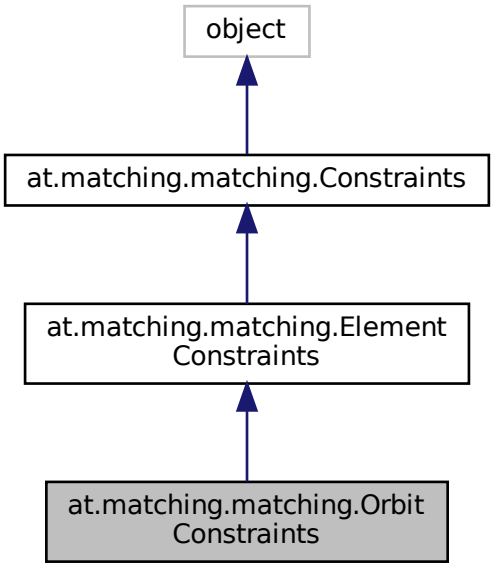
```
pyAT octupole element, with no changes from multipole at present
```

The documentation for this class was generated from the following file:

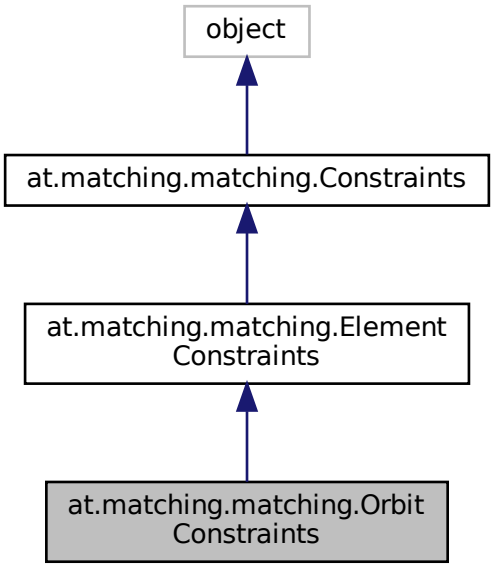
- at/lattice/elements.py

6.28 at.matching.matching.OrbitConstraints Class Reference

Inheritance diagram for at.matching.matching.OrbitConstraints:



Collaboration diagram for at.matching.matching.OrbitConstraints:



Public Member Functions

- def __init__ (self, ring, ∗args, ∗∗kwargs)
- def add (self, target, refpts=None, index=None, name=None, ∗∗kwargs)
- def compute (self, ring, ∗args, ∗∗kwargs)

Additional Inherited Members

6.28.1 Detailed Description

```
Container for orbit constraints:
The closed orbit can be handled with LinoptConstraints, but for problems
which do not involve parameters other than orbit, like steering or
orbit bumps, OrbitConstraints is much faster.

  at.find_orbit is called once before the evaluation of all constraints

Example:
    cnstrs = OrbitConstraints(ring, dp=0.01)

    # Add a bump (x=-0.004, x'=0) constraint at location ref_inj
    cnstrs.add([-0.004, 0.0], refpts=ref_inj, index=slice(2))
```

6.28.2 Constructor & Destructor Documentation

6.28.2.1 __init__()

```
def at.matching.matching.OrbitConstraints.__init__ (
              self,
              ring,
             * args,
             ** kwargs )


Build a OrbitConstraints container

KEYWORDS
dp=0            Momentum deviation, when radiation is OFF
dct=0            Path lengthening, when radiation ids OFF
orbit=None          Initial trajectory for transfer line: (6,) array
```

Reimplemented from at.matching.matching.ElementConstraints.

6.28.3 Member Function Documentation

6.28.3.1 add()

```
def at.matching.matching.OrbitConstraints.add (
              self,
              target,
              refpts = None,
              index = None,
              name = None,
             ** kwargs )
```

```
Add a target to the OrbitConstraints container

PARAMETERS
    target        desired value.

KEYWORDS
    refpts=None   location of the constraint. Several locations may be
          given to apply the same constraint at several points.
    index=None    index in the orbit vector. If None, the full orbit
          is used. Example:
            index=0         # x
            index=2         # z
            index=slice(4)  # x, x', z, z'
    name='orbit'  name of the constraint.
    weight=1.0    weight factor: the residual is (value-target)/weight.
    bounds=(0,0)  lower and upper bounds. The parameter is constrained
          in the interval [target-low_bound target+up_bound]

The target, weight and bounds values must be broadcastable to the shape
of value.
```

6.28.3.2 compute()

```
def at.matching.matching.OrbitConstraints.compute (
              self,
              ring,
             * args,
             ** kwargs )
```

```
Orbit computation before evaluation of all constraints
```

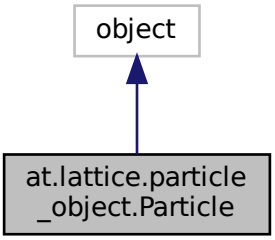
Reimplemented from at.matching.matching.ElementConstraints.

The documentation for this class was generated from the following file:

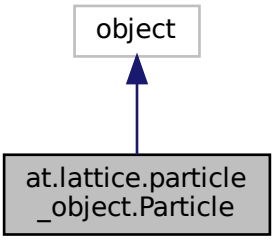
- at/matching/matching.py

6.29 at.lattice.particle_object.Particle Class Reference

Inheritance diagram for at.lattice.particle_object.Particle:



Collaboration diagram for at.lattice.particle_object.Particle:



Public Member Functions

- def **__init__** (self, name='relativistic', ∗∗kwargs)
- def **to_dict** (self)
- def **__repr__** (self)
- def **__str__** (self)
- def **rest_energy** (self)
- def **charge** (self)

Public Attributes

- **name**

6.29.1 Detailed Description

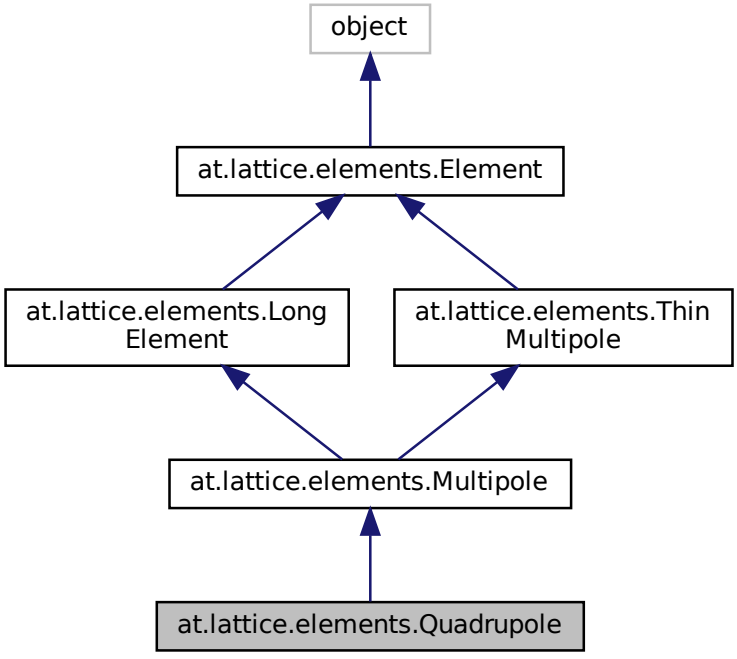
```
Particle object

This object defines the properties of the particles circulating in a ring

Particle(name, **params)

PARAMETERS
    name        Particle name. 'electron', 'positron and 'proton' are
                predefined. For other particles, the rest energy and charge
                must be provided as keywords.

KEYWORDS
    rest_energy Particle rest energy [ev]
    charge      Particle charge [elementary charge]
    *           Other keywords will be set as attributes of the particle
```

The documentation for this class was generated from the following file:

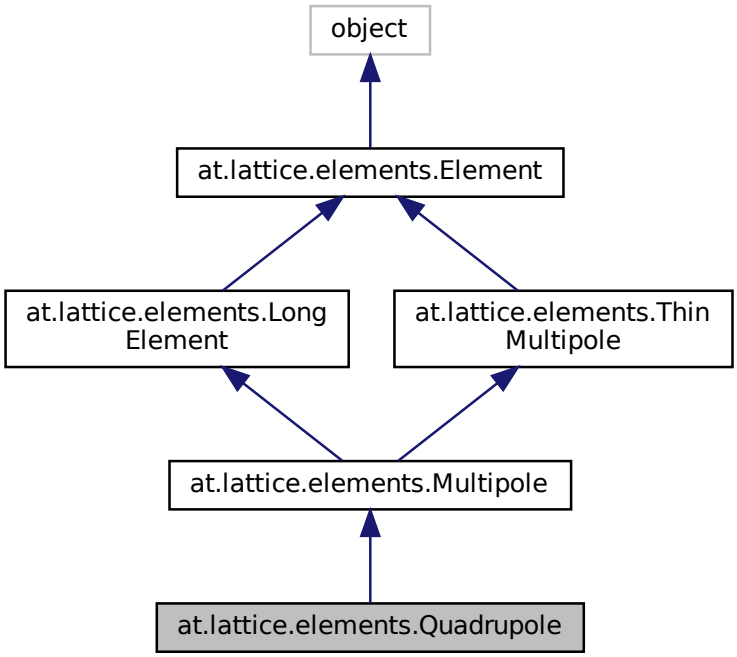
- at/lattice/particle_object.py

6.30 at.lattice.elements.Quadrupole Class Reference

Inheritance diagram for at.lattice.elements.Quadrupole:



Collaboration diagram for at.lattice.elements.Quadrupole:



Public Member Functions

- def __init__ (self, family_name, length, k=0.0, ∗∗kwargs)
- def **K** (self)
- def **K** (self, strength)

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = LongElement.REQUIRED_ATTRIBUTES + ['K']
- int **DefaultOrder** = 1

Additional Inherited Members

6.30.1 Detailed Description

```
pyAT quadrupole element
```

6.30.2 Constructor & Destructor Documentation

6.30.2.1 __init__()

```
def at.lattice.elements.Quadrupole.__init__ (
              self,
              family_name,
              length,
              k = 0.0,
             ** kwargs )
```

```
Quadrupole(FamName, Length, Strength=0, **keywords)

Available keywords:
PolynomB        straight multipoles
PolynomA        skew multipoles
MaxOrder        Number of desired multipoles
NumIntSteps     Number of integration steps (default: 10)
FringeQuadEntrance  0: no fringe fiels effect (default)
            1: Lee-Whiting's thin lens limit formula
            2: elegant-like
FringeQuadExit
fringeIntM0     Integrals for FringeQuad method 2
fringeIntP0
KickAngle       Correction deviation angles (H, V)
```

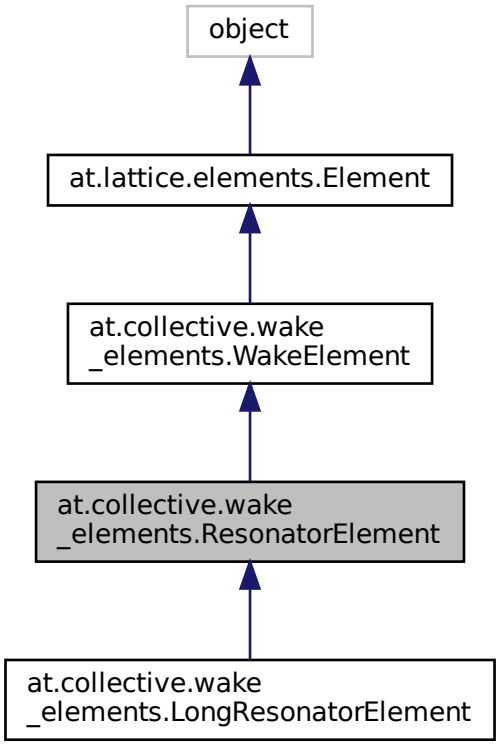
Reimplemented from at.lattice.elements.ThinMultipole.

The documentation for this class was generated from the following file:

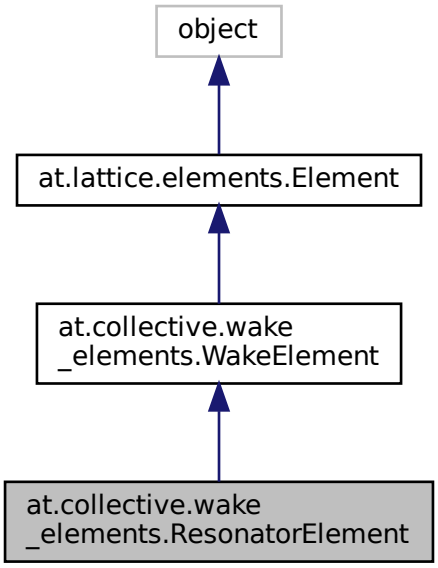
- at/lattice/elements.py

6.31 at.collective.wake_elements.ResonatorElement Class Reference

Inheritance diagram for at.collective.wake_elements.ResonatorElement:



Collaboration diagram for at.collective.wake_elements.ResonatorElement:



Public Member Functions

- def **__init__** (self, family_name, ring, srange, wakecomp, frequency, qfactor, rshunt, yokoya_factor=1, ∗∗kwargs)
- def **rebuild_wake** (self)
- def **ResFrequency** (self)
- def **ResFrequency** (self, frequency)
- def **Qfactor** (self)
- def **Qfactor** (self, qfactor)
- def **Rshunt** (self)
- def **Rshunt** (self, rshunt)
- def **Yokoya** (self)
- def **Yokoya** (self, yokoya)

Additional Inherited Members

6.31.1 Detailed Description

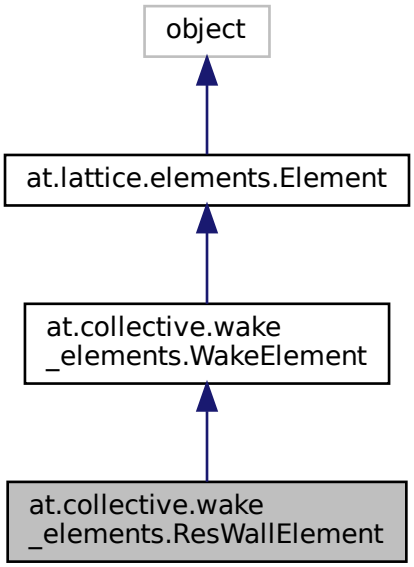
```
Class to generate a resonator, inherits from WakeElement
   additonal argument are frequency, qfactor, rshunt
```

The documentation for this class was generated from the following file:

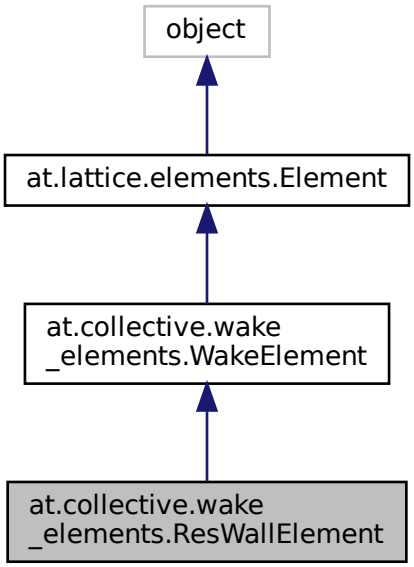
- at/collective/wake_elements.py

6.32 at.collective.wake_elements.ResWallElement Class Reference

Inheritance diagram for at.collective.wake_elements.ResWallElement:



Collaboration diagram for at.collective.wake_elements.ResWallElement:



Public Member Functions

- def **__init__** (self, family_name, ring, srange, wakecomp, rwlength, rvac, conduc, yokoya_factor=1, ∗∗kwargs)
- def **rebuild_wake** (self)
- def **RWLength** (self)
- def **RWLength** (self, length)
- def **Conductivity** (self)
- def **Conductivity** (self, conduct)
- def **Rvac** (self)
- def **Rvac** (self, rvac)
- def **Yokoya** (self)
- def **Yokoya** (self, yokoya)

Additional Inherited Members

6.32.1 Detailed Description

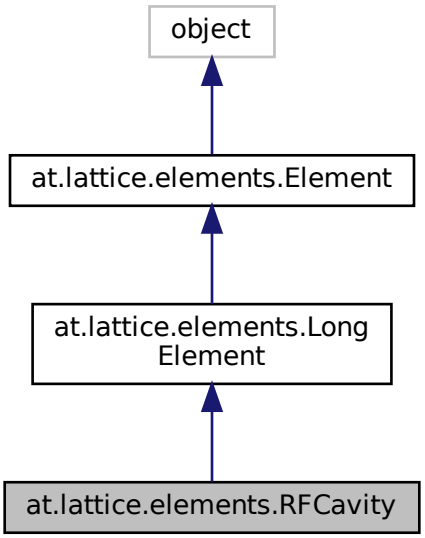
```
Class to generate a resistive wall element, inherits from WakeElement
   additional argument are yokoya_factor, length, pipe radius, conductivity
```

The documentation for this class was generated from the following file:

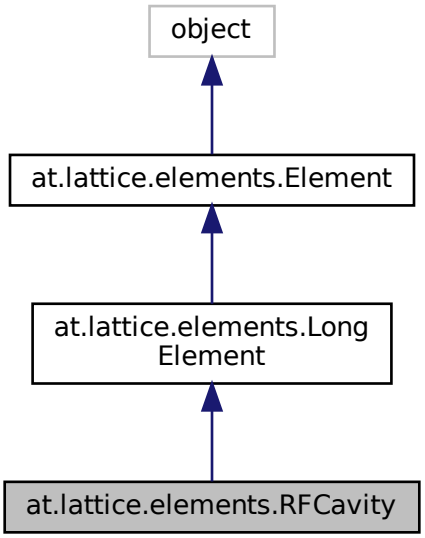
- at/collective/wake_elements.py

6.33 at.lattice.elements.RFCavity Class Reference

Inheritance diagram for at.lattice.elements.RFCavity:



Collaboration diagram for at.lattice.elements.RFCavity:



Public Member Functions

- def __init__ (self, family_name, length, voltage, frequency, harmonic_number, energy, ∗∗kwargs)

Static Public Attributes

- **REQUIRED_ATTRIBUTES**

Additional Inherited Members

6.33.1 Detailed Description

```
pyAT RF cavity element
```

6.33.2 Constructor & Destructor Documentation

6.33.2.1 __init__()

```
def at.lattice.elements.RFCavity.__init__ (
              self,
              family_name,
              length,
              voltage,
              frequency,
              harmonic_number,
              energy,
             ** kwargs )


Available keywords:
TimeLag   time lag with respect to the reference particle
```

6.33.3 Member Data Documentation

6.33.3.1 REQUIRED_ATTRIBUTES

```
at.lattice.elements.RFCavity.REQUIRED_ATTRIBUTES  [static]
```

Initial value:

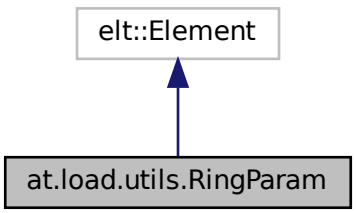
```
=  LongElement.REQUIRED_ATTRIBUTES + ['Voltage',
                                                    'Frequency',
                                                    'HarmNumber',
                                                    'Energy']
```

The documentation for this class was generated from the following file:

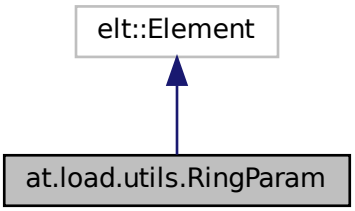
- at/lattice/elements.py

6.34 at.load.utils.RingParam Class Reference

Inheritance diagram for at.load.utils.RingParam:



Collaboration diagram for at.load.utils.RingParam:



Public Member Functions

- def **__init__** (self, family_name, energy, periodicity=1, ∗∗kwargs)

Static Public Attributes

- **REQUIRED_ATTRIBUTES**

6.34.1 Detailed Description

```
Private class for Matlab RingParam element
```

6.34.2 Member Data Documentation

6.34.2.1 REQUIRED_ATTRIBUTES

```
at.load.utils.RingParam.REQUIRED_ATTRIBUTES  [static]
```

Initial value:

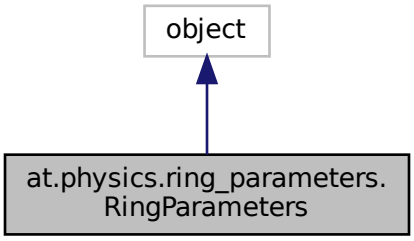
```
=  elt.Element.REQUIRED_ATTRIBUTES + ['Energy',
                                                    'Periodicity']
```

The documentation for this class was generated from the following file:

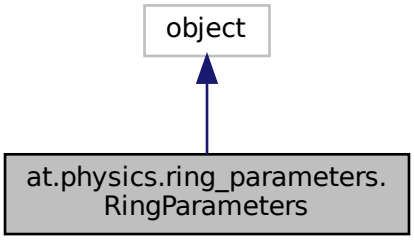
- at/load/utils.py

6.35 at.physics.ring_parameters.RingParameters Class Reference

Inheritance diagram for at.physics.ring_parameters.RingParameters:



Collaboration diagram for at.physics.ring_parameters.RingParameters:



Public Member Functions

- def __init__ (self, ∗∗kwargs)
- def **__str__** (self)

Static Public Attributes

- dictionary **props**

6.35.1 Detailed Description

```
Class for pretty printing the ring properties
```

6.35.2 Constructor & Destructor Documentation

6.35.2.1 __init__()

```
def at.physics.ring_parameters.RingParameters.__init__ (
              self,
             ** kwargs )
```

Initialisation

6.35.3 Member Data Documentation

6.35.3.1 props

dictionary at.physics.ring_parameters.RingParameters.props [static]

Initial value:

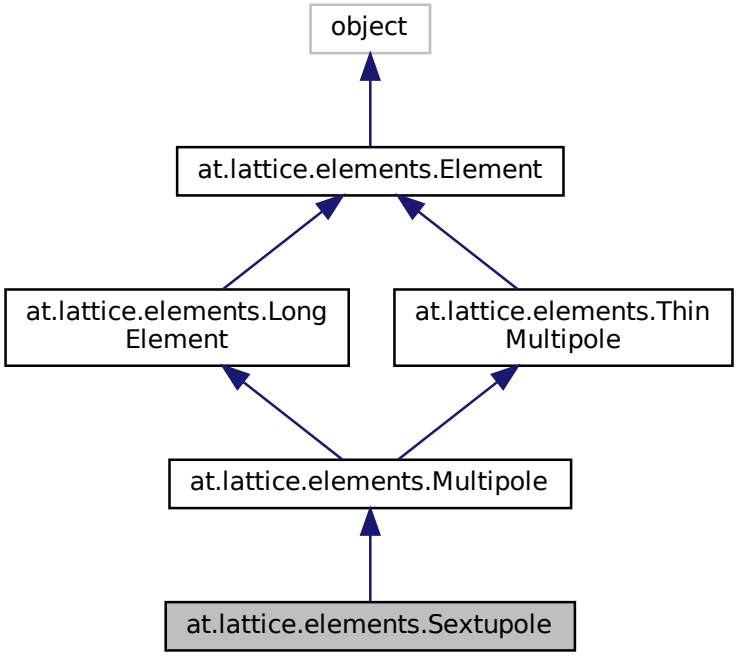
```
=  {
        'tunes':            '              Frac. tunes: {0}',
        'tunes6':           '  Frac. tunes (6D motion): {0}',
        'fulltunes':        '                    Tunes: {0}',
        'chromaticities':   '           Chromaticities: {0}',
        'alphac':           ' Momentum compact. factor: {0:e}',
        'etac':             '              Slip factor: {0:e}',
        'E0':               '                   Energy: {0:e} eV',
        'U0':               '       Energy loss / turn: {0:e} eV',
        'i1':               ' Radiation integrals - I1: {0} m',
        'i2':               '                      I2: {0} m^-1',
        'i3':               '                      I3: {0} m^-2',
        'i4':               '                      I4: {0} m^-1',
        'i5':               '                      I5: {0} m^-1',
        'emittances':       '          Mode emittances: {0}',
        'J':                'Damping partition numbers: {0}',
        'Tau':              '            Damping times: {0} s',
        'sigma_e':          '            Energy spread: {0:g}',
        'sigma_l':          '             Bunch length: {0:g} m',
        'voltage':          '         Cavities voltage: {0} V',
        'phi_s':            '        Synchrotron phase: {0:g} rd',
        'f_s':              '    Synchrotron frequency: {0:g} Hz'
    }
```

The documentation for this class was generated from the following file:

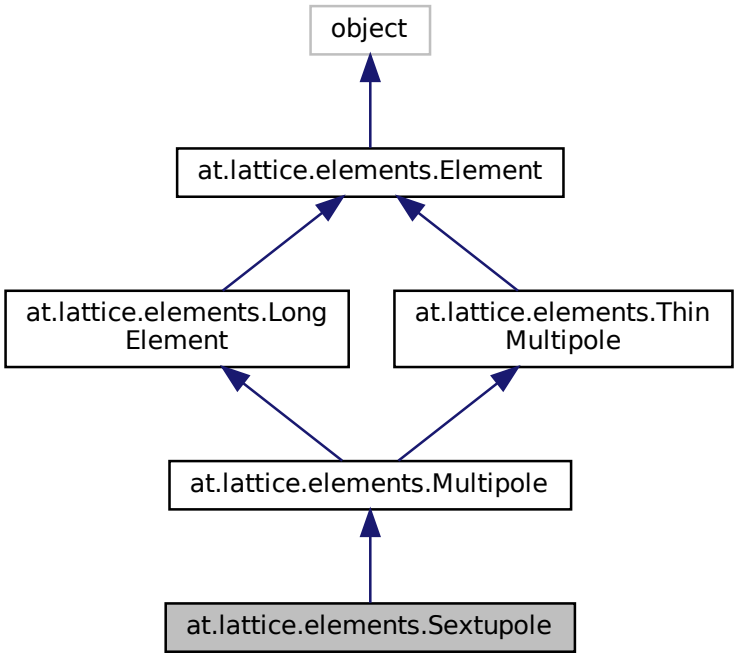
- at/physics/ring_parameters.py

6.36 at.lattice.elements.Sextupole Class Reference

Inheritance diagram for at.lattice.elements.Sextupole:



Collaboration diagram for at.lattice.elements.Sextupole:



Public Member Functions

- def __init__ (self, family_name, length, h=0.0, ∗∗kwargs)
- def **H** (self)
- def **H** (self, strength)

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = LongElement.REQUIRED_ATTRIBUTES + ['H']
- int **DefaultOrder** = 2

Additional Inherited Members

6.36.1 Detailed Description

```
pyAT sextupole element
```

6.36.2 Constructor & Destructor Documentation

6.36.2.1 __init__()

```
def at.lattice.elements.Sextupole.__init__ (
              self,
              family_name,
              length,
              h = 0.0,
             ** kwargs )

Sextupole(FamName, Length, Strength=0, **keywords)

Available keywords:
PolynomB        straight multipoles
PolynomA        skew multipoles
MaxOrder        Number of desired multipoles
NumIntSteps     Number of integration steps (default: 10)
KickAngle       Correction deviation angles (H, V)
```

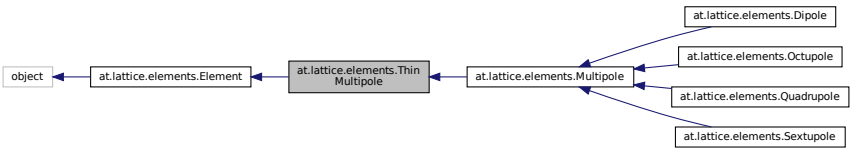
Reimplemented from at.lattice.elements.ThinMultipole.

The documentation for this class was generated from the following file:

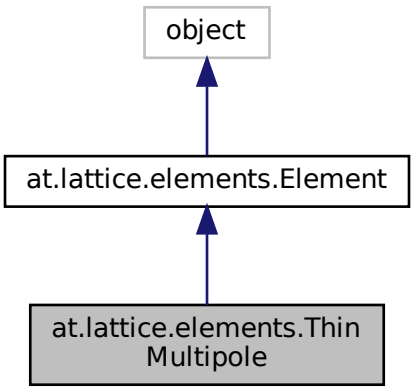
- at/lattice/elements.py

6.37 at.lattice.elements.ThinMultipole Class Reference

Inheritance diagram for at.lattice.elements.ThinMultipole:



Collaboration diagram for at.lattice.elements.ThinMultipole:



Public Member Functions

- def __init__ (self, family_name, poly_a, poly_b, ∗∗kwargs)
- def __setattr__ (self, key, value)

Public Attributes

- **PolynomA**
- **PolynomB**

Static Public Attributes

- **REQUIRED_ATTRIBUTES**

6.37.1 Detailed Description

```
pyAT thin multipole element
```

6.37.2 Constructor & Destructor Documentation

6.37.2.1 __init__()

```
def at.lattice.elements.ThinMultipole.__init__ (
              self,
              family_name,
              poly_a,
              poly_b,
             ** kwargs )
```

```
ThinMultipole(FamName, PolynomA, PolynomB, **keywords)

Available keywords:
MaxOrder        Number of desired multipoles. Default: highest index of
        non-zero polynomial coefficients
```

Reimplemented in at.lattice.elements.Quadrupole, and at.lattice.elements.Sextupole.

6.37.3 Member Function Documentation

6.37.3.1 __setattr__()

```
def at.lattice.elements.ThinMultipole.__setattr__ (
              self,
              key,
              value )
```

Check the compatibility of MaxOrder, PolynomA and PolynomB

Reimplemented from at.lattice.elements.Element.

6.37.4 Member Data Documentation

6.37.4.1 REQUIRED_ATTRIBUTES

```
at.lattice.elements.ThinMultipole.REQUIRED_ATTRIBUTES  [static]
```

Initial value:

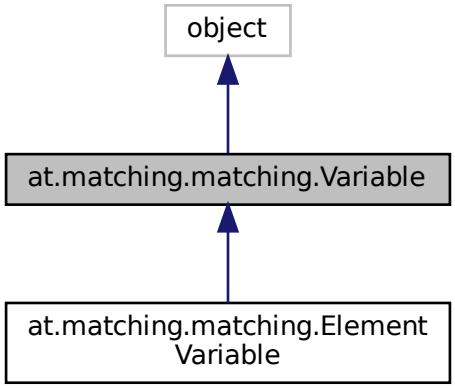
```
=  Element.REQUIRED_ATTRIBUTES + ['PolynomA',
                                                    'PolynomB']
```

The documentation for this class was generated from the following file:

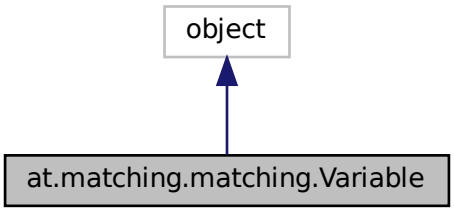
- at/lattice/elements.py

6.38 at.matching.matching.Variable Class Reference

Inheritance diagram for at.matching.matching.Variable:



Collaboration diagram for at.matching.matching.Variable:



Public Member Functions

- def **__init__** (self, setfun, getfun, name='', bounds=(-np.inf, np.inf), ∗args, ∗∗kwargs)
- def **set** (self, ring, value)
- def **get** (self, ring)
- def **status** (self, ring, vini=np.NaN)

Static Public Member Functions

- def **header** ()

Public Attributes

- **setfun**
- **getfun**
- **name**
- **bounds**
- **args**
- **kwargs**

6.38.1 Detailed Description

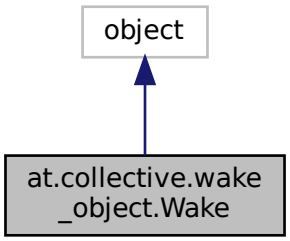
```
A Variable is a scalar value acting on a lattice through the
user-defined functions setfun and getfun
```

The documentation for this class was generated from the following file:

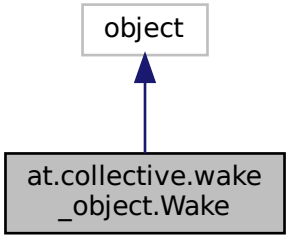
- at/matching/matching.py

6.39 at.collective.wake_object.Wake Class Reference

Inheritance diagram for at.collective.wake_object.Wake:



Collaboration diagram for at.collective.wake_object.Wake:



Public Member Functions

- def **__init__** (self, srange)
- def **srange** (self)
- def **DX** (self)
- def **DY** (self)
- def **QX** (self)
- def **QY** (self)
- def **Z** (self)
- def **add** (self, wtype, wcomp, ∗args, ∗∗kwargs)

Static Public Member Functions

- def resonator (srange, wakecomp, frequency, qfactor, rshunt, beta, yokoya_factor=1, nelems=1)
- def long_resonator (srange, frequency, qfactor, rshunt, beta, nelems=1)
- def resistive_wall (srange, wakecomp, length, rvac, conduct, beta, yokoya_factor=1, nelems=1)
- def build_srange (start, bunch_ext, short_step, long_step, bunch_interval, totallength)

Public Attributes

- **components**

6.39.1 Detailed Description

```
Class to generate a wake object
The wake object is define by its srange, specified
at initialization, and DX, DY, QY, Z corresponding
to transverse dipoles and quadrupoles and longitudinal

The srange is common to all components and cannot be changed
once initialized, all added component are resampled to the
srange

usage:
wake = Wake(srange)
wake.add(WakeType,WakeComponent, *args, *kwargs)

Component are WakeComponent.FILE (import from file),
WakeComponent.TABLE (provide vectors), WakeComponent.RESONATOR
(analytical resonator), WakeComponent.RESWALL (transverse RW)

Components are retrieved with Wake.DX for example
```

6.39.2 Member Function Documentation

6.39.2.1 build_srange()

```
def at.collective.wake_object.Wake.build_srange (
              start,
              bunch_ext,
              short_step,
              long_step,
              bunch_interval,
              totallength )  [static]
```

```
Function to build the wake table s column.
This is not the slicing but the look-up table,
however it generates data where bunches are located
to avoid using too much memory to store the table.

PARAMETERS
    start           starting s-coordinate of the table
            (can be negative for wake potential)
    bunch_ext       maximum bunch extension, function
            generates data at +/- bunch_ext
            around the bucket center
    short_step      step size for the short range wake table
    long_step       step size for the long range wake table
    bunch_interval  minimum bunch interval data will be generate
            for each bunch_inteval step
    totallength     total length of the wake table, has to contain
            the full bunch extension

OUTPUT
    srange          vector of s position where to sample the wake
```

6.39.2.2 long_resonator()

```
def at.collective.wake_object.Wake.long_resonator (
              srange,
              frequency,
              qfactor,
              rshunt,
              beta,
              nelems = 1 )  [static]
```

Method to build a longitudinal resonator wake object

6.39.2.3 resistive_wall()

```
def at.collective.wake_object.Wake.resistive_wall (
              srange,
              wakecomp,
              length,
              rvac,
              conduct,
              beta,
              yokoya_factor = 1,
              nelems = 1 )  [static]
```

Method to build a resistive wall wake object

6.39.2.4 resonator()

```
def at.collective.wake_object.Wake.resonator (
              srange,
              wakecomp,
              frequency,
              qfactor,
              rshunt,
              beta,
              yokoya_factor = 1,
              nelems = 1 )  [static]
```

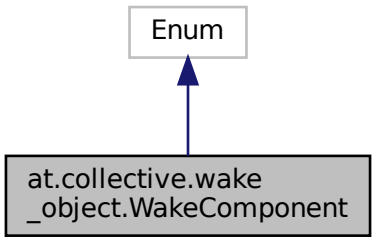
Method to build a resonator wake object

The documentation for this class was generated from the following file:

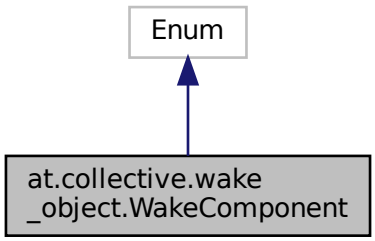
- at/collective/wake_object.py

6.40 at.collective.wake_object.WakeComponent Class Reference

Inheritance diagram for at.collective.wake_object.WakeComponent:



Collaboration diagram for at.collective.wake_object.WakeComponent:



Static Public Attributes

- int **DX** = 1
- int **DY** = 2
- int **QX** = 3
- int **QY** = 4
- int **Z** = 5

6.40.1 Detailed Description

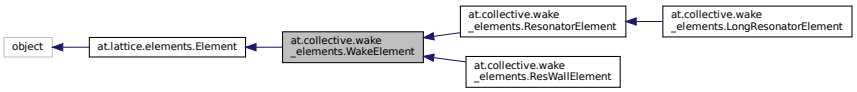
```
Enum class for wake component
```

The documentation for this class was generated from the following file:

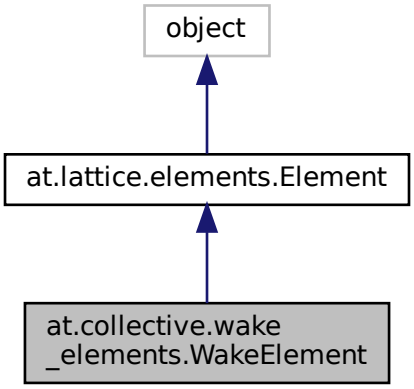
- at/collective/wake_object.py

6.41 at.collective.wake_elements.WakeElement Class Reference

Inheritance diagram for at.collective.wake_elements.WakeElement:



Collaboration diagram for at.collective.wake_elements.WakeElement:



Public Member Functions

- def **__init__** (self, family_name, ring, wake, ∗∗kwargs)
- def **rebuild_wake** (self, wake)
- def **clear_history** (self)
- def **set_normfactxy** (self, ring)
- def **WakeT** (self)
- def **WakeZ** (self)
- def **WakeDX** (self)
- def **WakeDY** (self)
- def **WakeQX** (self)
- def **WakeQY** (self)
- def **Nslice** (self)
- def **Nslice** (self, nslice)
- def **Nturns** (self)
- def **Nturns** (self, nslice)
- def **Current** (self)
- def **Current** (self, current)
- def __repr__ (self)

Public Attributes

- **NumParticles**
- **NormFact**
- **ZCuts**

Static Public Attributes

- **REQUIRED_ATTRIBUTES** = Element.REQUIRED_ATTRIBUTES

6.41.1 Detailed Description

```
Class to generate an AT wake element using the passmethod WakeFieldPass
args:  family name, ring, wake object
kwargs: PassMethod=WakeFieldPass
        Current=0   Bunch current [A]
        Nslice=101  Number of slices per bunch
        Nturns=1    Number of turn for the wake field
        ZCuts=None  Limits for fixed slicing, default is adaptive
        NormFact    (default=[1,1,1]) normalization for the 3 planes,
                    to account for beta function at the observation
                    point for example
```

6.41.2 Member Function Documentation

6.41.2.1 __repr__()

```
def at.collective.wake_elements.WakeElement.__repr__ (
              self )
```

```
Simplified __repr__ to avoid errors due to arguments
not defined as attributes
```

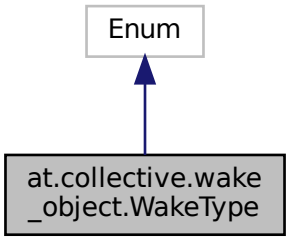
Reimplemented from at.lattice.elements.Element.

The documentation for this class was generated from the following file:

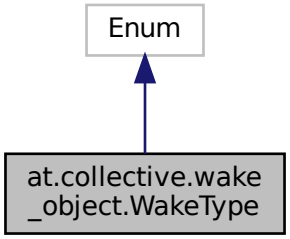
- at/collective/wake_elements.py

6.42 at.collective.wake_object.WakeType Class Reference

Inheritance diagram for at.collective.wake_object.WakeType:



Collaboration diagram for at.collective.wake_object.WakeType:



Static Public Attributes

- int **FILE** = 1
- int **TABLE** = 2
- int **RESONATOR** = 3
- int **RESWALL** = 4

6.42.1 Detailed Description

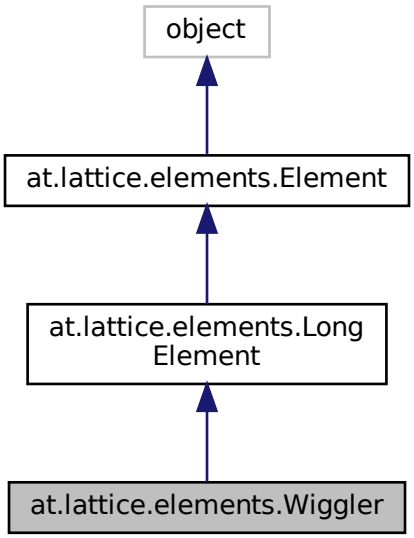
```
Enum class for wake type
```

The documentation for this class was generated from the following file:

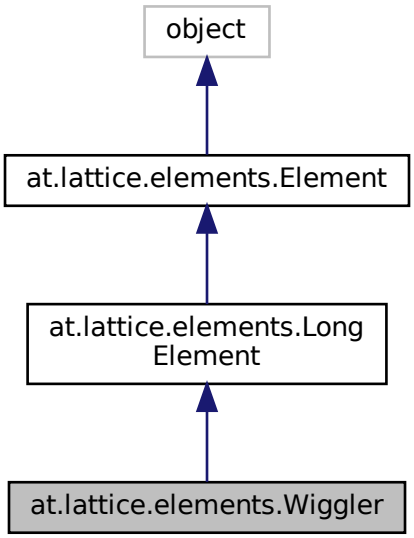
- at/collective/wake_object.py

6.43 at.lattice.elements.Wiggler Class Reference

Inheritance diagram for at.lattice.elements.Wiggler:



Collaboration diagram for at.lattice.elements.Wiggler:



Public Member Functions

- def __init__ (self, family_name, length, wiggle_period, b_max, energy, Nstep=5, Nmeth=4, By=(1, 1, 0, 1, 1, 0), Bx=(), **kwargs)

Public Attributes

- **NHharm**
- **NVharm**

Static Public Attributes

- **REQUIRED_ATTRIBUTES**

6.43.1 Detailed Description

```
pyAT wiggler element

See atwiggler.m
```

6.43.2 Constructor & Destructor Documentation

6.43.2.1 __init__()

```
def at.lattice.elements.Wiggler.__init__ (
              self,
              family_name,
              length,
              wiggle_period,
              b_max,
              energy,
              Nstep = 5,
              Nmeth = 4,
              By = (1, 1, 0, 1, 1, 0),
              Bx = (),
             ** kwargs )


Args:
    length: total length of the wiggler
    wiggle_period: length must be a multiple of this
    b_max: peak wiggler field [Tesla]
    energy: beam energy [eV]

Available keywords:
    Nstep: number of integration steps.
    Nmeth: symplectic integration order: 2 or 4
    Bx: harmonics for horizontal wiggler: (6,nHharm) array-like object
    By: harmonics for vertical wiggler (6,nHharm) array-like object
```

6.43.3 Member Data Documentation

6.43.3.1 REQUIRED_ATTRIBUTES

at.lattice.elements.Wiggler.REQUIRED_ATTRIBUTES [static]

Initial value:

```
=  LongElement.REQUIRED_ATTRIBUTES + ['Lw', 'Bmax',
                                                    'Energy']
```

The documentation for this class was generated from the following file:

- at/lattice/elements.py

Index