# wavepy documentation

April 28, 2023

This documents 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 interpolation of magnetic field data is incorporated.

# Chapter 1

# Description of Usage

## 1.1 Wave Axis Nomenclature

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

## 1.2 Spectrum Calculations

Import the wavepy project and create a wave instance by writing:

```
import wavepy as wpy
wave = wpy.create_wave_instance('wave_from_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 $B_y$ direction only by using `wave_from_by`, from $B_y$ and $B_z$ data by setting `wave_from_byz` and from all components by writing `wave_from_bxyz`. All field data should be in the format of two cols, the first one containing x in mm and the second B in T, no separators and header lines.

Next you can get standard wave parameters by writing:

```
paras = wave.get_paras()
```

The parameter object is a simple python dictionary with the following keys which correspond to wave settings:

```
b_type - which type of b-calculation? 'By' - only By given, 'Byz'
and 'Bxyz'. Each b-field in different file
wave_prog_folder - main folder where wave is stored (ending on /)
in_file_folder - folder in which the wave in-files are stored (ending on /)
in_files - the wave in-files to be used for different b_type situations, is
dic: { b_type : wave_in_file }
field_folder - folder where b-field files are stored
field_files - the b-field files list, file format: 2cols:
 x[mm] and B[T], no separator and no headers
res_folder - the folder in which the results are to be stored (ending on /)
wave_data_res_folder - Subfolder of res_folder where wave data is stored
pics_folder - subfolder of res_folder were pics are stored
res_summary_file - name of the summary_file to be written
no_copy - list of file names not to be copied/moved after simulation from wave stage folder
wave_ending_extract - file endings to move from wave stage folder after simu
wave_ending_copy - file endings of files to be copied (not moved)
wave_res_copy_behaviour:
'copy_all', 'copy_del_none' - only writes res_summary, 'copy_essentials'
only copies whats needed for: flux, flux_dens, flux_dens_distri, power, files
set in wave_files_essentials
wave_files_essentials - essential files, if wave_res_copy_behaviour set to 'wave_res_copy_behaviour'
only those files are stored
zip_res_folder - truth value, zip or not zip results
freq_low - lower frequency (actualy energy) of spectrum to calc [eV]
freq_high - upper frequency (actualy energy) of spectrum to calc [eV]
```

```
freq_num - number of frequencies to calc
beam_en - beam energy in [GeV]
current - current [A]
pinh_w - pinhole width (horizontal-z) [mm]
pinh_h - pinhole height (vertical-y) [mm]
spec_calc - truth value: calc spectrum or not, if not, only trajectory is written to file
pinh_x - position of pinhole along optical axis [m]
pinh_nz - number of points in pinhole horizontally
pinh_ny - number of points in pinhole vertically
```

Set parameters and run wave by:

```
wave.set_paras(paras)
wave.run_wave()
```

Extract and plot results by writing:

```
data = wave.extract_wave_results(results = ['flux_dens_distr','traj_magn',
'power_distr','flux','flux_dens',\
  'brill' ], plot = True, en_range = [300,500])
```

The result strings shown here are all that are permissible until now.

### 1.2.1 Summary File

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

## 1.3 B-Field Interpolation

Keep in mind that WAVE expects 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 do:

```
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 peak has to have to be identified as a peak - this is necessary to account for noise in the data. Save the processed 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.

# Chapter 2

# Troubleshooting

- WAVE cannot be executed: Is the file `wavepy/WAVE/bin/wave.exe` executable?
- If files cannot be copied after simulation: Is the result folder you specified existing?
- interpolated files cannot be saved: Is the folder where the files are supposed to go existing?
- WAVE is complaining about: "NEGATIVE OR ZERO PHOTON ENERGY OCCURED WHILE EXTENDING ENERGY " : Increase the number of energy points - parameter `freq_num`. Wave is extending the energy range you speficied 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.
- Wave complains it cannot find a zip file while trying to plot: Were the results files not properly stored before? Check if the data folder contains more than a zip file and delete everything but the zip.

# Chapter 3

# Documentation

### 3.0.1 file_folder_helpers.py

Contains functionality for handling files

- `def find_files_exptn(folder, hints = [], exptns = []) :`

  Finds all files whose name contains one of the strings in the list hints, if empty, all files are loaded, from those files expludes those whose file name contains one of the strings in the list exptns returns list of those filenames

- `def find_all_files_exptn(folder, exptns = []) :`

  Finds all files in a directory. from those files expludes those whose file name contains one of the strings in the list exptns returns list of those filenames

- `def zip_files_in_folder(folder_to_pack) :`

  Zips all files in folder_to_pack, names the zip as the folder, Moves the resulting zip to directory folder_to_pack And deletes all other files in this directory

- `def unzip_zip_in_folder(folder) :`

  Looks for a zip file in folder, unzips it there. Returns the extracted zip file list (empty if none extracted and found)

- `def mv_cp_files(hints, exptns, folder_in, folder_out, move = True, add_string = '') :`

  Moves files whose name contains some string from hints (excepting those whose name cont. some string from list exptns ) from folder_in to folder_out and adds add_string to name if move = False, then files are only copied Returns list of the names of the moved files (after adding add_string)

- `def del_files(hints, exptns, folder) :`

  Deletes files whose name contains some string from hints (excepting those whose name cont. some string from list exptns ) from folder Returns list of the names of the deleted files

- `def del_all_files(exptns, folder) :`

  Deletes all files in directory (excepting those whose name cont. some string from list exptns ) from folder Returns list of the names of the deleted files

### 3.0.2 wave_b_helper.py

Contains the functionality for loading and processing b-field data

- `def 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"

- `def checkIfList_Conv(data) :`

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

- `def checkIfDF_Conv(data) :`

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

- `def 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

- `def 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

- `def 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

- `def 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

- `def 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

- `def 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)

- `def plot_b_field_data( b_fields, gaps_plt = [] ) :`

- `def 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

- `def 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

- `def 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

- `def 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

### 3.0.3 wavepy_incl.py

Contains the import statements for wavepy modules

- `def to_scn(number,norm=True):`

  converts number to scientific notation

### 3.0.4 physical_constants.py

Definition of some physical constants

### 3.0.5 __init__.py

wavepy is a litter python wrapper build for easy use of the program WAVE by Michael Scheer Import by writing: import wavepy as wpy

### 3.0.6 wave.py

Contains the wave_from_b class that incorporates the API from python to WAVE and the function create_wave_instance that returns an instance of that class

- `class wave_from_b() :`

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

- `def __init__(self, b_type = 'By') :`

  Initiates WAVE class object with 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

- `def get_std_paras(self, b_type = 'By') :`

  Returns WAVE std. parameter dictionary. Entries are: b_type - which type of b-calculation? 'By' - only By given, 'Byz' and 'Bxyz'. Each b-field in different file wave_prog_folder - main folder where wave is stored (ending on /) in_file_folder - folder in which the wave in-files are stored (ending on /) in_files - the wave in-files to be used for different b_type situations, is dic: b_type : wave_in_file field_folder - folder where b-field files are stored field_files - the b-field files list, file format: 2cols: x[mm] and B[T], no separator and no headers res_folder - the folder in which the results are to be stored (ending on /) wave_data_res_folder - Subfolder of res_folder where wave data is stored pics_folder - subfolder of res_folder were pics are stored res_summary_file - name of the summary_file to be written no_copy - list of file names not to be copied/moved after simulation from wave stage folder wave_ending_extract - file endings to move from wave stage folder after simu wave_ending_copy - file endings of files to be copied (not moved) wave_res_copy_behaviour: 'copy_all', 'copy_del_none' - only writes res_summary, 'copy_essentials' only copies whats needed for: flux, flux_dens, flux_dens_distri, power, files set in wave_files_essentials wave_files_essentials - essential files, if wave_res_copy_behaviour set to 'wave_res_copy_behaviour' only those files are stored zip_res_folder - truth value, zip or not zip results freq_low - lower frequency (actualy energy) of spectrum to calc [eV] freq_high - upper frequency (actualy energy) of spectrum to calc [eV] freq_num - number of frequencies to calc beam_en - beam energy in [GeV] current - current [A] pinh_w - pinhole width (horizontal-z) [mm] pinh_h - pinhole height (vertical-y) [mm] spec_calc - truth value: calc spectrum or not, if not, only trajectory is written to file pinh_x - position of pinhole along optical axis [m] pinh_nz - number of points in pinhole horizontally pinh_ny - number of points in pinhole vertically

- `def get_paras(self) :`

  Returns current parameters

- `def set_paras(self,wave_paras) :`

  Sets new parameters

- `def run_wave(self) :`

  Runs wave and writes results

- `def prepare_b_files_for_wave(self) :`

  In dependence of b_type, copies and formats the b-field files needed

- `def edit_wave_results(self) :`

  According to what is set in wave_res_copy_behaviour cleans the wave-stage folder and copies the desired files to their location, deletes non-desired files and zips the results

- `def extract_summary(self, folder) :`

  Extract summary information from wave run from files in folder folder and stores results in file my_paras['res_summary_file'] of folder

- `def create_wave_input(self, wave_paras) :`

  loads the in-file set in wave_paras, changes the properties according to other wave_paras props and copies the result to the WAVE program folder

- `def extract_wave_results(self, results, plot = False, en_range = [], xlim = [], ylim = [], save_data_w_pics = False) :`

  Extracts the desired results and returns a dataframe with them results is a list containing either of : 'traj_magn' : Extracts Trajectory and magnetic field data 'power_distr' : power distribution [W/mm to the 2] 'flux' : Flux 'flux_dens' : Flux Density 'flux_dens_distr' : Flux Density Distribution 'brill' : Brilliance plot - determines if loaded data is plotted or not en_range - for distribution: gives the energy range over which the data is integrated xlim and ylim set the limits of the distribution plots (in horizontal and vertical) to cut the plots to the actual dimension of the pinhole used, and for 2d plots only xlim is considered if given save_data_w_pics - if True, saves the data for each plot alongside the pic returns a list of the loaded dataframes

- `def load_plot_stokes_distrib(self,folder, en_range = [], xlim = [], ylim = [], plot = True, save_folder = '',save_data_w_pics = 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

- `def create_wave_instance(instance_type) :`

  Takes the type of instance, instance_type, of wave that is returned. Possible valus are: 'wave_from_by' 'wave_from_byz' 'wave_from_bxyz' Returns instantiated instance of class

### 3.0.7 eds_fields.py

Example script file showing the usage of wavepy

### 3.0.8 annas_9th.py

Example script file showing the usage of wavepy

### 3.0.9 wavepy_spectrum_example_script.py

Example script file showing the usage of wavepy

### 3.0.10 wavepy_b_field_example_script.py

Example script file showing the usage of wavepy's b-field functionality