Unduwave

Release v0.0.1

Stefan J. Schäfer

Dec 04, 2024

CONTENTS

This package provides basic functionality to run WAVE and Undumag from python.

Note: This project is under active development. |

CHAPTER

ONE

CONTENTS

1.1 Installation

1.1.1 Linux

You need working git and python3 installations for thus guide to work.

Open a terminal in the folder where you want to have unduwave. Type:

```
git clone --recursive https://github.com/SteJSch/unduwave
```

After everything is finished, you can go to the unduwave folder and run in a terminal:

```
pip3 install -r requirements.txt
```

This installs all required packages (consider adding a virtual environment).

After this, you can go the "scripts" subfolder and run:

```
python3 example_wave.py
```

The script should run your first WAVE simulation.

If you encounter errors, you may have to recompile WAVE. | To do that, first go to the folder "\bin" inside the "External-Software/WAVE" directory and delete all files. Then, in the terminal, do

```
export WAVE_INCL="path/to/unduwave/"
```

Then type:

```
python3 python/make_wave.py
```

The compilation should now run. After that, go back to the scripts folder and re-run the "example_wave.py" file.

1.1.2 Windows

Preliminaries

First, read every step before you start doing things!

Then, start by installing windows if you havent done that :)

Now, we will begin:

Download and install Github. For example from here: Git .

Download and install Cygwin. For example from here: Cygwin .

Keep the Cygwin installer file. If you need it to install additional packages later, you can just run the installer file to do that. During the installation, choose: “Install from Internet” when asked and choose some mirror that suites you.

After the Cygwin installation, the package manager will automatically pop up:

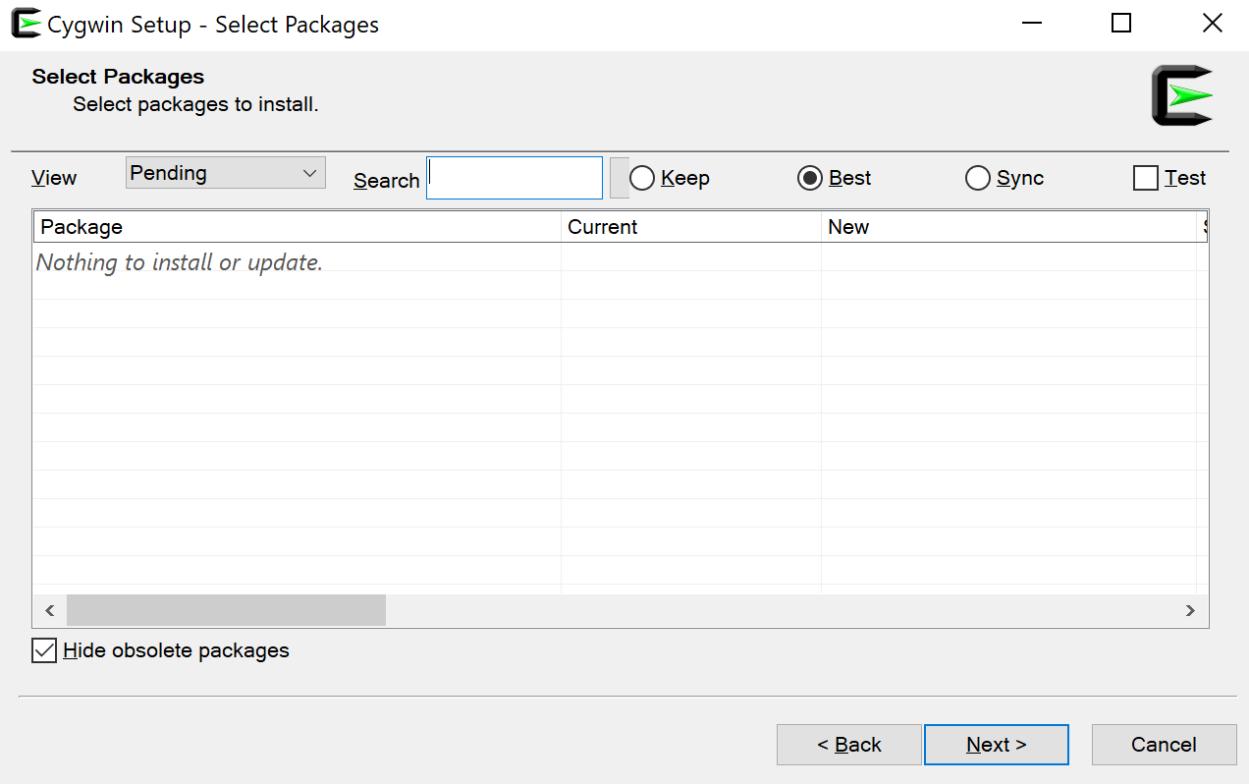

Fig. 1: Cygwin package manager.

Under “View” choose “Full”. Under “Search” search for the following packages:

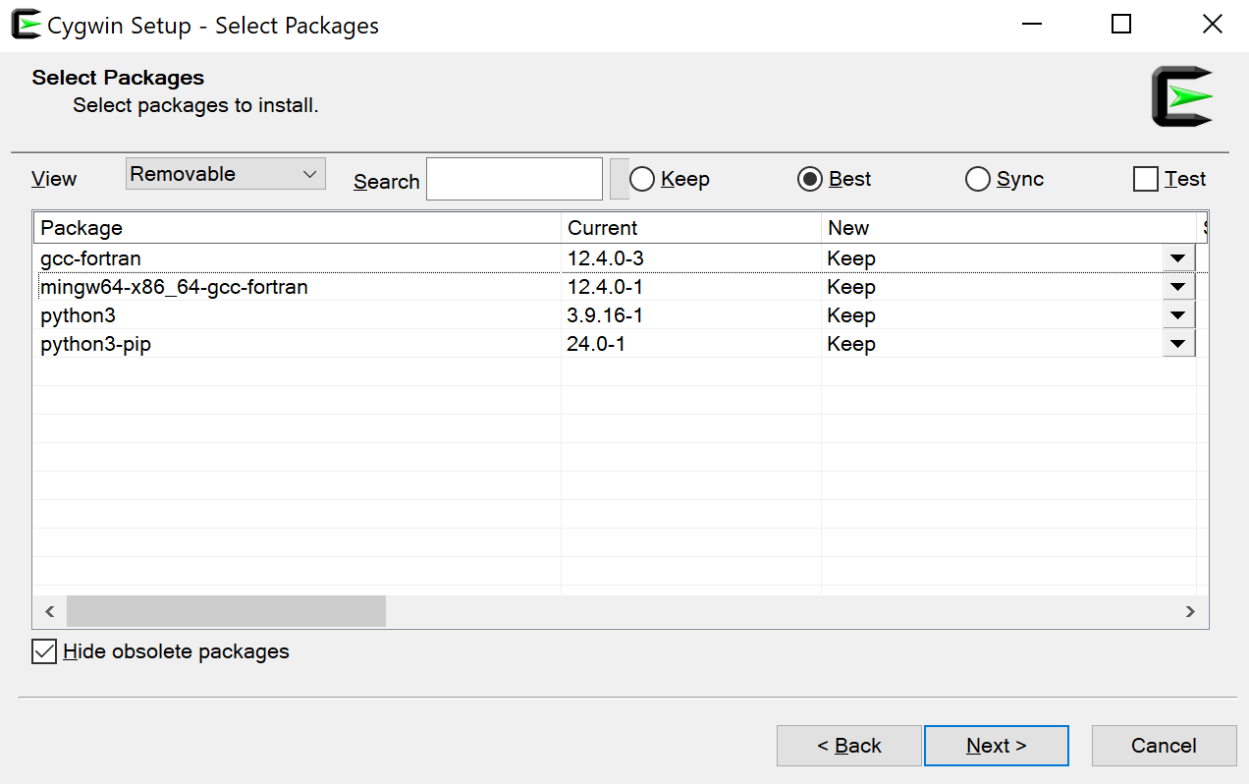

Fig. 2: Cygwin packages to install.

Select the drop-down arrow to the right to select a version of a given program and to add it to the installation list. After you added all packages you can click “next” and the installation will start.

The installation may take a long time. Do some sports meanwhile to get your head free :D

Once the installation is finished, go to the cygwin installation folder and double click on the “cygwin.bat” - Cygwin will be started the first time and does some setting-things. You can close the window afterwards if you like.

Depending on how you run python programs, you may have to install python again separately. In this guide I will run the scripts from the windows powershell. Search in the windows-search window on the lower left for powershell, start it and type “python3”. If python3 is not installed, you will be redirected to the windows store where you can install it. After installation restart the powershell and type again “python3” - the python console should now open.

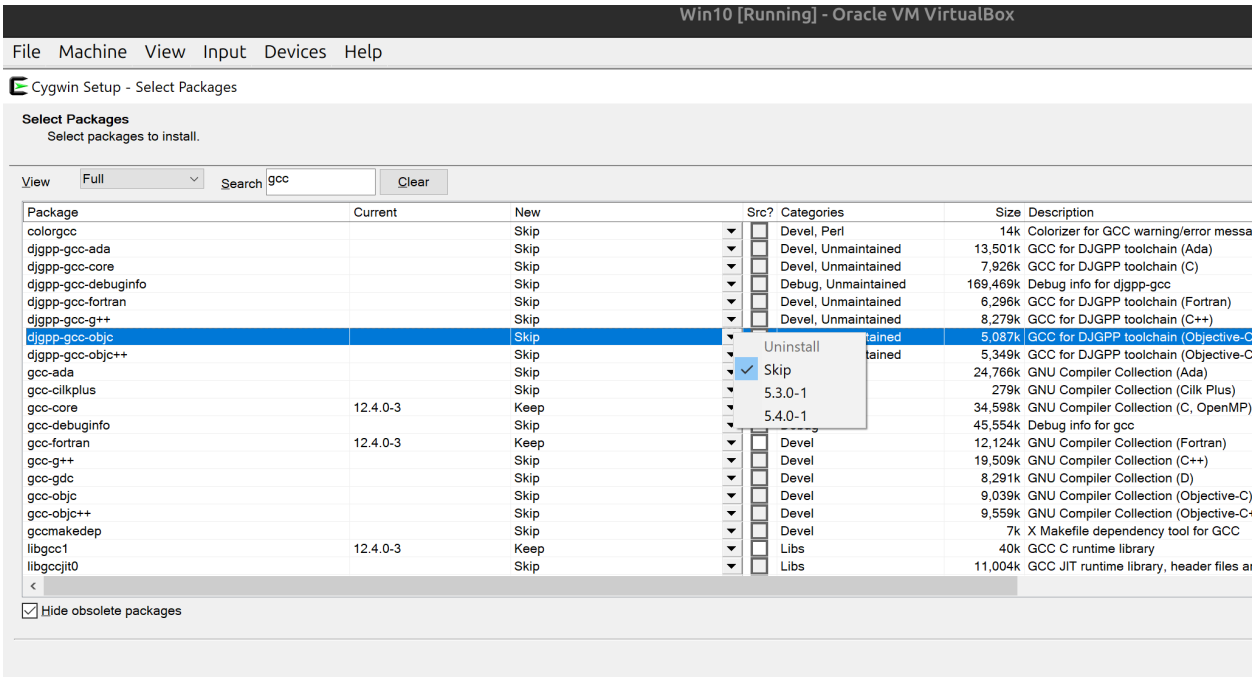

Fig. 3: How to choose to install a given package.

Github and Cloning

In the installation folder of Cygwin, you will find a folder “home/user” with user being your user name. Navigate into this folder with windows-file browser, right click and select “Open Git Bash here” In the bash write:

```
git clone --recursive https://github.com/SteJSch/unduwave
```

If you do not have the Open Git Bash option on right-click, search for the git-bash in the windows search function and navigate inside of it to the right folder in the Cygwin main folder (using +cd folder1/folder2/…/)

If this worked, the folder unduwave should appear and all submodules (wave and undumag) should be initialized (in the External-Software folder in unduwave) This may take some time. Get a coffee (or tea).

Test and Compilation

Go to the folder “unduwave/External-Software” and open the file “where_is_cygwin_installation.txt” . In the first line, write the path of your cygwin installation (this is a hard hack, I know. But only temporary :)). Use the format given, with the quotes and the path terminated by a backslash.

In the second line, write the path where you put unduwave inside the “home/user/” folder inside cygwin. If unduwave lies at the “cygwin/home/user/” you can leave it empty

This will install the required python packages to run the scripts.

To test the installation, go to the folder "unduwave/" inside the windows powershell (this is what I used to start the python scripts). Start the powershell by searching for it in the windows search field (lower left corner) and then navigate to the correct folder inside the powershell using "cd folder1/folder2/…" command. Then do:

```
pip3 install -r requirements.txt
```

This installs all the required python packages to run the scripts.

Then navigate to the sub-folder scripts and once you are there, do:

```
python3 example_wave.py
```

If you are (very) lucky, the program will run, wave will be called and execute normally. Otherwise, you will have to compile WAVE and Undumag.

To do that, first go to the folder "\bin" inside the "unduwave\External-Software\WAVE" directory and delete all files.

Then go to the main folder of your cygwin installation, double click on "cygwin.bat" - this opens the cygwin console

You start at your cygwin home-folder (i.e. in "cygwin/home/user/") and navigate to the folder "unduwave/External-Software/WAVE". In the cygwin console then type:

```
export WAVE_INCL="\path\to\unduwave\External-Software\WAVE"
```

Where you can use the relative path within you cygwin user directory. So, if unduwave lies in "C:\cygwin64\home\user\unduwave" you can simply write :

```
export WAVE_INCL="unduwave\External-Software\WAVE"
```

Then type:

```
python3 python/make_wave.py
```

The compilation should run smoothly, might take a long time. Get yourself whatever you need, a beer or cacao. After it is done, go back to the script folder (in the windows powershell) and retry to run the "example_wave.py" file.

If you are lucky, you now have a working unduwave version on you windows PC. If that installation guide was too much for you, perhaps you should consider changing to Linux :)

1.2 Quickguide

1.2.1 General Things

The Coordinate System

The x coordinate is in longitudinal direction, i.e. along the beam. The y-direction is vertical and z is horizontal.

1.2.2 First Run

First you have to tell python where unduwave is located. For this we do:

```
import sys
import os
sys.path.insert(0, 'absolute/path/to/unduwave/')
```

This is a hack and will be soon changed by making the package installable. (It actually already is installable, but I did not test it and it may be buggy if used in this way)

To import unduwave, do:

```
import unduwave as uw
```

To run a simple simulation:

```
1 wave = uw.wave(undu_mode='undu_easy')
2 wave_prog_paras = wave._wave_prog_paras
3 wave_prog_paras.res_folder.set('res/')
4 wave_prog_paras.spec_calc.set(0)
5 wave.run()
```

The first line creates a wave object which is used to control the parameter setting, the simulation and the creation of results. The parameter undu_mode determines what the undulator model is that is used in the background. “undu_easy” means a simple undulator model is used.

The second line gives you the general program parameter object. Changing the elements of this class will change the corresponding parameter. The third line gives an example of that, changing the “res_folder” element of the wave program paramters.

The fourth line sets the parameter “spec_calc” to 0, thereby switching off spectrum calculations. Only the magnetic fields and trajectories are generated.

Line 5 runs the wave simulation.

Congratulations, you just ran your first WAVE simulation in 5 lines of code.

1.2.3 The Parameter Structures

There are a couple of parameter structures that allow to control the program flow. These are _ebeam_parameters, _screen_parameters, _spectrometer_paras, _undu_paras and _wave_prog_parameters. These are accessible vie the wave-object returned by a "unduwave.wave()" call and can be changed via the set-method as seen in the example above.

The following tables show their contents.

Table 1: _ebeam_parameters

Parameter	Description (Vals)	Unit
beam_en	Beam Energy	GeV
current	Beam Current	A
bsigz	horizontal beam size	m
bsigzp	horizontal beam divergence	rad
bsigy	vertical beam size	m
bsigyp	vertical beam divergence	rad
espread	Energy Spread	%
emitt_h	Horizontal Emittance	m rad
emitt_v	Vertical Emittance	m rad
betfunh	Horizontal Beta Function	m
betfunv	Vertical Beta Function	m
circumference	Ring Circumference	m
rdipol	Bending Radius of Dipoles	m

Table 2: _screen_parameters

Parameter	Description (Vals)	Unit
pinh_w	Pinhole Width(z)	mm
pinh_h	Pinhole Height(y)	mm
pinh_x	Pinhole Position x	m
pinh_nz	Number of horiz. Segments	•
pinh_ny	Number of vert. Segments	•

Table 3: _spectrometer_paras

Parameter	Description (Vals)	Unit
freq_low	Lowest Photon Energy	eV
freq_high	Highest Photon Energy	eV
freq_num	Number of energies	•
undu	Undulator Mode (0/1)	•
wigg	Wiggler Mode (0/1)	•

The undulator mode means the electron is tracked through one period of the device and the radiation on the screen is calculated. The radiation from the other periods is added coherently.

Wiggler mode means the source points on the trajectory are identified and only the radiation from those is considered and added up incoherently.

If both modes are off, the expert mode is switched on. The whole trajectory is tracked and the radiation added together with appropriate phase-shifts between different points on the trajectory.

Table 4: _undu_paras

Parameter	Description (Vals)	Unit
pkHalbasy	K-Value	•
b0Halbasy	B-Amplitude	T
xlHalbasy	Period Length	m
ahwpolHalbasy	Number of Poles (2*periods+1)	•
b0y	B-Amplt. vert.	T
b0z	B-Amplt. hor.	T
nper	Num. Periods	•
perl_x	Period length	m
ell_shift	Shift	%

Different parameters for setting simple undulator models. The first 4 are used in conjunction with "undu_mode='undu_endp'". The other 5 with "undu_mode='undu_ellip'".

Table 5: _wave_prog_parameters

Parameter	Description (Vals)	Unit
wave_prog_folder	Folder where WAVE lies	•
in_file_folder	Folder with WAVE in-files	•
in_files	Name of wave-in file	•
field_folder	Folder where field files lie	•
field_files	List of magnetic field files	•
res_folder	Main result folder	•
wave_data_res_folder	Subfolder with WAVE-res-data	•
pics_folder	Subfolder holding result pics	•
no_copy	List of wave-results to not copy	•
wave_ending_extract	File endings to move	•
wave_ending_copy	File endings to copy	•
wave_files_essentials	Essential files to move	•
nthreads	Number of CPUs to use	•
wave_res_copy_behaviour	Copy behaviour	•
iefold	Fold spectrum (energy spread)	•
spec_calc	Switch on/off spectrum calculation	•
undu_mode	Undu mode	•

1.2.4 Undu Modes

The different modes in which unduwave can be used are described. When creating a wave object via:

```
wave = uw.wave(undu_mode=...)
```

You can specify the mode in which you want to use wave. The mode tells wave basically where you want to get your magnetic field from. The following options are available at the moment: “By”, “Byz”, “undu_ellip” and “undu_endp”.

The first two modes mean that you specify the magnetic field via files. The “undu_ellip” mode represents a simple elliptical undulator and “undu_endp” means a simple undulator with endpoles. See the following examples on how to specify the parameters for these use-cases.

1.2.5 User-defined B-Fields

```
import sys
import os
sys.path.insert(0, 'absolute/path/to/unduwave/')
import unduwave as uw

wave = uw.wave(undu_mode='By')
wave_prog_paras = wave._wave_prog_paras
wave_prog_paras.field_files.set( [ 'field.dat' ] )# The magnetic field files to be used␣
↪in the simulation
field_folder=''
wave_prog_paras.field_folder.set(field_folder)# Field Folder

wave.run()
```

The wave object is first organized and from it the program parameters. Then the field file is specified and the folder it lies in. Then the simulation is run. The field file format is two columns separated by whitespaces. The first holding the x-position in mm and the second the magnetic induction in T.

To load a By and Bz field,

```
import sys
import os
sys.path.insert(0, 'absolute/path/to/unduwave/')
import unduwave as uw

wave = uw.wave(undu_mode='Byz')
wave_prog_paras = wave._wave_prog_paras
wave_prog_paras.field_files.set( [ 'by_tmp.dat','bz_tmp.dat' ] ) # First file y-second z-
↪component of B
field_folder=''
wave_prog_paras.field_folder.set(field_folder)# Field Folder

wave.run()
```

This specifies two field files, one for the vertical and one for the horizontal components.

1.2.6 Elliptical Undulator

```
import sys
import os
sys.path.insert(0, 'absolute/path/to/unduwave/')
import unduwave as uw

wave = uw.wave(undu_mode='undu_ellip')
undu_paras = wave._undu_paras # getting parameter object
undu_paras.b0y.set(1.18)
undu_paras.b0z.set(0.0)
undu_paras.nper.set(10)
undu_paras.perl_x.set(0.020)
undu_paras.ell_shift.set(0.5)
wave.run()
```

An elliptical undulator is created. The parameters are set - b0y and b0z - the vertical and horizontal B-amplitude. nper - the number of periods, perl_x - the period length and ell_shift the shift of the elliptical undulator in % of the period-length.

1.2.7 Simple Undulator with Endpoles

```
import sys
import os
sys.path.insert(0, 'absolute/path/to/unduwave/')
import unduwave as uw

wave = uw.wave(undu_mode='undu_endp') # Simple undulator model with endpoles
undu_paras = wave._undu_paras # getting parameter object
undu_paras.b0Halbasy.set(1.2)
nperiods = 3
undu_paras.ahwpolHalbasy.set(2*nperiods+1) # we count the number of B-field peaks here - one extra for the end-fields (odd)
undu_paras.xlHalbasy.set(0.02)
```

A simple undulator model with endpoles is created. We set the B-field amplitude b0Halbasy, the period-length xlHalbasy and the number of main poles. Given the number of periods, the number of poles is 2*n+1.

1.2.8 Plotting and Results

Getting Results

After a succesful simulation, get the results by doing:

```
results = wave.get_results() # Return Result Object
```

From the results object we can get the quantities wich were calculated. These include:

```
quant = results.get_result(which=string_identifier)
```

Where string_identifier can be one of:

traj_x, traj_y, traj_z - The x,y,z values of the trajectory.

By, Bz - y and z components of B

power_y, power_z, power_distribution - The y and z coordinates for the power distribution on the screen and the values of the actual power distribution

en_flux, flux - the energy values at which the flux through the pinhole was calculated and the flux itself

en_brill, brill0, brill0e, brill0f, brill0ef - The energy values at which the Brilliance was calculated and the brilliance. brill0 - no energy spread and no emittance, brill0e - brilliance with energy spread, brill0f - brilliance with emittance folding, brill0ef - brilliance with energy spread and emittance.

en_fd, flux_density - Energy at which flux density on axis is calculated and flux density on axis

fd_y, fd_z, flux_density_distribution_{en} - the y and z coordinates of a given flux-density distribution (they are the same for all energies, so one pair suffices) and the actual flux density distribution - which contains the energy rounded to second decimal place. So, for example, at the energy 460 eV, the name would be: flux_density_distribution_460.00

Plotting Results

2D Plots

Each quantities data is saved under quantity._data. Each quantity has a number of plotting routines. To do a simple 2D plot, you can do:

```
results = wave.get_results() # Return Result Object
traj_x = results.get_result(which='traj_x')
By = results.get_result(which='By')
By.plot_over(x_quant=traj_x)
```

This plots the y-component of the B-field over the x-coordinate.

Merge Plots

To plot y and z components together, do:

```
nfig=0 # telling it to start with figure number 0
results = wave.get_results() # Return Result Object
traj_x = results.get_result(which='traj_x')
By = results.get_result(which='By')
By.plot_over(x_quant=traj_x,nfig=nfig,nosave=True)
Bz = results.get_result(which='Bz')
nfig=Bz.plot_over(x_quant=traj_x,nfig=nfig,title='B-Field')
```

This will plot By and Bz into the same window, setting a title and returning the new nfig number which can be used in the next plotting command as an argument.

Log Plots

To plot logarithmic axes, do:

```
nfig=0 # telling it to start with figure number 0
results = wave.get_results() # Return Result Object
en_flux = results.get_result(which='en_flux')
flux = results.get_result(which='flux')
nfig=flux.plot_over(x_quant=en_flux,file_name=None,nosave=False,nfig=nfig,loglog=True)
```

Parametric Plots

To do parametric plots, do:

```
nfig=0 # telling it to start with figure number 0
results = wave.get_results() # Return Result Object
traj_x = results.get_result(which='traj_x')
traj_y = results.get_result(which='traj_y')
traj_z = results.get_result(which='traj_z')

traj_y.plot_over(x_quant=traj_x,nfig=nfig,nosave=True)
nfig=traj_z.plot_over(x_quant=traj_x,nfig=nfig,title='Trajectory')
nfig=traj_z.plot_parametric_3d(x_quant=traj_x,y_quant=traj_y,title='Trajectory',
↪nfig=nfig)
```

3D Plots

To do 3D plots, do:

```
nfig=0 # telling it to start with figure number 0
results = wave.get_results() # Return Result Object
power_z = results.get_result(which='power_z')
power_y = results.get_result(which='power_y')
power_distro = results.get_result(which='power_distribution')
nfig=power_distro.plot_over_3d(x_quant=power_y,y_quant=power_z,file_name=None,
↪nosave=False,nfig=nfig)
```

Each call to plot_over_3d creates a 3D plot, a heat plot and plots of interpolated data.

Here are some plotting examples:

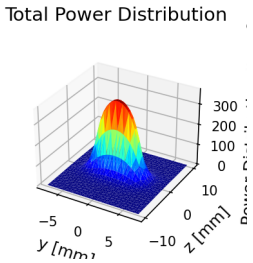

Fig. 4: Original 3D-plot of a power distribution.

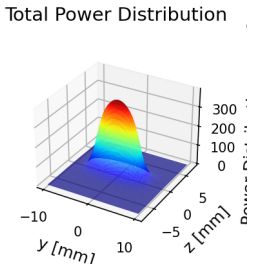

Fig. 5: Interpolated 3D-plot of the previous power distribution.

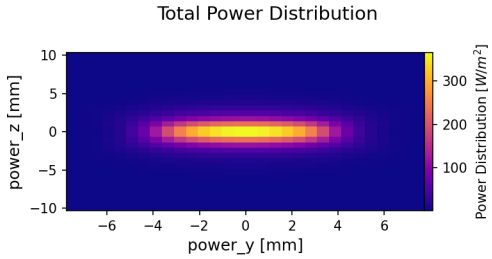

Fig. 6: Original heat map of the above power distribution.

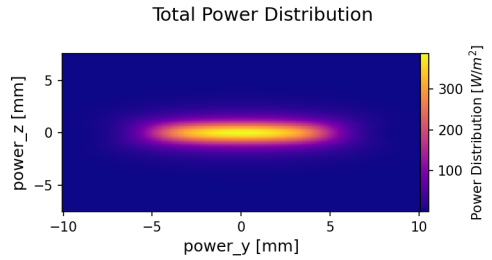

Fig. 7: Interpolated heat map of the above power distribution.

Plot Flux Density Distributions

To plot the flux density distribution, you have to specify the energy at which you want to do this.

```
nfig=0 # telling it to start with figure number 0
results = wave.get_results() # Return Result Object
flux_dens_distr_ens_loaded = results.find_load_flux_density_distribution(energies=[460,
↪504])
for en in flux_dens_distr_ens_loaded :
    fd = results.get_result(which=f'flux_density_distribution_{en:.2f}')
    nfig=fd.plot_over_3d(x_quant=fd_y,y_quant=fd_z,file_name=None,nosave=False,
↪nfig=nfig)
```

The call to find_load_flux_density_distribution searches for the closest distribution files to the energies specified and returns the corresponding energies it found as a list. From this returned list, the corresponding flux_densitie quantities can be returned and then plotted.

1.3 API Reference

This page contains auto-generated API reference documentation[1].

1.3.1 unduwave

Unduwave init statements

Subpackages

unduwave.api

Submodules

unduwave.api.undu_api_root

The basic api

[1] Created with sphinx-autoapi

Classes

point_coords	
undu_api	
point_coords	
undu_magnets	Collects a group of magnets into a list and implements some functionality on that list, like move,mirror,clc-creation,
undu_magnet_block_coords	Implements basic undumag magnet block

Functions

create_magnetization_unit_vec(magn_string)	
rotate(pnts, degrees[, axis, plane])	

Module Contents

unduwave.api.undu_api_root.**create_magnetization_unit_vec**(*magn_string*)

class unduwave.api.undu_api_root.**point_coords**(*x=0.0*, *y=0.0*, *z=0.0*)

__sub__(*pnt*)

__add__(*pnt*)

class unduwave.api.undu_api_root.**undu_api**(*undu_mode='undu_easy'*, *res_folder=''*)

add_element(*element*)

get_para()

set_para(*para*)

create_fresh_nam(*file_lines*, *undu_paras=None*)

create_fresh_clc(*file_lines*, *undu_paras=None*)

get_std_para()

run(*undu_paras=None*, *copy=''*, *freate_fresh_clc=True*)

load_on_axis_undumag_file(*file*)

load_force_undumag_file(*file*)

load_beff_undumag_file(*file*)

class unduwave.api.undu_api_root.**point_coords**(*x=0.0*, *y=0.0*, *z=0.0*)

__sub__(*pnt*)

__add__(*pnt*)

unduwave.api.undu_api_root.**rotate**(*pnts*, *degrees*, *axis=point_coords(0, 0, 0)*, *plane='yz'*)

class unduwave.api.undu_api_root.**undu_magnets**(*magnet_blocks*)

Collects a group of magnets into a list and implements some functionality on that list, like move,mirror,clc-creation, extent determination magnet_blocks - list of undu_magnet_block_coords objects

move_it(*vec*)

find_magn_name(*names*, *fnd_list=None*)

find_all_names(*names=None*)

find_all_mag_blocks(*mag_blocks=[]*)

rotate(*degrees*, *axis*, *plane='yz'*)

mirror(*coord='x'*)

change_segm(*segm_x*, *segm_y*, *segm_z*, *frac_y=None*, *frac_z=None*)

set_inactive()

add_to_clc(*clc_txt*, *magns_ignore=None*)

get_max_extent(*maxs=None*, *mins=None*)

set_magnetization(*magnetization*, *magn_unit_vec*)

get_center()

class unduwave.api.undu_api_root.**undu_magnet_block_coords**(*p_center*, *pnts=None*, *len_x=None*, *len_y=None*, *len_z=None*, *magnetization=None*, *magn_unit_vec=None*, *name='name'*, *mother='mother'*, *segm_x=1*, *segm_y=1*, *segm_z=1*, *frac_y=1*, *frac_z=1*, *material='mag'*, *chamf=None*)

Implements basic undumag magnet block can be moved, mirrored, incorporated into undumag-clc and the extent can be calculated

material - “magnet” or “pole” chamf - if some float - chamfer is added

find_all_names(*names=None*)

find_all_mag_blocks(*mag_blocks=[]*)

set_inactive()

change_segm(*segm_x*, *segm_y*, *segm_z*, *frac_y=None*, *frac_z=None*)

find_magn_name(*names*, *fnd_list=None*)

set_magnetization(*magnetization*, *magn_unit_vec*)

create_edge_points()

move_it(*vec*)

rotate(*degrees*, *axis*, *plane='yz'*)

mirror(*coord='x'*)

add_to_clc(*clc_txt*, *magns_ignore=None*)

create_clc_txt()

get_max_extent(*maxs=None*, *mins=None*)

determines the extent of this magnet block and compares to maxs and mins vals given, returns the max and min vals

unduwave.api.wave_api_root

Wave api definitions

Classes

wave_api	Wave API-class for controlling basic wave-functionality.

Module Contents

class unduwave.api.wave_api_root.**wave_api**(*undu_mode='undu_endp'*)

Wave API-class for controlling basic wave-functionality. Holds the basic parameter classes.

Initialize the WAVE parameters

Parameters

undu_mode (*str*) – can be one of the following: | 'By' : 'Byz' : 'Bxyz' : 'undu_ellip' : 'undu_easy' : 'undu_endp' : 'undu_gap' :

set_bessy_II_elliptical_undu(*nperiods*)

Sets standard settings for bessy II and some helical undulator

run()

Runs wave with the given settings, prepares and postprocesses data

get_results()

Returns the results from a given simulation as result-object.

unduwave.attribute_classes

Submodules

unduwave.attribute_classes.attributes

Definition of _attribute and _attribute_collection classes.

Classes

_attribute	Represents an attribute with a value.
_attribute_collection	Represents a collection of attributes with children.

Module Contents

class unduwave.attribute_classes.attributes.**_attribute**(*value=None*, *name=None*, *in_name=None*, *unit=''*, *fac=None*)

Represents an attribute with a value.

Args:
value: Initial value of the attribute. name (str, optional): Name of the attribute.

Initialize an attribute. :param value: The value held by this class :param str name: Name of the attribute :param str in_name: Name as used by Wave or Undumag :param str unit: The physical unit of the quantity ‘param float fac’: gauging factor

set(*value*)
Sets the value

get()
Returns the value

get_fac()
Returns scaling factor

set_name(*name*)
Setting the name

set_in_name(*in_name*)
Setting the in-name

get_in_name()
Returns in-name

property name

__str__()
Return str(self).

__repr__()
Return repr(self).

class unduwave.attribute_classes.attributes.**_attribute_collection**

Represents a collection of attributes with children. This is a class containing attributes as class-members. The class members are administered in unison.

Initialize the attribute collection

_add_attributes()

Scans all class members, finds those of type _attribute and sets those as new attributes

show_all_children()

Prints the names of the children

children()

Yields the children of the _attribute_collection.

unduwave.helpers

Submodules

unduwave.helpers.bfield_helpers

Contains the functionality for loading and processing b-field data

Functions

load_b_fields_gap(folder[, hints])	Load field files from "folder" containing files with fields for different gaps
checkIfList_Conv(data)	Takes a data object which can be either list of dics or dataframe and converts
checkIfDF_Conv(data)	Takes a data object which can be either list of dics or dataframe and converts
find_maxima(data, lim[, colx, coly])	finds maxima in data whose value is at least **|val|** >= lim
find_minima(data, lim[, colx, coly])	finds minima in data whose value is at least **|val|** >= lim (lim>=0!)
find_extrema(data, lim[, colx, coly])	finds extrema in data whose value is at least **|val|** >= lim
gauge_b_field_data(b_fields, col, gauge_fac)	takes a list of b-fields as returned by load_b_fields_gap and col, the name of the
center_b_field_data(b_fields, lim_peak)	takes b-field data list loaded with load_b_fields_gap and centers each field
convert_x_mm_b_T_file_to_wave_std(folder_in, file_in, ...)	Loads a file in folder_in called file_in with two cols: x[mm] and B[T] - no header to separator
plot_b_field_data(b_fields[, gaps_plt])	
cut_data_support(b_fields[, col_cut])	takes b-field data list loaded with load_b_fields_gap and
center_data(data, strat[, colx, coly])	takes data - which is list of dics of dataframe - and centers it according to strategy in strat
save_prepared_b_data(b_fields, folder[, name_add])	takes b-field data list loaded with load_b_fields_gap, a folder and a string name_add
interpolate_b_data(b_fields, gap, lim_peak[, ...])	interpolates b-field data for a given gap using already present dataframes for different gaps

Module Contents

unduwave.helpers.bfield_helpers.**load_b_fields_gap**(*folder*, *hints=[]*)

Load field files from "folder" containing files with fields for different gaps The file format should be two rows, the first one 'x' in [mm], the second one 'By' [T] The file-name format should be: file_name_front + '**gap_**' + str(gap) + '_' + file_name_back + '.' + file_ending Returns list of dictionaries: { 'gap' : gap,'data' : pd.DataFrame(data), 'file_name' : file_name }, where data is a panda dataframe with rows "x" and "By"

unduwave.helpers.bfield_helpers.**checkIfList_Conv**(*data*)

Takes a data object which can be either list of dics or dataframe and converts it to a list

unduwave.helpers.bfield_helpers.**checkIfDF_Conv**(*data*)

Takes a data object which can be either list of dics or dataframe and converts it to a dataframe

unduwave.helpers.bfield_helpers.**find_maxima**(*data*, *lim*, *colx=0*, *coly=1*)

finds maxima in data whose value is at least **|val|** >= lim data is supposed to be a list of dictionaries or a panda dataframe colx and coly give the column index of the x and y data returns a list containing 2 lists: one with x-coordinates and one with y-coordinates of the maxima positions

unduwave.helpers.bfield_helpers.**find_minima**(*data*, *lim*, *colx=0*, *coly=1*)

finds minima in data whose value is at least **|val|** >= lim (lim>=0!) data is supposed to be a list of dictionaries or a panda dataframe colx and coly give the column index of the x and y data returns a list containing 2 lists: one with x-coordinates and one with y-coordinates of the minima positions

unduwave.helpers.bfield_helpers.**find_extrema**(*data*, *lim*, *colx=0*, *coly=1*)

finds extrema in data whose value is at least **|val|** >= lim data is supposed to be a list of dictionaries or a panda dataframe colx and coly give the column index of the x and y data returns a list containing 2 lists: one with x-coordinates and one with y-coordinates of the extremal positions

unduwave.helpers.bfield_helpers.**gauge_b_field_data**(*b_fields*, *col*, *gauge_fac*)

takes a list of b-fields as returned by load_b_fields_gap and col, the name of the data column of the data objects to gauge and the number gauge_fac each element in the column col is then multiplied by gauge_fac

unduwave.helpers.bfield_helpers.**center_b_field_data**(*b_fields*, *lim_peak*)

takes b-field data list loaded with load_b_fields_gap and centers each field according to the position of the first and last peak identified for which **|peak|** >= lim_peak

unduwave.helpers.bfield_helpers.**convert_x_mm_b_T_file_to_wave_std**(*folder_in*, *file_in*, *out_path*)

Loads a file in folder_in called file_in with two cols: x[mm] and B[T] - no header to separator and converts, depending on b_type, to wave std and copies to out_path (path+filename)

unduwave.helpers.bfield_helpers.**plot_b_field_data**(*b_fields*, *gaps_plt=[]*)

unduwave.helpers.bfield_helpers.**cut_data_support**(*b_fields*, *col_cut='x'*)

takes b-field data list loaded with load_b_fields_gap and cuts the fields to the smallest common support in the column col_cut - afterwards all fiel-data is defined on the same col-values

unduwave.helpers.bfield_helpers.**center_data**(*data*, *strat*, *colx=0*, *coly=1*)

takes data - which is list of dics of dataframe - and centers it according to strategy in strat possible strat vals: { 'name' : 'peak', 'lim' : lim } - determines peaks of **|val|**>=lim and first and last one are centered colx/y are the number of the x and y columns

unduwave.helpers.bfield_helpers.**save_prepared_b_data**(*b_fields*, *folder*, *name_add=''*)

takes b-field data list loaded with load_b_fields_gap, a folder and a string name_add and saves all b-field data into the folder, the files are named according ot the file_name propertie in the b_fields dictionaries with name_add added to the file names

unduwave.helpers.bfield_helpers.**interpolate_b_data**(*b_fields*, *gap*, *lim_peak*, *num_support_per_extrema=20*)

interpolates b-field data for a given gap using already present dataframes for different gaps takes b-field data list loaded with load_b_fields_gap, a gap number, the lim_peak value used for findind the extrema in the data (for determination of the number of periods) and the number of support positions per extrema for the calculation of splines from the data Returns a list containing a dictionary: { 'gap' : gap,'data' : pd.DataFrame(data), 'file_name' : file_name }, where data is a panda dataframe containing the interpolated data and file_name contains the value of the gap at which this data is determined

unduwave.helpers.file_folder_helpers

Contains functionality for handling files.

Module to handle various file operations including finding, moving, copying, and deleting files.

Functions

convert_path_to_win(path)	Takes a path and converts it to win-standard.
find_files_exptn(folder[, hints, exptns])	Find files in the specified folder based on hints and exptns.
find_all_files_exptn(folder[, exptns])	Find all files in a directory while excluding specified patterns.
zip_files_in_folder(folder_to_pack)	Zip all files in a specified folder.
unzip_zip_in_folder(folder)	Unzip a zip file in the specified folder.
mv_cp_files(hints, exptns, folder_in, folder_out[, ...])	Move or copy files between folders based on specified criteria.
del_files(hints, exptns, folder)	Delete files from a specified folder based on specified criteria.
del_all_files(exptns, folder)	Delete all files from a specified folder.

Module Contents

unduwave.helpers.file_folder_helpers.**convert_path_to_win**(*path*)

Takes a path and converts it to win-standard. :param str path: The path. :return: windows path

unduwave.helpers.file_folder_helpers.**find_files_exptn**(*folder*, *hints=[]*, *exptns=[]*)

Find files in the specified folder based on hints and exptns.

Args:

folder (str): The folder in which to search for files. hints (list, optional): List of strings to search for in filenames. Default is []. exptns (list, optional): List of strings to exclude from filenames. Default is [].

Returns:

list: List of filenames matching the criteria.

unduwave.helpers.file_folder_helpers.**find_all_files_exptn**(*folder*, *exptns=[]*)

Find all files in a directory while excluding specified patterns.

Args:

folder (str): The folder in which to search for files. exptns (list, optional): List of strings to exclude from filenames. Default is [].

Returns:
list: List of filenames matching the criteria.

unduwave.helpers.file_folder_helpers.**zip_files_in_folder**(*folder_to_pack*)

Zip all files in a specified folder.

Zips all files in the specified folder, names the zip as the folder name, moves the resulting zip to the same directory, and deletes all other files in the directory.

Args:
folder_to_pack (str): The folder containing files to be zipped.

unduwave.helpers.file_folder_helpers.**unzip_zip_in_folder**(*folder*)

Unzip a zip file in the specified folder.

Looks for a zip file in the specified folder, unzips it there, and returns the list of extracted zip files.

Args:
folder (str): The folder in which to search for and extract zip files.

Returns:
list: List of extracted zip filenames.

unduwave.helpers.file_folder_helpers.**mv_cp_files**(*hints*, *exptns*, *folder_in*, *folder_out*, *move=True*, *add_string=''*)

Move or copy files between folders based on specified criteria.

Moves or copies files whose name contains a string from hints, excluding those whose name contains a string from the exptns list, from folder_in to folder_out and appends add_string to the name.

Args:
hints (list): List of strings to search for in filenames. exptns (list): List of strings to exclude from filenames. folder_in (str): The source folder. folder_out (str): The destination folder. move (bool, optional): Whether to move (True) or copy (False) files. Default is True. add_string (str, optional): String to append to the filenames. Default is ‘’.

Returns:
list: List of names of the moved or copied files (with add_string appended).

unduwave.helpers.file_folder_helpers.**del_files**(*hints*, *exptns*, *folder*)

Delete files from a specified folder based on specified criteria.

Deletes files whose name contains a string from hints, excluding those whose name contains a string from the exptns list, from the specified folder.

Args:
hints (list): List of strings to search for in filenames. exptns (list): List of strings to exclude from filenames. folder (str): The folder from which to delete files.

Returns:
list: List of names of the deleted files.

unduwave.helpers.file_folder_helpers.**del_all_files**(*exptns*, *folder*)

Delete all files from a specified folder.

Deletes all files in the specified folder, excluding those whose name contains a string from the exptns list.

Args:
exptns (list): List of strings to exclude from filenames. folder (str): The folder from which to delete files.

Returns:
list: List of names of the deleted files.

unduwave.quantities

Submodules

unduwave.quantities.quantities

defines quantities which hold data and implements some basic plot-functions

Classes

wave_quantity	wave_api - reference to the wave_api class

Module Contents

class unduwave.quantities.quantities.**wave_quantity**(*wave_api=None*, *name=None*, *description=None*, *unit=None*, *data=None*, *plot_name=None*)

wave_api - reference to the wave_api class name : name of the quantity description: some basic description unit : physical unit data: data-object plot_name: name shown in plot

plot_parametric_3d(*x_quant*, *y_quant*, *title=None*, *file_name=None*, *nosave=False*, *nfig=None*)

Basic plot of data, 2d, 3d

x_quant - the quantity plotted on the x-axis y_quant - quantity for y-axis plot title - title, if none is taken from description of this quantity

Draws a parametetric curve (x_quant,y_quant,self(x_quant,y_quant))

plot_over(*x_quant*, *title=None*, *file_name=None*, *nosave=False*, *nfig=None*, *loglog=False*)

Basic 2d plot of data

x_quant - the quantity to be on the x-axis loglog - True if both axes to be logarithmic

plot_over_3d(*x_quant*, *y_quant*, *title=None*, *file_name=None*, *nosave=False*, *nfig=None*)

Plots 3D plots and heat plots of data x_quant : quantity to be used as x-data y_quant : quantity to be used as y-data file_name : name under which to save plot nosave: True if you do not want to save plot

Creates a 3D plot and a heat plot of the original data and of interpolated data.

save_plot()

unduwave.wave_modules

Submodules

unduwave.wave_modules.wave_control

Attributes

dir_path

Classes

wave_control	Internal API for the WAVE program

Module Contents

unduwave.wave_modules.wave_control.**dir_path**

class unduwave.wave_modules.wave_control.**wave_control**(*wave_api*, *current_folder=None*)

Internal API for the WAVE program

Initialize the internal API wave_api : external wave_api class current_folder : folder to which you want to jump back after wave was run

run()

Run Wave from the self.wave_folder.

If given, change the directory back to self.current_folder

unduwave.wave_modules.wave_parameters

Classes

ebeam_parameters	Defining basic electron-beam parameters
screen_parameters	Basic screen parameters
spectrometer_paras	Basic spectrometer parameters
undu_paras	Parameters controlling the generation of the B-Field
bfield_paras	Represents a collection of attributes with children.
wave_prog_parameters	Represents standard parameters for wave simulations.

Module Contents

class unduwave.wave_modules.wave_parameters.**ebeam_parameters**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Defining basic electron-beam parameters beam_en - Beam energy in [GeV] current - current in [A] bsigz - horizontal beam size [m] bsigzp - Horizontal beam divergence [rad] bsigy - vertical beam size [m] bsigyp - vertical beam divergence [rad] espread - energy spread [%] emitt_h/v - horizontal and vertical emittance [mrad] betfunh/v - horizontal and vertical beta functions [m] circumference - Ring circumference in [m] rdipol - Bending radius of dipoles [m]

Initialize the attribute collection

beam_en

current

bsigz

bsigzp

bsigy

bsigyp

espread

emitt_h

emitt_v

betfunh

betfunv

circumference

rdipol

get_std_bessy_III_paras()

get_std_bessy_II_paras()

get_std_paras()

class unduwave.wave_modules.wave_parameters.**screen_parameters**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Basic screen parameters pinh_w/h - width and height of pinhole [mm] pinh_x - distance of pinhole from center of undu [m] pinh_nz - number of points in z-direction pinh_ny - number of points in y-direction

Initialize the attribute collection

pinh_w

pinh_h

pinh_x

pinh_nz

pinh_ny

get_std_paras()

class unduwave.wave_modules.wave_parameters.**spectrometer_paras**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Basic spectrometer parameters freq_low/high - Energie at which to start/end spectrum calculation [eV] freq_num - number of energies for which to calculate spectrum undu - undulator-mode (whole trajectory is source of radiation - coherent) wigg - wiggler-mode (only source-areas are considered and added incoherently)

Initialize the attribute collection

freq_low

freq_high

freq_num

undu

wigg

get_std_paras()

class unduwave.wave_modules.wave_parameters.**undu_paras**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Parameters controlling the generation of the B-Field

wave_prog_parameters.undu_endp = 1
pkHalbasy - K-Parameter of Machine b0Halbasy - B-Amplitude of Machine (either pkHalbasy or this) [T] xlHalbasy- period length in x-direction [m] ahwpolHalbasy - number of main poles (odd number)

wave_prog_parameters.undu_ellip = 1
b0y - B-Amplitude in y - [T] b0z - B-Amplitude in z - [T] nper- numer of periods perl_x - period length - [m] ell_shift - shift, % of period

Initialize the attribute collection

pkHalbasy

b0Halbasy

xlHalbasy

ahwpolHalbasy

undu_type

b0y

b0z

nper

perl_x

ell_shift

get_std_paras(*undu_mode*)
getting standard undu parameters undu_mode - same as wave_prog_parameters.undu_mode

class unduwave.wave_modules.wave_parameters.**bfield_paras**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Represents a collection of attributes with children. This is a class containing attributes as class-members. The class members are administered in unison.

Initialize the attribute collection

field_folder

get_std_paras()

class unduwave.wave_modules.wave_parameters.**wave_prog_parameters**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Represents standard parameters for wave simulations.

Initialize the attribute collection

wave_prog_folder

in_file_folder

in_files

field_folder

field_files

res_folder

wave_data_res_folder

pics_folder

res_summary_file

no_copy

wave_ending_extract

wave_ending_copy

wave_files_essentials

wave_res_copy_behaviour

zip_res_folder

nthreads

zipped

spec_calc

iemit

iefold

isigusr

ihisascii

b_type

irbtab

irbtabzy

irbtabxyz

undu_easy

undu_endp

undu_gap

undu_ellip

undu_mode

get_std_paras(*undu_mode='undu_easy'*)

Getting standard wave-parameters depending on mode - undu_mode = 'By' - takes by field data and runs with that - undu_mode = 'Byz' - undu_mode = 'Bxyz' - undu_mode = 'undu_ellip' - standard elliptical undulator - undu_mode = 'undu_easy' - undu_mode = 'undu_endp' - undu_mode = 'undu_gap'

unduwave.wave_modules.wave_postprocess

Classes

wave_postprocess	A class for postprocessing WAVE files.

Module Contents

class unduwave.wave_modules.wave_postprocess.**wave_postprocess**(*wave_api*)

A class for postprocessing WAVE files.

Args:
wave_api (StdParameters): An instance of the StdParameters class.

Takes external api and create postprocess class

copy_results()

Cleans the wave-stage folder and copies the desired files to their location, deletes non-desired files, and zips the results based on the wave_res_copy_behaviour setting.

extract_summary(*folder*)

Extracts summary information from a WAVE run's files in the specified folder and stores the results in the file self.wave_paras.res_summary_file within the folder.

Args:
folder (str): The folder containing the WAVE run files.

cleanup()

Cleans up the WAVE run by removing the 'WAVE.mhb' file if it exists in the specified folder.

Args:
wave_folder (str): The folder containing the WAVE run files.

unduwave.wave_modules.wave_prepare

Classes

wave_prepare	_summary_

Module Contents

class unduwave.wave_modules.wave_prepare.**wave_prepare**(*wave_api*)

summary

Args:
wave_api (wave_api): Standard Parameters used for the simulation

Ini wave-prepare class for setting everything up for calcs

create_wave_input()

Creates all the files needed as input fro Wave.

Loads the input file set in wave_paras, updates properties based on other wave_paras properties,and copies the resulting file to the WAVE program folder.

prepare_b_files_for_wave()

Prepare the files for WAVE depending on the b type.

Deoending on which b_type, copies and formats the b-field files needed

unduwave.wave_modules.wave_results

Classes

wave_results	Init wave-result class holding all the results

Module Contents

class unduwave.wave_modules.wave_results.**wave_results**(*wave_api*)

Init wave-result class holding all the results

load_from_res_folder()

goes through folder and loads all results it can find, flux, flux_d, brilliance

find_load_flux_density_distribution(*energies*)

Finds the distribution that is closed to each energy in energies, loads y/z data and flux distribution and adds them to the res_quantities list

energies - list of energies in eV at which flux distribution is to be plotted

find_load_flux_density_on_axis()

Loads on-axis flux-density

find_load_brilliance()

Loads brilliance data

find_load_flux()

Loads flux through pinhole

find_load_power_distribution()

Loads power-distribution data

find_load_trajectory_bfield_data()

Loads trajectory-bfield data

get_result(*which*)

Pass string “flux_dens”,… to get object containing data and some functionality

Submodules

unduwave.constants

Contains physical constants for unduwave

Attributes

fein_const
hbar
q_el
m_el
v_c
mu0
R_el

Module Contents

unduwave.constants.**fein_const**

unduwave.constants.**hbar = 1.054e-34**

unduwave.constants.**q_el = 1.602e-19**

unduwave.constants.**m_el = 9.109e-31**

unduwave.constants.**v_c = 299792458**

unduwave.constants.**mu0 = 8.854e-12**

unduwave.constants.**R_el**

unduwave.unduwave_incl

Contains the import statements for undupy modules

Attributes

cm_inch

Functions

to_scn(number[, norm])	converts number to scientific notation

Module Contents

unduwave.unduwave_incl.**cm_inch**

unduwave.unduwave_incl.**to_scn**(*number*, *norm=True*)

converts number to scientific notation

Attributes

cm_inch
fein_const
hbar
q_el
m_el
v_c
mu0
R_el
cm_inch
cm_inch

Classes

wave	Wave API-class for controlling basic wave-functionality.
undu	
_attribute	Represents an attribute with a value.
_attribute_collection	Represents a collection of attributes with children.
_attribute	Represents an attribute with a value.
_attribute_collection	Represents a collection of attributes with children.
ebeam_parameters	Defining basic electron-beam parameters
screen_parameters	Basic screen parameters
spectrometer_paras	Basic spectrometer parameters
undu_paras	Parameters controlling the generation of the B-Field
bfield_paras	Represents a collection of attributes with children.
wave_prog_parameters	Represents standard parameters for wave simulations.
wave_postprocess	A class for postprocessing WAVE files.
wave_prepare	_summary_
wave_control	Internal API for the WAVE program
wave_results	Init wave-result class holding all the results

Functions

to_scn(number[, norm])	converts number to scientific notation
to_scn(number[, norm])	converts number to scientific notation
to_scn(number[, norm])	converts number to scientific notation

Package Contents

unduwave.**cm_inch**

unduwave.**to_scn**(*number*, *norm=True*)

converts number to scientific notation

unduwave.**fein_const**

unduwave.**hbar = 1.054e-34**

unduwave.**q_el = 1.602e-19**

unduwave.**m_el = 9.109e-31**

unduwave.**v_c = 299792458**

unduwave.**mu0 = 8.854e-12**

unduwave.**R_el**

class unduwave.**wave**(*undu_mode='undu_endp'*)

Wave API-class for controlling basic wave-functionality. Holds the basic parameter classes.

Initialize the WAVE parameters

Parameters
undu_mode (*str*) – can be one of the following: | 'By' : 'Byz' : 'Bxyz' : 'undu_ellip' : 'undu_easy' : 'undu_endp' : 'undu_gap' :

set_bessy_II_elliptical_undu(*nperiods*)

Sets standard settings for bessy II and some helical undulator

run()

Runs wave with the given settings, prepares and postprocesses data

get_results()

Returns the results from a given simulation as result-object.

class unduwave.**undu**(*undu_mode='undu_easy'*, *res_folder=''*)

add_element(*element*)

get_para()

set_para(*para*)

create_fresh_nam(*file_lines*, *undu_paras=None*)

create_fresh_clc(*file_lines*, *undu_paras=None*)

get_std_para()

run(*undu_paras=None*, *copy=''*, *freate_fresh_clc=True*)

load_on_axis_undumag_file(*file*)

load_force_undumag_file(*file*)

load_beff_undumag_file(*file*)

class unduwave.**_attribute**(*value=None*, *name=None*, *in_name=None*, *unit=''*, *fac=None*)

Represents an attribute with a value.

Args:
value: Initial value of the attribute. name (str, optional): Name of the attribute.

Initialize an attribute. :param value: The value held by this class :param str name: Name of the attribute :param str in_name: Name as used by Wave or Undumag :param str unit: The physical unit of the quantity 'param float fac': gauging factor

set(*value*)

Sets the value

get()

Returns the value

get_fac()

Returns scaling factor

set_name(*name*)

Setting the name

set_in_name(*in_name*)

Setting the in-name

get_in_name()

Returns in-name

property name

__str__()

Return str(self).

__repr__()

Return repr(self).

class unduwave.**_attribute_collection**

Represents a collection of attributes with children. This is a class containing attributes as class-members. The class members are administered in unison.

Initialize the attribute collection

_add_attributes()

Scans all class members, finds those of type _attribute and sets those as new attributes

show_all_children()

Prints the names of the children

children()

Yields the children of the _attribute_collection.

unduwave.**cm_inch**

unduwave.**to_scn**(*number*, *norm=True*)

converts number to scientific notation

class unduwave.**_attribute**(*value=None*, *name=None*, *in_name=None*, *unit=''*, *fac=None*)

Represents an attribute with a value.

Args:

value: Initial value of the attribute. name (str, optional): Name of the attribute.

Initialize an attribute. :param value: The value held by this class :param str name: Name of the attribute :param str in_name: Name as used by Wave or Undumag :param str unit: The physical unit of the quantity ‘param float fac’: gauging factor

set(*value*)

Sets the value

get()

Returns the value

get_fac()

Returns scaling factor

set_name(*name*)

Setting the name

set_in_name(*in_name*)

Setting the in-name

get_in_name()

Returns in-name

property name

__str__()

Return str(self).

__repr__()

Return repr(self).

class unduwave._attribute_collection

Represents a collection of attributes with children. This is a class containing attributes as class-members. The class members are administered in unison.

Initialize the attribute collection

_add_attributes()

Scans all class members, finds those of type _attribute and sets those as new attributes

show_all_children()

Prints the names of the children

children()

Yields the children of the _attribute_collection.

class unduwave.ebeam_parameters

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Defining basic electron-beam parameters beam_en - Beam energy in [GeV] current - current in [A] bsigz - horizontal beam size [m] bsigzp - Horizontal beam divergence [rad] bsigy - vertical beam size [m] bsigyp - vertical beam divergence [rad] espread - energy spread [%] emitt_h/v - horizontal and vertical emittance [mrad] betfunh/v - horizontal and vertical beta functions [m] circumference - Ring circumference in [m] rdipol - Bending radius of dipoles [m]

Initialize the attribute collection

beam_en

current

bsigz

bsigzp

bsigy

bsigyp

espread

emitt_h

emitt_v

betfunh

betfunv

circumference

rdipol

get_std_bessy_III_paras()

get_std_bessy_II_paras()

get_std_paras()

class unduwave.**screen_parameters**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Basic screen parameters pinh_w/h - width and height of pinhole [mm] pinh_x - distance of pinhole from center of undu [m] pinh_nz - number of points in z-direction pinh_ny - number of points in y-direction

Initialize the attribute collection

pinh_w

pinh_h

pinh_x

pinh_nz

pinh_ny

get_std_paras()

class unduwave.**spectrometer_paras**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Basic spectrometer parameters freq_low/high - Energie at which to start/end spectrum calculation [eV] freq_num - number of energies for which to calculate spectrum undu - undulator-mode (whole trajectory is source of radiation - coherent) wigg - wiggler-mode (only source-areas are considered and added incoherently)

Initialize the attribute collection

freq_low

freq_high

freq_num

undu

wigg

get_std_paras()

class unduwave.**undu_paras**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Parameters controlling the generation of the B-Field

wave_prog_parameters.undu_endp = 1
pkHalbasy - K-Parameter of Machine b0Halbasy - B-Amplitude of Machine (either pkHalbasy or this) [T] xlHalbasy- period length in x-direction [m] ahwpolHalbasy - number of main poles (odd number)

wave_prog_parameters.undu_ellip = 1
b0y - B-Amplitude in y - [T] b0z - B-Amplitude in z - [T] nper- numer of periods perl_x - period length - [m] ell_shift - shift, % of period

Initialize the attribute collection

pkHalbasy

b0Halbasy

xlHalbasy

ahwpolHalbasy

undu_type

b0y

b0z

nper

perl_x

ell_shift

get_std_paras(*undu_mode*)

getting standard undu parameters undu_mode - same as wave_prog_parameters.undu_mode

class unduwave.**bfield_paras**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Represents a collection of attributes with children. This is a class containing attributes as class-members. The class members are administered in unison.

Initialize the attribute collection

field_folder

get_std_paras()

class unduwave.**wave_prog_parameters**

Bases: *unduwave.attribute_classes.attributes._attribute_collection*

Represents standard parameters for wave simulations.

Initialize the attribute collection

wave_prog_folder

in_file_folder

in_files

field_folder

field_files

res_folder

wave_data_res_folder

pics_folder

res_summary_file

no_copy

wave_ending_extract

wave_ending_copy

wave_files_essentials

wave_res_copy_behaviour

zip_res_folder

nthreads

zipped

spec_calc

iemit

iefold

isigusr

ihisascii

b_type

irbtab

irbtabzy

irbtabxyz

undu_easy

undu_endp

undu_gap

undu_ellip

undu_mode

get_std_paras(*undu_mode='undu_easy'*)

Getting standard wave-parameters depending on mode - undu_mode = 'By' - takes by field data and runs with that - undu_mode = 'Byz' - undu_mode = 'Bxyz' - undu_mode = 'undu_ellip' - standard elliptical undulator - undu_mode = 'undu_easy' - undu_mode = 'undu_endp' - undu_mode = 'undu_gap'

unduwave.**cm_inch**

unduwave.**to_scn**(*number*, *norm=True*)

converts number to scientific notation

class unduwave.**wave_postprocess**(*wave_api*)

A class for postprocessing WAVE files.

Args:

wave_api (StdParameters): An instance of the StdParameters class.

Takes external api and create postprocess class

copy_results()

Cleans the wave-stage folder and copies the desired files to their location, deletes non-desired files, and zips the results based on the wave_res_copy_behaviour setting.

extract_summary(*folder*)

Extracts summary information from a WAVE run's files in the specified folder and stores the results in the file self.wave_paras.res_summary_file within the folder.

Args:
folder (str): The folder containing the WAVE run files.

cleanup()

Cleans up the WAVE run by removing the 'WAVE.mhb' file if it exists in the specified folder.

Args:
wave_folder (str): The folder containing the WAVE run files.

class unduwave.**wave_prepare**(*wave_api*)

summary

Args:
wave_api (wave_api): Standard Parameters used for the simulation

Ini wave-prepare class for setting everything up for calcs

create_wave_input()

Creates all the files needed as input fro Wave.

Loads the input file set in wave_paras, updates properties based on other wave_paras properties,and copies the resulting file to the WAVE program folder.

prepare_b_files_for_wave()

Prepare the files for WAVE depending on the b type.

Deoending on which b_type, copies and formats the b-field files needed

class unduwave.**wave_control**(*wave_api*, *current_folder=None*)

Internal API for the WAVE program

Initialize the internal API wave_api : external wave_api class current_folder : folder to which you want to jump back after wave was run

run()

Run Wave from the self.wave_folder.

If given, change the directory back to self.current_folder

class unduwave.**wave_results**(*wave_api*)

Init wave-result class holding all the results

load_from_res_folder()

goes through folder and loads all results it can find, flux, flux_d, brilliance

find_load_flux_density_distribution(*energies*)

Finds the distribution that is closed to each energy in energies, loads y/z data and flux distribution and adds them to the res_quantities list

energies - list of energies in eV at which flux distribution is to be plotted

find_load_flux_density_on_axis()

Loads on-axis flux-density

find_load_brilliance()

Loads brilliance data

find_load_flux()

Loads flux through pinhole

find_load_power_distribution()

Loads power-distribution data

find_load_trajectory_bfield_data()

Loads trajectory-bfield data

get_result(*which*)

Pass string “flux_dens”,… to get object containing data and some functionality

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

P

Q

R

V

W

X

Z