# COBEA Manual

***Release 0.23***

**Bernard Riemann**

**Nov 13, 2017**

# CONTENTS

The *cobea* module [1] is a Python implementation of Closed-Orbit Bilinear-Exponential Analysis [2], an algorithm for studying closed-orbit response matrices of storage rings (particle accelerators).

If you publish material using this software, please cite one or more of the references [1-2].

For more information, visit https://bitbucket.org/b-riemann/cobea .

(References)

[1] B. Riemann et al., ''COBEA - Optical Parameters From Response Matrices Without Knowledge of Magnet Strengths'', in Proc. IPAC17, paper MOPIK066, 2017.

[2] B. Riemann, ''[The Bilinear-Exponential Closed-Orbit Model and its Application to Storage Ring Beam Diagnostics](http://dx.doi.org/10.17877/DE290R-17221)'', Ph.D. Dissertation, TU Dortmund University, 2016. DOI [10.17877/DE290R-17221](http://dx.doi.org/10.17877/DE290R-17221).

CHAPTER

ONE

# QUICK START TUTORIAL

This is a brief introduction on how to use the cobea module.

## 1.1 Reading in data

COBEAs input consists of

- A set of corrector names corresponding to rows of your matrix
- A set of monitor names corresponding to columns of your matrix
- The response matrix itself
- The ordering information, given as a list of monitor and corrector names, ordered along the beampath (downstream)

You need to convert your response data into a standardized input for COBEA. This is handled by the *Response* class:

**class** `cobea.model.`**`Response`**(*matrix*, *corr_names*, *mon_names*, *line*, *include_dispersion=True*, *unit='a.u./a.u.'*, *drift_space=None*, *corr_filters=()*, *name=''*, *assume_sorted=False*)

Representation of COBEA input, used as such for the function `cobea.cobea()`

During creation of the this object, py:data:*matrix* rows and columns, as well as the corresponding py:data:*corr_names* and py:data:*mon_names*, are resorted to their respective order in py:data:*line*.

**Parameters**

- **`matrix`** (*`array`*) – input response matrix of shape (correctors, monitors, directions). If only one direction is considered, the last dimension can be omitted.
- **`corr_names`** (*`list`*) – a list of corrector labels corresponding to each row of the matrix. See also corr_filters.
- **`mon_names`** (*`list`*) – a list of monitor labels corresponding to each column of the matrix
- **`line`** (*`list`*) – a list of element names in ascending s order
- **`include_dispersion`** (*`bool`*) – whether to use a model with or without dispersion for fitting. default: True
- **`unit`** (*`str`*) – (optional, default: '') unit for the input values of the matrix, containing a '/' character to separate monitor unit (e.g. mm) and corrector unit (e.g. A or mrad)
- **`drift_space`** (*`iterable`*) – (optional, default: None) a tuple or list with 3 elements (monitor name 1, monitor name 2, drift space length / m)
- **`corr_filters`** (*`list`*) – (optional) a list of filter strings with special character . *Example: To create one corrector set for all correctors with names starting with Cx, and another ending with dy, enter ('Cx','*dy')

- **name** (*str*) – (optional, default: '') a short description of the measured response
- **assume_sorted** (*str*) – cobea (especially cobea.mcs) requires rows and columns of the response to be sorted along the beam path. Only change this if you know what you are doing!

**topology**
*object* – A `Topology` object holding the re-ordered py:data:'corr_names', py:data:'mon_names', and py:data:'line' as attributes.

**input_matrix**
*array* – re-ordered input response matrix.

**mon_unit**
*str* – unit for BPM readings (extracted from unit argument)

**corr_unit**
*str* – unit for corrector strengths (extracted from unit argument)

**known_element**
*object* – Known-element object used for normalization. At the moment, only `DriftSpace` can be used.

## 1.2 Running the algorithm

The COBEA algorithm is then applied using the function `cobea.cobea()` (click link, return back to 1.3 afterwards)

## 1.3 Obtaining and plotting results

The *cobea* function returns a *Result* object. This object contains all computed information.

**class** `cobea.model.`**`Result`**(*response*, *additional={}*, ***kwargs*)
A container for all COBEA results that also computes secondary outputs on demand.

**input_matrix**
*array* – Original input response matrix

**error**
*object* – computed BE model errors, represented as `ErrorModel` object

**known_element**
*object* – Known-element object used for normalization. At the moment, only `DriftSpace` can be used.

**additional**
*dict* – may contain the following keywords

**coretime** [float] time used for computation in the start and optimization layer.

**err** [dict] dictionary with additional model parameter error estimates.

**conv** [dict] dictionary with L-BFGS convergence information (if convergence_info was True)

**invariants** [array] computed during normalization of monitor vectors if drift_space is given. These are just returned for completeness and do not contain information about beam physics.

**pca_singvals** [array] custom info from MCS algorithm

**pca_orbits** [array] custom info from MCS algorithm

**version** [str] version of the object

**cbeta_jmw**
Ripken-Mais beta parameters * constant. If self.R_jmw is normalized, constant = 1.

**cbeta_km**
const*beta at correctors assuming decoupled optics and thin correctors

**delphi_jmw**
Ripken-Mais phase advances per element

**delphi_km**
Betatron phase advances per corrector assuming decoupled optics and thin correctors

**flip_mu** (*m*)
switch the sign of mu_m for given m, simultaneously changing the conjugation of `R_jmw` and `A_km` so that the response matrix remains unchanged

**phase_integral** (*m*)
integrated phase from first to last BPM (not one turn!), used for `tune()` computation

**phi_jmw**
Compute Ripken-Mais betatron phases in units of degrees

**response_matrix** ()
generate a 'simulated' response matrix from the present model parameters

**Returns** **rsim_kjw** – response array of shape (`K`, `J`, `M`)

**Return type** array

**save** (*filename*)
save the Result object as a pickle file with the given filename. The object can be reloaded using *`cobea.load_result()`* (which simply uses pickle)

**tune** (*m*)
compute tune including integer part for a given mode m

**update_errors** ()
compute errors in attribute *`error`* for given BE-Model parameters and input response, including errors for Ripken-Mais parameters

The module *`cobea.plotting`* includes many helper functions to view these results. A summary of results is created by the *`cobea.plotting.plot_result()`* function.

CHAPTER

TWO

# API

This page contains automatic documentation of the complete cobea module.

## 2.1 cobea (main namespace)

Closed-Orbit Bilinear-Exponential Analysis (COBEA)

This is a Python implementation of the COBEA algorithm [1] to be used for studying betatron oscillations in particle accelerators by closed-orbit information.

**[1] B. Riemann. ''The Bilinear-Exponential Model and its Application to Storage Ring Beam Diagnostics'',**
PhD Dissertation (TU Dortmund University, 2016), DOI Link: (https://dx.doi.org/10.17877/DE290R-17221)

Bernard Riemann (bernard.riemann@tu-dortmund.de)

`cobea.`**`cobea`**(*response*, *convergence_info=False*)
Main COBEA function with pre- and postprocessing.

**Parameters**

- **`response`** (*`object`*) – A valid *`cobea.model.Response`* object representing the input.
- **`convergence_info`** (*`bool`*) – if True, convergence information from L-BFGS is added to the result dictionary (before saving).

**Returns result** – A *`cobea.model.Result`* object.

**Return type** object

`cobea.`**`load_result`**(*filename*)
Load (un-pickle) a Result object (or any other object)

`cobea.`**`optimization_layer`**(*result*, *iprint=-1*)
Implementation of the Optimization layer. It uses L-BFGS [1] as special case of L-BFGS-B [2] in scipy.optimize. The result object is modified to yield the optimal BEModel. A sub-dictionary with additional information is added under the key result.additional['Opt'].

[1] D.C. Liu and J. Nocedal, ''On the Limited Memory Method for Large Scale Optimization'', Math. Prog. B 45 (3), pp.~503–528, 1989. DOI 10.1007/BF01589116

[2] C. Zhu, R.H. Byrd and J. Nocedal, ''Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization'', ACM Trans. Math. Software 23 (4), pp.~550–560, 1997. DOI 10.1145/279232.279236

**Parameters** **`result`** (*`object`*) – A valid *`cobea.model.Result`* object. The object is modified during processing; the model variables are set to their optimal values.

**Returns result** – Identical to input object.

**Return type** object

`cobea.`**`read_elemnames`**(*fname*)

A helper function to read element names from text files into a list of strings. Standard input is a text file with linebreaks between elements.

## 2.2 cobea.model: COBEA classes and objects

This COBEA submodule defines all classes used by *`cobea`*. Besides input (*`Response`*) and output (*`Result`*) containers, this also includes gradient-based optimization procedures in `BE_Model`.

Bernard Riemann (bernard.riemann@tu-dortmund.de)

**class** `cobea.model.`**`BEModel`**(*K*, *J*, *M*, *topology*, *include_dispersion*, *init_fun=<built-in function empty>*)

Bases: *`cobea.model.BasicModel`*

Bilinear-Exponential model with topology information and optimization routines. Besides the attributes and methods contained in `Bare_Model`, the following information is included.

**Parameters**

- **`K,J,M`** (*`int`*) – dimensions of the model, with K being the number of correctors, J being the number of monitors, and M the number of modes respectively directions.
- **`init_fun`** (*`function`*) – a (possibly self-defined) initialization function like `zeros()` or `empty()` from numpy.

**`topology`**

*object* – input topology, represented as *`Topology`* object

**`cbeta_jmw`**

Ripken-Mais beta parameters * constant. If self.R_jmw is normalized, constant = 1.

**`cbeta_km`**

const*beta at correctors assuming decoupled optics and thin correctors

**`delphi_jmw`**

Ripken-Mais phase advances per element

**`delphi_km`**

Betatron phase advances per corrector assuming decoupled optics and thin correctors

**`flip_mu`**(*m*)

switch the sign of mu_m for given m, simultaneously changing the conjugation of `R_jmw` and `A_km` so that the response matrix remains unchanged

**`phase_integral`**(*m*)

integrated phase from first to last BPM (not one turn!), used for *`tune()`* computation

**`phi_jmw`**

Compute Ripken-Mais betatron phases in units of degrees

**`response_matrix`**()

generate a 'simulated' response matrix from the present model parameters

**Returns** **rsim_kjw** – response array of shape (`K`, `J`, `M`)

**Return type** array

**`tune`**(*m*)

compute tune including integer part for a given mode m

**class** `cobea.model.`**`BasicModel`**(*K*, *J*, *M*, *include_dispersion*, *init_fun=<built-in function empty>*)

simple representation of the Bilinear-Exponential model (without topology or optimization attributes).

**Parameters**

- **K,J,M** (*int*) – dimensions of the model, with K being the number of correctors, J being the number of monitors, and M the number of modes respectively directions.
- **init_fun** (*function*) – a (possibly self-defined) initialization function like `zeros()` or `empty()` from numpy.

**K**
*int* – total number of correctors. defines limit of corrector index k.

**J**
*int* – total number of monitors. defines limit of monitor index j.

**M**
*int* – number of directions respectively modes. defines limits of mode index m and direction index w.

**R_jmw**
*array* – monitor vectors in format [monitor, mode, direction]

**A_km**
*array* – corrector parameters, format [corrector, mode]

**d_jw**
*array* – unnormalized dispersion function at monitors, format [monitor, direction]

**b_k**
*array* – unnormalized dispersion coefficients at correctors, format [corrector]

**mu_m**
*array* – fractional phase advances per turn (in rad)

A reduced model class without topology or gradient computation

**class** `cobea.model.`**DriftSpace**(*mon_names*, *length*)
Representation of drift space information

**Parameters**

- **mon_names** (*list*) – list of monitor labels for the drift space
- **length** (*float*) – length of drift space

**r_prime_upstream**(*rj_drift*, *rj_drift_err=None*)
Spatial derivative of spatial vector at the beginning of the drift space given spatial vectors at its ends.

**Returns**

- **r_prime** (*array_like*) – spatial derivative
- **r_prime_err** (*array_like*) – error of spatial derivative for given rj_drift_err, else None

**class** `cobea.model.`**ErrorModel**(*K*, *J*, *M*, *include_dispersion*, *chi_squared*, *input_rms*, *unit*)
Bases: `cobea.model.BasicModel`

A class computing and storing all BE model errors, including additional attributes for Ripken-Mais errors.

**class** `cobea.model.`**Response**(*matrix*, *corr_names*, *mon_names*, *line*, *include_dispersion=True*, *unit='a.u./a.u.'*, *drift_space=None*, *corr_filters=()*, *name=''*, *assume_sorted=False*)
Representation of COBEA input, used as such for the function `cobea.cobea()`

During creation of the this object, py:data:*matrix* rows and columns, as well as the corresponding py:data:*corr_names* and py:data:*mon_names*, are resorted to their respective order in py:data:*line*.

**Parameters**

- **matrix** (*array*) – input response matrix of shape (correctors, monitors, directions). If only one direction is considered, the last dimension can be omitted.
- **corr_names** (*list*) – a list of corrector labels corresponding to each row of the matrix. See also corr_filters.

- **mon_names** (*list*) – a list of monitor labels corresponding to each column of the matrix
- **line** (*list*) – a list of element names in ascending s order
- **include_dispersion** (*bool*) – whether to use a model with or without dispersion for fitting. default: True
- **unit** (*str*) – (optional, default: '') unit for the input values of the matrix, containing a '/' character to separate monitor unit (e.g. mm) and corrector unit (e.g. A or mrad)
- **drift_space** (*iterable*) – (optional, default: None) a tuple or list with 3 elements (monitor name 1, monitor name 2, drift space length / m)
- **corr_filters** (*list*) – (optional) a list of filter strings with special character . *Example: To create one corrector set for all correctors with names starting with Cx, and another ending with dy, enter ('Cx*','*dy')*
- **name** (*str*) – (optional, default: '') a short description of the measured response
- **assume_sorted** (*str*) – cobea (especially cobea.mcs) requires rows and columns of the response to be sorted along the beam path. Only change this if you know what you are doing!

**topology**
*object* – A `Topology` object holding the re-ordered py:data:'corr_names', py:data:'mon_names', and py:data:'line' as attributes.

**input_matrix**
*array* – re-ordered input response matrix.

**mon_unit**
*str* – unit for BPM readings (extracted from unit argument)

**corr_unit**
*str* – unit for corrector strengths (extracted from unit argument)

**known_element**
*object* – Known-element object used for normalization. At the moment, only `DriftSpace` can be used.

**save** (*filename*)
save the Response object as a pickle file with the given filename. The object can be reloaded using `cobea.load_result()` (which simply uses pickle)

**class** cobea.model.**Result** (*response*, *additional={}*, ***kwargs*)
Bases: `cobea.model.BEModel`

A container for all COBEA results that also computes secondary outputs on demand.

**input_matrix**
*array* – Original input response matrix

**error**
*object* – computed BE model errors, represented as `ErrorModel` object

**known_element**
*object* – Known-element object used for normalization. At the moment, only `DriftSpace` can be used.

**additional**
*dict* – may contain the following keywords

**coretime** [float] time used for computation in the start and optimization layer.

**err** [dict] dictionary with additional model parameter error estimates.

**conv** [dict] dictionary with L-BFGS convergence information (if convergence_info was True)

**invariants** [array] computed during normalization of monitor vectors if drift_space is given. These are just returned for completeness and do not contain information about beam physics.

**pca_singvals** [array] custom info from MCS algorithm

**pca_orbits** [array] custom info from MCS algorithm

**version** [str] version of the object

**save** (*filename*)
save the Result object as a pickle file with the given filename. The object can be reloaded using `cobea.load_result()` (which simply uses pickle)

**update_errors** ()
compute errors in attribute `error` for given BE-Model parameters and input response, including errors for Ripken-Mais parameters

**class** cobea.model.**Topology** (*corr_names*, *mon_names*, *line*, *corr_filters=()*, *assume_sorted=False*)
Representation of corrector/monitor labels and the order between them along the ring. During creation, all columns and rows of the input matrix, together with their labels in corr_names, mon_names, are re-ordered in ascending s-position according to the line list.

**Parameters**

- **corr_names** (*list*) – corrector labels (strings), e.g. ['HK01', 'VCM1', 'special_Hcorr', ...]. The list index should correspond to the monitor_index, e.g. matrix[1,:,:] holds all information for the corrector named 'VCM1' in the above example.
- **mon_names** (*list*) – monitor labels (strings), e.g. ['BPM1','BPM2a','buggy_BPM',...,'important-bpm42']. the list index should correspond to the monitor_index, e.g. matrix[:,0,:] holds all information for the monitor named 'BPM1' in the above example.
- **line** (*list*) – corrector and monitor labels in ascending s position, downstream of the storage ring.
- **corr_filters** (*list*) – (optional) a list of filter strings with special character . *Example: To create one corrector set for all correctors with names starting with Cx, and another ending with dy, enter ('Cx*',*'dy')
- **assume_sorted** (*str*) – cobea (especially cobea.mcs) requires rows and columns of the response to be sorted along the beam path. Only change this if you know what you are doing!

**corrector_index** (*corrector_label*)
find the index/indices of corrector label(s) in self.monitor_names

**monitor_index** (*monitor_label*)
find the index/indices of monitor label(s) in self.monitor_names

## 2.3 cobea.mcs: Monitor-Corrector Subspace algorithm

Monitor-Corrector Subset (MCS) algorithm submodule

MCS can be used as start-value layer of COBEA.

cobea.mcs.**complexsolv** (*realvec*, *mat*)
solve the half-complex equation system realvec = real(compmat*conj(compsol)) for compsol. :returns: compsol, res (from lstsq),

realvec_rc: reconstructed realvec from compsol, s: singular values of compmat (from lstsq)

cobea.mcs.**composite_vectors** (*pcaDev*)
make two-orbit vectors (similar to phase space vectors) at beginning and end of partial orbits

`cobea.mcs.`**`corrector_matrix_k`**(*R*, *cE*)
output the complex corrector equation system matrix corrmat for a given corrector. corrmat.shape = [input_bpm*Directions+direction,mode] R: full input monitor array, R.shape = [input_bpm,mode,direction] cE: conj(E[:,k,:]) of Ejkm

`cobea.mcs.`**`corrector_systems`**(*Dev*, *monvec*, *bpm_s*, *corr_s*, *mus*, *printmsg=True*, *E=None*)
set up and solve the corrector equation systems. Dev[k,f,d]: Deviations at all correctors for fast BPMs. monvec: all input monitor vectors. returns: D[k,m]: corrector parameters complexsolv parameters as arrays

`cobea.mcs.`**`dice_splitpoints`**(*n*, *mon_idx*, *split_idx*)
Map the linear index n to bpm quadruplet index split_idx. As numpy arrays are passed by reference, split_idx is overwritten by this function.

`cobea.mcs.`**`flatten_Dev`**(*Dev*)
Index transform of Deviation matrix (k,j,w) to PCA processing matrix (k,j*w)

`cobea.mcs.`**`layer`**(*response*, *trials=-1*)
implementation of the Monitor-Corrector Subspace algorithm

**Parameters**

- **`result`** (*`object`*) – A valid *`cobea.model.Response`* object.
- **`trials`** (*`int`*) – Number of different monitor subsets tried for MCS. If set to -1, value is set automatically.

`cobea.mcs.`**`mcs_core`**(*result*, *mon_idx*, *cor_idx*, *split_idx*)
MCS routine for a given monitor quadruplet.

**Parameters**

- **`result`** (*`object`*) – A valid *`cobea.model.Result`* object.
- **`mon_idx`** (*`array`*) – 1d array of integer positions of all considered monitors in result.line
- **`cor_idx`** (*`array`*) – 1d array ... considered correctors in result.line
- **`split_idx`** (*`array_like`*) – a 2x2 array of monitor indices for the monitor quadruplet.

**Returns** **output** – ToDo for documentation

**Return type** list

`cobea.mcs.`**`monitor_matrix_j`**(*Y*, *E*)
output the complex monitor equation system matrix monmat for a given monitor AND direction. monmat.shape = [corrector,mode] Y: corrector parameters, Y.shape = [corrector,mode] E: E[j,:,:] of Ejkm

`cobea.mcs.`**`monitor_systems`**(*Dev*, *D*, *all_bpm_s*, *corr_s*, *mus*, *printmsg=True*, *E=[]*)
set up and solve the monitor equation systems, return R[j,m,d], the full monitor vector set for all monitors. Dev[k,j,d]: Deviations at all correctors for all BPMs. D[k,m]: all corrector parameters.

`cobea.mcs.`**`pca_core`**(*Dev*, *principal_orbits=True*)
Principal Component Analysis of a Deviation matrix.

`cobea.mcs.`**`unflatten_Dev`**(*pcaproc*, *Devshp*)
Index transform of PCA processing matrix (k,j*w) to Deviation matrix (k,j,w)

## 2.4 cobea.pproc: Standalone postprocessing functions

Small postprocessing and helper functions for COBEA results.

Bernard Riemann (bernard.riemann@tu-dortmund.de)

`cobea.pproc.`**`guess_mu_sign`**(*rslt*)
for weakly coupled setups, guess the sign of mu (quadrant) based on monitor phase advance

`cobea.pproc.`**`invariants_from_eigenvectors`**(*Q*)

Compute invariant of motion from a phase space eigenvector

**Parameters** **`Q`** (*`array`*) – phase space eigenvector with shape (1,2*M)

**Returns** **invariant** – invariants of motion

**Return type** float

`cobea.pproc.`**`invariants_of_motion`**(*R_drift*, *length*)

Compute invariant of motion from the eigenorbits around a known drift space

**Parameters**

- **`R_drift`** (*`array`*) – An array of two spatial vectors R_drift[0] and R_drift[1]
- **`length`** (*`float`*) – length of the drift space

**Returns** **invariant** – invariants of motion

**Return type** float

`cobea.pproc.`**`l_bfgs_iterate`**(*alloc_items=10000*)

convert the iterate.dat file produced by L-BFGS-B

**Parameters** **`alloc_items`** (*`int`*) – the number of maximum iterations for which memory is allocated.

**Returns**

**iter** –

**a dictionary with the following fields. The field names and descriptions have been copied from a demo outpu**

**'it'** [array] iteration number

**'nf'** [array] number of function evaluations

**'nseg'** [array] number of segments explored during the Cauchy search

**'nact'** [array] number of active bounds at the generalized Cauchy point

**'sub'** [str]

**manner in which the subspace minimization terminated** con = converged, bnd = a bound was reached

**'itls'** [int] number of iterations performed in the line search

**'stepl'** [float] step length used

**'tstep'** [float] norm of the displacement (total step)

**'projg'** [float] norm of the projected gradient

**'f'** [float] function value

**Return type** dict

`cobea.pproc.`**`layer`**(*result*, *convergence_info=False*)

Postprocessing layer

**Parameters**

- **`result`** (*`object`*) – A `cobea.model.Result` object. The object is modified during processing.
- **`convergence_info`** (*`bool`*) – if True, convergence information from L-BFGS is added to the result dictionary (before saving).

`cobea.pproc.`**`normalize_using_drift`**(*model*, *di*, *drift_length*)

Invariant postprocessing algorithm. The Result object is modified by information from a drift space. monitor vectors, corrector parameters and the sign of mu_m is changed accordingly.

**Parameters**

- **model** (*object*) – A valid *cobea.model.BEModel* object or descendant. The object is modified.
- **di** (*list*) – j indices of the used drift space
- **drift_length** (*float*) – length of the use drift space

`cobea.pproc.`**phasor_eigenvectors**(*R_drift*, *length*)

Compute phase space vector from the spatial vectors around a drif t space

**Parameters**

- **R_drift** (*array*) – An array of two spatial vectors R_drift[0] and R_drift[1]
- **length** (*float*) – length of the drift space

**Returns** **Q** – Phase space vector

**Return type** array

`cobea.pproc.`**symplectic_form**(*D=2*)

Compute the symplectic form.

**Parameters** **D** (*int*) – number of spatial dimensions of the phase space vectors considered.

**Returns** **Omega** – a matrix that can be used to compute invariants I from phase space eigenvectors Q via (* matrix product) I = Q.T * Omega * Q

**Return type** array

## 2.5 cobea.plotting: Routines for plotting results

routines for plotting cobea results

Bernard Riemann (bernard.riemann@tu-dortmund.de)

`cobea.plotting.`**A_km**(*result*, *m*, *ax=<matplotlib.axes._subplots.AxesSubplot object>*, *filter='all'*)

plot real and imaginary parts of corrector parameters (incl. errors) into an axis for a given mode m

`cobea.plotting.`**R_jmw**(*result*, *m*, *w=None*, *direction='xy'*, *ax=None*)

plot real and imaginary parts of monitor vectors (incl. errors) into an axis for a given mode m

`cobea.plotting.`**cbeta_jmw**(*result*, *m*, *w=None*, *comparison_data={}*, *direction='xy'*, *ax=None*)

plot beta resp. const*beta (incl. errors) into an axis for a given mode m

`cobea.plotting.`**cbeta_km**(*result*, *m*, *comparison_data={}*, *ax=<matplotlib.axes._subplots.AxesSubplot object>*, *filter='all'*)

plot const*beta at correctors assuming decoupled optics and thin correctors ToDo: errors for this quantity

`cobea.plotting.`**corrector_label**(*corr_labels=[]*, *spacing=0*, *dir='y'*, *ax=<matplotlib.axes._subplots.AxesSubplot object>*)

apply corrector labels to an axis

`cobea.plotting.`**corrector_results**(*result*, *m=0*, *comparison_data={}*, *direction='xy'*, *filter='all'*)

create a figure with corrector results for a given mode m

`cobea.plotting.`**d_jw**(*result*, *w*, *comparison_data*, *direction='xy'*, *ax=None*)

plot const*dispersion (incl. errors) into an axis for a given direction w (0: x, 1: y)

`cobea.plotting.`**delphi_jmw**(*result*, *m*, *w=None*, *comparison_data={}*, *direction='xy'*, *ax=None*)

plot phase-advance per monitor (incl. errors) into an axis for a given mode m

`cobea.plotting.`**monitor_label**(*mon_labels=0*, *spacing=0*, *ax=<matplotlib.axes._subplots.AxesSubplot object>*)

apply monitor labels to an axis

cobea.plotting.**monitor_results**(*result*, *m=0*, *w=None*, *comparison_data={}*, *direction='xy'*)
plot monitor results for mode m, optionally in comparison with comparison_data.

**Parameters**

- **result** (*object*) – A *cobea.model.Result* object.
- **m** (*int*) – mode index to plot results for
- **w** (*int*) – direction index to plot results for
- **comparison_data** (*dict*) – a dictionary containing optional data from alternative decoupled storage ring models, which may contain the following keys: 'name': name of the algorithm or model used 'beta': an array of shape (result.M,result.J) that contains Courant-Snyder beta values for each direction and monitor 'phi': an array of the same shape as 'beta', containing Courant-Snyder betatron phases 'dispersion': an array of the same shape, containing dispersion values

cobea.plotting.**plot_Dev_err**(*result*, *w=0*, *corr_filter='all'*)
create a figure that shows response matrix and residual error for a given direction w (0: x, 1: y)

cobea.plotting.**plot_matrix**(*Devdr*, *devlbl*, *cmap=('PRGn', 'Greens')*, *ax=<matplotlib.axes._subplots.AxesSubplot object>*)
plot an arbitrary matrix with divergent or sequential colormap (helper function)

cobea.plotting.**plot_residual**(*result*, *w=0*, *label='residual'*, *ax=<matplotlib.axes._subplots.AxesSubplot object>*, *corr_filter='all'*)
plot fit residual into an axis for a given direction w (0: x, 1: y)

cobea.plotting.**plot_response**(*response*, *w=0*, *label='deviation'*, *ax=<matplotlib.axes._subplots.AxesSubplot object>*, *corr_filter='all'*)
Plot response matrix into an axis for a given direction w (0: x, 1: y)

cobea.plotting.**plot_result**(*result*, *prefix=''*, *comparison_data={}*, *direction='xy'*, *plot_flags='mcdtvs'*)
plot cobea results.

**Parameters**

- **result** (*object*) – A *cobea.model.Result* object.
- **prefix** (*str*) – if print_figures=True, prefix contains the relative path to the current folder where results are printed.
- **comparison_data** (*dict*) – a dictionary containing optional data from alternative decoupled storage ring models, which may contain the following keys: 'name': name of the algorithm or model used 'beta': an array of shape (result.M,result.J) that contains Courant-Snyder beta values for each direction and monitor 'phi': an array of the same shape as 'beta', containing Courant-Snyder betatron phases 'dispersion': an array of the same shape, containing dispersion values
- **direction** (*str*) – direction characters for the result object. can be 'x','y', or 'xy'.
- **plot_flags** (*str*) – which plots are to be created. Each character represents a different result plot: 'm': monitor_results -> monitor_m*.pdf 'c': corrector_results -> corrector_m*.pdf 'd': plot_Dev_err -> Dev_err_w*.pdf, hist_w*.pdf 't': plot_topology -> topology.pdf 'v': convergence information -> convergence.pdf. Only works if convergence information is available.

cobea.plotting.**plot_topology**(*topology*)
create a figure that shows the accelerator topology. Input: Topology object

# PYTHON MODULE INDEX

## C

## N



## O



## P



## R



## S



## T



## U