WavePy

Stefan Schaefers

Aug 23, 2023

CONTENTS:

WavePy provides a simple python API to work with WAVE RAY-UI, a software for undulators simulations developed at Helmholtz-Zentrum Berlin.

This do cuments contains the api descriptions of WavePy and some general remarks. WavePy is a python wrapper for WAVE that allows easy calculation of spectra with wave from magnetic field data. Also functionality for the loading and interp olation of magnetic field data is incorporated.

CHAPTER

ONE

INSTALLATION

1.1 Install WavePy

Wavepy will works on Linux, macOS and Windows.

- You will need Python 3.9 or newer. From a shell ("Terminal" on OSX), check your current Python version.

  ```
  python3 --version
  ```

  If that version is less than 3.9, you must update it.

  We recommend installing wavepy into a "virtual environment" so that this installation will not interfere with any existing Python software:

  ```
  python3 -m venv ~/wavepy-tutorial
  source ~/wavepy-tutorial/bin/activate
  ```

  Alternatively, if you are a conda user, you can create a conda environment:

  ```
  conda create -n wavepy-tutorial "python>=3.9"
  conda activate wavepy-tutorial
  ```

  ```
  python3 -m pip install --upgrade wavepy
  ```

1.2 Troubleshooting

1.2.1 Wave can not be Executed

Make sure that `wavepy/WAVE/bin/wave.exe` is executable.

1.2.2 Files are not copied after the simulations runned

Does the result folder exist? If not, create it.

1.2.3 Interpolated files are not saved

Does the folder where they should be saved exist? If not, create it.

WAVE is complaining about: NEGATIVE OR ZERO PHOTON EN- ERGY OCCURED WHILE EXTENDING EN-
ERGY ~~~~~~
Increase the number of energy points - parameter `freq_num`. Wave is extending the energy range you specified in order to calculate the folding procedure and may, with too little points on which to calculate, run into negative energies. This is especially important at low energy values.

1.2.4 Wave complains it cannot find a zip file while trying to plot

Were the results files not prop erly stored before? Check if the data folder contains more than a zip file and delete everything but the zip file.

CHAPTER

TWO

HOW TO USE WAVEPY

2.1 Axis Nomenclature

Flight direction of the electron is `x` . The vertical direction (in planer undulators) is called `y` and the horizontal is `z` .

2.2 Spectrum Calculations

Import wavepy and create a wave instance

```
from wavepy import WaveFromB
wave = WaveFromB('By')
```

WAVE can perform many different tasks, one of which is calculating a spectrum from a given B-field. This specific functionality is so far covered in wavepy. You can calculate the spectra from a magnetic field in Bydirection only by using `By`, from By and Bz data by setting `Byz` and from all components by writing `Bxyz`. All field data should be in the format of two cols, the first one containing `x` in mm and the second B in Tesla, no separators and header lines. Next you can get standard wave parameters by writing:

```
# Get Wave-parameter
paras = wave.get_paras()
# Customize Parameters
paras.field_folder.set('..../Fields/Interpolated/') # Field Folder
paras.field_files.set( [ 'interp_b_field_gap_18.45_.dat'] ) # The magnetic field files␣
↪to be used in the simulation
paras.res_folder.set('..../Spectrum_Results/First_Simu/')   # where to store the wave␣
↪results
paras.spec_calc.set(1)    # calculate spectrum? if not, only trajectory is calculated
paras.freq_low.set(100)   # lower frequency for spectrum calc. [eV]
paras.freq_high.set(105)  # upper frequency for spectrum calc. [eV]
paras.freq_num.set(5)     # number of frequencies for spectrum calc. [eV]
paras.pinh_w.set(3)       # pinhole width (horizontal-z) [mm]
paras.pinh_h.set(3)       # pinhole height (vertical-y) [mm]
paras.pinh_x.set(10)      # pinhole distance in beam direction (x) [m]
paras.pinh_nz.set(21)     # number of horizontal points in pinhole for spec. calc.
paras.pinh_ny.set(21)     # number of vetical points in pinhole for spec. calc.
paras.wave_res_copy_behaviour.set('copy_essentials') #'copy_all'
```

Paras is a `StdParamenter` object. Then set the parameters and run wave:

```
# Set Parameters
wave.set_paras(paras)
# Run WAVE
wave.run_wave()
```

2.2.1 Summary file

WavePy then saves all the data that is set to b e saved into the designated folder and zips all data for storage. On plotting the data is unzipped and zipped again. Furthermore, a fiel `res_summary.txt` is stored inside the zip archive. This file contains some general information extracted from wave about the current simulation, including:

```
bmax [T] : max. b field
bmin [T] : min. b field
first_int [Tm] : first ingegral
scnd_int [Tmm] : second integral
power [kW] : total emitted power
pinhole_x [m] : dist. of pinhole from undulator centre
Fund. Freq. [eV] : fundamental frequency
Undu_Para : undulator parameter K
gamma : well, you guessed it
half_opening_angle [rad] : the half-opening angle
cone_radius_at_x [mm] : the cone-radius at the position
of the pinhole (tan(half_opening_angle)*pinhole_x)
```

2.3 B-Field Interpolation

Keep in mind that WAVE exp ects the magnetic field data over mm. You can load files containing magnetic field data for different gaps by writing:

```
b_field_data = wpy.load_b_fields_gap(folder)
```

The file name format should be `somename + gap_x_ + restname +.ending`. E.g. `myfile_g_23_.dat`. With the gap given in mm. To plot the b-field data:

```
wpy.plot_b_field_data( b_fields = b_field_data )
```

You can center the data and cut all loaded magnetic fields to the defined on the same interval by writing:

```
wpy.center_b_field_data( b_fields = b_field_data, lim_peak = 0.01 )
wpy.cut_data_support(b_fields = b_field_data, col_cut = 'x')
```

The centering is done by identifying the first and last peaks and centering those. The option `lim_peak` gives the minimal height a p eak has to have to be identified as a peak - this is necessary to account for noise in the data. Save the pro cessed fields by writing:

```
wpy.save_prepared_b_data(b_fields = b_field_data, folder = folder)
```

Next we can interpolate the data by writing:

```
interp_field = wpy.interpolate_b_data(b_fields = b_field_data, gap = gap, lim_peak = 0.
↪01)
```

Where `gap`` is the gap at which you'd like to interpolate.

2.4 Troubleshooting

2.4.1 Files are not copied after the simulations runned

Does the result folder exist? If not, create it.

2.4.2 Interpolated files are not saved

Does the folder where they should be saved exist? If not, create it.

WAVE is complaining about: NEGATIVE OR ZERO PHOTON EN- ERGY OCCURED WHILE EXTENDING EN- ERGY ~~~~~~ Increase the number of energy points - parameter `freq_num`. Wave is extending the energy range you specified in order to calculate the folding procedure and may, with too little points on which to calculate, run into negative energies. This is especially important at low energy values.

2.4.3 Wave complains it cannot find a zip file while trying to plot

Were the results files not prop erly stored before? Check if the data folder contains more than a zip file and delete everything but the zip file.

CHAPTER

THREE

HOW TO GUIDES

nothing here at the moment

CHAPTER

FOUR

API

4.1 Simulation

4.1.1 WAVE

`class wavepy.wave.`**`WaveFromB`**`(`*b_type='By'*`)`

WAVE class to perform spectrum calculations from b-field data files

Initiates WAVE class object with self.std WAVE parameters b_type sets which b-fields are loaded from file: ‘y’: By, ‘yz’ By and Bz, ‘xyz’ … . Each B-field from different file

Parameters

`b_type` (*`str, optional`*) – Field. possible values: By, Byz, Bxyz. Defaults to ‘By’.

`get_paras``()`

Returns current parameters

`run_wave``()`

Runs wave and postprocess the results

`set_paras``(`*wave_paras:* StdParameters`)`

Sets new parameters

Parameters

`wave_paras` (`StdParameters`) – Standard Parameters

4.1.2 WAVE API

`class wavepy.wave_API.`**`WaveAPI`**`(`*wave_folder: str*, *current_folder: str = False*`)`

API for the WAVE program

Parameters

- **`current_folder`** (*`str`*) – _description_
- **`wave_folder`** (*`str`*) – _description_

`run``()`

Run Wave from the self.wave_folder.

If given, change the directory back to self.current_folder

4.1.3 Elements

class wavepy.wave_elements.**WaveAttribute**(*value=None*, *name=None*)

Represents a wave attribute with a value.

Parameters

- **value** – Initial value of the attribute.
- **name** (*str, optional*) – Name of the attribute.

class wavepy.wave_elements.**WaveElement**(*b_type='By'*)

Represents a wave element with children.

Parameters
b_type (*str, optional*) – Type of the wave element.

children()

Yields the children of the WaveElement.

class wavepy.standard_parameters.**StdParameters**(*b_type='By'*)

Represents standard parameters for wave simulations.

Parameters
b_type (*str, optional*) – Type of the wave element.

get_std_paras(*b_type='By'*)

Returns a object containing WAVE standard parameters updated with standard values

Parameters

- **b_type** (*str*) – Type of b-calculation. Options: 'By' (only By given), 'Byz', 'Bxyz' (each b-field in different file).
- **wave_prog_folder** (*str*) – Main folder where wave is stored (ending on '/').
- **in_file_folder** (*str*) – Folder in which the wave in-files are stored (ending on '/').
- **in_files** (*dict*) – Dictionary of wave in-files for different b_type situations. Format: {b_type: wave_in_file}.
- **field_folder** (*str*) – Folder where b-field files are stored.
- **field_files** (*list*) – List of b-field files. Format: 2 cols - x[mm] and B[T], no separator, no headers.
- **res_folder** (*str*) – Folder where results are stored (ending on '/').
- **wave_data_res_folder** (*str*) – Subfolder of res_folder where wave data is stored.
- **pics_folder** (*str*) – Subfolder of res_folder where pictures are stored.
- **res_summary_file** (*str*) – Name of the summary file to be written.
- **no_copy** (*list*) – List of file names not to be copied/moved after simulation from wave stage folder.
- **wave_ending_extract** (*list*) – List of file endings to move from wave stage folder after simulation.
- **wave_ending_copy** (*list*) – List of file endings of files to be copied (not moved).
- **wave_res_copy_behaviour** (*str*) – Behavior for copying wave results: 'copy_all', 'copy_del_none', 'copy_essentials'.

- **wave_files_essentials** (*list*) – List of essential files when wave_res_copy_behaviour is set to 'copy_essentials'.
- **zip_res_folder** (*bool*) – Truth value, whether to zip results or not.
- **freq_low** (*float*) – Lower frequency (energy) of spectrum to calculate [eV].
- **freq_high** (*float*) – Upper frequency (energy) of spectrum to calculate [eV].
- **freq_num** (*int*) – Number of frequencies to calculate.
- **beam_en** (*float*) – Beam energy in [GeV].
- **current** (*float*) – Current in [A].
- **pinh_w** (*float*) – Pinhole width (horizontal-z) [mm].
- **pinh_h** (*float*) – Pinhole height (vertical-y) [mm].
- **spec_calc** (*bool*) – Truth value, whether to calculate spectrum or only write trajectory.
- **pinh_x** (*float*) – Position of pinhole along optical axis [m].
- **pinh_nz** (*int*) – Number of points in pinhole horizontally.
- **pinh_ny** (*int*) – Number of points in pinhole vertically.

4.1.4 Preprocess Files

class wavepy.preprocess_wave_files.**PreprocessWaveFiles**(*wave_paras:* StdParameters)

summary

Parameters

wave_paras (StdParameters) – Standard Parameters used for the simulation

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

4.1.5 Postprocess Files

class wavepy.postprocess_wave_files.**PostprocessWaveFiles**(*wave_paras:* StdParameters)

A class for postprocessing WAVE files.

Parameters

wave_paras (StdParameters) – An instance of the StdParameters class.

cleanup(*wave_folder*)

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

Parameters

wave_folder (*str*) – The folder containing the WAVE run files.

`edit_wave_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.

Parameters

`folder` (*str*) – The folder containing the WAVE run files.

4.1.6 Extract and Plot

`class wavepy.extract_plot_results.ExtractAndPlot`(*wave_paras*)

`extract_wave_results`(*results: list*, *plot: bool = False*, *en_range: list = []*, *xlim: list = []*, *ylim: list = []*, *save_data_w_pics: bool = False*)

Extracts the desired results and returns a DataFrame with them.

Parameters

- **results** (*list*) – A list containing the desired result types. Valid options are: - 'traj_magn': Extracts Trajectory and magnetic field data. - 'power_distr': Power distribution [W/mm^2]. - 'flux': Flux. - 'flux_dens': Flux Density. - 'flux_dens_distr': Flux Density Distribution. - 'power_dens_distr': Power density distribution. - 'brill': Brilliance.
- **plot** (*bool*) – Determines if loaded data is plotted or not.
- **en_range** (*list*) – For distribution, gives the energy range over which the data is integrated.
- **xlim** (*list*) – Sets the limits of the distribution plots in the horizontal direction.
- **ylim** (*list*) – Sets the limits of the distribution plots in the vertical direction. For 2D plots, only xlim is considered if given.
- **save_data_w_pics** (*bool*) – If True, saves the data for each plot alongside the picture.

Returns

A list of the loaded DataFrames.

Return type

list

`load_plot_stokes_distrib`(*folder*, *en_range=[]*, *xlim=[]*, *ylim=[]*, *plot=True*, *save_folder=''*, *save_data_w_pics=False*, *power_distr=False*)

loads stokes (flux-density) data from a folder, integrates the flux-density distribution over the en_range and cuts the plot horizontally and vertically to xlim and ylim, plots if plot = True and saves pic to save_folder returns the loaded and integrated data object If power_distr is True, then not the flux density but power density in the given energy range is calculated (multiplying all fluxes by the appropriate energies)

INDEX