PyNEC-Utilities

Release 0.1.alpha

Electro707

Oct 02, 2023

CONTENTS

CHAPTER

ONE

TABLE OF CONTENTS:

1.1 *pynec-utilities* Command Line Tool

This package also includes a command line utility to run simulations on an existing .nec file. This utility is invoked with *pynec-utilities*.

This command line tool allows for the following:

- Plot a 3D radiation pattern of the antenna

```
pynec-utilities FILE.nec plot-3d
```

1.2 *pynec-utilities* Examples

The following are some examples of how to use this utility

1.2.1 Simple Dipole 3D Simulation

The following code will plot a simple vertical dipole

```
# Import the package
import pynec_utilities
# Create a PyNECWrapper class object
dipole_sim = pynec_utilities.PyNECWrapper()
# Create the antenna
w_id = dipole_sim.add_wire([0, 0, -1], [0, 0, 1], 0.03, 36)
# This command MUST be ran after creating the antenna's geometry
dipole_sim.geometry_complete()
# Add an excitation in the dipole wire at the 18th segment, which is half-way
dipole_sim.add_excitation(w_id, 18)
# Set the frequency to simulate at a 144Mhz
dipole_sim.set_single_f(144)
# Calculate the antenna radiation pattern with 36 segments for theta and phi
dipole_sim.calculate(36)
# Plot the 3D radiation pattern
dipole_sim.plot_3d_radiation_pattern()
```

1.3 API Reference

1.3.1 Python Documentation

class pynec_utilities.**PyNECWrapper**

The PyNEC Utilities / Wrapper

class LoadingType(*value*)

The loading type enumeration. This is used for the *PyNECWrapper.add_loading()* function.

short_all = -1

series_rlc = 0

parallel_rlc = 1

series_rlc_per_m = 2

parallel_rlc_per_m = 3

impedance = 4

wire_conductivity = 5

class GoundType(*value*)

The ground type enumeration. This is used for the *PyNECWrapper.add_ground()* function

null = -1

reflection = 0

perfect = 1

finite_norton = 2

import_file(*file_name: str*, *do_calculation: bool = False*)

Imports a .nec file as the antenna model.

Parameters

- **file_name** (*str*) – The name of the .nec file
- **do_calculation** (*bool*) – Whether to execute the simulation, or leave it to the user

Returns

The arguments that can be given to an RP card with necwrapper.nec.rp_card(*returned_value)

add_wire(*coords_1: list*, *coords_2: list*, *wire_rad: float*, *numb_segments: int*, *manual_wire_id: int | None = None*) → int

Adds a wire to the antenna's geometry

Parameters

- **coords_1** (*list*) – The start coordinates of the wire as a list [x0, y0, z0]
- **coords_2** (*list*) – The start coordinates of the wire as a list [x1, y1, z1]
- **wire_rad** (*float*) – The radius of the wire
- **numb_segments** (*int*) – The number of segments to split the wire into for the simulation
- **manual_wire_id** (*int, optional*) – A geometry ID instead of it being auto assigned

Returns
The geometry ID of the created wire

add_arc(*radius: float*, *start_angle: float*, *end_angle: float*, *wire_radius: float*, *numb_segments: int*, *manual_arc_id: int | None = None*) → int

Adds an arc to the antenna's geometry

Parameters

- **radius** (*float*) – The radius of the arc
- **start_angle** (*float*) – The start angle for the arc
- **end_angle** (*float*) – The end angle for the arc
- **wire_radius** – The radius of the wire making up the arc
- **numb_segments** (*int*) – The number of segments to split the wire into for the simulation
- **manual_arc_id** (*int, optional*) – A geometry ID instead of it being auto assigned

Returns
The geometry ID of the created arc

geometry_complete(*is_gound_plane: bool = False*, *current_expansion: bool = True*)

Call this function when done with making the geometry

Parameters

- **is_gound_plane** (*bool, optional*) – Whether to add a ground plane to the simulation
- **current_expansion** (*bool, optional*) – Whether to use current expansion or not if there is a ground plane

add_excitation(*wire_id: int*, *place_seg: int*)

Adds an excitation source.

Parameters

- **wire_id** (*int*) – The WireID of the wire to apply the exitation on
- **place_seg** (*int*) – The segment of the wire to place the exitation on

add_ground(*gn_type:* GoundType, *radials: int = 0*, *dielectric_constant: float = 0*, *conductivity: float = 0*)

Adds a ground. See the NEC's GN for more details

Parameters

- **gn_type** (*GoundType*) – The ground type
- **radials** (*int*) – The number of radials
- **dielectric_constant** (*float*) – The ground's dielectric constant, used in *gn_type=reflection*
- **conductivity** (*float*) – The ground's conductivity, used in *gn_type=reflection*

coordinate_transform(*rot_x: float = 0*, *rot_y: float = 0*, *rot_z: float = 0*, *trans_x: float = 0*, *trans_y: float = 0*, *trans_z: float = 0*, *start_move_segment: int = 0*, *tag_increment: int = 0*, *numb_new_struct: int = 0*)

Apply a coordinate transform. See the NEC's GM for more details

Parameters

- **rot_x** (*float, optional*) – Apply rotation of this degree along the x axis

- **rot_y** (*float, optional*) – Apply rotation of this degree along the y axis
- **rot_z** (*float, optional*) – Apply rotation of this degree along the z axis
- **trans_x** (*float, optional*) – Apply translation of this distance along the x axis
- **trans_y** (*float, optional*) – Apply translation of this distance along the y axis
- **trans_z** (*float, optional*) – Apply translation of this distance along the z axis
- **start_move_segment** (*int, optional*) – The start segment ID to select to move
- **tag_increment** (*int, optional*) – The increment of the new structures if tag_increment is not zero
- **numb_new_struct** (*int, optional*) – The number of new structures to generate

add_loading(*loading_type: LoadingType*, *wire_id: int*, *start_seg: int*, *end_seg: int*, *resistance: float | None = None*, *capacitance: float | None = None*, *inductance: float | None = None*, *reactance: float | None = None*, *conductivity: float | None = None*)

Add loading to the geometry.

See NEC's LD card for more details on the input

Parameters

- **loading_type** (LoadingType) – The loading type
- **wire_id** – The geometry ID to apply the loading to
- **start_seg** – The start segment in the selected geometry to apply the loading to
- **end_seg** – The end segment in the selected geometry to apply the loading to
- **resistance** – The resistance of the loading, if applicable
- **capacitance** – The capacitance of the loading, if applicable
- **inductance** – The inductance of the loading, if applicable
- **reactance** – The reactance of the loading, if applicable
- **conductivity** – The conductivity of the loading, if applicable

Raises UserWarning: Raises this exception if the loading type and required parameters don't match

calculate(*n: float*, *theta_start: int = 0*, *phi_start: int = 0*, *theta_end: int = 180*, *phi_end: int = 360*)

Calculates the antenna

Parameters

- **n** – The number of segments for the delta and phi.
- **theta_start** – The starting angle for theta. Defaults to 0
- **phi_start** – The starting angle for phi. Defaults to 0
- **theta_end** – The ending angle for theta. Defaults to 180
- **phi_end** – The ending angle for phi. Defaults to 360

Note: If using a ground for the simulation, *theta_end* must be set to 90 as you cannot have a radiation pattern bellow ground.

---

set_single_f(*freq: float*)

Sets a single frequency for the calculation

Parameters
freq (*float*) – The frequency to simulate for, in Mhz

set_multiple_f(*min_f*, *max_f: float | None = None*, *n: int | None = None*, *step: float | None = None*)

Set multiple frequencies for the calculation

Parameters

- **min_f** (*float*) – The minimum simulation frequency, in Mhz
- **max_f** (*float*) – The maximum simulation frequency, in Mhz. Either this or step are required
- **n** (*int*) – The number of frequency steps to simulate for
- **step** (*float*) – The frequency step in Mhz. Required if max_f is not given

get_3d_radiation_surface(*freq_index: int = 0*) → *Radiation3DPatternSurface*

Get the radiation pattern data for a given frequency index

Parameters
freq_index (*int*) – The frequency index to get the radiation pattern data

Returns: *Radiation3DPatternSurface*

get_radiation_pattern(*freq_index: int = 0*) → *RadiationPatternData*

Gets the raw radiation pattern data from PyNEC, but with the gain return in linear values as opposed to db

Parameters
freq_index (*int*) – The frequency index to get the radiation pattern data

Returns: *RadiationPatternData*

get_2d_radiation_pattern(*freq_index: int = 0*, *elevation: float | None = None*, *azimuth: float | None = None*) → *Radiation2DPatternData*

Get the 2D radiation pattern data.

Parameters

- **freq_index** (*int*) – The frequency index for the radiation pattern. Defaults to zero
- **elevation** (*float*) – The elevation to set for the 2D radiation pattern
- **azimuth** (*float*) – The azimuth to set for the 2D radiation pattern

---

Note: You must set either *elevation* or *azimuth* to some angle value, but not both.

Returns: *Radiation2DPatternData*

get_all_freq_3d_radiation_surface() → List[*Radiation3DPatternSurface*]

Get 3D radiation pattern data for all frequencies simulated

Returns: A list of *Radiation3DPatternSurface*

get_all_frequencies() → list

Returns: A list of all frequencies simulated

plot_3d_radiation_pattern(*in_data:* Radiation3DPatternSurface | *None* = *None*, *freq_index: int* = *0*, *show: bool* = *True*) → *Graph3DRadiationPattern*

Function to plot the 3D radiation pattern of the antenna

Parameters

- **in_data** (*Radiation3DPatternSurface*) – A *Radiation3DPatternSurface* data set. If none, this function will calculate it using the simulated results
- **freq_index** (*int*) – The frequency index to plot the 3D radiation pattern for if *in_data* is not given
- **show** (*bool*) – Whether to show a plot of the 3D radiation plot to the user. Defaults to True

Returns

Graph3DRadiationPattern

plot_2d_radiation_pattern(*in_data:* Radiation2DPatternData | *None* = *None*, *freq_index: int* = *0*, *elevation: float* | *None* = *None*, *azimuth: float* | *None* = *None*, *show: bool* = *True*) → *Graph2DRadiationPattern*

Plots the 2D radiation pattern

Parameters

- **in_data** (*Radiation2DPatternData*) – A *Radiation2DPatternData* data set. If none, this function will calculate it using the simulated results
- **freq_index** (*int*) – The frequency index to plot the 3D radiation pattern for if *in_data* is not given
- **elevation** (*float*) – The elevation to set for the 2D radiation pattern
- **azimuth** (*float*) – The azimuth to set for the 2D radiation pattern

---

Note: You must set either *elevation* or *azimuth* to some angle value, but not both.

If *in_data* is given, you do not need to give *elevation* or *azimuth*

Returns

Graph2DRadiationPattern

static get_reflection_coefficient(*z: float*, *z0: float*) → float

Calculates the reflection coefficient

Partially copied from https://github.com/tmolteno/python-necpp/blob/master/PyNEC/example/antenna_util.py

Parameters

- **z** – The impedance of the device
- **z0** – The characteristic impedance

Returns

The reflection coefficient

calculate_vswr(*z: float*, *z0: float*) → float

Calculates the VSWR of a system

Parameters

- **z** – The impedance of the device
- **z0** – The characteristic impedance

Returns
The VSWR

get_vswr(*z0: float = 50.0*) → Tuple[list, list]

Gets the VSWR for the given calculation frequency or frequencies.

Parameters
z0 (*float*) – The impedance to calcular the VSRW over. Defaults to 50 ohms

Returns
A tuple 2 lists, one for frequencies and the other for the VSWR

plot_swr(*z0: float = 50.0*)

Plots the VSWR of the antenna

Parameters
z0 (*float*) – The impedance to calcular the VSRW over. Defaults to 50 ohms

add_antenna_to_axis(*ax: Axes*)

> **Warning:** Work-In-Progress Function

Adds surface plots for the antenna wires and elements to a MatPlotLib axis

class pynec_utilities.**RadiationPatternData**(*thetas: array | None = None*, *phis: array | None = None*, *gains: array | None = None*, *freq: float | None = None*)

A dataclass containing all data related to a 3D radiation pattern

thetas: array = None

A list of thetas

phis: array = None

A list of phis

gains: array = None

A 2D array containing the gains with an index of [theta, phi]

freq: float = None

The frequency for this radiation pattern data

class pynec_utilities.**Radiation3DPatternSurface**(*X: array | None = None*, *Y: array | None = None*, *Z: array | None = None*, *N: array | None = None*, *gains: array | None = None*, *freq: float | None = None*)

A dataclass for a 3D radiation pattern data used for a surface plot

X: array = None

An array of X data points for the radiation pattern

Y: array = None

An array of Y data points for the radiation pattern

Z: array = None

An array of Z data points for the radiation pattern

N: array = None

A array of normalized distance from the origin for any given X,Y,Z data point. Used for coloring the face

gains: array = None

A 2D array of all gains

freq: float = None

The frequency for this radiation pattern data

class pynec_utilities.**Radiation2DPatternData**(*plot_theta: array | None = None*, *plot_radius: array | None = None*, *constant_elevation: float | None = None*, *constant_azimuth: float | None = None*, *freq: float | None = None*)

A data class for a 2D radiation pattern

plot_theta: array = None

A list of angles for a polar plot

plot_radius: array = None

A list of radius for a polar plot

constant_elevation: float = None

If not None, this is the constant elevation for this data set

constant_azimuth: float = None

If not None, this is the constant azimuth for this data set

freq: float = None

The frequency for this radiation pattern data

class pynec_utilities.**Graph3DRadiationPattern**(*in_data: Radiation3DPatternSurface | List[Radiation3DPatternSurface]*, *rotate: bool = False*, *elevation: float = 30*)

A class for plotting 3D radiations patterns

log_tick_formatter(*val*, *pos=None*)

Internal formatter to format ticks in dB

static show()

Shows the plot

export_to_gif(*file_name: str*)

Exports the animation into a GIF

Parameters
file_name – The export file name

static export(*file_name: str*)

Calls Matplotlib's *savefig* function to export the plot

Parameters
file_name – The export file name with the desired extension

export_to_mp4(*file_name: str*)

Exports the animation into an MP4 file

Parameters
file_name (*str*) – The export file name

static export_to_latex(*file_name: str*)

Exports the plot in a .pgf file This function is experimental in the sense it must be called last, otherwise future plotting may not be possible

Parameters
file_name (*str*) – The export file name

class pynec_utilities.**Graph2DRadiationPattern**(*in_data:* Radiation2DPatternData | *List[*Radiation2DPatternData*]*)

A class for plotting 2D radiations patterns

static show()

Shows the plot

export_to_gif(*file_name: str*)

Exports the animation into a GIF

Parameters
file_name (*str*) – The export file name

static export(*file_name: str*)

Calls Matplotlib's *savefig* function to export the plot

Parameters
file_name (*str*) – The export file name with the desired extension

export_to_mp4(*file_name*)

Exports the animation into an MP4 file

Parameters
file_name (*str*) – The export file name

static export_to_latex(*file_name: str*)

Exports the plot in a .pgf file This function is experimental in the sense it must be called last, otherwise future plotting may not be possible

Parameters
file_name (*str*) – The export file name

PYTHON MODULE INDEX

p

INDEX